SALMON - An Architecture to Define, Store, Monitoring and Billing ISLAs in a Server Farm

Juliana Silva da Cunha (jsc@cin.ufpe.br)[1], Karen Appleby-Hougham (applebyk@us.ibm.com)[2], German Goldszmidt (german@us.ibm.com)[2] and Fabio Q. B. da Silva (fabio@cin.ufpe.br)[1]

ABSTRACT

Salmon (Service Agreement Levels for Monitoring Océano coNtracts) defines an architecture and prototype implementation of a system to specify and maintain Infrastructure Service Level Agreements (ISLAs). A contract is used to establish an ISLA between a customer and a service provider. Each contract includes multiple sections, such as report definition, violation policy descriptions, penalties for disruption of service and charging. Salmon will evaluate that the service provider has a sufficient number of resources to support the defined service level. Salmon will monitor the enforcement of the contract and will trigger the policy engine whenever a violation occurs. Contract violations are expressed as policies, which include a violation scenario, start and stop time, the monitor and an action that must be fired in order to calculate the violation penalty. The action is a procedure to correct the problem, and/or apply a monetary penalty on the service provider. A charging engine is responsible for the billing calculations. We address the problem of ISLA definition by using customer feedback and providing a flexible way to define and monitor the quality of service.

Keywords: Service Level Agreements, Service Billing, Network and System Monitoring

1. INTRODUCTION

Salmon defines an architecture to specify and maintain Infrastructure Service Level Agreements (ISLAs). It was developed as part of a larger project, Océano, which is a prototype of a highly available, scalable, and manageable infrastructure for an e-business computing utility [1]. It enables multiple customers to be hosted on a collection of shared resources. However, at any point of time, each resource is assigned for use to only a single hosted customer. That is, the hosting environment is divided into smaller, secure domains, each supporting one customer. These domains are dynamic: the resources assigned to them may be augmented when load increases and reduced when load dips. Océano manages the resources of the computing utility so that each customer has the resources necessary to provide a contracted level of service as specified by an Infrastructure Service Level Agreement (ISLA) [3]. ISLAs are defined using a Contract and then translated into system metrics that are monitored. Monitoring agents issue events when thresholds are exceeded or failures occur. Events are correlated to identify root causes [2].

The relationship between the customer and the provider begins with the definition of a contract that satisfies the service requirements. Ideally, the provider's infrastructure should offer the necessary resources, a fair pricing model and flexibility in dealing with changes.

This paper presents the approach used in Océano to define, store, monitor and bill ISLAs. A high-level contract, between the customer and the provider, specifies "business requirements" such as scenarios definition, appropriate QoS metrics, Infrastructure Service Level guarantees and a charging model. This contract is translated into a set of interpreted policies that are then monitored and enforced. A GUI interface is used to make updates in the contract, and to generate customized reports.

[1] Federal University of Pernambuco
[2] IBM – T.J. Watson Research Center

The rest of this paper is organized is follow: Section 2 describes the ISLA Contract, Section 3 describes the policy language that is used to represent the information specified in the contract; Section 4 presents the contract life-cycle process; Section 5 defines the architecture used to define, represent, monitor and charge ISLAs, and Section 6 discusses the prototype that was implemented.

2. THE ISLA CONTRACT

An Infrastructure Service Level Agreement is a contract between the provider and the customer that guarantees levels of services [12] and relates them to an economic model [4, 5, 6, 7]. The proposed contract supports the definition of partners, qualities of services, resources, violation policies, a charging model and reporting requirements. In this section we will present the Océano ISLA contract definition and the hierarchy that allows contract aggregation.

2.1. The Contract Definition

The contract template defined for Océano has 6 sections: Header, Customers, Scenarios, Violation, Charging and Report. These sections contain components that are purely informational and those that involve monitored elements. The focus of this work remains on monitored elements. A description of each section follows:

Informational

Header Section

The Header Section contains generic data about the contract, which is not monitored by the farm. Following is an example:

Contract Identification - Contract for ABC Company
Duration
    StartDate - 01/10/2001   EndDate - 02/10/2002
Version - 2.0
Level - Main
Responsible – John O’Connor
Role Players Definition:

RolePlayer 1	RolePlayer 2
Identification Name – Customer	Identification Name - Provider
Organization Name - ABC Company	Organization Name – Oceano Project
Main Contact	Main Contact
Name – Andrew White	Name – John O’Connor
Address – 99, ABC Road	Address – 99, XYZ Road
e-Mail – andreww@abc.com	e-Mail – john@abc.com
Telephone – (914)999-9999	Telephone – (914)555-5555
FAX - (914)888-8888	FAX - (914)666-6666

Services: Backup Service, Firewall

The contract level is used to build the contract hierarchy (see Section 2.2).

By default every contract has 2 role players: the customer and the provider. This section also includes a list of the services contracted by the customers, e.g., backup services.

Monitored Sections

Customer Configuration Section

This section has two purposes. The first is to configure the values that describe the customer segments. Each customer segment specifies different classes of traffic that may have specific ISLA requirements (ex: browsing traffic, e-commerce traffic). Some of the fields that must be filled are: the identification number, description, level, priority, port number, virtual IP address and server pool identification number, etc. For example:

```
CustomerID="11"
            CustomerSegID="9000111"
                    CustomerSegDS - Description of Segment 111
                    CustSegLevel - 1
                    VipAddr – 1.2.3.4
                    SegPortNo - 111
                    Priority - 1
                    ServerPoolID - 119000111
```

The second purpose is to specify the requirements that are defined for each customer segment. These requirements define the ISLA metrics that must be used in the load calculations and that characterize the desired QoS. The ISLA metrics chosen should be measurable and reliable. A more detailed explanation about the metrics is in [1]. Some examples are:

- **Active Connections/server** - The average number of active connections on a server.
- **Overall Response Time** - Average time it takes for any request to be processed.
- **Output Bandwidth** - Average number of outbound bytes per second per server.
- **DB Response Time** - Average time it takes for any request to be processed by the back-end DB.

Servers are allocated and deallocated based of threshold equations over these metrics. The following three parameters are used to control this process.

- **Maximum Server Load** – A value that is based on a combination of metrics that defines the point when the response time begins to deteriorate rapidly in response to load.
- **Minimum Server Load** – Arbitrary point below the maximum where the load curve is smooth.
- **Load Calculation Function** – This function is selected by the customers and is used to compute the load based on allocation and deallocation thresholds. These thresholds are a percentage of the minimum and maximum server load and are used to trigger the server allocation process.

ISLA Scenarios Definition Section

The set of defined scenarios specifies how to allocate and de-allocate servers in order to achieve the quality of service agreed upon. Each scenario is defined as follow:

- **Server Requirements** – defined in terms of floor, guaranteed-scalability-point, and ceiling values. The floor value defines the minimum number of servers that must be available to the customer at all times. The guaranteed-scalability-point defines the number of servers that are guaranteed to be available for allocation.

The ceiling value defines the maximum number of servers that can be allocated to the customer.

- **Allocation Thresholds** - defines the thresholds to be used to triger server allocations and deallocations. These are given in terms of the server minimum and maximum load.
- **Period of time** (start time and stop time) – defines the period of time where this scenario is valid and must be applied.
- **Priority** – enables the definition of override scenarios in order to support special and/or unexpected events.

Example:

```
Scenario 1: {[Server_Set(4, 4, 2)], 00:00 Dec/01/2000, 24:59 Dec/31/2000, 1}
Scenario 2: {[Server_Set(7, 7, 6)], 00:00 Dec/22/2000, 24:59 Dec/24/2000, 2}
```

The first scenario defines that the floor is 4 servers, the guaranteed-scalability-point is 8 servers and the ceiling is 13 during the year 2000, with priority 1. Scenario 2 specifies a different range of values during the 3 days that proceed Christmas day. Scenario 2 will have precedence over the first one because of its higher priority.

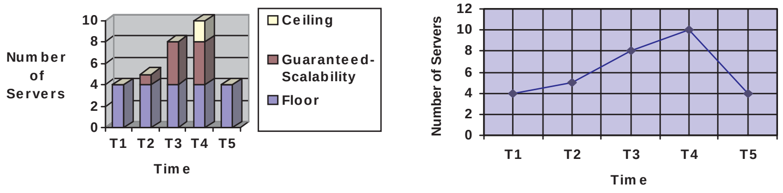

Figure 1 – Server Allocation over Time

Figure 1 presents two graphics that shows an example of server allocation for Scenario 1 described in the previous example. The first graphic shows that for every time T the floor is always available. New servers are allocated based on demand in the guaranteed-scalability-point range (T2 and T3) until it reaches the maximum of eighth servers, then new servers are allocated in the ceiling level range (T4). When the extra servers are not necessary they are deallocated and the number of servers falls back to the floor level.

The second graph shows the variation in allocated servers over time.

<u>Violation Policy Section</u>

Violation Policies describe what to do when a scenario is violated. Each policy is described in terms of:

- **Scenario ID** – The scenario identification (as described in the previous contract section).
- **Violator** – The policy violator, which must be on of the role players. Usually, this can be either the provider or the customer. A provider is considered the violator when the infrastructure did not support the agreed QoS for some period of time. The customer is the violator when it was not possible to maintain the ISLA because of some application problem or incorrect QoS estimation.
- **Grace Period** – The duration of time that the monitor must observe the event before it can classify it as a violation.

- **Penalty Action** - A charge or other penalty that will be imposed on the violator.

Example:

```
Violation 1: {Scenario 1, Provider, 30min, Penalty 1}
Violation 2: {Scenario 2, Provider, 15min, Penalty 1}
Violation 3: {Scenario 2, Customer, 30min, Penalty 2}
```

Charging Model Definition Section

This section defines how to calculate charges for the contract itself and the penalties to be applied in case of violations. Charges are classified as follows:

- **Base Cost -** δo (Fixed Operational Cost).
- **Contracted Services -** Cs_i (Cost per service$_i$).
- **Scenarios Cost** *(per scenario i)*- Cf_i (floor level), Cg_i (guaranteed level) and Cc_i (ceiling level).
- **Scenarios Violation** *(per scenario i)* **-** Pf_i (Penalty cost for server not available on the floor level) and Pg_i (Penalty cost for resource not available on guaranteed-scalability-point).

Example:

```
Scenario 1: {[Server_Set(5,7,3)], 00:00 Dec/01/2000, 24:59 Dec/31/2000, 1}
Scenario 2: {[Server_Set(7,7,6)], 00:00 Dec/22/2000, 24:59 Dec/24/2000, 2}
Base Cost:  $5000.00
Price per Scenario: Scenario 1 – floor = $5.00, guaranteed-scalability-point = $2.00, ceiling = $1.00
                    Scenario 2 – floor = $8.00, guaranteed-scalability-point = $5.00, ceiling = $2.00
Prices per Service:        Scenario 1 – $2.000,00
Prices per Penalty:        Scenario 1 – floor = $4.00, guaranteed-scalability-point = $1.00
                           Scenario 2 –  floor = $7.00, guaranteed-scalability-point = $2.00
```

Based on these prices we can calculate the following charges:

- **Contract Flat Charge** – Is based on the agreed services, reserved resources (e.g., those on the floor level) and a constant operational charge. This value doesn't change in the contract.
- **Usage-Based Charge** – This is a variable charge, based on the actual resources used.
- **Sub Contract Addition** – This value is calculated by the addition of sub-contracts, as explained in Section 2.2, to support the new scenarios.
- **Violation Penalty** –A penalty occurs when a service or resource is unavailable, i.e. resource unavailability occurs when the infrastructure did not allocate servers at the required level determined by the allocation algorithm. This value can be converted to an equivalent cash-back bonus or resource credit.

The equations to calculate these charges are shown in Section 3.1.3.

Report Customization Section

The Report Section defines reports as follow:

- **Type** – Currently we support the following pre-defined reports: Standard, Notification, Modification and Violation.
- **Recipient** – The report recipient can be a person, a URL, and/or an e-mail address.
- **Time Interval** – Time between the reports.

For example: Report: {Type = Violation, Interval = 24h, Recipient = http://www.customerA.com/reports}

2.2.The Contract Hierarchy

We defined a contract hierarchy that supports aggregation and overriding of contracts. Sub-contracts re-define or add information and inherit the rest from their ancestors. Figure 2 shows an example of a contract hierarchy that includes temporary special occasion contracts.

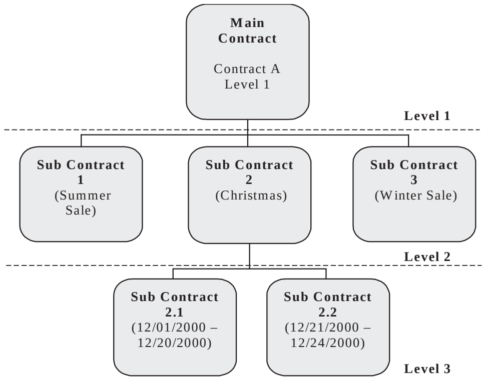
Main Contract
Contract A Level 1
Level 1
Sub Contract 1 (Summer Sale)
Sub Contract 2 (Christmas)
Sub Contract 3 (Winter Sale)
Level 2
Sub Contract 2.1 (12/01/2000 – 12/20/2000)
Sub Contract 2.2 (12/21/2000 – 12/24/2000)
Level 3

Figure 2 – Contract Hierarchy

3. THE POLICY LANGUAGE

The contract presented in Section 2 describes the ISLA Contract in a high-level, human readable language, appropriate for customer interaction. To perform conflict detection and contract validation a formal machine-usable language is required. From the Contract we extract a set of policies that are used by the following modules:

- Pricing
- Configuration Manager
- ISLA violation detector
- Reporting

The extraction process defines the following groups of policies: (1) Base, (2) Violation, and (3) Pricing.

3.1.1. Base Policies

These policies define QoS parameters and the scenarios supported for each customer, they correspond to the Customer Configuration and ISLA Scenarios Definition sections in the ISLA Contract, see Section 2. In this sense, we define two types of base policies: *Configuration Policies* and *Scenario Definition Policies*.

```
P1: Configure(CustomerID, CustomerSegmentID,
            [Parameter1:Value1,…,Parametern:Valuen]);
```

Where:

Customer ID – Customer identification.

Customer Segment ID – Customer segment identification.

Policy Name – Name of the policy.

Parameter List – List of configuration parameters and their values as defined in the Customer Configuration section of the contract.

Example:

```
P1: ConfigureCustomerSeg (11, 1101, [CustSegLevel:1, VipAddr:1.2.3.4, SegPortNo:111,
Priority:1, ServerPoolID:11101)
P2: ConfigureRequirements (11,1101,[ActiveConnections:100, OverallResponseTime:0.1,
        OutputBandwidth:0.025, DBResponseTime:0.075)
```

```
P2: DefineScenario(CustomerID, CustomerSegmentID, ScenarioID,
                   StartTime, EndTime, Priority,
                   ServerSet[floor, guaranteed, ceiling])
```

Where:

Customer ID – Customer identification.

Customer Segment ID – Customer segment identification.

ScenarioID – Scenario identification.

StartTime and EndTime – Define the period of time when the scenario is valid, each specified as hh:mm:ss mmddyy.

Priority – (Described in the contract section named ISLA Scenarios Definition).

Server Set – Specifies the values for the floor, guaranteed and ceiling.

3.1.2. Violation Policies

The Violation Policies define the penalties to be applied for a disruption in service. The policy syntax is:

```
P3: Violation(CustomerID, CustomerSegmentID, ScenarioID,
    GraceTime, Violator,PenaltyID)
```

3.1.3. Pricing Policies

The pricing policies can be used to support differentiation of services, specify how to calculate usage-based charges, and the penalty values for disruption of services. These policies encapsulate a set of equations that calculate the Contract Fixed Charge, Usage-Based Charge, Penalty Charges and Sub-Contract Charges [9,10,11]. New equations can be added to represent specific aspects of different customers.

ContractFixedCharge (per month)

$$CFC = \sum_{i=1}^{n} (Cf_i \times \# f_i) \times T_u + \sum_{j=1}^{m} Cs_j + \delta o$$

Where: Cf_I is the cost per server on the floor level per scenario i; f_i is the number of servers allocated on the f (floor) level; Cs_j is the cost per contracted service; δo is fixed Operational Cost and T_u the number of hours on one month (the basic time unit is hour).

UsageBasedCharge (per a T period of time)

$$USB = \sum_{i=1}^{n} (Cg_i \times \# g_i) \times Tg_i + \sum_{i=1}^{n} (Cc_i \times \# c_i) \times Tc_i$$

Where: $\text{R1}: Cf_i > Cg_i > Cc_i$ $\quad \text{R2}: \# c_i \leq c_i + g_i$

$\text{R3}: \# g_i \leq g_i + f_i$ $\quad \text{R4}: T = \sum_{i=1}^{n} Tg_i + \sum_{i=1}^{n} Tc_i$

Cf_i, Cg_i and Cc_i are respectively the cost per server on the different levels of guarantee (floor, guaranteed-scalability point and ceiling); *#g* and *#c* are the number of servers allocated on demand on the level *g* and on the level *c*; Tg_i and Tc_i are respectively the portion of time that the resources on the levels *g* and *c* were allocated.

Penalty (per a T period of time)

$$Penalty = \sum_{i=1}^{n}(Pf_i \times \# f_i) \times Tf_i + \sum_{i=1}^{n}(Pg_i \times \# g_i) \times Tg_i$$

Where: $\text{R5}: Pf_i > Pg_i$ $\qquad$ $\text{R6}: \# f_i \leq f_i$

$\text{R7}: f_i < \# g_i \leq f_i + g_i$ $\qquad$ $\text{R8}: \Delta T = \sum_{i=1}^{n} Tf_i + \sum_{i=1}^{n} Tg_i$

Pf_i and Pg_i are respectively the penalty cost per server not available on the levels floor and guaranteed; *#f* is the number of servers not allocated on the level *f*; *#g* is the number of servers not allocated on demand on the level *g*. Tf_i and Tg_i are respectively the portion of time that the resources on the levels *f* and *g* were not allocated.

Sub Contract Addition

Suppose that *m* new scenarios were created, then the cost of this modification will be a fixed cost calculated as follow:

$$SCA(m) = \sum_{i=1}^{m}((Cf_i \times \# f_i) \times T_{i2}) + \alpha$$

Where α is the fixed operational cost per modification. So, this charge is just the sum of fixed cost of each new scenario multiplied by the period of time that they apply plus a constant.

4. THE CONTRACT LIFE-CYCLE PROCESS

Figure 3 describes the data workflow, processes in the contract life cycle and the on-line process that implement violation processing.

Contract Definition (1) is the first step. The contract is then validated using the **Validation Process** (2). Contract validation itself has two phases:

- Conflict and Syntax Check – Checks that the specified contract does not have internal conflicts and that it is syntactically correct.
- Capacity Check - This establishes that the Océano farm has sufficient capacity to support this contract.

If a validation problem occurred, contract processing will return to the **Definition Process.** Once the contract has been validated, the **Translation Process** is started (3), which extracts the set of policies that must be enforced in the system. The policies are inactive until the **Controlling Process** (4) activates them, by storing them into the Océano database and initializing the appropriate monitors.

The policies drive a constant **Monitoring Process** (5) that detects violations. When a violation is detected a signal is sent to the **Violation Analysis Process** (6) that keeps track of the violation and sends requests to the **Penalty Calculator** (7). A copy of an active policy can be created and used for contract re-negotiation. When the new contract is ready the old contract will be deactivated and replaced.

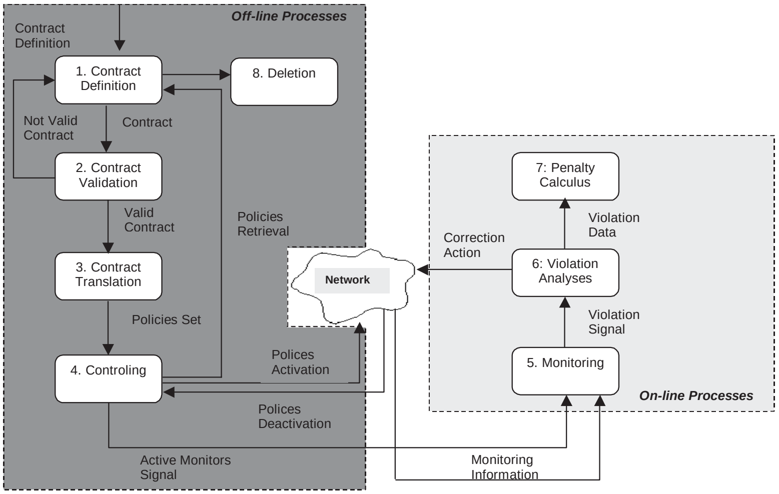

Figure 3 – Life-Cycle Process

5. THE ARCHITECTURE

Figure 4 shows the schema of the system components that support definition, storage, monitoring, billing and reporting of infrastructure service level agreements.

The numbered arrows represent the transfer of data in the system, dashed lines for on-line activities and solid for off-line. This architecture has 7 modules and 11 activities defined as following:

GUI Interface – The GUI captures interactions between the provider, the customer and the system. It allows one to edit contracts (1), visualize and modify existing contracts (11), visualize reports (10) and alternative quotes (8). It also permits the customer to make requests to add or change scenarios in order to support, for example, some marketing plan.

Contract Builder – This component produces a contract using the data provided through the interface, then sends the contract to the evaluator (2.1) and stores the valid contract into the database by generating SQL statements (2.2).

Contract Evaluator – The Contract Evaluator performs the validation process (as defined in the life cycle model).

ISLA Manager – The ISLA Manager has two components: The Response Automation Module and The Violation Detection Module. These modules perform the following:

1. Receives pertinent data from a new contract or from the policies defined in a previous contract (3).
2. Receives violation event notifications from the monitors (5). Poll for state information (6). Both activities are performed through Yemanja [2], a model-based event correlation engine for multi-layer fault diagnosis.
3. Keep track of violations until they finish.
4. Requests a penalty calculation from the Pricing Engine (7).

5. Finally, the collected information (violated scenario, start time, stop time, violator and penalty) is stored into the database (4).

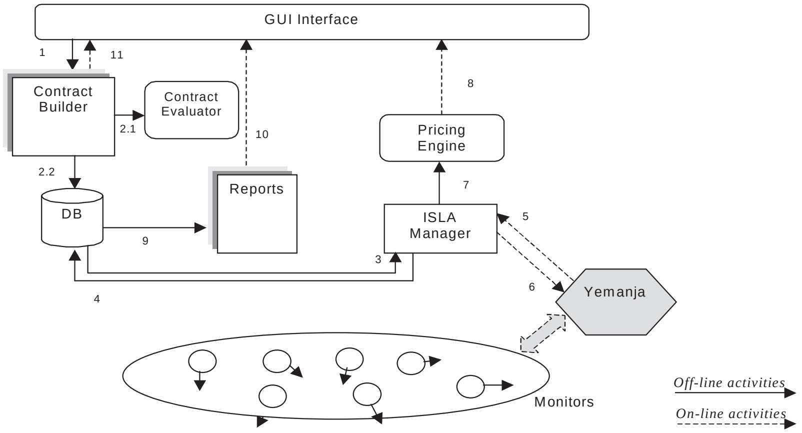

Figure 4 – The Salmon Architecture

Monitors – Monitors collect information about system behavior. The monitors send violation event notifications to the ISLA Manager (5). The ISLA Manager can also poll for state information (6).

Pricing Engine – The Pricing Engine calculates the contract value, penalty charges (to the customer and to the provider). It can also respond charging queries (8).

Report Generator – The Generator collects data from the database (9), formats the information into customized reports and sends them to the recipients. The reports are generated periodically or on demand.

6. THE SALMON PROTOTYPE

The Salmon prototype is currently [April 2001] still under construction. All the components, except the GUI interface, are being implemented using Jbuilder 3.5 [8] and DB2. Here we will describe only the most important components.

6.1.The GUI Interface

This interface is been implemented using HTML and provides contract visualization and editing. Either the provider and the customer can interact with the system, in this sense the GUI Interface provides a high-level language to specify and monitor ISLAs.

6.2.The Salmon Database

Figure 5, presents the most important classes and relationships defined by the database model. The Salmon Database supports the definition of customer configuration data, scenario definitions, violation policies, the violation log and pricing policies. Customer specific information is pulled from the configuration database whenever is needed.

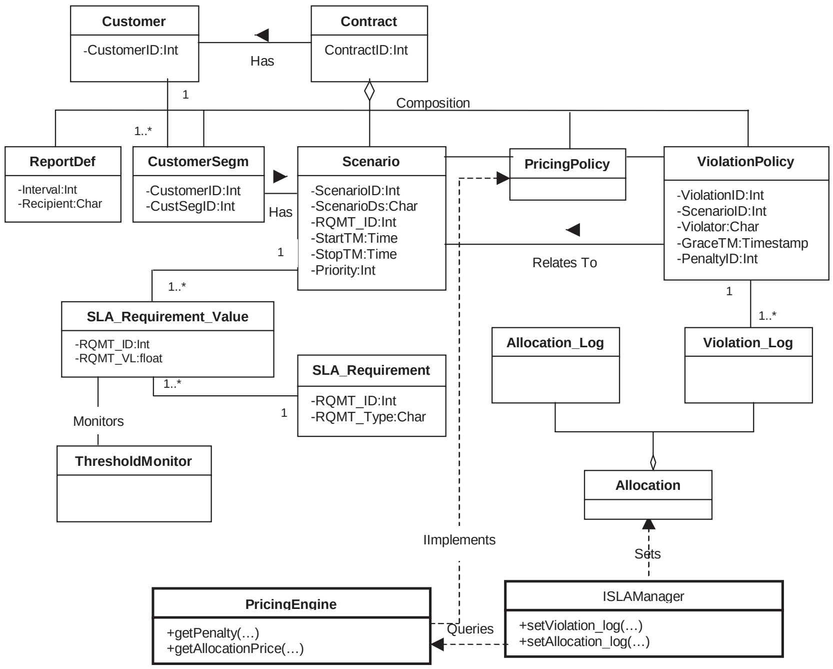

Figure 5 – The Salmon Database

As an example let's look at the ViolationLog class. This class contains information on every violation that was detected in the system. The violations are represented by the following attributes: policy violation identification (ViolationID), start and stop time of the violation, recovery time (time until the system came back to the correct behavior) and penalty value. For every new violation the ViolationLog class queries the pricing engine for the penalty value.

The Salmon Database is a repository of both static and dynamic data used by Océano to enforce, monitor and report on the contract in effect. This model was implemented in DB2.

6.3.The Pricing Engine

This component contains the pricing policies defined by the charging model, some equations were described previously in Section 3.1.3. In the Pricing Engine the policies are divided in two groups: contract policies and meta-policies. The contract policies are the ones defined per contract and the meta-policies are policies applied over all the supported contracts.

The Pricing Engine receives requests from others components, including: the Salmon GUI Interface, Salmon ISLA Manager and others Océano components that need pricing information for decision making. Some of the supported operations are:

getPenalty(ScenarioID:int, NRServers:int, NAServers:int):float – this function returns the penalty charge given the number of new servers requested and the number of servers not allocated (per scenario) once they were needed. The time unit is fixed in one hour.

getServerPrice(ScenarioID:int, CNServers:int, NRServers:int):float – this function returns the price for additional servers given the current number of servers and the number of added servers. The time unit is fixed in one hour.
getViolation(ContractID:int, StartTime:timestamp, EndTime:timestamp):float – returns the total penalty charge for all violations occurred to specific contract in a period of time.

7. CONCLUSIONS

Salmon is a prototype implementation of a system to specify and maintain Infrastructure Service Level Agreements (ISLAs) in server farms. In contrast to other earlier approaches, Salmon's emphasis is on providing ISLA enforcement and guarantees based on a set of scenarios and violation policies. The contract structure that we described combines the QoS definition with a charging model that supports penalties for disruption of service and charges based on demand. Salmon monitors contract violations, and triggers actions to calculate violation penalties. We expect Salmon to be fully functional by May 2001.

REFERENCES

[1] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. Pazel, J. Pershing, B. Rochwerger. Océano – SLA Based Management of a Computing Utility. *Integrated Network Management*, Atlanta, USA, May 2001.

[2] K. Appleby, G. Goldszmidt, M Steinder. Yemanja – A Layered Event Correlation Engine for Multi-domain Server Farms. *Integrated Network Management*, Atlanta, US, May 2001.

[3] Dinesh Verma. Supporting Service Level Agreements on IP Networks. *Macmillan Technology Series*, 1999.

[4] Gupta, A, D.O. Stahl and H.R. Varian. Pricing of Services on the Internet. *Technical Report*, University of Texas, Austin, Texas, 1995.

[5] Costas Courcoubetis and Vasilios A. Siris. Managing and Pricing Service Level Agreements for Differentiated Services. *Proceedings of IFIP/IEEE IWQoS'99*, London, UK, May 1999.

[6] Courcoubetis C. Pricing and Economics of Networks. *Infocom Presentation*, 1998, URL– http://www.ics.forth.gr/~courcou/

[7] Odlyzo, A. Internet Pricing and History of Communications. *AT&T Labs-Research*, URL - http://www.research.att.com/~amo

[8] Neal Ford, Ed Weber, Talal Azzouka, Casey Williams. JBuilder 3 Unleashed. *Sams*, August 1999.

[9] Courcoubetis, C., F. Kelly and R. Weber. Measurement-Based Usage Charges in Communications Networks. *Statistical Laboratory Research Report* 1997-19, University of Cambridge.

[10] Felix Hartano and Georg Carle. Policy-Based Billing Architecture for Intranet Differentiated Services. *Proceedings of IFIP Fifth International Conference on Broadband Communications*, Hong Kong, Japan, December 1999.

[11] Fishburn, P.C., Odlyzko, A. M. Dynamic Behavior of Differential Princing and Quality of Service Options for the Internet. *Proceeding First Intern. Conf. on Information and Computation Economies*, Charleston, USA, October 1998.

[12] Robert Kearney, Richard King, Martin Sachs, Asit Dan and Daniel Dias. Electronic Service Level Agreements (eSLA) for Application Hosting. *Internal Report*, IBM - T.J. Watson Research Center, May 2000.