e-CoBRA (electronic Contract-Based Resource Allocation): an Architecture to On-Demand Resource Allocation Management Based on Contracts

Juliana Cunha[(1)] *and Fabio Q. B. da Silva*[(2)]
(1) Federal University of Pernambuco, CESAR – Centro de Estudos e Sistemas Avançados do Recife
(2)Federal University of Pernambuco
juliana@cesar.org.br; fqbs@cin.ufpe.br

Keywords

On-demand Resource Allocation, Service Level Agreements, Network and System Monitoring

Abstract

This work presents the e-Cobra architecture, an on-demand resource allocation system based on service contracts, developed to address some of the problems in the deployment and management of network based application services. The architecture supports business rules representation extracted from negotiated contracts between the involved parts (providers and customers). From these contracts, policies are extracted and applied to the system in order to allow on-demand resource allocation to happen according to the submitted contracts' requirements. The proposed architecture covers not only the on-demand allocation task, but price, violations and penalties aspects. The business rules incorporated into the system model represent policies extracted from service contracts. Therefore, the present work applies the concepts contained in SLA and in the policy based management fields, as well as, extends them.

1. Introduction

Network management and system administration have changed from a technical job to become an strategy task in the overall enterprise management process. Therefore, formal methods and models must be developed to allow network management alignment with the market and business needs. With the network management growth in importance as a mission critical task for the success of companies and organizations new research areas appeared such as Quality of Service (QoS), Service Level Agreements (SLA) and Policy-Based Management. These areas aim to provide more control over the guarantees offered about the network behavior.

In the final years of the last decade, the growth in businesses done through the Internet opened new possibilities for resources, services and applications to be commercialized through the network. This opportunity created a new commercial channel called ASP - Application Services Providers. Figure 1, extracted from [Soul00], shows that, traditionally, the software market has worked around purchase contracts. Nowadays, this commercial way still exists but is evolving towards a usage-based payment process. This evolution in the software market shall allow the possibility of software rental contracts (of the use of whole packages or parts of it) instead of a one-off purchase of an entire package. That is, software, once considered a product, is becoming a service. The (software/service) provider must, then, create mechanisms to make the right contracted application available when an income request is received.

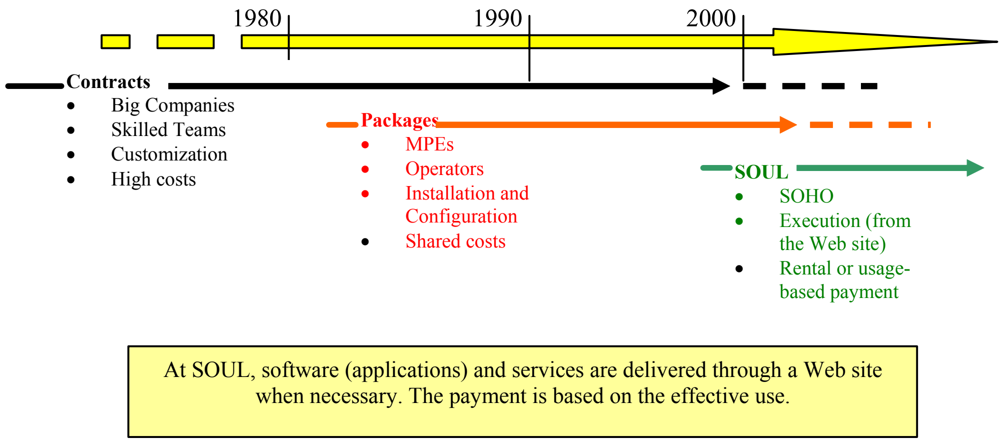
1980
1990
2000
Contracts
Big Companies
Skilled Teams
Customization
High costs
Packages
MPEs
Operators
Installation and Configuration
Shared costs
SOUL
SOHO
Execution (from the Web site)
Rental or usage-based payment
At SOUL, software (applications) and services are delivered through a Web site when necessary. The payment is based on the effective use.

Figure 1: Software Commercialization Evolution

Briefly, providers must be sure that the system reflects and implements the high-level business rules. The clients must be able to easily contract a resource abstracting themselves from implementation details but with conscience about their contract guarantees.

This work takes advantage of two main research areas: Policy-based Management and Service Level Agreements. The Policy-Based Management (PBNM) [Verma00] approach allows SLA and QoS guarantees in an active way. Monitoring is no longer a passive task. This facility allows network requirements to change in much faster and more accurate way. The main difference between PBNM and the OSI and SNMP approaches is the separation between the managements control tasks from its implementation. Policies define which actions must be taken and when, but not how. The agents that interpret polices can be elastic processes [YGY91, GY93, GY95] or even intelligent agents.

Some works are been developed in order to incorporate policies definition into the management tasks. In [Wies95a], Wies defends that corporate goals must be integrated with the network management process. Sloman presents a set of tools that constructs a framework for PBNM [Sloman94, YLS96, Yialelis96, YS96, SS97]. Since 1996, the IETF is working on a standard PBNM architecture [CIM, COPS]. Some commercial tools have been also developed such as: Cisco System QoS Policy Manager [CiscoQPM], IPHighway Open Policy System [Conover99], HPOpenView Policy Expert [HPPE], ExtremeWare Enterprise Manager [ExtremeWare], Network Executive [CGA$^+$01] and others.

Even though these works concentrate into the on-demand allocation problem, none of the above solutions covered all the necessary requirements. Some ASP such as LoudCloud, Digex, Exodus, Digital Fuel`s Service Track Platform provides SLA, but in a static way. These contracts cannot respond to demand changes over time, and their definitions are not part of the system architecture.

1.1 Océano Project

Recent advances in the area have been achieved in the Océano Project developed at the IBM T.J. Watson Center. Océano [AFF$^+$01, AGS01] is a prototype of a highly available, scalable, and manageable infrastructure for an e-business computing utility. It supports multiple customers in a collection of shared resources. For example, at any moment, a server can be allocated or de-allocated to/from a customer. This dynamic resource allocation enables flexible Service Level Agreements (SLAs). From the Océano project development, a set of new requirements were found that motivated the development of the present work:

- The need to establish a contract between costumers and providers.
- Océano's contracts should contain allocation domain definitions, violation policies descriptions, penalties for degradation on the level of service, price aspects and customized reports.
- The need of high-level contracts between the parties. This contract should specify business rules, QoS metrics and guarantees.
- Contracts should be defined in such a way that the customer's needs could be properly represented and that monitoring and enforcement could be translated into the system.
- The need for an architecture capable to define and maintain such contracts. This architecture should be able to monitor contract enforcement, detect violations, apply penalties and determine prices on a usage-basis.

This work presents the e-CoBRA architecture as one component of a comprehensive approach to treat policy-based resource allocations and the management of service contracts. This approach includes a contract description language called e-SAPo [Cunha02, CS03], the e-CoBRA architecture, a formal translation of e-SAPo constructs into e-CoBRA components, and an instantiation of e-CoBRA into a real implementation based on one of the Océano's component, the SALMON system. This paper concentrates on the description of e-CoBRA but briefly discuss the other aspects of the entire framework.

This paper is organized as follow: Section 2 discusses the e-CoBRA architecture requirements; the architecture business model is presented on Section 3. Section 4 shows the analysis model that supports the business model introduced. A real implementation of the e-CoBRA architecture is presented on Section 5; finally Section 6 concludes the paper.

2. A Policy-based Management System Architecture

A policy-based management system must have some functional requirements as described below:

- A graphical interface that allows access to the policy repository.
- A policy evaluator.
- A mechanism to notify the involved components about policy changes.
- A mechanism to translate policies in a format known by the devices
- A repository to store policies and its activity.

The related works presented on the previous section do not provide the necessary requirements to the conception of an architecture to the on-demand resource allocation systems supporting dynamic SLAs. In this sense, this work suggests a different approach by:

Defining a management system based on contract and not on policies - A contract is the object that formalizes the negotiation between providers and customers, thus describing the needs and terms that must be enforced. In this context, a contract is the object that must be monitored, guaranteed and that must regulate the infrastructure. Policies are very precise but also very specific to the low-level communication between the system components, as so, it is not suitable for the communication with customers. Instead, we suggest that contracts must have policies described inside its body allowing the integration with low-level management tools.

The Representation of Client's Needs – We propose the concept of scenarios to describe client's needs. Each scenario is set to a period of time and contains the policies to be applied in this period and the quality of service requirements. Each scenario can have different levels of guarantee with different behaviors. Also, a scenario must have a priority that is useful for scenario override and conflict resolution. Each client can define as many scenarios as he/she wishes in order to represent his/hers needs. Inside the system, a scenario is the customer avatar and the rules that it contains must be enforced.

Policies Representation - This work defines the following policies in order to cover the requirements of an on-demand allocation system: Configuration Policies, Monitoring Policies, Allocation Policies, De-allocation Polices, Violation Polices and Pricing Polices.

2.1 An Informal Representation of the Architecture

The proposed architecture was modeled in UML [BRJ99], a language to specify, model and document different aspects of systems. The fully completed specification is in [Cunha02].

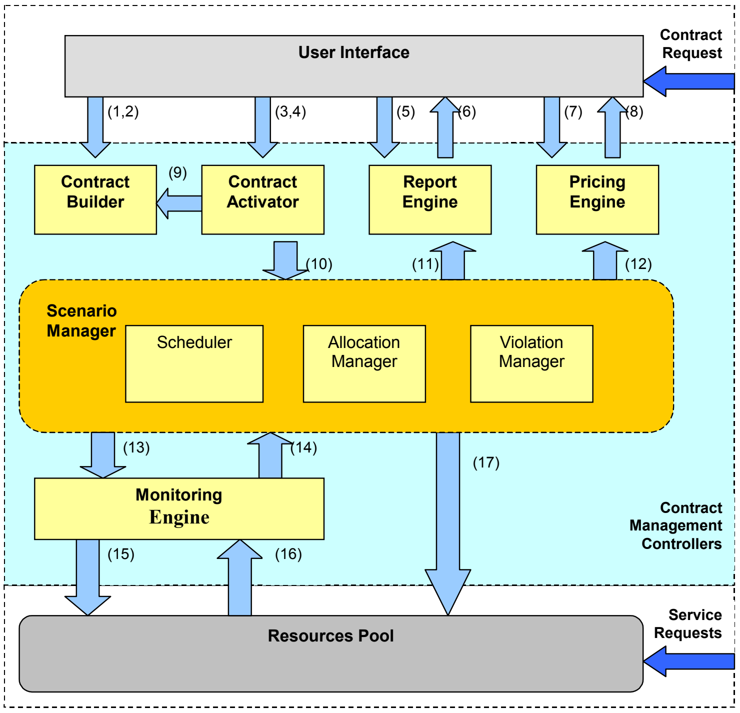

Figure 2 –Proposed Architecture

Figure 2 informally illustrates e-CoBRA architecture, which must support the following characteristics:

- Allows contract specification, storage and retrieval.
- Support multiple contract management that uses the same common infrastructure.
- Support violation policies for disruption of service.
- Provides accounting of used resources.
- Help the negotiation process by simulating scenarios usage and prices.
- Generate usage, violation and price reports.

The description of the components of the Architecture is provided below.

User Interface - The GUI interface is used to specify contracts (1), modify contracts (2), active and deactivate them (3, 4), report request and display (5, 6) and price queries and display (7, 8). This interface allows the interaction between the user and the system, so it must show contracts and policies in a human-readable form.

Contract Builder - This component retrieves a contract template and receives back the contract as defined by the user through the GUI Interface (1). It then checks its correctness and stores the contract in the repository. The Builder must "understand" the notation used to define contracts so it can store it properly in the database. The Builder also executes the process of retrieving a contract from the database in order to be visualized or modified (2).

Contract Activator - The Activator makes a defined contract active in the system. Once it receives an activation request (3), the Activator retrieves the contract through the Builder (7) and validates it. Once the contract is valid, the Activator sends a message to the Scenario Manager about a new contract income (9).

Scenario Manager - It's the most important component in the system. It is responsible for managing the contract enforcement. This component schedules the scenarios in order to proceed with their activation. Once a scenario is active, the Manager manages the monitors' thresholds and the policies applicability, stores the allocation changes information (10) and activates the underlying pricing policies (11). During a policy action, a resource allocation change may be required (15).

Monitoring Engine - This component is responsible for the infrastructure monitoring, verifying its behavior and detecting the thresholds used to trigger policies (14). When a scenario is activated, the Scenario Manager starts all the necessaries monitors (12). Once a monitor detects a threshold it sends a message to the Scenario Manager (13).

Report Engine - It builds (10) and outputs (6) reports, such as: usage, price, violation, etc.

Pricing Engine - This component executes pricing policies through a Scenario Manager request (11) and plays a role on the system tuning and negotiating by answering queries about contracts and usage-based scenarios prices (6).

3. Business Model

The main goal of this model is to find out the system requirements to support the underlying business. Rational Unified Process (RUP) methodology is use-case oriented, so this diagram is the process start point. The diagram on Figure 3 defines eight use-cases and three actors (one of them is a external system), and aims at capturing functional requirements and the relationship among use-cases and system actors. At this phase, the following artifacts will be described: use-case diagram, use-case specification and glossary.

Observe that each use-case returns some value to the user. All the others phases will be, in fact, a more detailed view of each use-case defined in this section.

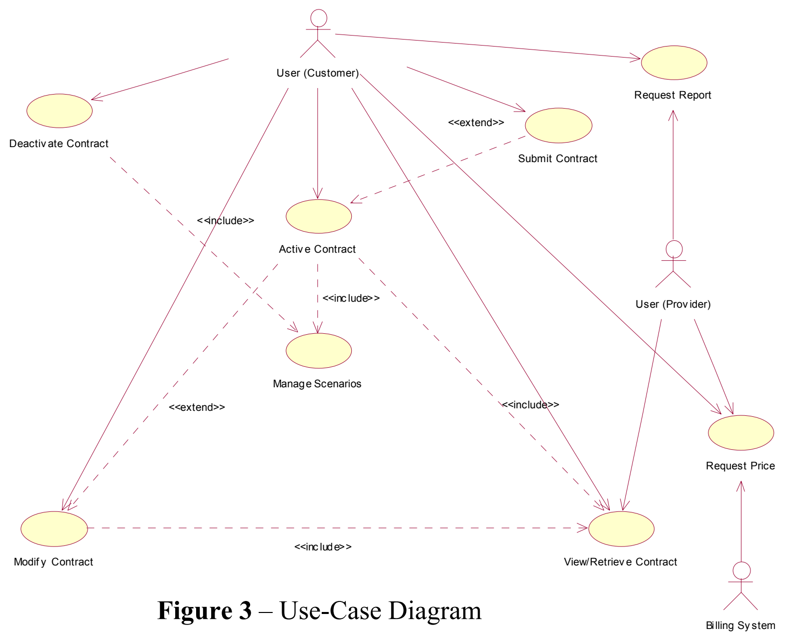

Figure 3 – Use-Case Diagram

This section specifies use-cases as functional requirements (FR) for negotiating, signing and managing contracts. Each use-case was described in term of its event flow, input and output. Below an example of a use-case description:

[FR007] Request Price Calculation

Inputs and Pre-Conditions: A contract must exist in the system.

Outputs and Pos-Conditions: Requested prices

Main Event Flow

1. The user requests a contract price calculation.
2. The Pricing Engine actives the pricing policies.
3. The policies return the requested prices.

Secondary Flow (alternative and exception):

1. Simulate Prices

Alternatively prices queries may be used in order to simulate or predict prices.

3.1 State Diagrams

In a resource allocation system based on e-CoBRA four elements are of fundamental importance:

- **Contract** – This is the contract itself. A contract is born when a provider defines its template and then publishes it waiting for requests. In order to contract a service or resource, a client must instantiate the template and activate the customized contract in the system.
- **Scenario** – This element defines when, how and under which circumstances a contract is applied.
- **Policy** – This element defines how to allocate, de-allocate and price scenarios, as well as how to treat a scenario violation.
- **Monitor** – The monitors verifies the network behavior and indicates allocation and de-allocation need.

For each of the above elements we created state diagrams that specify the sequence of states followed by each object during its life cycle while responding to pre-defined events.

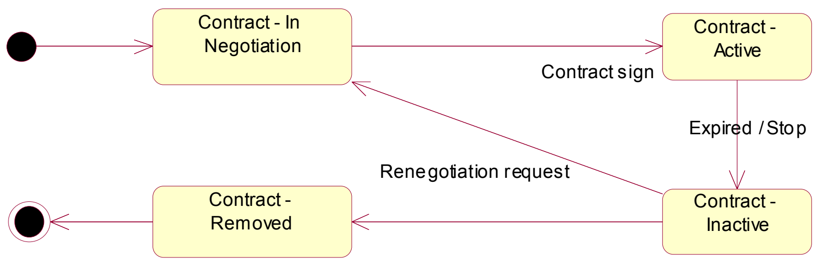

Figure 4 – Contract State Diagram

The contract object has 4 states: (1) In negotiation, (2) Active, (3) Inactive and (4) Removed. A contract is activated when the negotiation finishes by a sign activity. If a contract is expired or forced to finish its activity, it will assume the inactive state. Once in this state, the contract can be removed or re-negotiated.

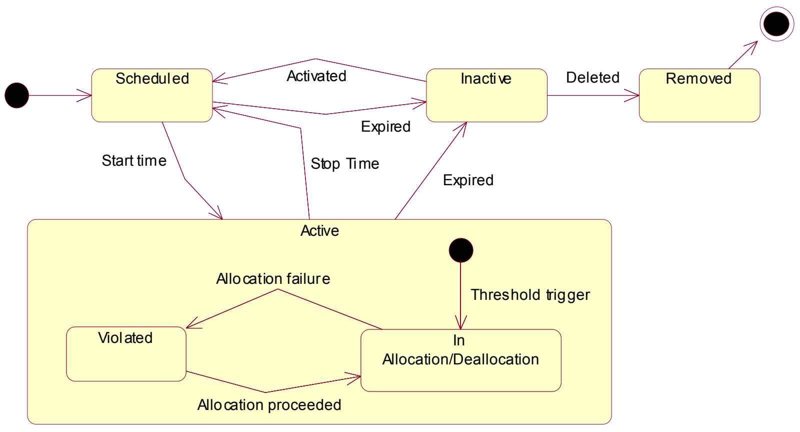

Figure 5 – Scenario State Diagram

The scenario object has four main states shown on Figure 5: (1) Scheduled: means that the scenario is waiting for its activation; (2) Active: a scenario starts when the scheduler reaches it start time, it remains on this state until its stop time; (3) Inactive: this state occurs when the scenario's execution finishes or the contract that contains it expires (represented by the Expired transition); (4) Removed: a scenario was deleted from the system.

The Active state has two sub-states: (1) In Allocation/De-allocation: scenario state during an allocation change; (4) Violated: if the allocation fails, the scenario remains on the violated state until the situation normalizes.

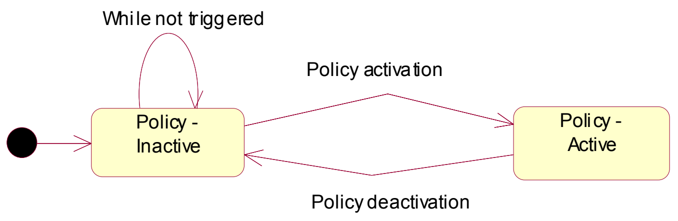

Figure 6 – Policy State Diagram

Policies have a very simple diagram with two states: inactive and active. When a trigger is received the state changes from inactive to active. The opposite happens the policy action is terminated.

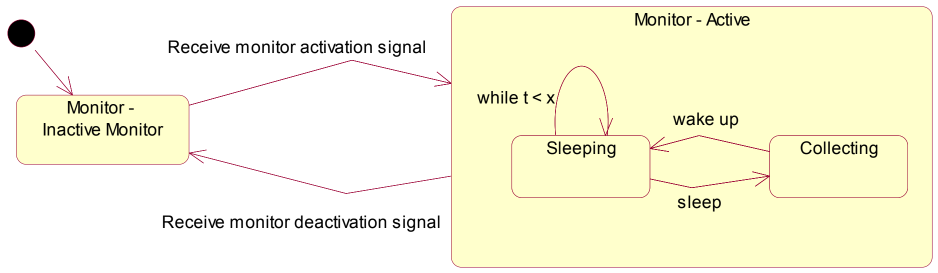
Monitor - Active
Receive monitor activation signal
Monitor - Inactive Monitor
while t < x
wake up
Sleeping
Collecting
sleep
Receive monitor deactivation signal

Figure 7 – Monitor State Diagram

The monitors have 2 states: Inactive and Active. However, while active, it can remain Sleeping in-between the executions. When the monitor is running, it changes to the Collecting state. When the collection finishes the monitor goes back to the sleeping state.

4. E-Cobra Architecture Analysis

Each use-case in the business model represents a set of classes and subsystems that are able to provide the business functionalities. The following figure presents the names of the elements that must exist in the e-CoBRA system and must have data persistence:

Abstractions Key		
Contract Header	Contract Hierarchy	Role Player
Monitor	Procedure	Resource
Level of Guarantee	Scenario	Service level Requirement
Allocation Policy	De-allocation Policy	Violation Policy
Usage Based Pricing Policy	Flat Pricing Policy	Report

The set of the above classes represents a contract. Each use-case was analyzed through class and sequence diagrams where their abstraction, relationships, data and messages have been introduced. One of the use-case analysis is presented below. The full description is in [Cunha02].

4.1 Use-Case Analysis - Active Contract

When a contract is signed it means that all the participants agreed with the terms that it includes and that after this very moment the contract must be enforced. The same concept is used here: when a service contract is activated, all the contracted resources and services must be correctly available. In order to proceed with the contract activation a set of boundary and control classes were defined. The boundary classes provide an interface between the external user and the system, and the control ones implement the logic that makes possible the request conclusion.

Boundary		**Control**	
Class	**Description**	**Class**	**Description**
Contract Activation Interface	Provides a contract activation interface to the user.	Contract Activator	Actives a contract in the system.
		Contract Builder	Manipulates the contract data.
		Scenario Manager	Manages scenarios.

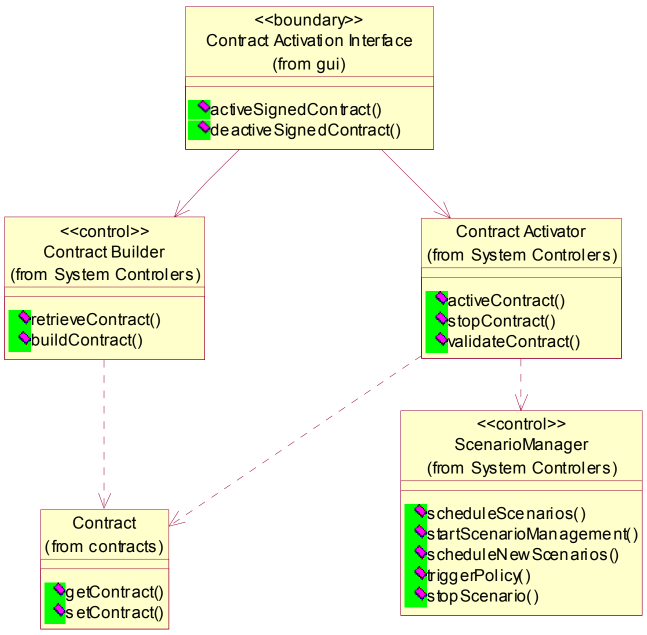

Figure 8 – Class Diagram: Active Contract

Figure 8 illustrates the relationships between the **ContractActivationInterface** class with the **ContractBuilder** and **ContractActivator** controllers. The first relationship allows contract retrieval and the second executes the activation. The class ContractActivator sends a message containing a new set of scenarios to be activated by the **ScenarioManager**. The class contract is an entity class and must have its persistence guaranteed:

Entity Class	
Class	**Description**
Contract	Stores contract information

The following diagram (Figure 9) illustrates the sequence of messages exchanged between the classes. The user requests activation through the interface that will send a message to the **ContractBuilder** asking for contract retrieval. Once a contract exists in the system, the ContractActivator must activate it by: (1) Checking the contract correctness; and (2) Sending a message to the **ScenarioManager** controller about the arrival of new scenarios.

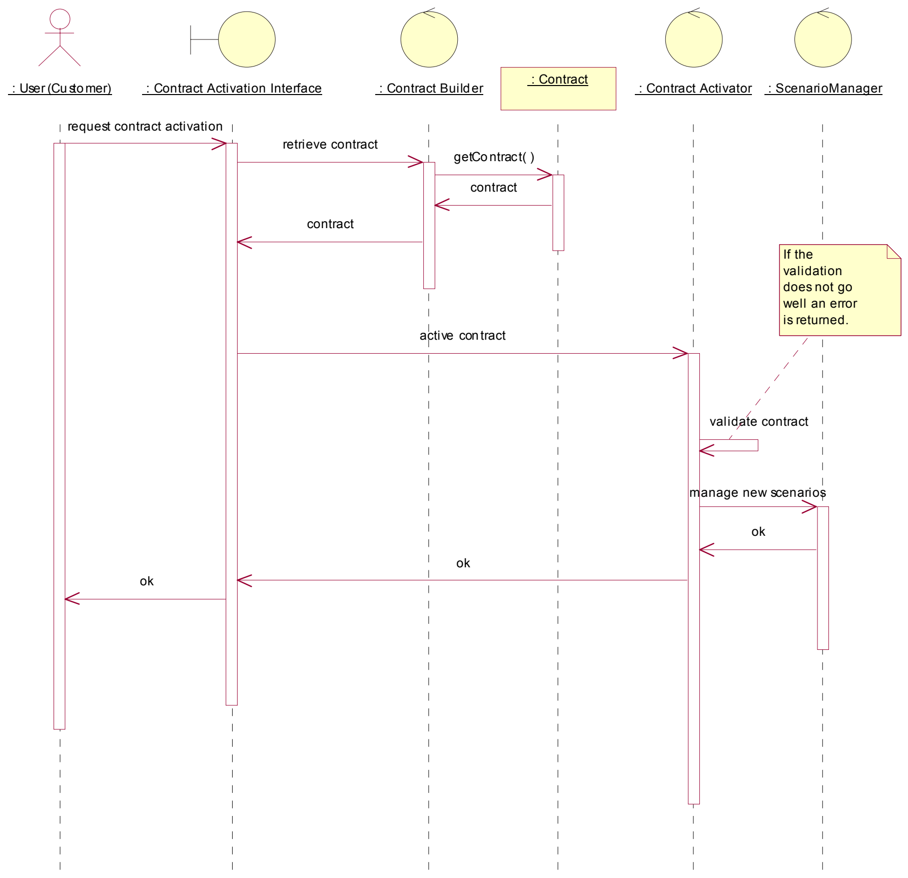

Figure 9 – Sequence Diagram: Active Contract

5. SALMON System

Salmon (Service Agreement Levels for Monitoring Océano coNtracts) uses an e-COBRA instantiation 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 whether the service provider has a sufficient 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 or to 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. SALMON is fully described in [CAGS01]. In Figure 10, we show SALMON architecture and its relations with another Oceano's components (Fortuna, Kelp, Yemanja and Neptune[1]):

[1] These components are not explained due to lack of space, but are described in [CAGS01].

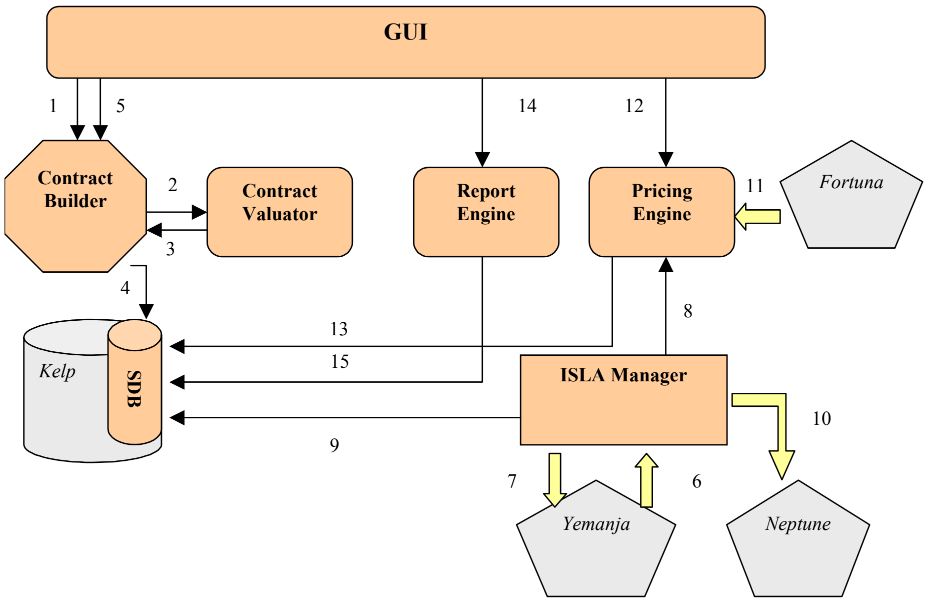

Figure 10 – Salmon's Architecture

5.1 The SALMON Prototype

All the components, except the GUI interface, were implemented using Jbuilder 3.5 and DB2. Here we briefly list only the most important components. 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. 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. Salmon was developed with the support of Oceano's Team at IBM T. J. Watson Research Center.

6. Concluding Remarks

This article presented the e-CoBRA architecture requirements and the modeling of an on-demand resource allocation management system based on contracts. The UML model allowed the definition of the system main functionalities and components. The presented architecture describes the components responsible for the execution of important tasks such as: building, activating, and monitoring contracts, scenario management, price calculation and reporting of activities. However, the system architecture was designed in a way that interfaces with third-part systems can be easily done (for example, the use of a monitoring tool).

We developed an implementation using a web-based interface between the users and the system, an object-oriented Java language to system implementation and a relational or object-oriented database. The organization of project elements in layers facilitates the model extensions in terms of support of additional classes.

e-CoBRA architecture models a contract driven system but does not specify a language to define these contracts. Contracts can be defined in variety of ways and using some or none formalism. Even a GUI interface can be viewed as contract definition language. In our research, contracts are specified in a language called e-SAPo, a contract specification notation that allows business and operational rules definition, which shall be describe in future work [CS03]. This notation can be used with the architecture defined here, but also has a more broadly applicability on helping resource/application providers to define their own contracts.

References

[AFF+01] Appleby, K.; Fakhouri, S., Fong, L.; et al. *Océano – SLA Based Management of a Computing Utility*. In Proc. IFIP/IEEE International Symposium on Integrated Network Management. Atlanta, USA, May 2001.

[AGS01] Appleby, K.; Goldszmidt, G.; Steinder, M. *Yemanja – A Layered Event Correlation Engine for Multi-domain Server Farms*. In Proc. IFIP/IEEE International Symposium on Integrated Network Management. Seattle, USA, May 2001.

[BRJ99] Grady Booch, James Rumbaugh and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison Wesley Longman Inc., 1999.

[CAGS01] Juliana Silva da Cunha, Karen Appleby-Hougham, German Goldszmidt, Fabio Q. B. da Silva. *SALMON – Architecture to Define, Store, Monitoring and Billing ISLAs in a Server Farm*. In Proc. Second Latin American Network Operation Management System Symposium, LANOMS'2001. Belo Horizonte, Brasil, August 2001.

[CGA+01] Gustavo Coelho, Lisandro Granville, Maria Almeida, Liane Tarouco. *Network Executive: A Policy-Based Network Management Tool*. In Proc. Second Latin American Network Operation and Management Symposium, Lanoms'01. Belo Horizonte, August, 2001.

[CIM] Policy Core Information Model. Available at: http://www.ietf.org/rfc/rfc3060.txt . As accessed in February 2002.

[CiscoQPM] Cisco. QoS Policy Manager. Available at: http://www.cisco.com/warp/public/cc/pd/wr2k/qoppmn/index.shtml. As accessed in March 2002.

[Conover99] Joel Conover. *Policy-Based Network Management*. Available at http://www.networkcomputing.com. As accessed in March 2001.

[COPS] *An Architecture for COPS Based Policy Control Management Framework*. Available at: http://search.ietf.org/internet-drafts/draft-ietf-rap-cops-frwk-00.txt. As accessed in February 2002.

[CS03] Juliana Silva da Cunha and Fabio Q. B. da Silva. *A Contract Specification Language for On-Demand Resource Allocation*. (paper in preparation). June 2003.

[Cunha02] Juliana Silva da Cunha. *Resource Management Using Service Level Contracts: An Approach Based on the Definition and Implementation of Allocation Policies*. PhD Thesis – Federal University of Pernambuco. June 2003 (available in Portuguese and English).

[ExtremeWare] Extreme Networks. ExtremeWare. Available at: http://www.extremenetworks.com/products/datasheets/exware.asp. As accessed in February 2002.

[GY93] Germán Goldszmidt and Yechiam Yemini. *Evaluating Management Decisions via Delegation*. In Proc. IFIP International Symposium on Network Management. San Francisco, USA, April 1993.

[GY95] Germán Goldszmidt and Yechiam Yemini. *Distributed Management by Delegation*. In Proc.15th International Conference on Distributed Computing Systems. Vancouver, Canada, June 1995.

[HPPE] Hewlett-Packard. *HP Policy Expert Homepage*. Available at: http://managementsoftware.hp.com/products/policyexpert/. As accessed in February 2002.

[Soul00] *Plataforma TIS-BR: Uma Aposta Estratégica para a Indústria de Tecnologias da Informação e Comunicação no Brasil*. Sociedade Brasileira Para Promoção da Exportação de Software – SOFTEX. 2000.

[Sloman94] Morris Sloman. *Policy Driven Management for Distributed Systems*. Journal of Network and Systems Management, 1994.

[SS97] M. Mansouri-Samani and M.Sloman. *A Generalized Event Monitoring Language for Distributed Systems*. IEEE/IOP/BCS Distributed Systems Engineering Journal, June 1997.

[Verma00] Dinesh Verma. *Policy Based Networking*. New Riders Publishing, Macmillan Technology Series, 2000.

[Wies95a] René Wies. *Using a Classification of Management Policies for Policy Specification and Policy Transformation*. In Proc. 4th International Symposium on Integrated Network Management. Washington DC, April, 1994.

[YGY91] Y. Yemini, Germán Goldszmidt, S. Yemini. *Network Management by Delegation*. In Proc. 2nd International Symposium on Integrated Network Management. Washington DC, April, 1991.

[Yialelis96] Nicholas Yialelis. *Domain-Based Security for Distributed Object Systems*. PhD Thesis, Imperial College, 1996.

[YLS96] N. Yialelis, E. Lupu, M. Sloman. *Role-Based Security for Distributed Object Systems.* In Proc. Workshop on Project Coordination IEEE WET-ICE. Stanford University, California, June1996.

[YS96] Yialelis, N. and Sloman, M. *A Security Framework Supporting Domain Based Access Control in Distributed Systems*. In Proc. Internet Society Symposium on Network and Distributed System Security. San Diego, February 1996.

Acknowledgments

Fabio Q. B. da Silva receives a research grant from the Brazilian National Council for Scientific and Technological Development – CNPq.

Juliana Silva da Cunha received a research grant from the Brazilian National Council for Scientific and Technological Development – CNPq during the PhD program.