Policy-Based Automation of SLA Establishment for Cloud Computing Services

Mohan Baruwal Chhetri
Center for Complex Software Systems and Services
Swinburne University of Technology
Hawthorn, Australia
mchhetri@swin.edu.au

Quoc Bao Vo
Center for Complex Software Systems and Services
Swinburne University of Technology
Hawthorn, Australia
bvo@swin.edu.au

Ryszard Kowalczyk
Center for Complex Software Systems and Services
Swinburne University of Technology
Hawthorn, Australia
rkowalczyk@swin.edu.au

***Abstract*—We propose a policy-based framework for the automated establishment of SLAs for cloud computing services. The proposed framework supports multiple interaction models for SLA establishment giving consumers and providers the flexibility to choose one that is most appropriate in a given context, while simultaneously supporting multiple concurrent SLA interactions using different interaction models. We describe the underlying policies, focussing on the key features and contributions of the framework. We also validate our framework through a real-world use-case scenario using the Amazon EC2 service.**

***Keywords*-policy-based framework, decision-making strategy, interaction protocol, SLA**

I. Introduction

Cloud computing offers a realization of SOA in which IT resources can be dynamically provisioned to consumers on-demand and using a pay-as-you-go model. When consuming or providing such services, entities establish business relationships with their counterparts which are formally captured in Service Level Agreements (SLAs). These SLAs, include, among other things, the *usage terms and conditions* for the provisioned service, which are a key differentiator in an increasingly competitive cloud services market that is characterised by its diversity and dynamism. *Diversity* results from consumers and providers having varying requirements, capabilities, constraints and preferences over the service usage terms and conditions. *Dynamism* arises from varying supply and demand of the computing resources. Given the diversity and dynamism of the cloud environment, using a single interaction model for SLA establishment such as auction, commodity market, or one-on-one negotiation in all scenarios and contexts may not always be appropriate. Service consumers and providers can benefit from supporting multiple models for SLA establishment, giving them the flexibility to choose the most appropriate interaction model in a given context while at the same time participating in multiple concurrent SLA interactions using different models.

During the process of SLA establishment, all participants have to interact with one another in order to reach an agreement over the service usage terms and conditions. These interactions are usually in the form of message exchanges and are governed by an interaction protocol which defines the "rules of procedure" for the conversation and enables automation and rational decision-making. Service consumers and providers usually have varying and potentially conflicting preferences over the usage terms and conditions, and the process of SLA negotiation and establishment can be viewed as a distributed search through a space of potential agreements [1]. Depending upon the type of SLA interaction model used, the entities use different *decision-making strategies* to try and reach an agreement. For example, if the interaction model is an auction based on the *sealed bid first-price auction protocol*, then all the bidders submit a single sealed bid and hence the strategy has to determine what the bidding price should be. Alternatively, if the service provider and consumer are involved in bilateral negotiation using the *alternating offers protocol*, they have to make decisions about what initial offer to make, what counter-offer to make, whether to accept an offer and when to terminate negotiation.

In this paper, we propose a policy-based framework to support multiple interaction models for the automated establishment of SLAs in diverse and dynamic environments such as the cloud. Policies can be used to capture domain knowledge relating to the interaction protocols, and, the decision-making strategies used during the SLA interactions under different scenarios and contexts. An autonomous policy engine can evaluate incoming requests and the relevant context against the pre-defined policies and initialise the most appropriate SLA interaction model. This paper is complementary to our work presented in [2], where we focussed on policy based preference specification.

The main contributions of this paper are as follows:

- We propose a policy-based framework that enables service consumers and providers to choose the most appropriate SLA interaction model in a given context while at the same time supporting multiple concurrent SLA interactions using different interaction models. By allowing the policy authors to refer to externally defined decision-making strategies, we keep the policy language light-weight while allowing reuse of existing work in the area of automated SLA establishment, particularly the decision-making models [1][5][6][7][8].

- We introduce a policy-based model to support the automated establishment of SLAs. Our proposed model is based on three types of assertions – *context assertions*, *interaction policy (IP) assertions* and *strategy assertions*. These assertions are used in two types of policies - the *interaction protocol (IP) policies*, which specify the interaction protocols supported for automated SLA establishment, and the *strategy policies*, which specify the decision making strategies to use under different scenarios and contexts.
- We extend the WS-Policy framework to provide a domain-independent policy language for specifying the IP policies and strategy policies.

The rest of the paper is organized as follows. Section II gives a brief overview of the main aspects of automated SLA establishment. Section III discusses the Amazon EC2 service which we use as a motivating scenario for our research. We present our formal policy model in Section IV. We present a brief description of the reference architecture and prototype implementation in Section V . We demonstrate the usefulness of our approach through the Amazon EC2 service in Section VI. Section VII discusses related work in the area of policy-based SLA establishment. Section VIII concludes the paper.

II. Overview of Automated SLA Establishment

In the process of SLA establishment, all participating entities (service consumer, service provider, mediator etc.) have to interact with one other in order to reach a common agreement over the service usage terms and conditions. The interactions are in the form of message exchanges with the interaction models varying from auctions to commodity markets to bilateral and multi lateral negotiations. Irrespective of the complexity of the interactions, the process of automated SLA establishment is characterised by four key aspects - the *service usage terms and conditions* and preferences over them, the *interaction protocols*, the *decision-making strategies* [1] and the *interaction context*.

- **Service attribute preferences:** Every service is characterised by a number of attributes that may be customizable and can take on one or more possible values. A service request or offer is essentially an assignment of values to some or all of the service attributes. These preferences and constraints are used to evaluate incoming service requests, to generate offers and counter-offers and to make bids.
- **Interaction Protocols:** Interaction protocols are sets of rules which regulate the different aspects of the interactions including the permissible type of participants, the different states of interaction, the valid actions in the different states and the content of the messages exchanged. All entities participating in the SLA establishment process have to conform to a common protocol to enable automation and rational decision-making.
- **Decision-making strategies:** Service consumers and providers usually have *varying* and *potentially conflicting* preferences over the service attribute values. The process of SLA negotiation can be viewed as a distributed search through a space of potential agreements [1] and the specific strategy chosen determines the traversal path towards the preferred agreement. It helps participants make several decisions such as - what initial offer to make? what counter offer to generate? when to abandon negotiation? when is a proposal acceptable? and when is an agreement reached?
- **Interaction Context:** The interaction context refers to the states and conditions of an enterprise's business which influence the SLA interactions. It can include information about the counterparts such as the size of the company, credit rating of the company, history of previous interactions etc. It can include the business objectives and goals that the enterprise wishes to achieve through the interaction. It can also include the time and resources available to carry out the business interactions i.e. time available to negotiate with the counterparts, deadline by which an agreement has to be reached etc. The interaction context has a very strong influence on the decision-making strategies used as explained in Section III.

With respect to SLA interactions, the protocols are essentially *public documents* that specify the rules of interaction that all participating entities should follow. The preferences and decision-making strategies on the other hand are *private* and not disclosed to the other parties. Each participant uses its own decision-making strategy which is compliant with the selected interaction protocol. The interaction model chosen for SLA establishment depends upon the interaction context as explained in Section IV.

III. Motivating Scenario

We consider the case of Amazon Elastic Cloud Compute (EC2) as a motivating scenario for our research work. We first describe the purchasing models (or using our terminology, SLA interaction models) currently supported by Amazon EC2.

A. Amazon EC2 – Service Provider

One of the key features of the Amazon EC2 service is the flexibility it offers to its customers. Customers have the choice of multiple instance types, operating systems, software packages and geographical locations. In addition to this, Amazon EC2 also provides its customers flexibility in optimising running costs by offering three different purchasing models.

- **On-Demand Instances** – this model lets customers pay for compute capacity by the hour with no long-term commitments or upfront costs. Consumers can increase

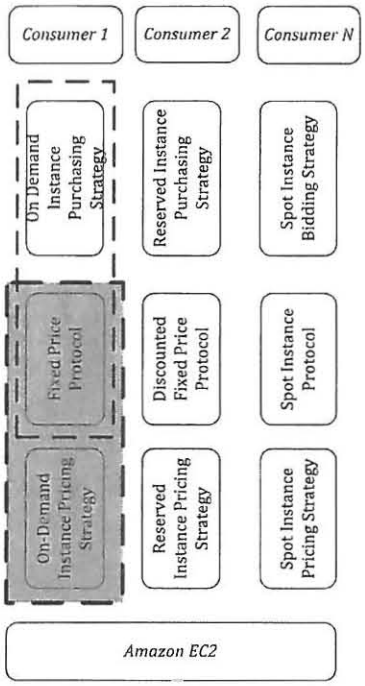


Figure 1. Multiple Concurrent SLA Interactions

or decrease compute capacity on demand and have to pay the fixed hourly rate for the instances used.

- **Reserved Instances** – this model lets customers pay a small one-time, upfront payment for an instance, reserve it for a fixed period of time (one year or three years), and then, pay a significantly lower fixed rate for each hour that the instance is used.
- **Spot Instances** – this model allows customers to bid for unused Amazon EC2 capacity. Customers can specify the maximum hourly price they are willing to pay for a particular instance type. Amazon determines the *Spot Price* based on the bids received and the quantity of unused/idle resources. Customers can access the requested resource as long as their bid price is above the spot price. However, if the bid price drops below the spot price, Amazon shuts down the instance immediately.

In order to automate these three purchasing models, Amazon uses three interaction protocols. The first is the *fixed-price protocol* which is applicable to the on-demand purchasing model, and the second is the *discounted fixed-price protocol* which is applicable to the reserved instance purchasing models. If using the on-demand and reserved instance models, consumers have no flexibility in terms of the price they pay for the resources. But they do have guaranteed and uninterrupted access to the computing resources. The third interaction protocol is the *spot instance protocol* that is used in the spot instance purchasing model, which is based on a *uniform price, sealed-bid, market-driven auction*. *Uniform price* implies that all bidders pay the same price for the resource if they are successful in their bid. *Sealed bid* means that the bids are unknown to other participants and *market-driven* means that the spot price is set according to the client's bids. Using this model, consumers bid the maximum price they are willing to pay for the resource. If they are successful, they have access to the resource and are able to use it until either they choose to terminate it or the new Spot Price becomes higher than their bid. As the service provider, Amazon publicly advertises the SLA interaction models and the associated interaction protocols. But it has its own internal strategy to determine the Spot Price based on all the bids and the available supply of unused resources [16]. Similarly all consumers have their own strategies to select and purchase the resources from Amazon.

B. Service Consumer

Let us consider the scenario where an entity executes jobs on behalf of its customers on the Amazon EC2 infrastructure. In order to to do so, it rents the computing resources on Amazon EC2 as and when required. Each time the entity receives a request, it has to decide how many instances to rent and whether to purchase an on-demand instance or to go for a spot-instance. If purchasing spot instances, it also has to determine the best bid value to use. For the sake of simplicity, we assume that the type of instance required is already fixed as part of the incoming request and the task runs only on a single instance. Each job has to be completed within a certain time, which we refer to as completion time. We refer to the actual time taken to process the job on the specific instance as processing time. Depending upon the current context, the entity can use a number of different strategies to rent the resources and fulfil the incoming request. Let us look at a few of the possible interaction contexts or scenarios and the corresponding strategies that could be used to purchase computing resources from Amazon. The rules for strategy selection based on context take the form *if c then s* and can be described as follows – under a certain context specified by condition c use strategy s. Strategies 2, 3 and 4 are currently being used by Amazon EC2 customers as explained in the video Deciding on Your Spot Bidding Strategy[1].

- **Scenario 1** – *Client wants immediate access to the resource for a short duration.* If the customer wants immediate and uninterrupted access to the computing resource, and the *completion time* is equal to the *processing time*, then the best strategy is to purchase an on-demand instance. In this case, the interaction model chosen is the on-demand model and the price payable is the on-demand price.

$$S_1 : P = P^i_{od}, \text{ where } i \text{ denotes instance type} \quad (1)$$

[1]http://www.youtube.com/embed/WD9N73F3Fao

- **Scenario 2** – *Client wants to minimize the computing cost and job completion time is not a constraint.* If the customer submitting a request is interested in cost optimization and job completion time is not a constraint, then the strategy is to try and pay the lowest price possible for the resources i.e. bid around the reserved instance usage price.

$$S_2 : P_{max} = \kappa \cdot P_r^i, \text{ where } 1 \leq \kappa \leq \frac{P_{od}^i}{P_r^i} \quad (2)$$

where κ is a constant, i denotes instance type, P_r^i denotes reserved instance price and P_{od}^i denotes on-demand price.

- **Scenario 3** – *Client wants to complete the job as quickly as possible and minimize the cost.* If the customer wants to optimize both cost and completion time, then the most appropriate strategy is to use the *Price History Momentum* strategy since it takes into account the previous trends in the pricing history.

$$S_3 : P_{max} = \kappa \cdot P_{avg_n}^i, \text{ where } \kappa \leq 1 \quad (3)$$

where κ is a constant and $P_{avg_n}^i$ is the average spot instance price for the last n hours.

- **Scenario 4** – *Client wants uninterrupted access to the resource for a long duration.* If the customer wants uninterrupted access to the resource to complete the task and still wants to pay lower than the on-demand price, then the strategy is to bid a maximum price which is significantly higher than the on-demand price.

$$S_4 : P_{max} = \kappa \cdot P_{od}^i, \text{ where } \kappa > 1 \quad (4)$$

where κ is a constant and P_{od}^i is the on-demand price for the instance type i.

If we consider more complex scenarios where the task can be executed on multiple instances and the *clients want to make a tradeoff between computation cost and performance*, then more complex strategies [20] can be defined for bidding for resources on Amazon EC2 as explained below.

- **Fixed Bid Strategy** - Given a fixed bid value b and a fixed on-demand rate σ, the strategy is to assign c_j instances for a job j as follows: request $\sigma * c_j$ instances as on-demand requests at price p and (1- σ) * c_j instances as spot instances at price b. If there is a risk of not completing the job on time, the strategy is to switch to on-demand instances only.
- **Variable Bid Strategy** - In this strategy, the bid price for spot instances in each round is set as the weighted average of past spot prices. It has two parameters - a weight γ and a safety parameter ϵ. The bid price is given by $b_s = \frac{1}{Z} \int_y p_s(y) \gamma^{\tau - y} dy + \epsilon$ where b_s is the bid price in the current round, $p_s(y)$ is the history sequence for the past y hours and $Z = \int_y \gamma^{\tau - y} dy$ is the normalization constant

In the examples below we show how policies can be used to specify which strategy to use in a given context.

Some example strategy policies

```
if (completion_time= processing_time )
    then use on_demand_strategy;
if (uninterrupted_access = true && immediate_access =
true)
    then use on_demand_strategy;
if (completion_time >> processing_time)
    then use cost_optimization_strategy;
if (minimize_cost && completion_time)
    then use price_history_momentum_strategy;
if (uninterrupted_access = true && minimize_cost)
    then use minimize_interruption_strategy;
```

IV. SLA Establishment Model

In this section we introduce our formal policy framework for automated SLA establishment. It comprises of two models as explained below:

- *Interaction model* - which allows policy authors to specify the interaction protocols that are supported for automated SLA establishment, and
- *Strategy model* - which allows policy authors to specify which decision-making strategies to use with which interaction protocols under specific interaction contexts.

A. Formal interaction model

In our framework we assume that any entity participating in the service provisioning process can support at least one interaction protocol for SLA establishment. It publicly advertises its list of supported protocols so that other participants can choose the protocol they want to use. By default, the entity initiating the interaction has the right to choose the interaction protocol and the other participant is bound to this selection. Similarly, an entity can have one or more decision-making strategies it can use during its interactions with counterparts in different negotiation contexts. Each strategy conforms to one or more interaction protocols.

Let us assume that an entity participating in the service provisioning process supports a set of interaction protocols $P = \{p_1, p_2, \ldots, p_n\}$ for SLA establishment. Similarly, let $S = \{s_1, s_2, \ldots, s_m\}$ represent the set of available decision-making strategies. Each strategy is a *parametric function* given by $s = f(p, v_1, v_2 \ldots v_k)$ where parameter $p \in P$ refers to the interaction protocol and $v_1, v_2, \ldots v_k$ are the configurable parameters of the strategy. The remaining parameters take their value from a finite domain such that $\mathcal{D} = D_2 \times D_2 \ldots \times D_k$ represents the corresponding set of strategy parameter domains where D_i is the finite set of values that parameter p_i can take.

B. Formal assertion model

- *Context assertion:* A *context assertion* is a triple $A_c \stackrel{def}{=} \langle x, \varphi, v \rangle$ where x is a context attribute, $\varphi \in \Phi$, where

$\Phi = \{<, >, \leq, \geq, \ldots\}$ and $v \in D(x)$ where D is the domain for context attribute x.

- *Interaction-protocol (IP) assertion:* The Interaction Protocol assertion is defined as:

$$A_{ip} \stackrel{def}{=} A_c \rightarrow \{\rho_1, \rho_2, \ldots, \rho_n\} \quad (5)$$

where A_{ip} is the interaction protocol assertion, A_c is the context assertion and $\rho_i \in P$.

- *Strategy assertion:* The strategy assertion is defined as:

$$A_s \stackrel{def}{=} A_c \rightarrow \bigvee_{i \in \{0,..,q\}} \tau(\rho, \sigma_1, \ldots \sigma_k) \quad (6)$$

where A_s is the strategy assertion, A_c is the context assertion, $\tau \in S$ is the applicable strategy, $\rho \in P$ is the protocol to use, and $\sigma_i \in D(v_i)$ is the concrete value for the strategy parameter v_i.

In our current model we assume that all the policies are consistent and there are no conflicts. While conflict detection and resolution is an necessary and important aspect of policy-based management, it is out of the scope of this paper.

C. Formal policy model

There are two types of policies - the *interaction protocol (IP) policies* which specify which protocols are supported, and the private *strategy policies* which specify which strategies to use under different contextual conditions.

1) Policy alternative: A policy alternative is a logical conjunction of zero or more assertions. The *interaction protocol (IP) alternative* consists of a single IP assertion as shown below:

$$P_{alt_{ip}} = A_{ip} \quad (7)$$

Similarly, the *strategy alternative* consists of zero or more strategy assertions and zero or more conditional-strategy assertions as shown below:

$$P_{alt_s} = A_s \quad (8)$$

2) Policy: A policy is a collection of alternatives combined using different policy operators. In its normal form, the *interaction protocol (IP) policy* can be represented as an enumeration of its alternatives as shown below:

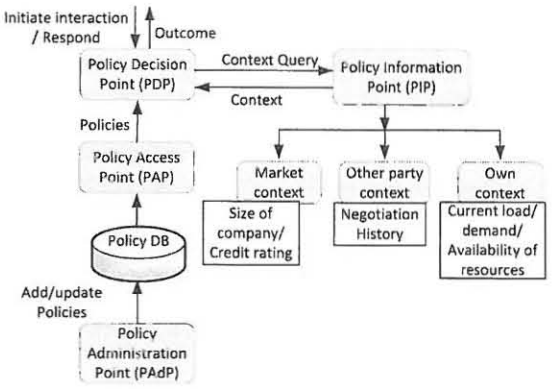
$$P_S = \begin{cases} \bigvee_{i \in \{0,..,q\}} P_{alt_{ip_i}} & : \text{Any} \\ \bigoplus_{i \in \{0,..,q\}} P_{alt_{ip_i}} & : \text{ExactlyOne} \end{cases} \quad (9)$$

where $P_{alt_{ip_i}}$ is given by Equation (7) and $q \in \mathbb{N}$ meaning that a policy can have 0 or more alternatives.

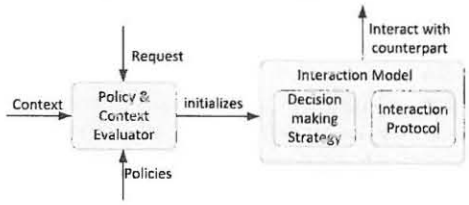
Similarly, the *strategy policy* can be represented as an enumeration of its alternatives as shown below:

$$P_S = \begin{cases} \bigvee_{i \in \{0,..,q\}} P_{alt_{s_i}} & : \text{Any} \\ \bigoplus_{i \in \{0,..,q\}} P_{alt_{s_i}} & : \text{ExactlyOne} \end{cases} \quad (10)$$

where $P_{alt_{s_i}}$ is given by Equation (8) and $q \in \mathbb{N}$ meaning that a policy can have 0 or more alternatives.



(a) Main components of Policy Engine



(b) Policy & Context Evaluation

Figure 2. Reference Architecture of Policy Processing Middleware

V. Reference Architecture & Prototype Implementation

In this section we present the reference architecture (Figures 2(a) and 2(b)) and a proof-of-concept prototype (Figures 3(a) and 3(b)) for our policy-based SLA establishment engine.

A. Reference Architecture

The main components that comprise the policy engine are shown in Figure 2(a):

- **Policy Decision Point (PDP).** This component receives the incoming request and evaluates all the policies that are applicable in the current context. The outcome of the evaluation is sent back as the response to the incoming request.
- **Policy Access Point (PAP).** This component makes available to the PDP all the policies and rules that are applicable in the current context.
- **Policy Information Point (PIP).** This component retrieves all information about the current context.
- **Policy Administration Point (PAdP).** This component is the one through which the business experts, negotiation experts and the domain experts specify their policies.

As shown in Figure 2(a), when an entity initiates the SLA interaction process or responds to a request, the PDP retrieves all the current policies from the PAP and selects the ones which are applicable in the current context by

evaluating the contextual information retrieved from the PIP. Based on the outcome of the policy evaluation, the PDP instantiates the appropriate interaction model with the corresponding decision making strategy and interaction protocol as shown in Figure 2(b). Depending upon whether it is a one-round interaction or multi-round interaction, the interaction module then exchanges messages with the SLA counterpart to try and obtain an outcome. If a common agreement is reached during the interaction, then the policy engine returns a decision to form a SLA. If an acceptable outcome is not achieved, then the PDP returns a failure decision.

B. Prototype Implementation

In order to validate our policy-based approach, we have implemented a proof-of-concept prototype of the policy middleware for automated SLA establishment. It comprises of three key components - a parser which parses WS-Policy rules to the popular Drools[2] format, an embeddable Drools rule engine which evaluates these rules, and a library of executable strategies for purchasing instances from EC2. Each time a request comes in, the policy middleware translates the WS-Policy policies to Drools rules, and passes the incoming request along with the parsed rules to the Drools rule engine. The rule engine determines the most appropriate SLA interaction model to use for the particular SLA interaction and triggers the corresponding strategy. A more detailed description of the overall architecture of the policy-based middleware as well as the prototype implementation will be presented in a future publication.

VI. Use case validation

We have used the Amazon EC2 scenario described in Section III to validate our policy-based approach for automated SLA establishment. In this scenario end-consumers submit their requests to the *Smart Cloud Agent* whenever they have a job to process on EC2. They know which instance type they want and how many instances of it. They have preferences and constraints over the task completion time and the total cost payable, which they specify when they submit their request. The cloud agent (policy engine) evaluates each incoming request against its policy base (Figure 3(a) shows the WS-Policy policy and Figure 3(b) shows the corresponding Drools rule) and determines the most appropriate purchasing model as well as the best bidding strategy. It then initiates the interaction with Amazon EC2 and if purchasing on-demand instances, initiaates the process and starts up the instance. If going for spot-instances, it starts bidding for resources using the selected bidding strategy. If the bid is successful, it starts up the specific instance.

For the input request shown in Figure 4(a), the policy engine chooses the spot instance purchasing model and

[2]http://www.jboss.org/drools

```
▼<tns:Policy xmlns:slam="http://www.swin.edu.au/ws-slampolicy"
 xsi:schemaLocation="http://www.w3.org/ns/ws-policy ws-policy.xsd
 ▼<tns:All>
  ▶<slam:Rule name="Minimize job completion time and cost">...</slam:Rule>
  ▶<slam:Rule name="Immediate Access for Short Duration">...</slam:Rule>
  ▼<slam:Rule name="Minimize Cost and Job Completion Time not a constraint">
   ▼<slam:If>
    ▼<slam:Context identifier="context" objectType="Context">
     ▼<slam:AndConstraintConnective>
      ▼<slam:FieldConstraint field-name="minCost">
        <slam:LiteralRestriction value="yes" evaluator="=="/>
       </slam:FieldConstraint>
      ▼<slam:FieldConstraint field-name="duration">
        <slam:LiteralRestriction value="short" evaluator="=="/>
       </slam:FieldConstraint>
      </slam:AndConstraintConnective>
     </slam:Context>
    </slam:If>
   ▼<slam:Then>
     <slam:Strategy name="CostOptimizationStrategy"/>
    </slam:Then>
   </slam:Rule>
  ▶<slam:Rule name="Uninterrupted Access with Minimum Cost">...</slam:Rule>
  ▶<slam:Rule name="Immediate Access to instance">...</slam:Rule>
  </tns:All>
 </tns:Policy>
```

(a) WS-Policy Example

```
package au.edu.swin.cb.core

import au.edu.swin.cb.context.Context;
import au.edu.swin.cb.ec2.strategies.OnDemandStrategy;
import au.edu.swin.cb.ec2.strategies.PriceMomentumStrategy;
import au.edu.swin.cb.ec2.strategies.CostOptimizationStrategy;
import au.edu.swin.cb.ec2.strategies.MinimizeInterruptionStrategy;
import au.edu.swin.cb.drools.DroolsRuleEngine;

rule "Immediate Access for Short Duration"
when
    context : Context( (immediateAccess == "yes") && (duration == "short") )
then
    OnDemandStrategy odS = new OnDemandStrategy();
    DroolsRuleEngine.getInstance().addStrategy(odS);
end

rule "Minimize job completion time and cost"
when
    context : Context( ((minCost == "yes") && (minCompletionTime == "yes")) )
then
    PriceMomentumStrategy pmS = new PriceMomentumStrategy();
    DroolsRuleEngine.getInstance().addStrategy(pmS);
end

rule "Minimize Cost and Job Completion Time not a constraint"
when
    context : Context( (minCost == "yes") && (duration != "short") )
then
    CostOptimizationStrategy coC = new CostOptimizationStrategy();
    DroolsRuleEngine.getInstance().addStrategy(coC);
end

rule "Uninterrupted Access with Minimum Cost"
when
```

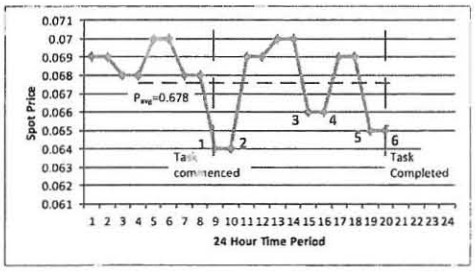
(b) Parsed Drools Rule

Figure 3. Example Policies

chooses the price momentum strategy. The policy engine computes the maximum bidding price as $0.678 based on the past 12 hours spot pricing history which is obtained by querying the Amazon EC2 web service. With the bid price of $0.678, the user is able to start and use the resource when the bid price is above the spot price as shown in the graph in Figure 4(b).

The Smart Cloud Agent is able to make purchasing decisions on behalf of the end-users based on the domain knowledge captured in the form of strategy policies.

(a) Client UI



(b) Spot Price History

Figure 4. Smart Cloud Agent - Automated Selection of EC2 Purchasing Model

VII. Related Work

The three important aspects related to decision making for automated SLA establishment are the *service usage terms and conditions* and *preferences* over them, the *interaction protocols* that govern the interactions between the participating entities, and the *decision making strategies* that are used to try and obtain an agreement [1]. Most research in the area of policy-based automation of SLA establishment focusses on these aspects individually. A lot of work has been done on policy-based preference specification over the negotiation objects [9][10][13][14]. There are fewer works on the use of policy-based approaches for strategy specification and interaction protocol specificaton.

There are several research proposals on policy-based specification of decision-making strategies for automated negotiation. [11][12], [15], [17] and [18] propose the use of declarative rules to capture the decision-making strategies. [17] and [18] do not provide any formal models or concrete examples to illustrate how this can be done. The main limitation of defining strategies declaratively via rules is that while it is sufficient for simple strategies, it is not straightforward for complex strategies which could be based on a number of different approaches such as game-theoretic approaches [5][1], heuristic approaches [6][7] and evolutionary approaches [8]. There has to be a tradeoff between the expressive power of the policy language and the ease of usage. In [19], the authors have proposed the declarative specification of decision-making strategies using an extension of the WS-Policy specification language where the decision-making strategies are defined as parametric functions where the parameter values are specified via the strategy policy. To the best of our knowledge, our paper is the first to support multiple interaction models through the use of a policy-based approach. We allow the policy authors to specify which strategy to use under different contexts, so that the policy engine can autonomously make decisions that conform to these policies at run-time. Our approach also enables reuse of existing research results since we allow externally defined strategies to be referred to within our policies and separate the strategy reference from the actual implementation.

VIII. Conclusion

In this paper, we have presented a novel policy-based framework for the automated establishment of SLAs in open, diverse and dynamic environments such as the cloud. Using our framework, entities have the flexibility to choose the most appropriate SLA interaction model in a given context while at the same time participating in multiple concurrent interactions using different SLA interaction models. This is possible through the use of three new policy assertions - context assertion, interaction protocol assertion and strategy assertion. We have extended the WS-Policy framework to provide a light-weight and simple yet expressive and flexible policy language for policy specification. We have implemented a proof-of-concept prototype and validated our approach with the Amazon EC2 service where EC2 consumers can delegate the task of purchasing instances to a smart cloud agent which makes use of the pre-defined policies to choose the most appropriate purchasing model and bidding strategy based on relevant context.

The process of automated SLA establishment is characterised by the preferences over the service usage terms and conditions, and the decision-making strategies and protocols. While we have developed policy models for capturing preferences over the service usage terms and conditions, and for supporting multiple SLA interaction models, the two models are currently independent of each other. But if we consider the decision-making strategies as parametric functions, the configurable parameters of the strategy are usually dependant upon, or can be derived from the preferences over the usage terms and conditions. Hence, it makes sense to

integrate the two models into a common framework. Future work will investigate into the different ways in which the different types of policies i.e. preference policies, strategy policies and interaction protocol policies can be combined to provide a unified policy framework for automated SLA establishment in dynamic and diverse SOA environments such as the cloud.

ACKNOWLEDGMENT

This work was partially funded by the Service Aggregation Project within the Smart Services CRC which is proudly supported by the Australian Federal Government's CRC Grant Program.

REFERENCES

[1] N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge, *Automated Negotiation: Prospects, Methods and Challenges*, International Journal of Group Decision and Negotiation, 10 (2). pp–199-215, (2001)

[2] M. Baruwal Chhetri, B. Q. Vo, R. Kowalczyk, *A Flexible Policy-based Framework for the QoS Differentiated Provisioning of Services*, In *The 11th International Symposium on Cluster, Grid and Cloud Computing*, NewPort, California, USA, May 23-26, (2011)

[3] M. Baruwal Chhetri, B. Q. Vo, R. Kowalczyk, *Policy-based management of QoS in Service Aggregations*, In *The 10th International Symposium on Cluster, Grid and Cloud Computing*, Melbourne, Australia, May 17-20, (2010)

[4] J. O. Kephart and R Das: *Achieving Self-Management via Utility Functions*, IEEE Internet COmputing, Vol. 11, No. 1, pp–40-51 (2007)

[5] K. Binmore, and V. Nir, *Applying game theory to automated negotiation*, NETNOMICS, Vol.1, No.1 (1999)

[6] P. Faratin, C. Sieera, and N. R. Jennings: *Using similarity criteria to make trade-offs in automated negotiations*, Artificial Intelligence, Vol. 142, No. 2, pp–205-237 (2002)

[7] R. Kowalczyk, *Fuzzy e-negotiation agents*, Soft Computing, Vol 6. No. 5, pp–337-347 (2002)

[8] S. S. Fatima, M. Wooldridge, and N. R. Jennings, *A comparative study of game theoretic and evolutionary models of bargaining for software agents*, Artificial Intelligence Review, Vol. 23, No. 2, pp–187-208 (2005)

[9] E. M. Maximilien, M. P. Singh, *A Framework and Ontology for Dynamic Web Services Selection* in *IEEE Internet Computing*, Sep./Oct. 2004, vol. 8, no. 5, pp. 84-93

[10] S. Chaari, Y. Badr and F. Biennier, *Enhancing Web Service Selection by QoS-Based Ontology and WS-Policy*, In Proceedings of the 2008 ACM Symposium on Applied Computing, pp. 2426-2431, 2008

[11] H. Gimpel, H. Ludwig, A. Dan and B. Kearney, *PANDA: Specifying Policies for Automated Negotiations of Service Contracts*, In Proceedings of ICSOC 2003, pp. 287-302 (2003)

[12] T. Skylogiannis, G. Antoniou and N. Bassiliades, *DR-NEGOTIATE - A System for Automated Negotiation With Defeasible Logic-Based Strategies* in Proceedings of IEEE International Conference on e-Technology, e-Commerce and e-Service, pp. 44-49 (2005)

[13] S. Lamparter and D. Oberle, *Approximating service utility from policies and value function patterns*, In the Proceedings of *6th IEEE International Workshop on Policies for Distributed Systems and Networks*, IEEE Computer Society, 2005, pp. 159-168

[14] S. Lamparter, D. Oberle and C. Weinhardt, *A Policy Framework for Trading Configurable Goods and Services in Open Electronic Markets*, In Proceedings of *ICEC'06*, August 2006, Frederiction, Canada, pp. 162-173

[15] H. Li, S. Y. W. Su, and H. Lam, *On Automated e-Business Negotiations: Goal, Policy, Strategy, and Plans of Decision and Action*, Journal of Organizational Computing and Electronic Commerce, Vol. 13, No. 1, pp–1-29 (2006)

[16] O. A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster and D. Tsafrir, *Deconstructing Amazon EC2 Spot Instance Pricing*, In Proceedings of 3rd IEEE International Conference on Cloud Computing Technology and Science,

[17] F. Zulkernine, P. Martin, C. Craddock, et.al., *A Policy-Based Middleware for Web Services SLA Negotiation*, In Proceedings of IEEE International Conference on Web Services (ICWS2009), pp. 1043–1050, (2008)

[18] Z. Xiao, D. Cao, C. You and H. Mei, *A Policy-based Framework for Automated Service Level Agreement Negotiation*, In Proceedings of IEEE International Conference on Web Services, pp. 682–689 (2011)

[19] M. Comuzzi and B. Pernici, *An Architecture for Flexible Web Service QoS Negotiation*, In Proceedings of Ninth IEEE International EDOC Enterprise Computing Conferences, pp. 70–79 (2005)

[20] N. Jain, I. Menache, O. Shamir, *On-demand or Spot? Learning-based Resource Allocation for Delay-Tolerant Batch Computing*, available online at http://research.microsoft.com/en-us/um/people/navendu/papers/lbr_infocom.pdf, accessed on 25 November 2011

[21] M. Baruwal Chhetri, Q. B. Vo, R. Kowalczyk and C. L. Do, *Cloud Broker: Helping You Buy Better*, In Proceedings of Web Service System Engineering Conference (WISE2011), pp. 341–342 (2011)