

Managing Dynamic Services: A Contract Based Approach to a Conceptual Architecture

Alexander Keller, Gautam Kar, Heiko Ludwig, Asit Dan, Joseph L. Hellerstein
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598, USA
{alexk|gkar|hludwig|asit|hellers}@us.ibm.com

Abstract

This paper describes a novel contract based approach for defining, deploying, monitoring and enforcing service level agreements (SLA) in a dynamic e-Business environment. The current trend in application service delivery is to move away from tightly coupled systems towards structures of loosely coupled, dynamically bound systems with long and short business relationships. It appears highly likely that the next generation of e-Business systems will consist of an interconnection of services, each provided by a possibly different service provider, that are coupled together to offer an end to end service to a customer. Such an environment, which we call Dynamic e-Business (DeB), will be governed by dynamically negotiated electronic contracts between service providers and service customers. From a management viewpoint, this development poses new challenges, such as contract based provisioning of management systems, monitoring and violation detection of dynamically agreed upon QoS parameters, problem determination and resolution, according to the terms and conditions specified in the contract. This paper proposes a management architecture for specifying, deploying and monitoring service contracts in a DeB environment, with a view to providing a basis for SLA management.

Keywords

Service Management, Contract, SLA, dynamic e-Business, Inter-Domain Management

1 Introduction

The widespread use of the Internet within the business community is leading to a fundamental change in the way that services are being delivered by service providers and consumed by service customers. The past year, in particular, has seen an extensive amount of activity on research and technical specifications that enable the development of interoperable e-Business interactions, both B2C and B2B, using the Web and the Internet as the underpinning technologies. Standards and initiatives such as ebXML [6], UDDI [20] and Web Services [13] are gaining increasing acceptance. A trend that is beginning to emerge from this activity is the development of an environment where service customers and service providers can locate each other over the Internet, negotiate terms and conditions of business electronically, connect with each other dynamically, transact business and tear down their relationship when it is no longer needed. We call this development **Dynamic e-Business (DeB)**; in other work [9], it is also referred to as virtual enterprise. A DeB consists of a variety of **dynamic services**, offered by different service providers

and selected on-demand, that cooperate for a short period of time. One of the very important elements of dynamic e-Business is an electronic contract that describes the role of the various parties involved in a DeB environment and the service level agreements (SLA) that are negotiated between them. Considerable research has been ongoing on the content and structure of electronic contracts [16, 9]; however, existing work focuses often on business-level interactions, thus on an abstraction level that is too high for the purposes of (technical) service management. On the other hand, work in the area of Service Management [21, 3] often fails to take the business impact of SLAs into account.

The approach described in this paper attempts to combine both research areas and leverage their strengths; it is based on the premise that in DeB environments, management in general – and SLA management in particular – will be driven by the constituent elements of a contract. We present an object oriented model for a contract and use it as the basis for developing an architecture for representing, provisioning and monitoring the elements of a contract in a dynamic e-Business environment. In particular, our architecture is geared towards building management services that allow the flexible definition and measurement of QoS parameters, service level guarantees and detection of violations during the interaction between a customer and a service provider.

The paper is structured as follows: Section 2 introduces the management requirements in a DeB environment through an example scenario. Section 3 depicts the role of a contract and its implications in a DeB. Section 4 introduces a model for a contract, and provides details on the provisioning and deployment of a contract. A management architecture for detecting contract violations in a DeB is also presented in this section. Section 5 concludes the paper with an outlook on the future development of this area.

2 Dynamic Services: Elements of Dynamic e-Business

In order to set the stage for the rest of the paper, we will describe a scenario that will illustrate the formation of a DeB. Our example is directed towards a small to medium size business that wishes to outsource its business process (entirely or in part) to various service providers. The parties that participate in a DeB are described below.

2.1 Outsourcing an Internet Storefront

We assume a dynamic e-Business infrastructure for a *Storefront Owner* that wishes to construct and operate an electronic storefront for selling goods. Figure 1 depicts a typical configuration where the business process associated with the operation of an Internet storefront has been set up to be hosted by a number of different service providers. In the example shown, the business process consists of a fairly straightforward flow: catalog presentation, shopping cart handling, user account maintenance and profiling, credit and payment processing, and shipment. Potentially each of these sub-processes could be outsourced to a *Service Provider*, depending on the ability of the Storefront Owner to maintain the IT infrastructure needed for supporting them. The Storefront Owner is then a customer of a *Service Integrator*; both negotiate and sign an electronic contract (a.k.a. Service Level Agreement, SLA) that specifies the guaranteed quality of the subscribed services and the compensating actions in case of a violation. Additional players are needed when service customers buy services to be consumed by third parties. In our

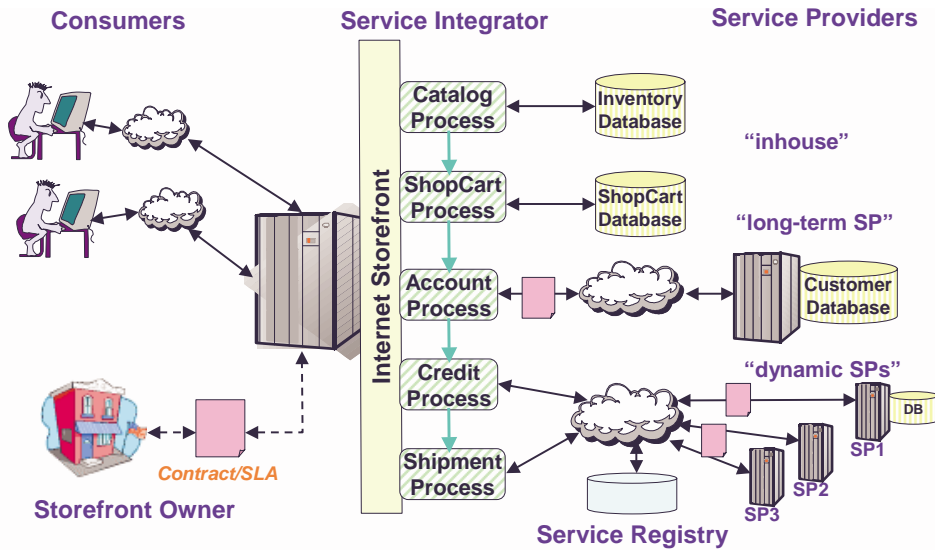


Figure 1: Architecture of an Internet storefront with multiple Service Providers

example, the storefront is outsourced by a Storefront Owner who intends the storefront to be used by its *Consumers*. In this case, the customer has neither a sophisticated infrastructure to supervise the service nor does he access the services from his environment on a regular basis. A typical real-life example for such a storefront service offering, which caters to Small and Medium Businesses (SMB), is *Yahoo! Store* [22].

In our scenario, the catalog representation and shopping cart handling processes are provided “inhouse” by the service integrator. However, a service integrator may choose not to implement all the services needed to support the business processes himself, but to subcontract parts of the fulfillment to other service providers. Each outsourced process is governed by an electronic contract, formed by negotiations. In our example, account maintenance, credit processing and shipment are subcontracted to other service providers. The relationship of a service integrator to its service provider is the same as the relationship of a service customer (here: the Storefront Owner) to a service provider. Note that the service relationships are bilateral, i.e., the relationship between a service customer and service integrator is decoupled from the relationship between service integrator and actual service provider. The mapping of subcontracted services to the service provided to the customer is done internally by the service integrator. By treating each service relationship and the corresponding management independently, this model of contractual relationships supports value chains of arbitrary depth, which may be implemented by a hierarchy of service providers. In the case of our scenario, the Service Integrator has chosen to outsource the maintenance of the customer database (needed for the fulfillment of the account maintenance business process) to another service provider “long-term SP”, because such a “classical” outsourcing contract is usually valid for a longer period of time.

Dynamic Service Providers (depicted in the lower right part of figure 1) enter the scene if we consider the emerging Web Services [13] and UDDI [20] architectures, which define mechanisms for advertising, finding, and binding to services over the Internet. While UDDI defines the architecture of a *Service Registry* and the mechanisms to access them, Web Services specifies a component model for describing and invoking services over the Internet. UDDI and Web Services are the building blocks that enable a Service Integrator to perform very late binding to other Service Providers, in the extreme case on a per-transaction basis. In our example, the Credit and Shipment business processes are fulfilled by Dynamic Service Providers.

2.2 Management Service Providers in a Dynamic e-Business

Extending the DeB with service management capabilities introduces additional parties to monitor and enforce the negotiated electronic contracts; these parties may be selected on-demand, too, and are subject to the dynamics of the underlying DeB. A *Management Service Provider (MSP)* is the most common example of an entity that may be selected by e.g., Storefront Owner and Service Integrator to oversee the electronic contracts between the various customer/provider relationships. Typical functions performed by an MSP are:

- Provisioning the management functionality such that the various MSPs are instructed to measure the relevant metrics in support of the agreed upon SLAs.
- Performance data reporting; Keynote Systems [12] is an example of such an MSP.
- Contract monitoring for contract violation detection.
- Reporting of contract violations to the various interested parties.
- Problem determination and resolution.

These functions can be performed either by one or by multiple entities. For instance, provisioning and problem determination could be done by one entity that can be an integral part of the Service Integrator, whereas contract monitoring and SLA violation detection could be done by an agreed upon third party, such as an MSP. This DeB environment poses a number of management challenges, which an MSP needs to address:

- interpretation of a contract to set up monitoring policies in support of negotiated QoS parameters,
- provisioning of management systems in the customer and service provider environments to enable SLA monitoring. This step needs to be done dynamically as new service providers replace or join existing ones in the DeB.
- detection of contract violations, finding the source of the violation and initiating corrective measures.
- problem determination, which is a difficult enough task in large distributed systems, becomes even more so in the context of dynamic e-Business. This is because of the multiple organizational boundaries involved in the completion of an e-Business

transaction. In the event of a problem, the management systems of the different organizations have to cooperate by exchanging relevant information (e.g., monitored data, log file extracts, etc.) and map the external problem report to their internal resource data in order to find the root cause of the problem.

In this paper we concentrate on providing an approach for contract based management. The following sections introduce the structure and model of a contract and describe a management architecture for violation detection and contract deployment.

2.3 Related Work

The Application Service Provider (ASP) model of making software accessible as a service has been gaining increasing acceptance over the past years; however, current ASP customer/provider relationships tend to be fairly static and restricted to a pair of customers and providers because the infrastructure for selecting services on-demand from multiple service providers is not yet in place. [19] gives an overview over the current state of the art in application service provisioning.

The dynamic selection of services and the automated setup of service relationships is the subject of intensive work in the recent years, e.g., in the Web Services Architecture [13], tpaML based business protocol deployment [5] and the CrossFlow project that supports dynamic integration of business processes [9, 15]. However, these approaches do not consider establishing a service management relationship along with the steps of provisioning the service. There are a number of approaches to formal contract languages in general [16, 18] that do not provide constructs to define management relationships. Work on electronic SLAs for application hosting [11] specifies QoS associated with a service; however, it lacks the flexibility in SLA metric definition and doesn't distinguish between the different kinds of functionality an MSP can provide.

Work in the area of service management for multi-party environments [14, 10], first described in [3], often lacks adequate formal SLA representations to facilitate the automated negotiation, deployment and enforcement of contracts. In addition, work on the assessment of the SLAs' business impact on an organization has been carried out in the recent past and focuses on telecommunication environments [4, 17].

3 Contractual Relationships and Roles

3.1 Contractual Scope of Services and their Management

In this section we introduce a model of the contractual relationships that helps us to understand and structure the involvement of multiple parties in the delivery of a service and its management. Figure 2 illustrates the main elements of our model based on our Internet storefront scenario, described in section 2.1.

In general, a contract defines the relationships between a number of parties for delivering and managing a service: All rights and obligations of the parties are defined in the contract, such as the obligation to deliver a service at the specified quality and the obligation to pay for it. This also includes the interaction relationship between the contracting parties, e.g., the right to send a measurement probe and the obligation to make measurement data available. We call the set of rights and obligations the *scope of a contract*.

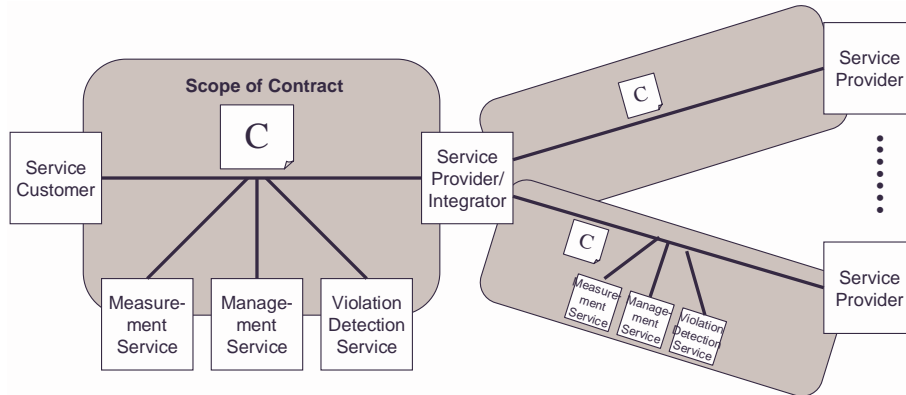


Figure 2: Contractual scopes in a dynamic e-Business environment

There is no relationship between two parties unless it is on the basis of a contract (We ignore the establishment and dismantling of the contractual relationship at this point).

From the point of view of service management, the scope of the contract encompasses two important aspects: (1) The bilateral service relationship between a service provider and its customer and (2) the service management, which may include additional parties that contribute in various roles.

The *service relationship* is defined in a *bilateral contract* between a service provider and a service customer. It defines the service that the service provider must deliver and how the fulfillment of the contract is going to be managed. This includes the specification of the QoS parameters, the way they are measured, their guaranteed values and the punitive action that is entailed by a guarantee violation e.g., the crediting of a penalty. A service provider is fully accountable for the delivery of the service as specified by the QoS parameters.

The *service management aspect* complements the service relationship. It defines the necessary interactions to measure the relevant QoS parameters, exchange measured values, determine problems and launch corrective actions. The contracting parties may decide to *include other parties* in the monitoring and management of their contractual relationship. In addition to the (mandatory) service provider and service customer, third parties in additional roles may be part of the contractual relationship: They perform activities that providers and customers prefer not to do or are unable to carry out by themselves e.g., the management of the QoS parameters specified in a contract. Depending on the particular environment, one or more MSPs might embody various roles, depending on the needs of the respective parties and the viability of a particular role being run as an independent business. The following services are typical roles of an MSP:

- **Measurement Service:** This service measures QoS parameters such as availability, throughput or response time from outside the service provider, e.g., by probing or interception of client invocations. A measurement service may measure all or a subset of the QoS parameters. Multiple measurement services may be involved.

- **Violation Detection Service:** This service obtains measured values of QoS parameters from the service provider or a measurement service and tests them against the guarantees given in the contract. This can be done each time a new value is available, or periodically.
- **Management Service:** The purpose of this service is to execute corrective actions on behalf of the managed environment if the Violation Detection Service discovers that a term of the contract has been violated. While such corrective actions are limited today to opening a trouble ticket or sending an event to the provider's management system, we envision this component playing a crucial role in the future by acting as an automated mediator between customer and provider, according to the terms of the contract. This includes the submission of proposals to the management system of a service provider on how a performance problem could be resolved (e.g., proposing to assign a different traffic category to a customer if several categories have been defined in the contract).

Despite the fact that a multitude of parties may be involved in providing a service, these interactions may be broken down into chained customer/provider relationships. Every interaction therefore involves only two parties, a customer and a provider. Consequently, we do not see a need for multi-party contracts [18] in a DeB (i.e., contracts that are negotiated and signed by more than two parties), which do not provide enough value to justify their added complexity.

3.2 Implications of Contract based Management

The foregoing discussion makes it clear that dynamic e-Business has major implications on its management, which make Service Management very different from traditional Enterprise Management. First, virtual enterprises can be formed (and dissolved) on a per-transaction basis, thus implying dynamics several orders of magnitude larger than found in today's corporate networks, whose topology tends to be fairly static. The second major issue comes from the privacy concerns of the various providers: A service provider is, in general, neither interested in disclosing which of his business processes have been outsourced to other providers, nor the names of these providers. DeB customers, on the other hand, will not necessarily see a need anymore to know the exact reason of a performance degradation as long as their service provider is able to take appropriate remedies. Consequently, the goal of "End-to-End Management", which has been the target of several enterprise management efforts, becomes unachievable in a DeB environment spanning multiple organizational domains.

In order to develop a generic model for service contracts, we have analyzed close to three dozen state-of-the-art SLAs currently used throughout the industry in the areas of application service provisioning (ASP) [2], web hosting and information technology (IT) outsourcing. The first observation is that a large part of the current contracts deals with legal (non-IT related) terms and conditions, such as the scope of work, the legal responsibilities and proprietary rights of the parties, or the modes of invoicing and payment. These terms and conditions are best kept outside the actual IT-related contract because they cannot be observed/enforced by a management system.

Another observation is that every analyzed contract contains (in a more or less straightforward way) the involved parties, the QoS parameters, the raw metrics used as

input to compute the QoS parameters, the algorithm for computing the QoS parameters, the service guarantees and the appropriate actions to be taken if a violation of these guarantees has been detected. Therefore, the structure of a contract is very similar among our (statistically small, but representative) samples, which makes these contract artifacts suitable candidates for inclusion in a generic model representing service contracts.

Finally, and surprisingly enough, almost all ASP contracts focus on a single QoS parameter, namely **Availability**; other important QoS parameters, such as assigned bandwidth, response time or throughput are almost never mentioned. We believe this reflects the current early stage of ASPs, since availability is the most fundamental metric. However, it turns out that measuring availability does already represent a challenge on its own because the definitions of this QoS parameter come in various forms and shapes: some ASPs focus on the infrastructure to define service availability (“user(s) being able to establish a TCP connection to the appropriate server”), while others refer to the application that implements the service (“Customer’s ability to access the software application on the server”). Still others rely on the results obtained from monitoring tools (“the application is accessible if the server is responding to HTTP requests issued by a specific monitoring software”), while another approach uses elaborate formulas consisting of various metrics, which are sampled over fixed time intervals. These base clauses are then usually annotated with exceptions, such as maintenance intervals, weekend/holiday schedules, or even the business impact of an outage (“An outage has been detected by the ASP but no material, detrimental impact on the customer has occurred as a result”). While we do not intend to comment on the usefulness of some of the clauses or the ease of proving their violation in an objective way, we would like to stress the point that despite the fact that all the clauses refer to the same QoS parameter (“Availability”), the only commonality among these clauses is — their diversity! The important implication is that a suitable contract model must not constrain the parties in the way they formulate their clauses but instead allow for a high degree of flexibility. In the aforementioned examples, a management tool that implements only a non-modifiable “textbook definition” of availability would not be considered helpful by today’s ASPs and their customers.

Nevertheless, while the nature of the clauses may differ considerably among different contracts, the general structure of all the different contracts remains the same. This implies that there is a way to come up with a unified contract model, which can be applied to a multitude of bilateral customer/provider relationships. The following section discusses our efforts towards finding such a flexible contract model.

4 Contract Model and Management Architecture

We will now describe our approach to developing a generic model for service contracts (in section 4.1) and its operational issues, such as the detection of violations and the contract deployment in sections 4.2 and 4.3, respectively.

4.1 A flexible Model for describing Service Contracts

The discussion in sections 2.1 and 3.2 has shown on the one hand that the general structure of a contract is the same for all interactions between a customer and a service provider even if the customer/provider relationships are nested to form a hierarchy. On the other

In addition, the algorithm for computing QoS Parameters from Metrics is captured in the contract; the class `MeasurementDirectiveDefinition` (not depicted in the figure due to space constraints) holds this information. `Guarantee` contains the threshold definitions for the `QoSParameterDefinitions`. These two classes are connected by the association `GuaranteeParmDef`.

The most important artifact of a service contract is the `Contract` itself. Since a `Contract` is composed of all the previously mentioned artifacts – even other contracts – we can aggregate all of them by means of the aggregation `ContractCSDComposite`. This simple, but elegant construct of an aggregation combined with an inheritance hierarchy is known as the “Composite” design pattern, described in [7]. Note that a contract can, in addition to aggregating other contracts, also be associated with other contracts (by means of the association `AssociatedContract`), thus making it possible for a management system user to navigate from one contract to another. All these classes contain the definitions of the contract artifacts, i.e., the content of the contract document. This simple model meets the requirement of representing the definition aspects of a service contract, as described in section 3.

The second important aspect of our model takes care of the fact that we need to relate the definitions contained in the contract to the instances of the elements that represent the actual values of the QoS Parameters, Metrics, Algorithms and Guarantees. This is the purpose of all the classes depicted in the lower part of the figure.

`RawMetrics` are the values retrieved directly from the managed resources residing in the service provider’s administrative domain. They can be considered as having no notion of time; typical examples are counters and gauges, and resource status information. `CompositeMetrics` are created by combining several raw (or other composite) metrics according to a specific algorithm, such as averaging one or more metrics over a specific amount of time. This computation is represented by the class `MetricAlgorithm`, which retrieves its input parameters (instances of `RawMetric` or `CompositeMetric`) by traversing the association `InParameters`. The result of the computation is a `CompositeMetric`, as can be seen from the association `OutParameters`. Such computations are usually being done within the service provider’s domain but can be outsourced to a measurement service. Composite metrics are exposed by a service provider by means of a well-defined interface either to customers or a measurement service.

`QoSParameters` put the metrics available from a service provider into the context of a specific customer, according to the contract specification. In contrast to the previous metrics, every QoS parameter is associated with high/low watermarks, which enables the customer, provider, or a third-party violation detection service to evaluate the retrieved metrics whether they meet/exceed/fall below specific QoS objectives (`Guarantee`). As mentioned before, every QoS parameter and its permitted range are defined in the service contract. The actual comparison between the value of a `QoSParameter` and its `Guarantee` is carried out by a `MeasurementDirective`. This comparison is done by navigating the associations `ActualQoSParameter` and `GuaranteedParameter`, respectively.

Finally, we need to relate a specific `Customer` and a `Provider` to the `Service` in question, and also to all the computed and measured values. This is the purpose of the various associations in the lower left corner of the figure. The part dealing with the definition of appropriate corrective actions that need to be undertaken once the violation of a guarantee has been detected is omitted from the figure for the sake of brevity. How-

ever, the underlying concepts remain the same as for the elements relating to clauses and guarantees.

The model allows one to express that a customer requests customized data, i.e., he selects the metrics and QoS parameters, and also works properly in cases where a customer specifies the way by which data is collected. However, this is highly dependent on the extent a customer requires the customization of parameters exposed by the service provider (or measurement service) and how much he is willing to pay for it. This, in turn, depends on the degree of customization the provider is willing to apply to its metrics.

The probably biggest advantage of this model over the existing state of the art is the fact that our mechanism for recursive aggregation of metrics and QoS parameters into new QoS parameters allows the creation of measurable quantities not only at any service access point of the customer/provider hierarchy, but also within a single administrative domain. This is due to the fact that our model avoids to distinguish between customer-specific, provider-specific and side-independent artifacts (in contrast to e.g., [8]): A service provider may use this model to aggregate the various counters obtained from the instrumentation of its managed resources to expose aggregated metrics to a customer, which will then be used as input parameters for defining the QoS parameters in the contract clauses. A customer may then take the QoS parameters defined in the contract to aggregate them into higher-level business metrics that are meaningful to him. This procedure can be repeated across value chains of arbitrary length, thus taking into account nested customer/provider relationships and eliminating the disconnect between business-level and IT-level metrics.

4.2 Violation Detection

The architecture defined in this section describes the structure and behavior of the contract management system and its relationship to the service production system. It also defines the runtime relationships to external services that are involved in the monitoring and management of a contractually defined service. Figure 4 outlines an example of a runtime architecture that involves an external Violation Detection Service and a Measurement Service. The architecture consists of two parts: (1) A *Service Production System*, which hosts or implements the service and (2) a corresponding *Contract Management System* that deals with the monitoring and management of the contractual obligations defined in the contract clauses.

The Service Production System is an application server that consists of a servlet engine, a monitoring and management interface and an administration console that facilitates local management tasks. The application server hosts the servlets that implement the service for the user e.g., servlets that expose their functionality as a Web Service.

The Contract Management System comprises the following functional components:

- *Contract Repository*: This component maintains the contracts of an organization. Once the contract establishment functionality has signed a new contract, this new instance of the contract model (described in section 4.1) is submitted to the contract repository. The repository also maintains pointers to all other functional components and external partners that are involved in the management of the contract (referred to as “link” in the figure). It is the entry point for administrative activities associated with contracts.

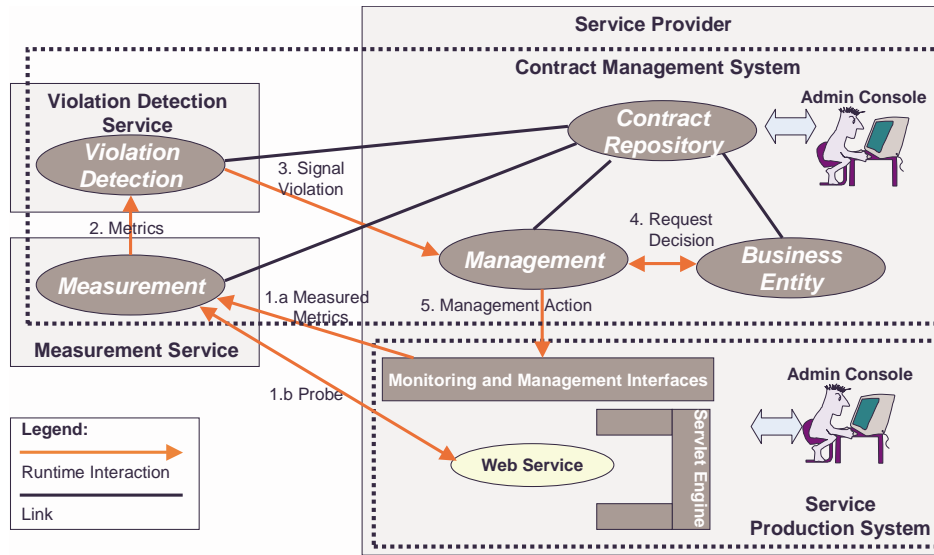


Figure 4: Violation Detection Architecture

- Measurement:** The responsibility of the measurement component is the collection of raw data that is relevant for QoS parameters. Also, the measurement component aggregates the raw data into higher-level metrics that are used as QoS parameters in the guarantees of the contract. To collect data, the measurement component can either read data from the service production system through its monitoring and management interface (interaction 1.a in the figure) or it can probe the service independently of the management interface of the service production system (1.b). This enables independent measurement of data and metrics that indicate how the performance of a service is experienced from outside the service provider organization.
- Violation Detection:** This component supervises the guarantees defined in the contract. It retrieves metrics relevant for the QoS parameters from the measurement component (2.) and evaluates the guarantees with the current set of data. This can either be performed every time new data is available, or periodically. If a violation occurs, the violation detection notifies the management component (3.).
- Management:** Upon reception of a violation notification, the management component determines the problem and derives a proposal for a management action that remedies the cause(s) of the contract violation. However, the management component does not implement the business logic to decide whether the management action is in the best interest of the service provider from a business point of view. Thus, it sends a decision request to the business entity (4.), which embodies this business logic. When a decision is made to perform the management action, the management component triggers the appropriate function of the monitoring and management interfaces of the service production system.

- *Business Entity*: The business entity component performs the decision-making about which management action to take, if there is a choice, and when to execute it. For example, the financial terms of the contract could specify a low penalty if the agreed downtime is not much longer than a pre-defined value. The business entity can then decide to allocate resources to fulfill another contract, which is to be serviced in parallel, that entails a higher penalty for violation. This component can be implemented in different ways e.g., by exposing a user interface and involving employees in the decision-making process.
- *Deployment*: A component not involved at the runtime of a contractual service is the deployment component. This component provisions the service and sets up the contract management infrastructure, which comprises both organization-internal and external components. The procedure used by the deployment component is detailed in section 4.3.

Some functional components of the contract management system can be implemented by external service providers, thereby extending the contract management system beyond the scope of a single organizational domain. Good candidates are measurement, violation detection and, potentially, management services, as the business entity is invoked prior to the execution of a management action.

The contract management system of the service customer is similar to that of the service provider. Service customers can run their own measurement and violation detection functionality or outsource it – partially or completely – to an external service shared with the service provider. In any case, service customers require their own contract repository. Since service customers do not run a service production system, they do not require the management functionality.

4.3 Contract Deployment

Contract deployment refers to the process of provisioning the service and setting up a contract monitoring and management infrastructure. Figure 5 explains this process by depicting a scenario in which a service provider deploys a contract. In addition to internal components, a third-party *Measurement Service* and a *Violation Detection Service* are also involved in the measurement and enforcement of the contract. The service provider runs his *Service Production System* and the internal components of the corresponding *Contract Management System*. In a different configuration there could be no external roles involved – all functionality implemented by the provider – or even multiple measurement and violation detection services.

The deployment procedure contains four steps, referring to interactions in Figure 5:

1. In a first step, the deployment component receives the contract from the contract repository. It decides how to provision the service in the service production system and which components to include in the contract management system. While external components are defined in the contract, there is some degree of freedom to decide which local components will be used for the contract management system and the provisioning of the service production system.

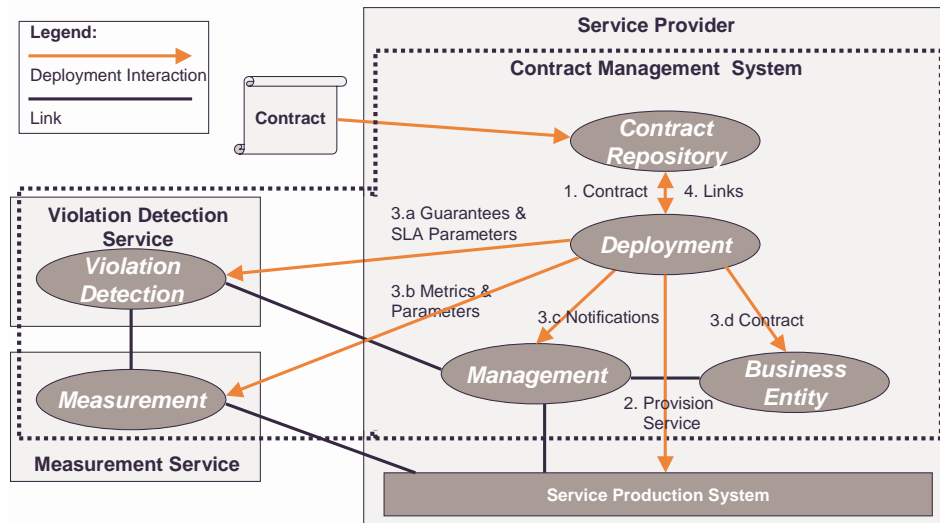


Figure 5: Contract deployment process

2. Subsequently, the service is provisioned in the service production system. Depending on the type of service, different procedures are performed and may even involve processes that need to be performed manually.
3. The third step is the setup of the components of the contract management system. The deployment component instantiates other necessary components and supplies them with the parts of the contract that are needed for them to perform their tasks. Different component types receive different contract artifacts.
 - 3.a** A violation detection component receives the guarantees and the parts of the QoS parameter definition that enables it to retrieve them from the measurement component.
 - 3.b** A measurement component retrieves the definition of the metrics to be measured and the corresponding measurement directives and metric algorithms.
 - 3.c** The management component needs to know which notifications of violations to expect and which actions should be taken.
 - 3.d** The business entity receives the complete contract to enable the necessary decision-making in case violations occur.

In addition to contractual information, internal components need further information to establish links to other internal components, which are not contractually defined.

4. In the final step, the deployment component reports all linking information back to the contract repository to facilitate the administration of the contract management system.

A service customer's deployment is a simplified version of a service provider's. Obviously, there is no service production system to provision, hence we do not need step two. Also, the deployment of all additional parties to the contract is triggered by the service provider. No involvement of the service customer is necessary.

5 Conclusion and Outlook

This paper presents the results of our research in a new and evolving area: Contract based management of dynamic services. At the core of our work is an object-oriented model for a bilateral contract that addresses the need for flexibility required by today's SLAs.

During the prototyping of the contract model and the various components presented in this paper, we were able to fully automate the steps of deployment, monitoring and violation detection of the SLA. However, the contract negotiation and establishment phases and the provisioning of the dynamic services may require manual intervention. Another issue that needs to be addressed before dynamic e-Business can become a reality is to enhance the functionality of the management service: Currently, its task is limited to obtaining and evaluating the events received from the violation detection component and to forward them to the provider's management system. In the future, we envision the management service as a focal point for inter-domain problem resolution: It would carry out problem determination tasks, perform root cause analysis and initiate corrective measures when a contract violation is detected. However, the fact that such measures need to be implemented across different organizational boundaries and with multiple related contracts makes this problem particularly challenging.

Finally, the paper has introduced the concept of third party measurement, violation detection and management services, which play an important role in monitoring and enforcing a contract; these services go beyond the functionality offered by traditional Management Service Providers. Here, important questions related to security and trust need to be addressed so that service providers are willing to expose their performance metrics to external measurement services.

References

- [1] N. Anerousis, G. Pavlou, and A. Liotta, editors. *Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management (IM 2001)*, Seattle, WA, USA, May 2001. IEEE Publishing.
- [2] ASP Industry Consortium. *White Paper on Service Level Agreements*, 2000.
- [3] P. Bhoj, S. Singhal, and S. Chutani. SLA Management in Federated Environments. In M. Sloman, S. Mazumdar, and E. Lupu, editors, *Proceedings of the 6th IFIP/IEEE International Symposium on Integrated Network Management*, pages 293–308, Boston, MA, USA, May 1999. IEEE Publishing.
- [4] B. Bhushan, M. Tschichholz, E. Leray, and W. Donnelly. Federated Accounting: Service Charging and Billing in a Business-To-Business Environment. In Anerousis et al. [1], pages 107–121.

- [5] A. Dan, D. Dias, R. Kearney, T. Lau, T. Nguyen, F. Parr, M. Sachs, and H. Shaikh. Business-to-Business Integration with tpaML and a B2B Protocol Framework. *IBM Systems Journal*, 40(1), February 2001.
- [6] *ebXML – Creating a Single Global Electronic Market*. <http://www.ebxml.org>.
- [7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
- [8] M. Garschhammer, R. Hauck, H.-G. Hegering, B. Kempter, M. Langer, M. Nerb, I. Radisic, H. Roelle, and H. Schmidt. Towards generic Service Management Concepts: A Service Model Based Approach. In Anerousis et al. [1], pages 719–732.
- [9] Y. Hoffner, S. Field, P. Grefen, and H. Ludwig. Contract-driven creation and operation of virtual enterprises. *Computer Networks*, 37:111–136, 2001.
- [10] G. Kar, A. Keller, and S.B. Calo. Managing Application Services over Service Provider Networks: Architecture and Dependency Analysis. In J.W. Hong and R. Weihmayer, editors, *Proceedings of the 7th IEEE/IFIP Network Operations and Management Symposium*, pages 61–75. IEEE Press, April 2000.
- [11] R. Kearney, R. King, M. Sachs, A. Dan, and D. Dias. Electronic Service Level Agreement (eSLA) for Application Hosting. IBM Research Report RC 22071, IBM Corporation, May 2001.
- [12] *Keynote – The Internet Performance Authority*. <http://www.keynote.com>.
- [13] H. Kreger. *Web Services Conceptual Architecture 1.0*. IBM Software Group, May 2001.
- [14] L. Lewis. *Managing Business and Service Networks*. Kluwer Academic Publishers, 2001.
- [15] H. Ludwig and Y. Hoffner. The Role of Contract and Component Semantics in Dynamic E-Contract Enactment Configuration. In *Proceedings of the 9th IFIP Workshop on Data Semantics (DS9)*, pages 26–40, 2001.
- [16] M. Merz, F. Griffel, T. Tu, S. Müller-Wilken, H. Weinreich, M. Boger, and W. Lamersdorf. Supporting Electronic Commerce Transactions with contracting Services. *International Journal of Cooperative Information Systems*, 7(4):249–274, 1998.
- [17] Open Service Marketplace White Paper. Version 1.1 telecom/00-06-05, Object Management Group, June 2000.
- [18] B. Schopp, A. Runge, and K. Stanoevska-Slabeva. The Management of Business Transactions through Electronic Contracts . In A. Camelli, A. Min Tjoa, and R.R. Wagner, editors, *Proceedings for the 10th International Workshop on Database and Expert Systems Applications*, pages 824–831, Florence, Italy, 1999. IEEE Computer Society Press.
- [19] L. Tao. Shifting Paradigms with the Application Service Provider Model. *IEEE Computer*, 34(10):32–39, October 2001.
- [20] UDDI Version 2.0 API Specification. Universal Description, Discovery and Integration, uddi.org, June 2001.
- [21] K. White. Definition of Managed Objects for Service Level Agreements Performance Monitoring. RFC 2758, IETF, February 2000.
- [22] *Yahoo! Store*. <http://store.yahoo.com>.