SPECIAL ISSUE PAPER

AutoSLAM – A policy-based framework for automated SLA establishment in cloud environments

M. Baruwal Chhetri*,†, Q. Bao. Vo and R. Kowalczyk

Faculty of Information & Communication Technologies, Swinburne University of Technology, Melbourne, Australia

SUMMARY

Cloud computing offers a realization of SOA in which IT resources are dynamically provisioned as services to consumers using flexible provisioning and pricing models. When provisioning such services, providers and consumers must first agree over the service usage terms and conditions, which are captured in Service Level Agreements (SLAs). In this paper, we propose a policy-based framework with corresponding models, mechanisms and tools for the automated establishment of SLAs in open, diverse and dynamic cloud environments. The Automated SLA Management framework allows entities to specify their requirements and capabilities, and preferences over them in a flexible and expressive manner. It also supports multiple interaction models for SLA establishment, giving consumers and providers the flexibility to select the one that is most appropriate in a given context, while simultaneously participating in multiple concurrent SLA interactions using different interaction models. As part of the framework, we define a formal model for the underlying policies, a corresponding physical model WS-SLAM that extends WS-Policy and a reference architecture that can be easily implemented. We validate the practicability of our framework through the Smart CloudPurchaser prototype that can automatically purchase computing resources from Amazon EC2 under different scenarios and contexts. Copyright © 2013 John Wiley & Sons, Ltd.

Received 15 October 2012; Revised 19 July 2013; Accepted 17 September 2013

KEY WORDS: policy-based automation; automated SLA establishment; cloud computing; preference policy; interaction policy; strategy policy

1. INTRODUCTION

Cloud computing offers a realization of SOA in which IT resources can be dynamically provisioned as services to consumers using flexible provisioning and pricing models. When consuming or providing such services, entities establish Service Level Agreements (SLAs) with their counterparts. These SLAs, include, among other things, the usage terms and conditions for the provisioned service. These usage terms and conditions are a key differentiator in an increasingly competitive cloud services market, which is characterized by its diversity and dynamism. Diversity results from consumers and providers having varying requirements, preferences and constraints over the service usage terms and conditions. Dynamism arises from varying supply and demand of the computing resources. Given the diversity and dynamism, using a static set of preferences and a single interaction model for SLA establishment is not appropriate in all scenarios and contexts. Service consumers and providers can benefit if they have flexible and expressive preference models and support multiple interaction models and decision-making strategies for SLA establishment, so that they can dynamically adapt to changing scenarios and contexts.

*Correspondence to: M. Baruwal Chhetri, Faculty of Information & Communication Technologies, Swinburne University of Technology, John Street, Hawthorn, VIC 3122, Australia.
†E-mail: mchhetri@swin.edu.au

Copyright © 2013 John Wiley & Sons, Ltd.
During the process of SLA establishment, participants have to interact with one another in order to reach an agreement over the service usage terms and conditions. These interactions are generally characterized by three key aspects—preferences over the usage terms and conditions, the interaction protocols that govern the interactions and the decision-making strategies that guide these interactions. The preferences are used to evaluate requests and offers, and to generate offers and counter offers. The interaction protocols define the rules of procedure for the conversation that all participating entities have to conform to, enabling automation and rational decision-making. Because service consumers and providers usually have varying and potentially conflicting preferences over the usage terms and conditions, the process of SLA establishment can be viewed as a distributed search through a space of potential agreements [1]. Depending upon the type of interaction model used, the entities can 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 where all bidders have to submit a single sealed bid, then the strategy has to determine what the bid 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. A fourth aspect that is critical to the SLA establishment process is the diverse, dynamic, volatile and transient interaction context, which has a profound impact on the decision-making behavior.

We propose a policy-based approach for the automation of SLA establishment in open, diverse and dynamic environments. From an Artificial Intelligence perspective, policies can be regarded as a form of guidance from human administrators that determine the decisions and actions of computer systems [2]. The goal of autonomic computing is to shrink the gap between an enterprise’s business goals and objectives, and the IT implementation necessary to achieve them [3]. Automated SLA establishment presents a rich source of challenges for autonomic computing [4] with enterprises needing flexible ways to capture multi-attribute preferences, effective decision-making strategies and appropriate protocols that help establish the rules of interaction and govern the flow of messages among participants. Entities can specify their requirements and capabilities, and preferences over them in a flexible and expressive manner using preference policies. Similarly, interaction policies can specify the supported interaction protocols, and strategy policies can specify the decision-making strategies to be used during SLA interactions under different scenarios and contexts. These policies can be interpreted by an autonomous policy engine, which can exhibit flexible problem solving behavior with minimal human intervention. Our proposed approach has two key benefits. Firstly, the use of policies bridges the gap between the high-level business goals and the low-level strategies used to achieve them. Mapping business goals to negotiation strategies ensures that the SLA negotiation is driven by the business goals of an enterprise. Any change in the business goal directly impacts the negotiation strategy and tactics used, ensuring that the negotiation outcomes are consistent with the enterprises’ strategic goals. Secondly, using policies to manage the SLA establishment process allows the reuse of existing results in the different areas of automated SLA establishment including the preference models, negotiation protocols and decision-making algorithms.

1.1. Contributions

In this paper, we present Automated SLA Management (AutoSLAM) – our unified and flexible policy-based framework for the automated establishment of SLAs in highly dynamic and diverse environments such as the cloud. While our previous works including [5–8] discussed different aspects of the AutoSLAM framework, this paper complements them by providing a unified view of our results and give additional details about our recent extensions. Our key contributions in the area of policy-based SLA establishment are as follows:

- We present a formal generic QoS preference model that improves the state-of-the-art in policy-based preference specification by combining cardinal and ordinal preferences. We introduce the utility-value assertion, which, when combined with a comprehensive cardinal utility function covering the entire configuration space, is sufficient on its own for specifying preferences. If the cardinal utility function is not comprehensive (which is generally the case) and covers...
only a subset of the configuration space, additional attribute-value and conditional attribute-value assertions can be used to specify ordinal preferences (ordinal utility allows the ranking or ordering of different service configurations). Policy authors can combine cardinal and ordinal preferences to allow maximum flexibility in the offered service levels thereby maximizing the chances of forming agreements.

- We present a formal interaction model that supports multiple interaction models for SLA establishment, giving service consumers and providers the flexibility to choose the one that is most appropriate in a given context while simultaneously participating in multiple concurrent interactions using different SLA interaction models. We make use of condition-action rules to specify conditional assertions over the supported SLA interaction models and strategies. The condition part captures the context surrounding the SLA interaction, while the action part specifies the interaction model to be used for SLA establishment. We define three types of assertions – context assertions, interaction policy (IP) assertions and strategy assertions. These assertions are used in two types of policies – the interaction protocols (IP) policy, which specifies 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 provide a reference architecture for the AutoSLAM middleware, which can be easily implemented and extended by others. In AutoSLAM, we formally describe preference policies and interaction policies using WS-SLAM, our novel extension of the WS-Policy language [9]. We also provide a reference implementation of the policy-processing middleware, which uses the Drools Rule Engine [10] to reason about interaction policies. We do this by first parsing the WS-SLAM policies into Drools rules, which are then fed to the rule engine. The AutoSLAM middleware intercepts each incoming request and determines which SLA interaction model to use for SLA establishment. We validate the practicability of our framework by implementing the Smart CloudPurchaser prototype, which automatically purchases computing resources from Amazon EC2 under different contexts as specified by the interaction rules.

1.2. Paper organization

The rest of the paper is organized as follows. We give an overview of automated SLA establishment in Section 2 followed by a discussion of the Amazon EC2 service, which we use to motivate our research work in Section 3. In Section 4, we briefly discuss the relevant concepts associated with our proposed approaches for specifying QoS preferences and interaction rules. We present our formal policy model in Section 5 and the reference architecture for the policy-processing middleware in Section 6. We briefly describe the AutoSLAM middleware implementation in Section 7. In Section 8, we demonstrate the practicability and usefulness of our approach through the Smart CloudPurchaser prototype that automatically interacts with the Amazon EC2 service to purchase computing resources under different contexts. We discuss some limitations of our work in Section 9 and present related work in the area of policy-based SLA establishment in Section 10. Section 11 concludes the paper.

2. OVERVIEW OF AUTOMATED SLA ESTABLISHMENT

When entities participating in the service provisioning process (i.e., service providers and service consumers) enter into service partnerships, the specifics of these business relationships are captured in SLAs or service contracts. The business relationships can either be one-off partnerships or ongoing, long-term relationships, which involve multiple interactions. Irrespective of the nature and duration of the partnership, the service contracts formally capture the specifics of the relationship including a description of the provided service, the responsibilities and guarantees of all involved parties, and the usage terms and conditions (e.g., payment for the service, or service level objectives). Each SLO is related to a specific metric and has a guaranteed value, a way to measure it and a penalty incurred in case of non-fulfillment\textsuperscript{2}. During the process of SLA establishment, all

\textsuperscript{2}Please refer to [11] for a comprehensive list of the most featured QoS metrics in literature.
participating entities interact with one other in order to reach a common agreement over the service usage terms and conditions, both from a functional and non-functional, or QoS perspective. The interactions can vary in their complexity ranging from the simple offers-based approach and dynamic pricing, to more complex bilateral and multilateral negotiations, auctions and commodity markets. Irrespective of the complexity of the interactions, the process of automated SLA establishment is characterized by four key aspects – the **preferences** over the service usage terms and conditions, the **interaction models** comprising **interaction protocols** and **decision-making strategies**, and the **interaction context**.

- **Service attribute preferences**: A service is normally characterized by multiple customizable attributes that can take on one or more possible values. Consumers and providers usually have varying and potentially conflicting preferences over the values that the service attributes can take. Hence, the process of SLA establishment can been seen as a form of decision-making where consumers and providers try to find a mutually acceptable solution through the exchange of offers and counter-offers. Preferences are used to evaluate incoming service requests, to generate offers and counter-offers and to make bids. Hence, one of the key requirements of automated SLA establishment is the support for expressive and flexible preference statements.

- **Interaction protocols**: In order to reach an agreement, participating entities have to interact with each other. The interaction protocol defines how an entity interacts with other entities during the service provisioning process. It provides the rules that regulate the different aspects of the interactions including the different states of interaction, the valid actions in the different states and the content of the message exchanged. All entities participating in the SLA establishment process must conform to a common protocol to enable automated and rational decision making, that is, the interaction protocol is a public shared specification. An AutoSLAM system should be able to support multiple interaction protocols as there is no standard protocol that can be used for automated SLA establishment in all scenarios and contexts.

- **Decision-making strategy**: The process of SLA negotiation can be viewed as a distributed search through a space of potential agreements [1], during which participants make several decisions such as what initial offer to make? what counter offer to generate? when to abandon negotiation? and when to accept an offer. The specific strategy chosen determines the traversal path towards the preferred agreement. An enterprise may choose to use different decision-making strategies and algorithms depending upon the service provisioning context, that is, the decision-making strategies are private and not shared with the counterparts. Hence, an AutoSLAM system should support a wide range of decision-making strategies and algorithms and be able to decide which strategy to use with which protocol depending upon the contextual conditions.

- **Interaction context**: There is no universally best approach or technique for automated negotiation. Rather, participants have an eclectic bag of interaction protocols and decision-making strategies they can choose from; the choice depends upon the interaction context. The interaction context refers to any situational information that affects the interaction process. It captures the states and conditions that are relevant to an enterprise during the process of SLA establishment. It can include information about the counterparts (size of the company, credit rating of the company, history of previous negotiations etc.), about itself (current load, current demand, availability of resources etc.) and the market in general. In open and dynamic environments like the cloud, the interaction context is likely to change constantly. Depending upon the context, different strategies may have to be used in different scenarios.

### 3. 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. We then use simple examples to show how EC2 clients can specify their requirements and preferences through preference policies and use context conditions to determine selection of purchasing model through interaction rules.
3.1. 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 optimizing 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 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* that is applicable to the on-demand purchasing model, and the second is the *discounted fixed-price protocol* that 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 and 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 clients’ 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 on-demand prices, the reserved prices and the spot prices [12]. Similarly, all consumers have their own strategies to select and purchase the resources from Amazon as shown in Figure 1.

```
Figure 1. Multiple concurrent Service Level Agreement interactions.
```
Table I. Example preference statements.

<table>
<thead>
<tr>
<th>Preference Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Memory $\geq$ 4 GB;</td>
</tr>
<tr>
<td>2. Processing-power $\geq$ 5 ECU;</td>
</tr>
<tr>
<td>3. If (io-performance = very-high) then memory $\geq$ 10 GB;</td>
</tr>
<tr>
<td>4. Job deadline $\leq$ 6 h;</td>
</tr>
<tr>
<td>5. If (job_deadline $\leq$ 6 h) then memory $\geq$ 15 GB &amp;&amp; processing-power $\geq$ 10 ECU;</td>
</tr>
<tr>
<td>6. If (job_deadline $\geq$ 24 h) then 8 GB memory $\leq$ 15 GB &amp;&amp; 7 ECU processing-power $\leq$ 15 ECU;</td>
</tr>
</tbody>
</table>

3.2. Service consumer

Consumers can choose any one of the three purchasing models to purchase computing resources on Amazon EC2 based on their specific situation. As a simple illustrative example, let us consider the scenario where an entity executes jobs on behalf of its customers on the Amazon EC2 infrastructure. In order to do so, it rents the computing resources as and when required. Each time the entity receives a request, it has to decide which instance type to rent, 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. Depending upon the context, the entity can use a number of different strategies to rent the resources. Of course, in-depth knowledge of the application domain would be required in order to specify such decision rules.

3.2.1. Preference specification. Preferences may be specified in terms of the absolute minimum and maximum values for individual attributes – for example, the minimum memory required, the minimum storage space, the geographic location or the job completion time. Conditional preferences may also be specified in terms of the infrastructure requirements under different job completion deadlines. For example, if the deadline is too short and the job is high-memory intensive, then it is better to go with a more powerful instance. On the other hand, if the deadline is long, then the objective may be to minimize the cost and go for the cheapest machine that will complete the job within the stipulated deadline. Table I shows some example preference statements. Based on such preference statements, it is possible to find the instance type offered by Amazon EC2 that best matches the consumer requirements. The preferences could be specified at the infrastructure level or at the application level (in which case there would be appropriate mapping to the infrastructure level).

3.2.2. Selection of purchasing model. Once the appropriate instance type to be used has been selected, the EC2 client has to next decide which purchasing model to use in order to procure the resource. Depending upon the current context, the entity can use either the on-demand purchasing model or the spot-instance purchasing model to rent the resources and fulfill the incoming request. If using the spot-instance purchasing model, it can choose from a number of different strategies. Let us look at a few possible interaction contexts or scenarios, and the corresponding interaction models that could be used to purchase computing resources from Amazon. The interaction rules take the form $\text{if } c \text{ then } m(p,s)$ and can be described as follows – under a certain context specified by condition $c$, use specific interaction model $m$ with interaction protocol $p$ and decision-making 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.

- **Scenario 1 – Context:** The client wants immediate and uninterrupted access to the computing resource for a short duration, that is, the job completion time is approximately equal to the request processing time. **Interaction Model & Strategy:** The best interaction model is the

---

1We do not consider the reserved instance purchasing model as it requires a prior subscription and does not afford the just-in-time purchasing of resources that we consider in this scenario.

2http://www.youtube.com/embed/WD9N73F3Fao
on-demand purchasing model and the maximum price the client has to pay is the on-demand price.

\[ P_{\text{max}} = P_{\text{od}}^i, \text{ where } P_{\text{od}}^i \text{ denotes on-demand price} \]  

(1)

- **Scenario 2 – Context:** The client wants to minimize the computing cost, and job completion time is not a constraint. **Interaction Model and Strategy:** The best purchasing model is the spot-instance purchasing model, and the best bidding strategy is to be conservative and try and pay the lowest price possible by bidding around the reserved instance usage price.

\[ P_{\text{max}} = \kappa \cdot P_{r}^i, \text{ where } 1 \leq \kappa \leq \frac{P_{r}^i}{P_{\text{od}}^i} \]  

(2)

where \(\kappa\) is a constant, \(i\) denotes instance type, \(P_{r}^i\) denotes reserved instance price and \(P_{\text{od}}^i\) denotes on-demand price.

- **Scenario 3 – Context:** The client wants to complete the job as quickly as possible and at the same time wants to minimize cost. **Interaction Model and Strategy:** The most appropriate purchasing model is the spot-instance purchasing model, and the most appropriate strategy is the Price History Momentum strategy, which takes into account the previous trends in the pricing history.

\[ P_{\text{max}} = \kappa \cdot P_{\text{avg}}^i, \text{ where } \kappa \leq 1 \]  

(3)

where \(\kappa\) is a constant and \(P_{\text{avg}}^i\) is the average spot instance price for the last \(n\) hours.

- **Scenario 4 – Context:** The client wants uninterrupted access to the resource for a long duration. **Interaction Model and Strategy:** 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 purchasing model to be used is the spot-instance purchasing model and the strategy is to bid a maximum price, which is significantly higher than the on-demand price. This strategy maximizes the chances of having uninterrupted access, while the actual spot price is likely to be much lower than the on-demand price.

\[ P_{\text{max}} = \kappa \cdot P_{\text{od}}^i, \text{ where } \kappa > 1 \]  

(4)

where \(\kappa\) is a constant and \(P_{\text{od}}^i\) is the on-demand price for the instance type \(i\).

4. **UTILITY THEORY AND RULE-BASED EXPERT SYSTEMS**

The AutoSLAM framework is based on two paradigms – utility theory and rule-based expert systems. In this section, we first describe the concepts of utility theory and utility functions and show through simple examples how we can combine cardinal and ordinal utility to improve preference specification. We then discuss the AI concept of production rules, which is one of the more popular approaches for knowledge representation and use simple examples to illustrate how condition-action rules can be used to specify which interaction model and decision-making strategy to use under different contexts and scenarios.

4.1. **Utility theory and utility function**

In Economics [13], utility is defined as a measure of the relative satisfaction from or desirability of consumption of a product or service. It can generally be expressed in two ways – as ordinal utility and as cardinal utility. Ordinal utility theory states that while the utility of a particular good or service cannot be measured using a numerical scale, different alternatives can be ordered or ranked. Alternatively, cardinal utility allows the measurement of the strength of preference of a good or service with precision through the use of some objective criteria. Theoretically, an objective function can be defined, which can assign a scalar utility value to every possible service configuration. Such an objective utility function \(U\) is represented as \(U : C \rightarrow [0,1]\) and allows for unambiguous and rational decision making, facilitating automation of the decision-making process. When service
consumers and providers express their preferences using policies, they try to elicit this function \( U \). However, in practice, it is seldom possible to define such a comprehensive function, and hence, it may be beneficial to combine cardinal and ordinal utility to allow for greater coverage of the configuration space.

### 4.1.1. Cardinal utility
We can assume that every service configuration has an associated utility, but whether stakeholders can assign a value to it depends upon how comprehensive and inclusive their utility function is. There are different ways in which the utility function can be defined, and it can cover either the entire configuration space or only a part of it. To illustrate this, let us consider a service that has only one customizable QoS attribute \( Availability \) with a domain in the range \([0–1]\). Let us now see how the stakeholders can specify their preference over \( Availability \) using utility functions.

- **Point-based utility function:** One way to define the utility for \( Availability \) is by using the point-based utility function approach as shown in Equation 5. In this example, the point-based utility function assigns utility values for three specific values for \( Availability \). Obviously, this definition does not cover the entire assignment space.

\[
U_{Availability}(a) = \begin{cases} 
0.7 & a = 0.8 \\
0.65 & a = 0.65 \\
0.6 & a = 0.6
\end{cases}
\] (5)

- **Numeric utility function:** Another way to define the utility for \( Availability \) is to use a numeric function shown as follows:

\[
U_{Availability}(a) = \begin{cases} 
0 & a < 0.7 \\
0.5 & 0.7 \leq a < 0.9 \\
0.7 & 0.9 \leq a < 0.95 \\
1 & 0.95 \leq a < 1
\end{cases}
\] (6)

An alternate numeric function definition from [14] shown as follows:

\[
U_{Availability}(a) = K \frac{e^{\alpha(\beta-a)}}{1 + e^{\alpha(\beta-a)}}
\] (7)

where \( K \) is a normalization factor, which is given as

\[
K = \left(1 + e^{\alpha(\beta-1)}\right) / e^{\alpha(\beta-1)}
\] (8)

In Equation 8, \( \beta \) is the best preferred value for \( Availability \), and \( \alpha \) is a sensitivity parameter that defines the sharpness of the utility curve. It should be noted that Equation 7 is just one of many possible utility function definitions. As can be seen from Equations 6 and 7, instead of explicitly specifying the utility for specific values of \( Availability \) as in Equation 5, a numeric function can be defined, which can compute the utility for a number of possible values of \( Availability \). Again, this utility function can be constrained to a subset of all possible values by specifying the upper and lower bounds using constraint operators. As a simple example, the utility function in Equation 6 can be constrained to a subset of the potential assignment space shown as follows:

\[
U_{Availability}(a) = \begin{cases} 
0.7 & 0.9 \leq a < 0.95 \\
1 & 0.95 \leq a < 1
\end{cases}
\] (9)

This particular definition in Equation 9 specifies the cardinal utility for all values above 0.9. However it does not specify the utility for values below 0.9. Hence, it cannot be considered as a comprehensive utility function and cannot be used to evaluate service configurations with \( Availability \) below 0.9.

Thus, while numeric and point-based utility functions provide an expressive approach to preference specification, they may not cover the entire configuration space and hence may not be sufficient to assign a utility value for every service configuration. We propose combining the numeric utility...
function with utility-value assertion, attribute-value assertions and conditional attribute-value assertions to provide a more flexible way to express the stakeholder’s utility. This allows policy authors greater flexibility in constraining or expanding the acceptable configuration space.

4.1.2. Combining assertions. We now show how we can combine different types of assertions to improve the elicitation of utility. For simplicity, we continue with our example of a service with a single customizable QoS attribute Availability. We use the utility-value assertion and the attribute-value assertion separately to illustrate the benefit of complementing numeric utility functions with either of these assertions.

- **Utility-value assertion**: The utility function specifies how the utility is to be computed for a specific configuration and can cover a subset of or the entire configuration space. The utility-value assertion specifies the minimum acceptable utility and constrains the size of the acceptable configuration space by imposing a lower bound on the utility. Let us consider the numeric utility function given in Equation 6, which covers the entire configuration space between 0 and 1. A stakeholder (either the service provider or the consumer) can specify the minimum acceptable utility through the utility-value assertion shown as follows.

\[ U_{Availability_{min}}(a) \geq 0.6 \]  

(10)

This constrains the acceptable configuration space as shown in Figure 2(a). The value for the minimum acceptable utility can be changed at any time thereby constraining or relaxing the acceptable configuration space.

- **Attribute-value assertion**: Let us consider the numeric utility function given in Equation 9. It covers only a subset of the entire configuration space. It gives a scalar utility only for values of Availability between 0.9 and 1.0. In such a scenario, the stakeholder can further relax the acceptable configuration space by using a simple attribute-value assertion shown as follows:

\[ a \geq 0.8 \]  

(11)

This attribute-value assertion relaxes the acceptable configuration space as shown in Figure 2(b). The numeric utility function allows computing the utility for all values of availability between 0.9 and 1.0. In addition, the attribute-value assertion states that a service configuration with availability greater than 0.8 is also acceptable even though the exact utility value cannot be computed. Thus, combining the utility-value assertion and attribute-value assertion allows elicitation of both cardinal and ordinal utility.

- **Conditional attribute-value assertion**: When the numeric utility function is defined over a set of QoS attributes and covers only a subset of the potential configuration space, the conditional attribute-value assertion, which is of the form if p then q, can be used to further relax the size of the acceptable configuration space. There are different ways in which the conditional attribute-value assertions can be used as illustrated by the following three simple examples:
The simplest usage of conditional attribute-value assertion is to enforce additional constraints under a certain context. It can be described as follows – under a certain context (specified by the condition $p$), further constraints (specified by the attribute-value assertion $q$) need to be enforced and/or introduced as part of the agreement. A simple example is if ($\text{ResponseTime} > 100 \text{ ms}$), then ($A \geq 0.99$), which states that if response time is greater than 100 ms, then availability is required to be at least 0.99.

Another usage of the conditional attribute-value assertion is to specify that an attribute-value assertion $p$ cannot be accepted in general unless it is introduced in a specific context specified by the attribute-value assertion $q$. A simple example is – if ($A = 1.00$) then ($\text{Premium} \geq 70\$), which implies that availability of 100% is acceptable only if the customer is paying a premium of at least 70$.

A third usage of the conditional attribute-value assertion is for specifying preferences over interdependent QoS attributes. For example, when execution time is very high, it does not make sense to have a very high throughput; hence, an example conditional attribute-value assertion could be – if ($\text{ET} > 100 \text{ ms}$), then ($T \leq 10 \text{ tps}$); or in the case of a video-on-demand application scenario, if the content is in low-resolution or audio-only, then the connection using shared channels is sufficient; on the other hand, if the content is high-resolution video, the connection using dedicated channels is preferred.

4.2. Expert systems and production rules

In Artificial Intelligence, reactive planning refers to a group of techniques used by autonomous entities for action selection. One such technique uses Condition-action rules (or if–then rules), which take the form: if condition then action. They are also referred to as production rules and are interpreted as follows: if the condition holds, perform the associated action. Production rules capture expert knowledge and form the basis for rule-based expert systems, which are widely used in many fields. Table II shows some example decision-rules that an EC2 consumer may use to capture appropriate knowledge that influences the purchasing decision of computing resources on Amazon EC2.

A rule-based system uses a set of assertions, collectively referred to as the ‘working memory’, and a set of rules that specify how to act on the assertion set (Figure 3). A typical rule-based system has the following four components:

- **Rule base** – which captures all of the appropriate knowledge in the form of if–then rules. The conditions are expressed logically as conjunctions or disjunctions of predicates. The conditions of the rules are testing against the working memory.
- **Fact base/Working Memory** – which contains all known information that is relevant to the reasoning by the inference engine
- **Inference engine** – which applies the rules to the working memory and infers the actions to be taken. It does this in two-phases. First, it matches the left-hand side conditions of all rules against the content of the working memory. As a result, a conflict set of applicable rules is obtained. In the second phase, it chooses a single rule for execution using a conflict-resolution strategy. Some common conflict resolutions strategies include the following:

Table II. Example interaction rules.

<table>
<thead>
<tr>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. If ($\text{completion_time} \approx \text{processing_time}$) then use od_purchasing_model;</td>
</tr>
<tr>
<td>2. If ($\text{uninterrupted_access} = \text{true} &amp;&amp; \text{immediate_access} = \text{true}$) then use od_purchasing_model;</td>
</tr>
<tr>
<td>3. If ($\text{completion_time} \gg \text{processing_time}$) then use si_purchasing_model with cost_optimization_strategy;</td>
</tr>
<tr>
<td>4. If ($\text{minimize_cost} = \text{true} &amp;&amp; \text{minimize_completion_time} = \text{true}$) then use si_purchasing_model with price_history_momentum_strategy;</td>
</tr>
<tr>
<td>5. If ($\text{uninterrupted_access} = \text{true} &amp;&amp; \text{minimize_cost} = \text{true}$) then use si_purchasing_model with minimize_interruption_strategy.</td>
</tr>
</tbody>
</table>
First applicable: If the rules are in a specified order, firing the first applicable one allows control over the order in which rules fire.

Random: In this case, a single rule is randomly selected from the set of applicable rules.

Most specific: This strategy is based on the number of conditions of the rules. From the set of applicable rules, the rule with the most conditions is chosen. This is based on the assumption that if it has the most conditions, then it has the most relevance to the current context.

Most important rule: In this strategy, each rule is given a weight, which specifies how important it is compared to the other rules. The rule with the highest weight is chosen from the conflict set.

User interface – which connects the rule-based system to the outside world and through which inputs are received and outputs are sent.

5. SLA ESTABLISHMENT MODEL

In this section, we introduce our formal policy framework for automated SLA establishment. We first present the preference policy model, followed by the interaction policy model. Policy authors use the preference model to make preference statements over the customizable service attributes. Similarly, they use the interaction model to specify the supported interaction models and also to help context-driven interaction model and strategy selection.

5.1. Preference policy model

Our QoS preference model is based on the theory of utility functions, which offers the most expressive form of preference specification and has very strong theoretical properties [13, 15]. The utility function is an objective function that assigns a scalar value to all possible service configurations. Our preference model allows service providers to combine cardinal utility functions with utility-value assertions, attribute-value assertions and conditional attribute-value assertions to specify the preferences more comprehensively. It also allows service consumers sufficient expressivity to formulate service requests using attribute-value assertions and conditional attribute-value assertions.

5.1.1. Formal QoS model. Let us assume that a service has a set of QoS attributes \( X = \{x_1, x_2, x_3, \ldots, x_n\} \). Each attribute takes its value from a finite domain so that \( D = \{D_1, D_2, \ldots, D_n\} \) represents the corresponding set of QoS attribute domains where \( D_i \) is the finite set of values that variable \( x_i \) can take. The potential configuration space \( C \) is given by \( C = \{(c_1, \ldots, c_n) | c_i \in D_i \} \). A utility function \( U \) assigns a scalar utility value for every possible service configuration and is given by \( U : C \rightarrow [0,1] \). Service providers or consumers can restrict the region of interest by using constraint operators (mathematical or semantic), such that the set \( \Phi = \{<, >, \leq, \geq, \ldots, \} \) represents all the applicable constraint operators for the QoS attributes associated with a given service.
5.1.2. Formal assertion model. A policy assertion represents an individual QoS requirement or capability statement. We support three types of assertions based on the QoS model defined previously. They are as follows:

- **Utility-value assertion**: A utility-value assertion is a triple \( A_u \) \( \equiv (u, \geq, v) \), where \( u \) represents the utility and \( v \) is the minimum acceptable value assigned to it. The utility-value assertion is used to specify the minimum acceptable utility.

- **Attribute-value assertion**: An attribute value assertion is a triple \( A_a \) \( \equiv (x, \varphi, v) \), where \( x \in X, \varphi \in \Phi \), and \( v \in D(x) \). It specifies preferences over individual QoS attributes.

- **Conditional attribute-value assertion**: A conditional attribute-value assertion is defined as

  \[
  A_c \equiv D_x^{i \in 1, \ldots, k} A_{a_i} \rightarrow A_{a_{k+1}}
  \]

  where \( k \geq 1 \) and \( A_{a_1}, \ldots, A_{a_k} \) are attribute-value assertions about the attributes \( x_{c_1}, \ldots, x_{c_k} \) and \( A_{a_{k+1}} \) is an attribute-value assertion about an attribute \( x_{c_{k+1}} \notin \{x_{c_1}, \ldots, x_{c_k}\} \). The condition needs to be non-empty (i.e., \( k \geq 1 \)) for it to be a conditional attribute-value assertion and the right-hand side is an assertion about an attribute \( x_{c_{k+1}} \) that is not specified in the condition.

5.1.3. Formal policy model.

- **Policy alternative**: A policy alternative is a logical conjunction of zero or one utility-value assertion, zero or more attribute-value assertions, and zero or more conditional attribute-value assertions. It is given as follows depending upon whether the utility-value assertion is defined or not:

  \[
  P_{alt} = \begin{cases} 
  A_u \land \bigwedge_{i \in \{1, \ldots, p\}} A_{a_i} \land \bigwedge_{j \in \{1, \ldots, q\}} A_{c_j}, & \text{where } p, q \geq 0 \\
  \bigwedge_{i \in \{1, \ldots, p\}} A_{a_i} \land \bigwedge_{j \in \{1, \ldots, q\}} A_{c_j}, & \text{where } p + q > 0
  \end{cases}
  \]  

  (12)

- **Policy**: A policy is a collection of policy alternatives which can be combined using the following two policy operators:

  - **Any**, which is equivalent to the logical OR construct and enforces the rule that at least one of the listed policies has to be satisfied in order to satisfy the QoS preferences, and
  - **ExactlyOne**, which is equivalent to the logical XOR construct and enforces the rule that ‘exactly one’ of the listed policies can be true.

  In its normal form, a preference policy can be represented as an enumeration of its alternatives that in turn enumerate each of their assertions. Thus, the normal form policy expression is given as

  \[
  P_{QoS} = \begin{cases} 
  \bigvee_{i \in \{0, \ldots, q\}} P_{alt_i} & \text{: Any} \\
  \bigoplus_{i \in \{0, \ldots, q\}} P_{alt_i} & \text{: ExactlyOne}
  \end{cases}
  \]  

  (13)

  where \( P_{alt} \) is given by Equation (17) and \( q \in \mathbb{N} \) meaning that a policy can have 0 or more alternatives.

5.2. Interaction policy model

In our framework, we assume that any entity participating in the service provisioning process has to 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 \textit{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 $D = D_2 \times D_3 \times \cdots \times D_k$ represents the corresponding set of strategy parameter domains where $D_i$ is the finite set of values that parameter $p_i$ can take.

5.2.1. Formal assertion model. We support three types of assertions in our interaction policy model.

- \textit{Context assertion:} A context assertion is a triple $A_c \stackrel{\text{def}}{=} \langle x, \varphi, v \rangle$ 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 $x$.
- \textit{Interaction-protocol (IP) assertion:} The interaction-protocol assertion is defined as
  \[ A_{ip} \stackrel{\text{def}}{=} A_c \rightarrow \{\rho_1, \rho_2, \ldots, \rho_n\} \]  
  where $A_{ip}$ is the interaction protocol assertion, $A_c$ is the context assertion and $\rho_i \in P$.
- \textit{Strategy assertion:} The strategy assertion is defined as
  \[ A_s \stackrel{\text{def}}{=} A_c \rightarrow \bigvee_{i \in \{0..q\}} \tau(\rho, \sigma_1, \ldots, \sigma_k) \]  
  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 $v_i$.

5.2.2. Formal policy model. There are two types of policies – the interaction protocol (IP) policies that specify which protocols are supported and the strategy policies that specify which strategies are used with which interaction protocol under different contextual conditions.

- \textit{Policy alternative:} A policy alternative is a logical conjunction of zero or more assertions. The \textit{interaction protocol (IP) alternative} consists of a single IP assertion shown as follows:
  \[ P_{alt_{ip}} = A_{ip} \]  
  Similarly, the \textit{strategy alternative} consists of zero or more strategy assertions, and zero or more conditional-strategy assertions shown as follows:
  \[ P_{alt_s} = A_s \]

- \textit{Policy:} A policy is a collection of alternatives combined using different policy operators. In its normal form, the \textit{interaction protocol (IP) policy} can be represented as an enumeration of its alternatives shown as follows:
  \[ P_I = \{ \bigvee_{i \in \{0..q\}} P_{alt_{ip_i}} : \text{Any} \} \bigoplus_{i \in \{0..q\}} P_{alt_{ip_i}} : \text{ExactlyOne} \]  
  where $P_{alt_{ip_i}}$ is given by Equation (16) and $q \in \mathbb{N}$ meaning that a policy can have 0 or more alternatives. Similarly, the \textit{interaction policy} can be represented as an enumeration of its alternatives shown as follows:
  \[ P_S = \{ \bigvee_{i \in \{0..q\}} P_{alt_{si_i}} : \text{Any} \} \bigoplus_{i \in \{0..q\}} P_{alt_{si_i}} : \text{ExactlyOne} \]  
  where $P_{alt_{si_i}}$ is given by Equation (17) and $q \in \mathbb{N}$ meaning that a policy can have 0 or more alternatives.
6. AUTOSLAM REFERENCE ARCHITECTURE

In this section, we present the AutoSLAM reference architecture that provides the foundation to build policy-driven AutoSLAM systems such as the one we have implemented to purchase computing resources from Amazon EC2. The main benefit of our model is twofold. On the one hand, it allows the reuse of existing elements of automated SLA establishment so that they can be freely integrated into the system. On the other hand, the model is flexible enough to adapt to the SLA interaction model that is best suited for each SLA interaction scenario. We base our reference architecture on the eXtensible Access Control Markup Language (XACML) architecture [16, 17]. The XACML framework is an authorization and access control framework that defines a declarative access control policy language and a processing model to evaluate authorization requests according to the rules defined in XACML policies. An XACML request is usually made by a subject to perform a certain action on a given resource. The output of the XACML policy-processing model is a permit or deny decision based on which the authorization or access is approved or disapproved.

The AutoSLAM framework defines a declarative policy language WS-SLAM for specifying the supported SLA interaction models. It also defines a policy-processing model, which can evaluate incoming service requests (and the relevant context) against the SLA interaction policies to determine the most appropriate interaction model to instantiate. The main components of AutoSLAM are shown in Figure 4. The grayed box shows the AutoSLAM extension to the XACML architecture.

- **Policy Enforcement Point (PEP).** PEP is the entry point to the AutoSLAM policy-processing middleware. Initially, it receives the service request and forwards it to the Policy Decision Point (PDP). It then interprets the decision of the PDP and instantiates the appropriate SLA interaction model as shown in Figure 4.

![Figure 4. Automated Service Level Agreement Management reference architecture.](image-url)
- **Policy Decision Point (PDP).** PDP evaluates the incoming request and the relevant context against all the policies that are applicable in the current context. The outcome of the evaluation is the selected interaction model, which is sent back to the PEP.

- **Policy Access Point (PAP).** PAP makes available to the PDP all the policies and rules that are in the policy database.

- **Policy Information Point (PIP).** PIP retrieves all the information about the relevant context surrounding the current service request.

- **Policy Administration Point (PAdP).** Policy authors manage the policies in the policy database through the PAdP. They can add new policies, and remove or edit existing policies to update the knowledge base of the AutoSLAM decision model.

As shown in Figure 4, when an entity initiates the SLA interaction process or responds to a request, the PEP forwards the request it receives to the PDP, which in turn retrieves all the current policies from the PAP, evaluates them against the contextual information retrieved from the PIP and, based on the evaluation, selects the appropriate SLA interaction model with the corresponding decision-making strategy and interaction protocol. It then forwards this decision to the PEP, which instantiates the selected SLA interaction model. Depending upon whether it is a one-round interaction or multi-round interaction, the interaction module 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 an SLA, and the service entity is provided access to the service. If an acceptable outcome is not achieved, then the PEP returns a failure decision. The PDP shown is essentially a forward chaining inference engine as shown in Figure 5. It is a rule-based system that has a working-memory that consists of a set of assertions or facts, and a rule-base that contains all the appropriate knowledge encoded into if–then rules. The system examines all the rule conditions and determines a subset of rules whose conditions are satisfied based on the working memory. This subset is referred to as the conflict set. One of the rules from this conflict set is triggered. Which one is triggered is based on the conflict resolution strategy used. When the rule is fired, the SLA interaction model specified in the then clause is initialised and the preferences are passed as input to it.

![Figure 5. Policy decision point.](Concurrency Computat.: Pract. Exper. (2013) DOI: 10.1002/cpe)
7. AUTOSLAM MIDDLEWARE IMPLEMENTATION

In this section, we present our implementation of the AutoSLAM middleware. We first present the WS-SLAM policy specification language followed by the implementation of the policy-processing middleware, which is used to evaluate WS-SLAM rules.

7.1. WS-SLAM Specification language

The WS-SLAM is designed as an extension to the WS-Policy language that allows web services to advertise their capabilities, requirements and general characteristics in a flexible and extensible grammar using XML format. In WS-Policy, a policy is essentially a collection of policy alternatives. Each policy alternative is in turn a collection of policy assertions. The policy assertion represents a specific requirement, capability, constraint or behavior of the service. The policy assertions are not provided by the WS-Policy specification but instead can be provided for specific domains. WS-Policy operators (wsp:Policy, wsp:All, wsp:ExactlyOne) are used to group policy assertions into policy alternatives. WS-SLAM provides two sets of assertion models – one for specifying preference policies and one for specifying interaction rules. Table III presents the XML infoset representation of the preference assertions showing the three types of assertions supported by WS-SLAM, namely, attribute-value assertion, utility-value assertion and conditional attribute-value assertion. Table IV shows a concrete example of a WS-SLAM preference policy.

The assertion model for specifying interaction rules combines the context assertions and strategy assertions in condition-action rules. The core element of WS-SLAM is the Rule element. Each rule has an If part that captures the contextual conditions specified in the form of Context Assertions, which can be combined using the and, or and not logical operators. The Then part specifies the executable strategy that is to be invoked when the conditional part is satisfied. Each rule is identified by a unique name and can have a number of optional attributes to provide additional information. The XML infoset representation of a WS-SLAM Rule is Table V. A number of WS-SLAM rules can be combined into a single policy alternative using the All policy operator. A WS-SLAM rule makes use of three key assertions to declaratively specify the supported SLA interaction models, namely, context assertion, strategy assertion and interaction protocol assertion. These assertions have been formally defined in Section 5. There are two ways in which a strategy assertion can be made over the parametric strategy function:

<table>
<thead>
<tr>
<th>Attribute-Value Assertions</th>
</tr>
</thead>
<tbody>
<tr>
<td>`&lt;qosp:[QoS]Assertion unit=&quot;xs:string&quot; predicate=&quot;qosp:PredicateType&quot; value=&quot;xs:integer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Utility-Value Assertion</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;qosp:UVAssertion predicate=&quot;tns:GreaterEquals&quot; value=&quot;xs:double &quot;/&gt;</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conditional Attribute-Value Assertion</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;qosp:If&gt; (&lt;wsp:All&gt;(&lt;Assertion../&gt;)*)&lt;/wsp:All&gt; * </code><a href="">qosp:If</a> <code>&lt;qosp:Then&gt; (&lt;wsp:All&gt;(&lt;Assertion../&gt;)*)&lt;/wsp:All&gt; * </code><a href="">qosp:Then</a>`</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Normal form of WS-Policy expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;wsp:Policy&gt;</code> <code>&lt;wsp:ExactlyOne&gt; (&lt;wsp:All&gt;(&lt;Assertion... &gt; ...&lt;Assertion/&gt; ) * &lt;/wsp:All&gt; ) * &lt;/wsp:ExactlyOne&gt; &lt;/wsp:Policy&gt;</code></td>
</tr>
</tbody>
</table>
Example 7. Concrete WS-SLAM preference policy.

```
<wsp:Policy>
  <wsp:All>
    <qosp:UVAssertion predicate="GreaterEquals" value="0.7"/>
    <qosp:MemoryAssertion predicate="GreaterEquals" unit="GB" value="4"/>
    <qosp:If>
      <wsp:All>
        <qosp:IOPerformanceAssertion unit="tps" predicate="Equals" value="Very-high"/>
      </wsp:All>
    </qosp:If>
    <qosp:Then>
      <wsp:All>
        <qosp:MemoryAssertion unit="GB" predicate="GreaterEquals" value="10"/>
      </wsp:All>
    </qosp:Then>
  </wsp:All>
</wsp:Policy>
```

Table V. WS-SLAM interaction rules syntax (normal form).

### WS-SLAM Rule

```
<slam:Rule name="xs:string" type="xs:string">*
  <slam:RuleAttribute name="xs:string" value="xs:string"/>
  <slam:If>
  </slam:If>
  <slam:Then>
    <slam:Strategy.../>
  </slam:Then>
</slam:Rule>
```

### WS-SLAM policy expression

```
<wsp:Policy>
  <wsp:All>
    (<slam:Rule.../>)*
  </wsp:All>
</wsp:Policy>
```

### WS-SLAM Context Assertion

```
<slam:Context identifier="xs:string">
  
  (<slam:FieldConstraint/> | <slam:AndConstraintConnective/> | <slam:OrConstraintConnective/>)
</slam:Context>
```

### WS-SLAM Strategy Assertion

```
<slam:Strategy name="xs:string">
  (<slam:StrategyAttribute name="xs:string" value="xs:string" />)*
</slam:Strategy>
```

- **By reference**: In this case the WS-SLAM merely refers to an externally defined SLA interaction model that is to be invoked if the context holds true.
• **By reference with values:** In this case, the strategy assertion not only refers to the externally defined strategy but also specifies the specific values for the strategy parameters.

In WS-SLAM, the context is represented by the Context element, which can have an unrestricted number of fields (or context attributes). Constraints can be specified on the values these fields can take by using the FieldConstraint element. Multiple FieldConstraint elements can be combined using the logical and and or connectives. Atomic context assertions can be combined to compose complex context assertions using the `<slam:AndConditionalElement/>` and the `<slam:OrConditionalElement/>`. The XML infoset representation of the strategy assertions and the Interaction Protocol assertions are shown in Table V.

7.1.1. *A basic example of WS-SLAM.* Figure 6 shows a simple example of a policy document, which is compliant with the WS-SLAM policy language specification. In order to improve readability, we have removed the namespace declarations of both WS-Policy and WS-SLAM. The example policy

```xml
<xml version="1.0" encoding="UTF-8">
<tns:Policy>
<tns:All>
<slam:Rule name="Rule 1">
<slam:If>
<slam:Context identifier="context" objectType="Context">
<slam:AndConstraintConnective>
<slam:FieldConstraint field-name="uninterruptedAccess"><slam:LiteralRestriction value="true" evaluator="eq"/></slam:FieldConstraint>
<slam:FieldConstraint field-name="minCost"><slam:LiteralRestriction value="true" evaluator="eq"/></slam:FieldConstraint>
<slam:FieldConstraint field-name="accessInFuture"><slam:LiteralRestriction value="true" evaluator="eq"/></slam:FieldConstraint>
</slam:AndConstraintConnective>
</slam:Context>
<slam:If>
<slam:Then>
<slam:Strategy name="BlockPurchasingStrategy"/>
</slam:Then>
</slam:If>
</slam:Rule>

<slam:Rule name="Rule 2">
<slam:If>
<slam:Context identifier="context" objectType="Context">
<slam:OrConstraintConnective>
<slam:AndConstraintConnective>
<slam:FieldConstraint field-name="minCost"><slam:LiteralRestriction value="true" evaluator="eq"/></slam:FieldConstraint>
<slam:FieldConstraint field-name="minCompletionTime"><slam:LiteralRestriction value="true" evaluator="eq"/></slam:FieldConstraint>
</slam:AndConstraintConnective>
<slam:OrConstraintConnective>
<slam:Context>
<slam:If>
<slam:Then>
<slam:Strategy name="PriceMomentumStrategy"/>
<slam:Then>
</slam:If>
</slam:Rule>
</slam:All>
```

Figure 6. Example WS-SLAM policy.
shows two rules, where each rule specifies the SLA interaction model and decision-making strategy to use in a given context. This example policy defines rules to make decisions for purchasing instances on Amazon EC2.

7.2. AutoSLAM policy-processing middleware

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 has been integrated with the Smart CloudPurchaser for purchasing instances on Amazon EC2. It comprises three key components:

- WS-SLAM2DrlParser, which parses WS-SLAM policies into Drools\(^1\) rules.
- An embeddable Drools Rule Engine, which evaluates the incoming requests and the relevant context against the predefined WS-SLAM policies.
- A library of executable SLA interaction models that are used to purchase instances from Amazon EC2, including the decision-making strategies given in the motivating scenario in Section 3.

7.2.1. WS-SLAM2Drl Parser. In the current version of the AutoSLAM middleware, we have implemented a WS-SLAM2Drl Parser, which parses WS-SLAM policies and rules into Drools rules as shown in Figure 7. The parser makes use of mapping rules to map from WS-SLAM constructs to the Drools constructs. As illustrated in the figure, a parser may be implemented to parse the WS-SLAM rules into Jess format in which case the JESS Rule Engine could be used to evaluate the request and relevant context against the rules.

Table VI shows the correspondence between the main constructs in WS-SLAM and Drools. The WS-SLAM policy specification is a light-weight language, which is intended to be used by non-technical policy authors and hence does not support low-level executable code expressions. On the other hand, in Drools, the action part refers to executable actions and supports the insertion of executable Java code. Hence, there has to be a mapping file (wsslaml2drl.mapping), which can map the abstract rules and constructs in WS-SLAM to more concrete executable classes and objects in Drools. A technical expert has to define the mapping between the abstract Context and Strategy names in WS-SLAM to the corresponding executable Java implementations, which are inserted into the Drools file. Figure 8 shows a snippet of the mapping from WS-SLAM to Drools. Figure 9 shows the output of the WS-SLAM2Drl Parser when the input is the file shown in Figure 6.

\(^1\)http://www.jboss.org/drools

Table VI. Correspondence between language constructs.

<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
<th>WS-SLAM</th>
<th>Drools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule</td>
<td>if CONDITION then ACTION end</td>
<td>&lt;IF&gt;</td>
<td>rule name</td>
</tr>
<tr>
<td></td>
<td>THEN&gt; CONTEXT &lt;/THEN&gt;</td>
<td>WHEN</td>
<td></td>
</tr>
<tr>
<td>Formula</td>
<td>Conditional part of the rule</td>
<td>atomic</td>
<td>and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OR</td>
<td>not</td>
</tr>
<tr>
<td>Atomic</td>
<td>Atomic member of formula</td>
<td>&lt;CONTEXT&gt;</td>
<td>pattern</td>
</tr>
<tr>
<td></td>
<td>&lt;/CONTEXT&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Action</td>
<td>Action refers to externally defined functions or methods</td>
<td>&lt;STRATEGY&gt;</td>
<td>Objects imported into rule package and invoked from within rules</td>
</tr>
</tbody>
</table>

Figure 8. WS-SLAM to Drl mapping.

8. USE CASE VALIDATION

We have used the Amazon EC2 scenario described in Section 3 to validate our AutoSLAM policy-driven middleware for automated SLA establishment. In this scenario, end-consumers submit their requests to the *Smart CloudPurchaser* whenever they have a job to process on EC2. To simplify the scenario, the end-consumers know the exact EC2 instance type that they want and the number of instances required to complete the task. They have preferences and constraints over the task completion time and the total cost payable, which they specify when they submit their request. The Smart CloudPurchaser is able to make purchasing decisions on behalf of the end-users based on the domain knowledge captured in the form of strategy policies. It uses the AutoSLAM policy engine to evaluate each incoming request against its policy base and determines the most appropriate purchasing
model as well as the best bidding strategy. Different mechanisms can be used to resolve deadlocks that result when more than one rule (and hence SLA interaction model) is applicable. The simplest solution is to choose the first applicable rule or a randomly selected rule. Alternatively, rules can be assigned individual scores (for e.g., based on the number of context attributes satisfied) and then the rule with the highest score is executed. Once an applicable rule is selected, the Smart CloudPurchaser initiates the interaction with Amazon EC2. If purchasing on-demand instances, it requests for and uses the on-demand instance. If going for spot-instances, it starts bidding for resources using the selected bidding strategy. If the bid is successful, it uses the spot-instance to process the job.

8.1. Scenario 1: immediate access to computing resources

For the request shown in Figure 10(a), where immediate access is one of the constraints, the only applicable purchasing model is the on-demand model. Based on the user request, the policy engine purchases the instances using the on-demand purchasing model, and the job is executed on the purchased resources. Once the job is completed, the instances are disposed.

8.2. Scenario 2: short duration and minimize cost

In this scenario, the user wants the resources for a short duration and wants to minimize the total cost payable for the resources. When the policy engine receives this request, it determines that the cost optimization strategy is the best strategy where the strategy is to bid at a high price (very close to the on-demand price). Hence, it uses an on-demand price of $0.035 for the t1.micro instance.
8.3. Scenario 3: uninterrupted access and minimize cost

For the input request shown in Figure 11(a), where the objective is to minimize cost while having uninterrupted access to the resource, the policy engine chooses the spot instance purchasing model with the minimize interruption strategy. This strategy computes a bid that is much higher than the on-demand price, in this particular instance, twice the on-demand price.

8.4. Scenario 4: minimize cost and minimize completion time

For the input request shown in Figure 11(b), where the objective is to minimize cost of renting the computing resources and minimize the job completion time, the policy engine chooses the spot instance purchasing model and the price momentum strategy. Having selected this strategy, it computes the maximum bidding price as $0.678 based on the past 12-h 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 12.
9. LIMITATIONS OF CURRENT WORK

In this section, we discuss some of the limitations of our current work. We first discuss our work on preference specification and identify some limitations. We then look at policy-based specification of interaction models and strategies and identify the limitations of our current work. We then discuss how these limitations can and will be addressed in future work.

9.1. Policy-based preference specification

While combining attribute-value assertions, conditional attribute-value assertions and utility-value assertions gives policy authors greater flexibility in specifying their preferences, it can also lead to potential conflicts. We illustrate this with the use of a simple example. Let us assume that the domain expert has defined a point-based utility function for a set of six specific configurations for service that has three attributes – availability $A$, response time $RT$ and throughput $T$, as shown in Table VII. Let us also assume that the policy author has defined a utility-value assertion as shown in Equation (20) based on which the service configurations $c_1$, $c_3$ and $c_4$ are acceptable.

$$ AU_{min} = (U > 0.7) $$

Let us now assume that in response to changing business conditions, the policy author decides to define an additional conditional attribute-value assertion shown as follows:

$$(A \geq 95) \rightarrow (T \leq 10)$$

According to the new conditional attribute-value assertion, the service configuration $c_5$ is acceptable. However, the utility-value assertion only allows service configurations with a utility greater than 0.7 to be accepted. Thus, we can see through this simple example that potential conflicts can arise when combining utility-value assertions, attribute-value assertions and conditional attribute-value assertions, and support is required for detecting and/or resolving such conflicts.

Table VII. Example showing conflicting preference statements.

<table>
<thead>
<tr>
<th>Service Configuration</th>
<th>Utility Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>0.75</td>
</tr>
<tr>
<td>$c_2$</td>
<td>0.65</td>
</tr>
<tr>
<td>$c_3$</td>
<td>0.8</td>
</tr>
<tr>
<td>$c_4$</td>
<td>0.7</td>
</tr>
<tr>
<td>$c_5$</td>
<td>0.6</td>
</tr>
<tr>
<td>$c_6$</td>
<td>0.55</td>
</tr>
</tbody>
</table>
Service providers would benefit from a built-in reasoning mechanism that can detect any conflicting preferences and resolve them. One simple way to avoid such conflicts is to use assertion combining algorithms similar to the rule-combining algorithms such as permit-overrides and deny-overrides in XACML [17]. Such an approach allows policy authors to specify potentially conflicting assertions but then uses rule-combining algorithms to resolve conflicts. While this solution is sufficient for avoiding conflicts, it does not provide any support for conflict detection. Thus, one avenue for future research will be to look into the issue of conflict detection and resolution when dealing with QoS preference policies. Another more interesting area of future research could be looking into how the service provider can negotiate with the service consumers when no solutions exist, which satisfy both the consumer request and the provider preferences. In case there is no acceptable solution, the autonomous reasoner has to be able to reason about its own tradeoffs as well as that of the other party and try to make or propose tradeoffs. This might mean relaxing own preferences or requesting the other party to relax its requirements.

9.2. Policy-based interaction model specification

In our current model, we use interaction policies to specify which interaction model (with corresponding strategy) to use under specific contexts. We have shown simple examples of how different bidding strategies can be used to bid for spot instances from Amazon EC2 under different contexts. However, in the current approach, there is a tight coupling between the interaction protocol and the decision-making strategy, and the actual SLA interactions are external to the policy engine as shown in Figure 4. Once the interaction model has been selected, the policy engine has no control over the SLA interaction process – it is restricted to selection of interaction model and decision strategy. While this allows the easy plug-n-play mechanism by using black box implementations of SLA interaction models, a potential problem with this approach is that there is no guarantee that by selecting a specific interaction model and decision-making strategy, an acceptable agreement will be achieved. In fact, it is quite possible that at the end of the interaction, the best possible agreement is not achieved, or worse still, no agreement is achieved.

As a simple example, let us consider a scenario where a client wants to procure spot instances from Amazon EC2 and wants to minimize the computing cost, and the job completion time is much larger than processing time albeit with a firm deadline. Based on the request, the most appropriate purchasing model is the spot-instance purchasing model. If the bidding strategy is to bid as low as possible, it is quite possible that the client will not be successful in procuring spot instances by the deadline. If the policy engine could monitor the progress of the SLA interaction, it could self-adapt to maximize the chances of obtaining a successful agreement. For the simple example described here, there can be an adaptive action to switch the interaction model from spot-instance to on-demand when the remaining time is close to the job processing time. Alternatively, a new bid can be re-submitted using the same spot instance purchasing model but at a higher price before the deadline is reached.

In order to introduce self-adaptation into the system, all the mechanisms necessary to enact the adaptation strategies have to be built into the policy engine. To enable this, the policy engine should not only be able to select the appropriate SLA interaction model based on the current context but also be able to follow the progress of the SLA interaction. During the process of SLA interaction, the SLA goes through different states. If we consider the Spot Instance purchasing model, the SLA goes through the following states – bid-generation, bid-submission, pending-evaluation, pending-fulfillment and bid-fulfilled (Figure 13). These states are essentially captured in the interaction protocol. The policy engine can use an interceptor-style monitoring to monitor the status of the SLA interaction in each state and use appropriate decision-tactics (9a and 9b in Figure 13) in that state. For instance, in the pending-evaluation state, if the status is price-too-low, it can choose a different bidding strategy to compute a new bid. If the status is capacity-not-available, the policy engine might choose another instance type that fulfills the user preferences and submit a new bid. Using such an approach, the policy engine intervenes only when a deviation from the expected outcome is observed and adaptive mediation becomes necessary. We can write new interaction policies to enable the adaptive action as shown in Table VIII.
AUTOSLAM – A POLICY-BASED FRAMEWORK FOR AUTOMATED SLA ESTABLISHMENT

Figure 13. Spot request status update.

Table VIII. Example adaptive interaction rule.

\[
\begin{align*}
\text{if}(\text{bid-price-too-low} \land \text{remaining_time} > \text{processing_time}) & \Rightarrow \text{use price_history_momentum_strategy}; \\
\text{if}(\text{bid-price-too-low} \land \text{remaining_time} \approx \text{processing_time}) & \Rightarrow \text{use od_purchasing_model};
\end{align*}
\]

10. RELATED WORK

Cloud computing offers a realization of SOA in which IT resources can be provisioned as QoS guaranteed services to consumers on demand using flexible and scalable provisioning models, and dynamic pricing models [18]. Resource orchestration – the process of provisioning computing resources comprises the following phases – selection, assembly and deployment of computing resources, and monitoring of deployed resources [19]. During the selection phase, the cloud service provider and the cloud service consumer have to reach an agreement over the service usage terms and conditions that are formally captured in the SLA. Once an agreement is reached and the resource is assembled and deployed, it has to be monitored to ensure that the contracted service level objectives are fulfilled. Given the diversity and heterogeneity of cloud resource types, and the uncertainty of the underlying cloud environment, there is a need for automated SLA establishment and monitoring [20], and adaptive management to ensure SLA fulfillment. In this section, we present related work in the area of automated SLA establishment, with a particular focus on policy-based approaches for SLA establishment.

The four important aspects related to automated SLA establishment in open, diverse and dynamic SOA environments are the preferences over the service usage terms and conditions, the interaction protocols that govern the interactions between the participating entities, the decision-making strategies that are used to try and obtain agreements [1], and the dynamic, transient, distributed and
multi-dimensional interaction context. Most research in the area of automated SLA establishment and in particular policy-based SLA establishment focusses only on the first three aspects – that too in isolation. Research tends to focus on the design and development of interaction protocols, decision-making algorithms or preference models and does not look at all three aspects collectively. More importantly, no work considers the interaction context and its impact on the decision-making behavior and the SLA establishment process. Thus, there is very limited research on how to combine the different results into a unified solution that supports flexible and expressive preference models, a wide range of interaction protocols and a wide range of decision-making strategies to support automated SLA establishment in different contexts and scenarios. In this Section, we look at related work in the area of policy-based preference specification, policy-based specification of decision-making strategies and SLA interaction models, and also policy-based frameworks for automated SLA establishment.

10.1. Preference specification

A significant amount of research work has been previously carried out on policy-based QoS preference specification [21–25]. Most existing work on policy-based preference models uses ad-hoc approaches with no strong theoretical foundations, while others have limitations and are not easy to use as shown in [6]. In the simplest approach to policy-based preference specification, policy authors specify their preferences over non-functional attributes by using attribute-value assertions. These assertions are defined for single attributes and collectively can be used to classify a given alternative as either being acceptable or not-acceptable [21, 22]. Different types of mathematical operators such as =, <, >, ≤, ≥, and/or semantic relationship operators such as atMost, atLeast and around can be used to make more sophisticated and flexible preference statements [26]. Additionally, attribute-importance statements can be used to assign weights to the different attributes. This approach allows the ordinal ranking of the different alternatives and helps in choosing the most acceptable service offering or service request. An improvement on this is to use utility functions that allow the measurement of the strength of preference of a good or service with precision through the use of some objective criteria, thus enabling cardinal ranking. In this approach presented in [23,24], policy authors specify preferences through conditional and unconditional attribute-value assertions. They also provide attribute-importance statements. From these assertions, the corresponding value functions (or utility functions) are estimated. The authors have defined a fixed set of value functions – point-based functions, piecewise linear functions and pattern based functions. The utility function for every QoS attribute is estimated from the attribute-value assertions, and the total utility is computed using weighted sums. While this approach permits the evaluation and ranking of offers, the major limitation of estimating utility functions from preference statements is that it restricts the utility function definition to a limited set of predefined functions that may or may not capture the actual preferences of the policy authors. In [25], policy authors directly insert built-in numeric utility functions in the preference rules. The authors assume that the attributes are preferentially independent and have a linear utility. The overall utility for a service configuration is then computed as the weighted sum of the individual utilities.

We base our QoS preference model on utility theory and utility functions, the most expressive form of preference specification with very strong theoretical properties [13, 15]. The key contributions of our proposed preference model are twofold. We introduce the utility-value assertion, which, when combined with a comprehensive cardinal utility function, is sufficient by itself for specifying preferences over the entire configuration space. We also allow policy authors to use cardinal and ordinal preferences together so that they can show maximum flexibility in the preference specification. If the cardinal utility function is not comprehensive and covers only a subset of the configuration space, additional attribute-value and conditional attribute-value assertions can be used to specify ordinal preferences over the remaining configuration space thereby maximizing the chances of forming agreements. More importantly, these assertions can be easily modified on-the-fly, and the policy engine can easily determine the modified set of acceptable service configurations. We have intentionally kept the definition of the numeric functions separate from the preference policies in contrast to the approach in [25] because of two reasons. Firstly, the inclusion
of numeric functions in preference policies makes the policy language complex. Numeric functions can vary from simple mathematical expressions as in Equation (5) to very complex ones (as in Equations (7) and (8)) and providing support for all possible definitions makes the policy language very heavyweight. The idea behind autonomic computing and policy-based computing is to allow policy authors to specify their high-level objectives in a natural, intuitive manner using a simple, expressive and flexible policy language, which is lightweight. Hence, in our approach, we allow complex utility functions to be defined separately and referred to within the policies. Secondly, in order to enable automated SLA establishment, it is important that the policy engine can reason about and analyze QoS preference policies. Because attribute-value assertions, conditional attribute-value assertions and utility-value assertion are based on well-defined predicates and logical connectives, a reasoning engine should have no problem reasoning about them. However, reasoning about complex numeric utility functions requires additional capabilities to deal with mathematical expressions and their semantics.

10.2. Decision-making strategies

There are several research proposals on policy-based specification of decision-making strategies for automated negotiation. The authors in [27–30] and [31] propose the use of declarative rules to capture the decision-making strategies. In [27], the authors propose a policy-based approach to facilitate automated decision-making by combining rules and utility functions. They explicitly specify the negotiation strategy in the form of rules defined within the negotiation policy. The utility function is kept separate from the negotiation policies. The authors claim that their approach is agnostic to specific negotiation protocols, but this is not true in all cases. In [30] and [31], the authors propose a policy-based middleware for the automated negotiation of web service SLAs, but they do not provide any formal models or concrete examples to illustrate how this can be carried out. 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 [1, 32], heuristic approaches [33, 34] and evolutionary approaches [35]. There has to be a tradeoff between the expressive power of the policy language and the ease of usage. In [36], 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. Some work has also been carried out on using logic formalism to model automated negotiation [37, 38] and [39]. In [37], the authors use a formal and executable approach to capture the behavior of the parties involved in the negotiation. The negotiation strategies are expressed in a declarative rules language – defeasible logic. In [38], the authors use propositional logic to model preferences over attributes as well as the relations among the attributes. In this work, the authors restrict themselves to Rubenstein’s alternating-offers protocol and use a mediator to solve the multi-objective optimization problem. In [39], the authors extend their work by using Description Logic, which offers greater expressivity than propositional logic. However, in both [38] and [39], the authors only focus on preference specification and restrict themselves to the alternating-offers protocol. While context-dependent utility and the impact of context on decision-making behavior have been investigated in the fields of human judgment and decision-making [40], not much work has been carried out on studying the impact of context on SLA establishment. To the best of our knowledge, our proposed solution is the first to support multiple interaction models and enable context-driven selection of interaction models for SLA establishment. 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. We allow policy authors to refer to externally defined decision-making strategies within the interaction rules. This makes the policy language light-weight and enables reuse of existing results in the area of automated SLA establishment, particularly decision-making models. Another key improvement that our approach offers is that it accounts for the impact of the diverse, transient, volatile and multi-dimensional interaction context and its impact on the decision-making behavior that guides the SLA interactions.
10.3. Unified solutions

In [41], the authors propose WS-Negotiation as a declarative language that contains three parts – Negotiation Message, which specifies the format for the messages exchanged between the negotiation parties, Negotiation Protocol, which describes the mechanisms and rules that the participants should follow, and Negotiation Decision Making, which is used to specify the decision-making strategies. The main limitation of this work is that it only supports bilateral and multi-issue bargaining, that is, it supports only one interaction protocol. Another limitation of this work is that it does not consider the SLA interaction context and its impact on the decision-making behavior, and it does not have a reference architecture that can be implemented and evaluated. WS-Agreement [42] is another framework which supports automated SLA establishment. While it provides a generic and extensible model for specifying the service usage terms and conditions, the WS-Agreement protocol is based on a single round ‘offer’, ‘accept’ message exchange. WS-AgreementNegotiation [43] is an extension of WS-Agreement that is based on the alternating-offers protocol and supports multi-round interactions for SLA establishment. The main limitation of WS-AgreementNegotiation is that it is restricted to a single interaction protocol and does not have a publicly available reference implementation. Mandi [44] is a light-weight market exchange framework that gives its users the flexibility of choosing the appropriate negotiation protocol while supporting simultaneous coexistence of multiple trading negotiations. The key difference between Mandi and our work is that Mandi is currently restricted to three negotiation protocols – commodity market, one-sided auction and two-sided auction. Also, there is no support for preference specification and context-dependent selection of interaction protocol. In [29], the authors also propose a policy-based approach to automated e-business negotiations. They define two types of policies – negotiation policies, which map the negotiation context to high-level negotiation goals. Policy makers define the negotiation goals and policies. At the same time, the negotiation experts define the strategy policies, which map the negotiation goals to a set of decision-action rules. The main contribution of this work is that they provide a formal model to map high-level negotiation goals to low-level tactics. The main limitation is that the negotiation is restricted to only one negotiation protocol, which they model and describe using state-transition diagrams. Also, they do not have a preference model for capturing the preferences and constraints over the customizable service attributes. They talk about the preferences in terms of high-level goals such as profitability, desirability to form contract and time for negotiation.

11. CONCLUSION

In this paper, we have presented AutoSLAM – our novel policy-based framework for the automated establishment of SLAs in open, diverse and dynamic environments such as the cloud. To the best of our knowledge, it is the first such framework that covers all key aspects of automated SLA establishment. As part of AutoSLAM, we have presented a flexible QoS preference model that improves the state-of-the-art in policy-based preference specification by combining cardinal and ordinal preferences. Combining utility-value assertions with attribute-value assertions and conditional-attribute value assertions provides policy authors greater flexibility and expressivity in specifying their preferences. The task of evaluating service requests and offers can then be delegated to autonomous agents that can reason about the preferences and takes decisions on behalf of the service consumers and providers. Similarly, our policy-based interaction model gives entities the flexibility to choose the SLA interaction model that is most appropriate in a given context while simultaneously participating in multiple concurrent SLA interactions using different SLA interactions. Policy authors can make conditional assertions over the supported interaction models and decision-making strategies by using context assertions with interaction model assertions.

We have extended the WS-Policy framework to provide a light-weight and simple yet expressive and flexible policy specification language. We have used the Amazon EC2 example to illustrate why service entities may require flexibility in terms of preference specification as well as choice of SLA interaction model in different context and scenarios. We have implemented a proof-of-concept prototype of our AutoSLAM middleware, which makes use of WS-SLAM, our extension of the WS-Policy framework, to specify preference and interaction policies. We have implemented
the decision-making module using the standard Drools Rule Engine. We have validated our framework by implementing the Smart CloudPurchaser that evaluates incoming requests for computing resources on Amazon EC2 and the context surrounding them, against the preference and interaction policies in order to select the best instance type and the best purchasing option.

Currently, the AutoSLAM framework uses preference policies and interaction policies to manage the process of automated SLA establishment. The preference policies specify what the preferred outcome should be and the interaction policies specify how to achieve this outcome under different context conditions. However, there is no guarantee that by using a particular interaction model and decision-making strategy, an acceptable outcome will be achieved. In fact, it is quite possible that during the actual interaction, the best possible outcome is not achieved or the preferences are violated. As future work, we would like to investigate the use of policies to enable dynamic adaptation of preferences and strategies during the SLA interaction process to ensure that the chances of achieving an acceptable SLA outcome is maximized.

ACKNOWLEDGEMENTS

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

2. Kephart JO, Walsh WE. An artificial intelligence perspective on autonomic computing policies. 5th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2004), NY, USA, 2004; 3–12.