AMPol-Q: Adaptive Middleware Policy to Support QoS*

Raja Afandi, Jianqing Zhang, and Carl A. Gunter

University of Illinois Urbana-Champaign, Urbana, IL 61801, USA,
afandi@illinoisalumni.org,{jzhang24,cgunter}@cs.uiuc.edu

Abstract. There are many problems hindering the design and development of Service-Oriented Architectures (SOAs), which can dynamically discover and compose multiple services so that the quality of the composite service is measured by its End-to-End (E2E) quality, rather than that of individual services in isolation. The diversity and complexity of QoS constraints further limit the wide-scale adoption of QoS-aware SOA. We propose extensions to current OWL-S service description mechanisms to describe QoS information of all the candidate services. Our middleware based solution, *AMPol-Q*, enables clients to discover, select, compose, and monitor services that fulfil E2E QoS constraints. Our implementation and case studies demonstrate how AMPol-Q can accomplish these goals for web services that implement messaging.

Key words: AMPol-Q, WSEmail, Adaptive Middleware, Policy, Service Oriented Architecture, QoS, Dynamic Service Discovery, Security, Ontologies.

1 Introduction

Although there has been considerable attention devoted to the composition of functional properties in Service Oriented Architectures (SOAs), more work is needed to deal with *non-functional* Quality of Service (QoS) properties such as reliability, performance and security required by clients. Issues that need attention include providing QoS features at the level of the individual service and client, discovering and composing candidate services on the basis of QoS features, monitoring and ensuring that a promised QoS is actually provided during execution, and adopting and using QoS-aware SOAs on a large scale. At least three problems must be overcome. First, current approaches [1–3] for dynamic service discovery and composition do not provide a *global* view of QoS features about all candidate services prior to invocation. They are limited to discovering first-level immediate services, and each individual service is responsible for discovering other services independently. They also lack the comprehensive specification of QoS features. Second, QoS is not compositional in the sense that functional features expressed through interfaces or functional components are composed to achieve a composite functionality (*e.g.* workflow systems). QoS-based composition requires complex calculations of *aggregate* QoS values of multiple entities involved in a transaction. Participants are interested in the final aggregate value of the runtime global QoS (*e.g.* end-to-end delay, overall cost, global integrity and confidentiality). However, current

* 4th International Conference on Service Oriented Computing (ICSOC'06), Chicago, USA, December, 2006

QoS-aware systems are not able to support global QoS behavior. Third, and finally, QoS-aware service composition and negotiation may not be effective without monitoring. Most QoS-aware systems do not guarantee that an agreed quality of service is actually provided during execution. Existing QoS-monitoring approaches [4, **?**,5] rely on trusted third parties to centrally monitor QoS delivered by service providers. This is technically difficult and limited to QoS features like availability and performance, while security and privacy cannot be covered. Moreover, monitoring involves complex and domain-specific logic for measuring and verifying QoS, which make the task harder.

To address these problems, we have developed an *Adaptive Middleware Policy to Support QoS (AMPol-Q)*. Our approach is based on an integrated collection of reference frameworks for description, discovery, and monitoring that are specially suited to handle QoS features in a SOA. The *description framework* includes semantic model for capturing QoS requirements, constraints and capabilities. We extend current service description and advertisement mechanisms (OWL-S and UDDI) to gather QoS information about all the candidate services. For efficient implementation, we represent these QoS requirements as policy rules. In the *discovery framework*, AMPol-Q serves as a broker (at the client end) for dynamically discovering and composing matched services on the basis of functional as well as non-functional features. The candidate services are first discovered on the basis of their functional capabilities and the final set of services is selected according to their QoS features. This approach is capable of evaluating global quality requirements and applying different types of optimizations (such as context-aware optimizations) to select the best-matched services. It negotiates the QoS properties between service providers and consumers to create an agreement. The *monitoring framework* provides an agile and adaptive mechanism to automatically plug in customized modules for measuring, verifying and ensuring QoS features without modifying the baseline system. We use a technique [6] in which the QoS contracts are monitored at each individual participant. Furthermore we improve on distributed monitoring approaches by providing support for two-way specialization.

We validated AMPol-Q with a prototype implementation and a case study on WSEmail [7] that shows how AMPol-Q can enhance the function of email messaging systems by enabling automatic deployment and use of complex QoS features like cycle exhaustion puzzles, reverse Turing tests and identity based encryption without the need for global deployment or changes to the baseline system. This case study shows how SOA can support QoS-aware service discovery, selection and monitoring.

2 Description Framework

The AMPol-Q *description* framework is a collection of interoperable semantic models used to represent QoS features of all entities in SOA. These QoS ontology and policy models, which are extensions to current service description frameworks [8–11], are intended for global discovery and selection of candidate services on the basis of QoS features. They are based on layered semantic models (*QoS Ontology, Policy* and *Entity Profile*). The steps of describing QoS features are a series of bottom up instantiations of these models. We use semantic models because they can be easily extended with new concepts. Furthermore, existing reasoning tools can be applied on the semantic models to detect ambiguity or inconsistency. Our discussion focus on novel features related to capturing global QoS behavior and to achieve support for E2E Global QoS.

Semantic QoS Ontology Model Our semantic QoS ontology model provides a standard generic ontology for arbitrary QoS features. It defines the nature of associations between QoS concepts, QoS metrics, and the way they are measured and monitored. Figure 1 shows the detailed ontology model. To facilitate reusability and extensibility,

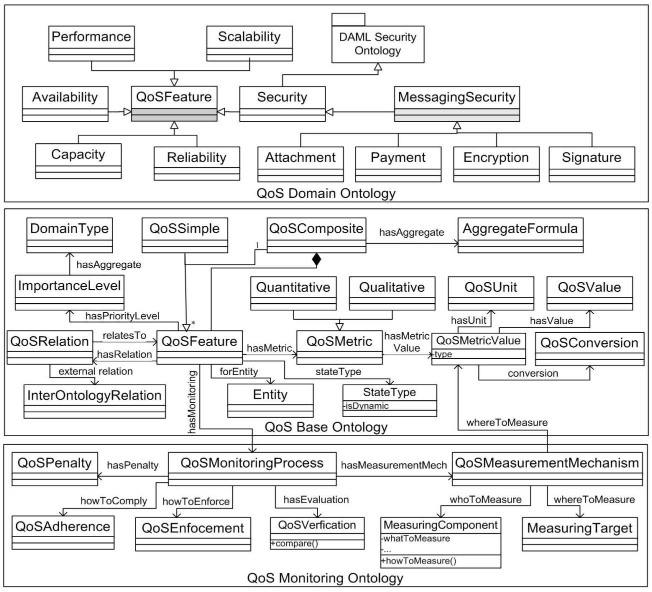

Fig. 1. AMPol-Q QoS Model

the ontology has a modular design and is categorized into three models: *base, monitoring* and *domain*.

In the QoS base ontology model, each QoS feature is an instance of a class *QoSFeature*, and it is associated to a *Quantitative* or *Qualitative* property. *Quantitative* relates the attributes which can be measured by numbers with a particular unit. For example, the percentage availability of a service. *Qualitative* relates to attributes which cannot necessarily be measured by exact amount. For example, the obligation features such as requirement of data encryption or providing an X.509 certificate.

In the context of global QoS, we define *QoSSimple* and *QoSCopmosite* as subclasses of *QoSFeature*. *QoSComposite* represents complex global QoS features which are drawn from calculation of aggregate QoS values. For example, the formula for composite service availability is the *product* of availability measure of each participant service. The computational logic is captured by *AggregateFormula*. Different entities may specify QoS values (*QoSMetricValue*) with different units (*e.g.* 90% versus 0.90 or 50F versus 10C). The unit conversion is done by *QoSConversion*, which captures the conversion logic. In global QoS, there are dependencies and correlations between

QoS features. For example, some QoS values are inversely proportional each other, *e.g.* the service response time and the throughput; some are directly proportional, *e.g.* accessibility and availability. *QoSRelation* class captures these relationship types. Some composite QoS is measured from aggregate values of different types of related QoS feature. For example, response time at a client is a sum of network latency and service processing time. This behavior is captured by the *has-a* object property.

Current QoS modeling approaches [3, 8, 9] do not have ontologies to support measurement, verification or monitoring of QoS features. We propose a QoS monitoring ontology model, which binds QoS features with their corresponding monitoring process (*QoSMonitoringProcess*). The QoS monitoring process involves measurement of QoS features, verification by evaluating measured QoS values against required policy values, adherence logic to provide required QoS features, and enforcement logic to *e.g.* permit or deny the requests. Domain specific ontologies can be defined by extending QoS base ontology model. We sketch a domain ontology for our case study later.

Policy Model AMPol-Q represents QoS features in the form of policy rules. The policy model specifies rules that use QoS ontologies to define QoS features of a particular entity. These policy rules are then used to describe, discover and compose services and to monitor QoS. See [12] for details of AMPol-Q *policy model*.

Policy rules are defined as an *implication property* in the form of *antecedent implies consequent*, *e.g. [(a:QoSFeature o:Operator a:QoSValue) connective (b:QoSFeature o:Operator b:QoSValue)] implies [ACTION]*. The *Rule* property uses QoS ontology to represent antecedent conditions; action can be *permit* or *deny*. Both QoS constraints and capabilities are described as rules.

For dynamic service composition based on global QoS, the advertised QoSValue can be calculated only if the QoS values of all dependent services are determined. For example, a loan processing service LP provides functionality for acquiring loans from banks. In order to process loan requests it talks to credit reporting agency CR to verify a client credit history and coordinate with bank B for loan processing. Processing time for acquiring a loan (the functionality of the LP service) can be calculated by adding its processing time (*P:QoSFeature*) and processing times of all the dependent services (CR and B). If CR and B are dynamically discovered then LP's processing time cannot be calculated beforehand. Current description languages are not able to handle these kinds of complex QoS features. To solve this problem we introduce a concept of *rule templates*. Rule templates can specify antecedents containing unresolved *template variables*. Antecedents can be evaluated only if all the template variables are determined (during runtime). In the above scenario, say, LA processing time is *50ms*, the capability rule of LA can be represented as *[P:QoSFeature = (50ms:QoSValue + p1:T1 + p2:T2)]*, where *p1* and *p2* are template variables, *T1* and *T2* are templates which are defined as *T1 = ((B.P):QoSValue)* and *T2=((CR.P):QoSValue)*. This problem can also be solved by modeling each QoS feature as a *QoSComposite* object with a *has-a* object property to represent dependent QoS feature values and an *AggregrateFormula* object to represent aggregation logics. But our policy engine implementation has shown that rule templates are simpler to construct and more efficient to evaluate.

AMPol uses meta-specification (the policies of a policy) to specify how polices are evaluated and enforced. For example, in a service oriented environment for monitoring global QoS, the policy model should be able to specify which entities the policy is applied to and which entities enforce them. In a distributed system, the creator of the rule or the policy might not be the entity who will check the enforcement of the policy. So it is necessary to indicate the subject and target of the policies explicitly. Furthermore, by explicitly relating rules to their enforcement and adherence components (QoS monitoring components), our adaptive policy model can take the policy conformance and enforcement logics for each individual quality requirement out of the core application. This is beneficial for monitoring QoS features in a flexible and dynamic manner. Each *Rule* or *RuleSet* has associated meta-information, which is captured through the class *MetaSpecification*. *MetaSpecification* has *Subject*, which is the entity the rule or rules set will be applicable to (entity providing QoS feature), and *Target*, which is the entity enforcing the rule or rules set (entity assuring QoS is met). It uses *Transformation* and *QoSMonitoringProcess* for policy enforcement.

The policy model aids wide-scale adoption of complex and dynamic QoS features. The policy language is generic enough so that the policy semantic schema and core components (policy engine, inference engine, merging, comparison, conflict resolution and so on) do not need to be modified by the addition of new assertions. Addition and execution of associated third-party components is also policy driven (*extension policies*).

Entity Profile Model Finally, we propose a construct named *profile* which captures everything required to specify QoS features. It can be associated with a system entity and can be advertised. Thus, clients can use it to discover desired services. Entity profiles represents entities' QoS capabilities, constraints, extension constraints, service dependencies and dependent request templates. The client profile contains only QoS capabilities, QoS constraints, and extension constraints.

The entity profile model supports end-to-end global QoS better than current service description and advertisement mechanisms such as OWL-S. Unlike current approaches, every service description in AMPol-Q explicitly specifies a list of its dependent services so that the discovery mechanisms can gather global QoS information about all the candidate services. Furthermore, we propose *service request templates*, a functional request based on IOPE attributes [13], to enhance dynamic services discovery. These templates have static IOPE attributes and dynamic IOPE template variables which can be instantiated using the client functional request's IOPE attributes. Each service provides the templates for their dependent services and the third party can use them to discover other services.

We use OWL to implement the QoS model and core policy model constructs. Policy rules are written using SWRL language constructs, which use an ontology vocabulary described by the QoS model in OWL. The benefit of using this two layer approach is that, first by using OWL, it is possible to perform reasoning over the knowledge model (QoS model) and the policy rules, and second, by the use of SWRL policy rules and underlying policy framework, the system's QoS behavior can be controlled without any ambiguity. Details of the implementation are given in [12].

3 Discovery Framework

Discovery framework consists of *Service Discovery and Chaining*, *Global QoS Analysis and Policy Agreement* and *Contract Negotiation*. It provides mechanisms for discovering global QoS information about all the candidate services, selecting best matched services and binding selected parties in a QoS contract. As mentioned in Section 1, QoS based service composition requires complex calculations of aggregate and global QoS values, which makes it hard to work with QoS features without global analysis. We will show how this section addresses issues related toGlobal QoS based service composition.

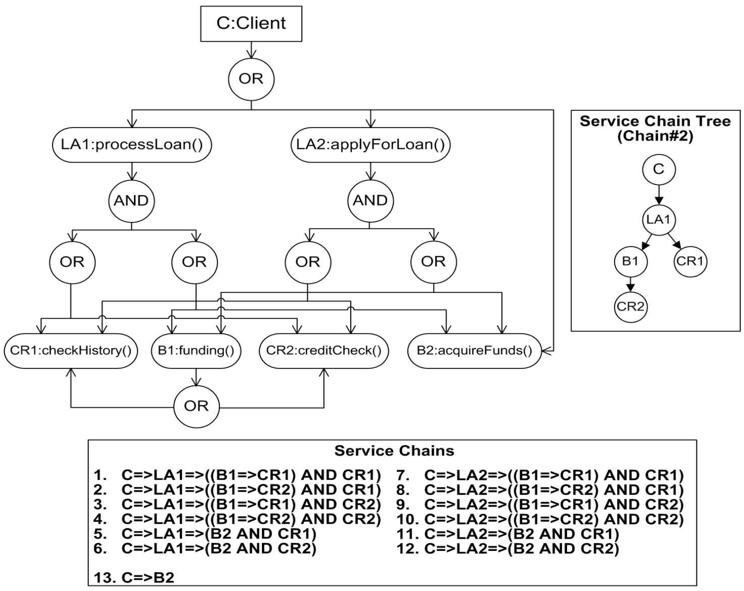

Fig. 2. Service Chain Graph

Service Discovery and Chaining The framework initiates a discovery process on behalf of a client. First the immediate-level services are discovered by using conventional IOPE based discovery approach. IOPE based request is send to a registry or directory service, which returns a list of services matched on the basis of functional IOPE attributes. We extend the discovery approach proposed by [14] to return AMPol-Q entity profile for the selected services. For each first-level service, the IOPE base discovery process is re-run to gather profiles of its dependent services. The IOPE request for discovering dependant services is generated from the request templates associated with a dependent service. The template variables are first assigned values from the available IOPE information of client or other services and then fully populated request is used for discovering profiles of dependent services.

Service discovery process continues until the profiles of all the candidate services are discovered. This global information can be modeled as an AND-OR graph called

Service Chain Graph (SCG). In the SCG, an OR combination shows the option of choosing one of the candidate service and an AND combination represents dependent services which must be composed. Figure 2 shows a SCG for the example of loan processing agency we discussed before. In this example, we have an option of two candidate services for each type. Client has an option of getting loan either from loan processing agencies or directly from a bank. Only bank B2 directly deals with small business clients. Loan processing agencies are dependant on credit reporting agencies and banks. Bank B1 independently verifies the credit score of a client from an external credit reporting service, while bank B2 has its own internal credit reporting department. By doing a traversal on SCG we can easily extract service chains (SC). Service chain represents a set of services which can provide a required service functionality. Global QoS analysis is done on each service chain to select a best candidate chain for final execution. For the above example, we have thirteen possible service chains. Service chain are further modeled as a tree to simplify the global QoS analysis and policy matching.

Global QoS Analysis and Policy Agreement Global QoS analysis has two steps: 1) pre-process QoS information; 2) match the policies and create a contract. These steps are repeated for each service chain in a SCG to create a list of policy contracts with associated agreement value.

Pre-processing is to map the global QoS requirements and capabilities to each individual node so that policy matching and agreement can be done independently between two nodes. It involves normalizing ontologies, filling rule templates, calculating aggregate QoS values, propagating rules and associating different entities with constraint rules. For example, for service chain 5 in Fig 2, the aggregate availability of the composite services (LA1, B2 and CR2) will be calculated by the product of availability value of each individual service, and then either a new capability rule is added to represent this value (e.g. in case of a broker) or the capability value of first level service (LA1) is replaced by the calculated aggregate value. Similarly, suppose client has a requirement of end-to-end message confidentiality then this constraint is propagated to all the services in the chain, so that during policy matching phase it can be compared against capabilities of each service.

Next, to find out whether a node fulfills the QoS requirements or not, QoS constraints are matched with QoS capabilities. For any constraint, if there is no matching capability (or capability is not sufficient enough) then there must be an associated capability module (adherence logic). Every rule can have associated adherence, verification and enforcement modules. If external capability is required then it must be checked against extension policy restrictions of that node. QoS requirement rule can only be satisfied if there is a matching capability rule available or there is an extension module available to provide the QoS capability and there are no extension restrictions on this module.

At last, a policy contract is created and an agreement value is assigned. Policy contract contains all entities in a service chain along with their capabilities and imposed constraints. Agreement value is penalized for every non-resolvable conflict, missing associated capability, no associated monitoring module, restricted extension modules *etc*. The service chains in which entities cannot fulfill the QoS requirements of each other are heavily penalized and hence have less chance of getting selected.

Contract Negotiation The contract with maximum agreement value in the policy contract list is selected, verified and signed from each entity in the service chain. The terms of the contract imply that the entities in question will comply with all the QoS constraints and will provide agreed upon QoS behavior. Policy contract is sent to each party in a service chain. Each individual entity verifies the contract policies against its private policies (if any). If a contract is rejected by any entity in a service chain then a next best contract is chosen for agreement. Negotiation process continues until all the entities agree on a particular contract. Because our service selection approach is based on global QoS information, it is able to select best set of services, while most existing approaches [14, 13, 15, 3, 1] can only select the first available matched service(s). Contract negotiation phase is optional but it provides assurance of a desired QoS from all entities even if capabilities or constraints are not advertised.

The discovery framework uses the Jena ontology inference engine and SWRL virtual machine to parse, normalize and compare policies. More implementation details are given in the [12].

4 QoS Monitoring Framework

Adaptive Middleware for QoS Monitoring Monitoring involves measuring delivered QoS, verifying QoS features and taking enforcement actions. AMPol-Q is an agile and adaptive middleware framework that enables the participants to adapt to QoS features of others during runtime. It is realized by two-way specialization, which extracts the logic of measuring QoS values and verifying and enforcing QoS policies by third party customized and pluggable components. These components are called *extensions*. This is executed in the way described by *extension policies*. The QoS features in a policy contract are associated with these extensions and can be dynamically added or removed per collaboration. In order to support a new QoS behavior, we do not need to change the core of the application. Instead AMPol-Q middleware can locate, load and execute new extensions automatically. The whole procedure is called *system extension*. We have used a middleware approach to mask problems of heterogeneity and distribution. Its flexibility and extensibility helps to support dynamic QoS, fine-grained policy control and seamless system evolution. It hides the implementation complexity from the core application logic and the functionality provided can be re-used by different applications. The discovery framework is also a part of the AMPol-Q middleware, which acts as a broker at the client end for discovering and selecting services. Figure 3 shows different components of the AMPol-Q middleware.

Entities in a service chain must be capable of providing requested QoS features, fulfilling QoS requirements, or complying with QoS constraints. We call this an *adherence logic*. First we need to distinguish between two types of QoS features, pluggable and non-pluggable. Pluggable QoS can be supported independently without any significant change to the core application, *e.g.* an encryption algorithm. Non-pluggable QoS features that cannot be supported by just adding an external capability, *e.g.* processing time or network bandwidth. Generally, qualitative features (capabilities) are likely to be pluggable more often than quantitative ones. A specialization can only be applied to a pluggable QoS feature. QoS capabilities may be pluggable through adherence

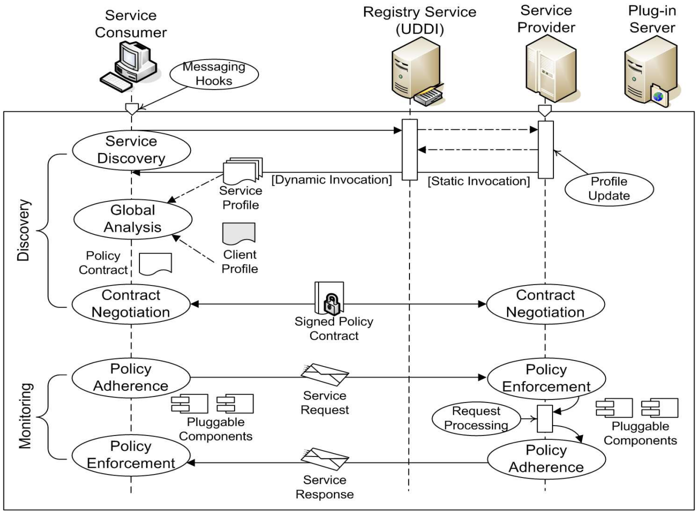

Fig. 3. AMPol-Q Middleware

components, while logic for QoS measurements, verification and enforcement for both qualitative and quantitative QoS features are easily pluggable.

The monitoring framework has three core components: QoS measurement, policy verification, and policy enforcement. Each component is heavily reliant on extensions. The service invocation process starts with the interpretation of policy contract at the client side. It executes a series of verification and adherence extensions on a request message to provide required QoS for a target service. On receiving a request, the service middleware first verifies the QoS constraint imposed by a service on the client. According to verification result the enforcement logic either rejects the request or forwards it to the service. Once the response is ready, the verification logic verifies that a response complies with client constraints. If the verification fails, the pluggable adherence logic is executed to conform the response message with the client constraints. On receiving a response, the client verifies the QoS delivered by the service, which may involve measuring QoS through extensions. If verification fails, then the enforcement mechanism will take actions accordingly. The QoS policies are verified, adhered or enforced on a point-to-point basis, but eventually they all comply with global QoS constraint and requirements.

Extension Manager The extension manager manages extension components and the system extension process. Extension management is controlled by extension policies, in which extensions are downloaded and executed only if extension policies allow doing so. Extension policies may restrict a type of extension to be only downloadable from a particular trusted extension server or may restrict the execution of an extension to allow limited access to the system resources (such as sandbox execution). Additionally, The system extension has a meta-level control over the adaptation process to ensure that the changes are effective.

Modules of the monitoring framework are implemented in C# and the extensions are packaged in separate DLLs. Details are given in the [12].

5 Validation and Case Studies

Policy-based WSEmail In this case study, we integrate AMPol-Q with WSEmail [7] to show how the email services could be enhanced to support QoS features in an end-to-end adaptive manner. In particular, our implementation is able to add new QoS requirements for availability and security. It deploys and uses plug-ins for puzzles [16] to raise burdens for email spammers [17, 18], and identity-based encryption [19] to allow senders to encrypt mail for recipients based on email addresses or other strings. As with the puzzles, our goal is to show how AMPol-Q can aid the deployment of IBE without requiring universal adoption of IBE by users. This case study is an extension of our implementation in [20] and illustrates the application of AMPol-Q to systems based on static service invocation rather than purely discovering other service dynamically.

The case study uses security domain QoS ontologies named APES [20] (*Attachment*, *Payment*, *Encryption* and *Signature*). *Encryption* and *Signature* classes specify the cryptographic parameters used for encryption or signature. For availability, *Payment* class specifies the type of cost (puzzles) imposed on the message sender. *Attachment* class specifies the patterns of the messages and attachment files, which is the primary medium for spreading viruses.

There are four entities involved in the system, the Sender Mail User Agent (SMUA), the Sender Mail Transfer Agent (SMTA), the Recipient MTA (RMTA) and the Recipient MUA (RMUA). MTAs advertise their clients and their own entity profiles, which are merged with client profiles for simplicity. MTAs entity profiles also contain dependent services (Relays or RMTAs) and their request templates, which can be used to dynamically discover dependent MTAs. These request templates also specify a mechanism to discover relaying MTAs by providing a reference to an extension e.g. a plugin for querying local DNS server for finding next hop MTA. In the example settings we map a MTA to a single relay per email address domain, which is in fact a target RMTA. So in this case we only have one service chain with three entities (SMUA $\Longrightarrow$ SMTA $\Longrightarrow$ RMTA). Also there is a third-party trusted plugin-server which hosts the extensions. For the current setup we show how the SMUA can automatically adapt to the QoS constraints of the target services (SMTA, RMTA and RMUA).

The MUA's AMPol-Q middleware is configured as a broker for discovering profiles of other entities. AMPol-Q first requests an SMTA entity profile and then fills in the dependency request templates; this only requires email addresses for the users. It invokes a pluggable discovery component to retrieve the merged entity profile of the RMTA. Because there is only one service chain, a single contract is created with an agreement value and simply send to other entities for QoS monitoring. Messages sent by the SMUA are verified against the contract and accordingly adherence extensions are downloaded and executed to conform the message with required QoS constraints. At the SMTA, the received message is first verified by the middleware and then processed by the SMTA application (if the verification succeeds). When the message is relayed to the RMTA, it is again verified and then forwarded to the RMUA. QoS discovery, verification, measurement, adherence and enforcement mechanisms are provided through pluggable extensions which are automatically downloaded from a trusted third party plug-in servers.

Web based WSEmail Based on WSEmail, this case study realizes AMPol-Q for typical web-based applications. Here a web browser client (CB) and a web application server (AS) adapt themselves to accommodate QoS aware service discovery and monitoring. The motivation behind this case study is that most of the client applications in SOA are web based and we try to show that how easily AMPol-Q can enable these client applications to be QoS aware.

We extended WSEmail by providing an application server and a browser-based MUA instead of the WSEmail MUA. We also extended it to provide a multi-hop and multi-relay topology to dynamically discover relays. Profiles are advertised on a UDDI-based server instead of relying on DNS entries. On receiving an HTTP request from a MUA browser, the application server internally talks to the WSEmail MTA and replies with an HTML page. In contrast to previous case study, it is not possible for the web client to do dynamic discovery and selection of services and to publish or advertise its QoS policies.

Our implementation considers AS and CB to be two independent entities with their own QoS features. CB does not need to discover any services as it statically invokes AS, while AS dynamically discovers other services. HTTP request from a CB is intercepted by AMPol-Q middleware and it first sends a modified HTTP request for service selection along with CB's QoS policies and functional intent to AS. The corresponding AMPol-Q middleware component at AS receives the request and initiates the service discovery based on CB request. We consider each AS application (for example, servlet or asp pages) to be a service interface and like other services, AS should also provide a complete entity profile including request templates to discover other dependent services. In the web-based scenario these profiles do not need to be advertised at registry service as the AS is never dynamically invoked by clients. The final service chain is selected and the contract is negotiated by AS. The communication between AS and CB is done through HTTP requests and responses. Finally the original HTTP request from CB is evaluated against an agreed contract and the final modified HTTP request is sent to AS. On receiving a response message, it is monitored by verifying against agreed contract.

We used Firefox Mozilla v1.5 as the browser and Apache Tomcat (v4.1) as AS. See [12] for the implementation details and video demonstration.

6 Related Work

Different service description (*e.g.* OWL/OWLS, Web Service Modeling Ontology) and QoS models [1, 8, 9] represent services with both functional and non-functional requirements, but they do not provide explicit support for compositional QoS and E2E service discovery. The OWL-S process model has implicit information about dependent services, but this information is not useful for discovering other services. Additionally, the QoS models in these works do not capture monitoring and compositional aspects. There are studies [2, 3, 14, 21] on QoS aware dynamic discovery and composition of services, but these are not able to discover or compose services on the basis of E2E global QoS features and do not provide sufficient support for continuously changing QoS requirements. There is no comprehensive specification that states how dynamic selection and invocation of services is to be performed on the bases of QoS features.

There are efforts on contract monitoring [5, 6] and mediating services [4, 15] through trusted third parties, but these approaches are based on local criteria and do not address the global end-to-end QoS assurance problem of the composite business services. Different policy frameworks [10, 11] are used to enforce requirements for individual entities. Adaptability is achieved by adding, customizing or replacing entities such as aspects [22], components, or concerns [23]. Existing efforts assume a built-in logic to support and ensure QoS policy constraints (QoS requirements) or have a static binding with external processing components to handle policy rules. AMPol-Q provides a more flexible approach because it takes the QoS logic out of the core application and provides it in a form of pluggable extensions.

There is a work [24] on a broker-based framework for QoS-aware Web Service (QCWS) composition. It is based on several service selection algorithms used to ensure the E2E QoS of a composite web services. This work addresses the problem of evaluating E2E QoS, but leaves open questions about how to support and ensure them. It also does not address the issue of how to dynamically discover E2E global QoS information.

There is work [25, 26] on dynamic adaptation in a service-oriented framework that addresses entities that have different QoS requirements on a per session basis. This work does not provide concrete negotiation protocols and does not explicitly specify which system entity will enforce the policy. [27] is another policy-based effort to achieve E2E adaptability, but it also does not support negotiation of requirements and focuses more on system extensibility and policy framework. DySOA [28] provides a framework for monitoring the application system, evaluating acquired data against the QoS requirements, and adapting the application configuration at runtime. It has a simple manual policy negotiation between the requester and the provider but does not support runtime negotiation. It does not address system extensibility beyond the capability of reconfiguring system parameters. GlueQoS [29] proposes a declarative language based on WS-Policy to specify QoS features and a policy mediation meta-protocol for exchanging and negotiating QoS features. One obvious limitation of GlueQoS is that it does not support dynamic system extensibility. All of above efforts only can handle simple QoS features because WS-Policy framework they use is not generic and adaptive enough to support new types of QoS constraints.

In a related work [30] on messaging systems we explored using XACML to model policies for email systems. In this work policies are used for controlling access to mailing lists. A related effort [20] on adaptive policies uses a ‘non-semantic’ policy language to model security features. AMPol-Q uses a semantic approach to support more complex policies. [20] is similar to AMPol-Q but it is based on systems with static binding and a more domain-specific focus, while AMPol-Q has a more generic formulation. In other work [31], we explored more sophisticated policy merging mechanisms than the ones in AMPol-Q, but these could perhaps be used for AMPol-Q policies as well.

7 Conclusion

We have introduced AMPol-Q, a policy-driven adaptive middleware for providing E2E support for dynamic QoS features in SOA. Its main contributions are its E2E solution, its adaptive middleware framework for supporting and monitoring QoS features, its generic semantics-aware reference architecture for describing, discovering and composing services on the basis of their non-functional features, and its application of this middleware to the systems based on web services. AMPol-Q differs from other work on adaptation in its focus on exploring an E2E solution for QoS features that incorporates all of the necessary support features. This work also provides one of the most complete studies to date of a proof-of-concept QoS-aware policy system based on Web services. Our future work includes formal security analysis, improved security measures such as sandbox protection, features to support privacy, models for negotiating policies, policy conflict resolution and performance testing.

8 Acknowledgements

We are grateful for help and encouragement we received from Anne Anderson, Noam Artz, Mike Berry, Jodie Boyer, Rakesh Bobba, Jon Doyle, Omid Fatemieh, Munawar Hafiz, Fariba Khan, Himanshu Khurana, Steve Lumetta, Adam Lee, Kevin D. Lux, Michael J. May, Anoop Singhal, Kaijun Tan. This research was partially supported by NSF CCR02-08996, CNS05-5170, CNS05-09268, and CNS05-24695 and ONR N00014-04-1-0562 and N00014-02-1-0715.

References

1. Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., Chang, H.: Qos-aware middleware for web services composition. In: ITSE'04: IEEE Trans. on Software Engr. (2004)
2. Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., Shan, M.: Adaptive and dynamic service composition in eflow. In: Tech. Report, HPL-200039, Software Tech. Lab. (2000)
3. Zeng, L., Benatallah, B.and Dumas, M., Kalagnanam, J., Sheng, Q.: Quality driven web service composition. In: WWW'03:Proc. of 12th Int. World Wide Web Conf. (2003)
4. Piccinelli, G., Stefanelli, C., Trastour, D.: Trusted mediation for e-service provision in electronic marketplaces. In: Lecture Notes in Computer Science, 2232:39. (2001)
5. Mahbub, K., Spanoudakis, G.: A framework for requirements monitoring of service based systems. (In: ICSOC'04: In Proc. of the 2nd Int. Conf. on Service Oriented Computing)
6. Jurca, R., Faltings, B.: Reputation-based service level agreements for web services. (In: ICSOC'05: In Proc. of the 3rd International Conference on Service Oriented Computing)
7. Lux, K.D., May, M.J., Bhattad, N.L., Gunter, C.A.: WSEmail: Secure Internet messaging based on Web services. In: Int. Conf. on Web Services (ICWS '05), IEEE (2005)
8. Tsesmetzis, D., Roussaki, I.G., Papaioannou, I., Anagnostou, M.E.: Qos awareness support in web-service semantics. In: AICT-ICIW'06. (2006)
9. Dobson, G., Lock, R., Sommerville, I.: Qosont: a qos ontology for service-centric systems. In: EUROMICRO-SEAA'05. (2005)
10. Kagal, L., Paolucci, M., Srinivasan, N., Denker, G., Finin, T., Sycara, K.: Authorization and privacy for semantic web services. (In: AAAI'04: Workshop on Semantic Web Services)

11. Uszok, A., Bradshaw, J.M., Jeffers, R., Johnson, M., Tate, A., Dalton, J., Aitken, S.: Kaos policy management for semantic web services. In: IIS'04: IEEE Intelligent Systems. (2004)
12. AMPol-Q: website. `http://seclab.cs.uiuc.edu/ampol/AMPol-Q` (2006)
13. Sirin, E., Parsia, B., Hendler, J.: Filtering and selecting semantic web services with interactive composition techiques. In: IEEE Intelligent Systems, 19(4). (2004)
14. Pathak, J., Koul, N., Caragea, D., Honavar, V.G.: A framework for semantic web services discovery. In: WIDM05. (2005)
15. Shuping, R.: A model for web service discovery with qos. In: ACM SIGecom. (2003)
16. von Ahn, L., Blum, M., Hopper, N., Langford, J.: CAPTCHA: Using hard AI problems for security. In: Proceedings of Eurocrypt. (2003) 294–311
17. Juels, A., Brainard, J.: Client puzzles: A cryptographic defense against connection depletion attacks. In: NDSS99: Networks and Distributed Security Systems. (1999)
18. Dwork, C., Naor., M.: Pricing via processing or combatting junk mail. In Brickell, E.F., ed.: Proc. CRYPTO 92, Springer-Verlag (1992) 139–147
19. Boneh, D., Franklin, M.: Identity based ecncryption from the Weil pairing. SIAM J. of Computing **32**(3) (2003) 586–615
20. Afandi, R., Zhang, J., Hafiz, M., Gunter, C.A.: AMPol: Adaptive Messaging Policy. In: 4th IEEE European Conference on Web Services (ECOWS'06), Zurich, Switzerland, IEEE, IEEE Conference Publishing Services (2006)
21. Karastoyanova, D., Buchmann, A.: Development life cycle of web service-based business processes. enabling dynamic invocation of web services at run time. In: ICSOC'05: In Proc. of the 3rd International Conference on Service Oriented Computing. (2005)
22. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-oriented programming. In Aksit, M., Matsuoka, S., eds.: Proceedings ECOOP '97. Volume 1241 of LNCS., Jyvaskyla, Finland, Springer-Verlag (1997) 220–242
23. Hürsch, W., Lopes, C.V.: Separation of concerns. Technical Report NU-CCS-95-03, College of Computer Science, Northeastern University, Boston, Massachusetts (1995)
24. Yu, T., Lin, K.: Service selection algorithms for composing complex services with multiple qos constraints. In: ICSOC'05: 3rd Int. Conf. on Service Oriented Computing. (2005)
25. Mukhi, N.K., Konuru, R., Curbera, F.: Cooperative middleware specialization for service oriented architectures. In: WWW '04, IEEE Computer Society (2004)
26. Mukhi, N., Plebanni, P., Silva-Lepe, I., Mikalsen, T.: Supporting policy-driven behaviors in web services: Experiences and issues. In: ICSOC '04, IEEE Computer Society (2004)
27. Baligand, F., Monfort, V.: A concrete solution for web services adaptability using policies and aspects. In: WISE'03: Proceedings of the Fourth International Conference on Web Information Systems Engineering, IEEE Computer Society (2004)
28. Bosloper, I., Siljee, J., Nijhuis, J., Hammer, D.: Creating self-adaptive service systems with dysoa. (In: ECOWS'05, Proceedings of the 3rd European Conference on Web Services)
29. Wohlstadter, E., Tai, S., Mikalsen, T., I.Rouvellou, Devanbu, P.: Glueqos: Middleware to sweeten quality-of-service policy interaction. In: ICSE '04: Proceedings of the 26th International Conference on Software Engineering, IEEE Computer Society (2004)
30. Bobba, R., Fatemieh, O., Khan, F., Gunter, C.A., Khurana, H.: Using attribute-based access control to enable attribute-based messaging. In: Annual Computer Security Applications Conference (ACSAC '06), Miami Beach, FL, Applied Computer Security Associates (2006)
31. Lee, A.J., Boyer, J.P., Olson, L.E., Gunter, C.A.: Defeasible security policy composition for web services. In: Formal Methods in Software Engineering (FMSE '06), Alexandria, VA, ACM (2006)