

# Using a Lease to Manage Service Contracts in Service Oriented Architectures

**Michael E. Stevens**

Rensselaer Polytechnic Institute  
mestevens@acm.org

**Heidi J. C. Ellis**

Rensselaer Polytechnic Institute  
heidic@rh.edu

## ABSTRACT

Service-oriented architecture concepts and Web Services specifications allow for an infinite binding between the service requestor and the service provider. This open-ended form of interaction constrains the service provider from changing service contracts, policies and endpoints. Other distributed technologies and protocols such as Jini and DHCP provide for the use of a lease to more easily manage service provider resources and to manage change in the environment more efficiently. This paper introduces the concept of a *lease* as it applies to service-oriented architectures and the concepts of an *active lease* which maintains state, and a *passive lease* to eliminate the need for the service provider to maintain a conversational state between service requestor and service provider. Lastly, this paper identifies the components of a service description that would require a rebinding should the components change, and it explores the impact of an active and a passive lease on several quality attributes.

## Keywords

Lease, service-oriented architecture, Web Services

## INTRODUCTION

A service-oriented architecture (SOA) is a distributed architecture consisting of components and connectors that form a conceptual basis for implementing distributed computing systems that minimize coupling between distributed components (Baresi, Heckel, Thone and Varro, 2003). SOAs permit the instant and dynamic assembly of services into larger grained processes at the time that the processes are needed and processes are then unbound when they are no longer needed (Bennet, Xu, Monro, Hong, Layzell, Gold, Budgen and Bereton, 2002). SOAs consist of three types of actors: service requestors, service providers and discovery agencies (Champion, Ferris, Newcomer and Orchard, 2002).

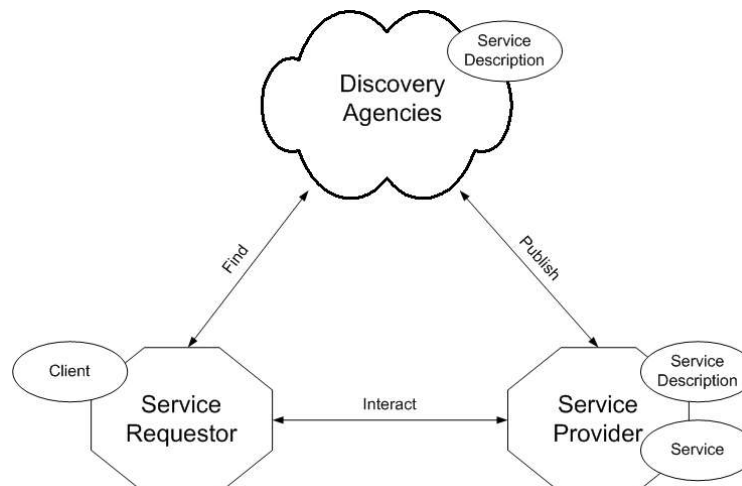


Figure 1. Service Oriented Architecture (Champion, et al., 2002)

A service requestor is a client that needs to invoke a service. It is the actor that initiates the discovery of a service in a discovery agency, binding to the service over a transport and executing the service function. The service provider exposes functionality that may be used by one or more service requestors. The discovery agencies contain service descriptions that

have been published by service providers and are queried and used by service requestors. The discovery agencies provide two sets of interfaces that are used by both service providers and service requestors. A service provider uses one set of interfaces to publish descriptions of the services it offers (Stevens, 2003). Service requestors use the other set of interfaces to lookup the available services in the discovery agency. Figure 1. depicts the service requestor finding a service provider description and interacting with a service provider according to the published description of the service.

Service descriptions contain information about the service. The service requestor uses this information in two ways. First, the service requestor uses this information to match against its lookup criteria to find a service that meets the desired characteristics. Second, the service requestor must also adhere to the interface contract contained in the service description in order to interact with the service provider.

SOAs have the following loosely coupled characteristics (Stevens, 2003):

- 1) Knowledge of the service provider is limited to the service description.
- 2) Service descriptions are discovered dynamically at run-time.

A service requestor uses a discovery agency to find the description of the service. This description contains the interface contract that the service requestor needs in order to interact with the service. The service requestor uses this interface contract at run-time to interact with the service provider. The service provider interface contract is independent of the implementation of the service provider. These properties provide a loose coupling between service requestors and service providers. This loose coupling allows the service provider to change the service implementation as long as the new implementation adheres to the same interface contract. A SOA maximizes the information hiding and modularity of the service provider by making the service provider implementation inaccessible except through the published service interface.

SOAs support the run-time discovery of service implementations and late binding of service requestors to service providers. Run-time discovery of services improves loose coupling because it delays the point at which the service requestor needs to know the details of the service description until run-time. This ability gives the service provider more flexibility to change service implementations because clients are not statically bound to the service.

#### **Shortcomings of SOAs and Web Services – Infinite Binding**

Many current SOA architectures and Web Service implementations allow service requestors to bind to the service contract and execute the service according to that contract for an infinite period of time. A service provider that needs to change the service description is limited by the possibility of an *infinite binding* of service requestors to the service provider. The problems that arise when infinite binding occurs include:

- 1) A service provider cannot change the interface contract for the service because multiple service requestors may be bound to the current interface contract.
- 2) A service provider has no means of identifying currently bound service requestors.
- 3) A service provider has no method of ensuring that a change to the service description will not disrupt one or more service requestors.

These properties of SOAs and Web Services reduce their loosely coupled potential.

#### **Lease Constructs for SOAs**

In this paper, we introduce the concept of a *lease* as it applies to SOAs. A lease is a constraint on the binding of a service requestor to a service provider and limits the length of time that a service requestor is bound to a service provider. The lease specifies the conditions under which a service requestor may use the service. A lease enhances the low coupling between SOA components by allowing components the flexibility to change interface contracts over time. When a lease elapses, the lease is said to have expired and the service requestor may not access the service. The service requestor must obtain new lease information to continue to access the service provider.

SOA implementations using Sun's Jini (Sun, 2001) technology use a lease to reduce the infinite binding of service requestors to service providers. This approach allows the service provider to better manage service provider resources and more easily update service interface contracts. It also provides a facility to automatically remove stale service provider references from the lookup service. The dynamic host configuration protocol (DHCP) (Droms, 1993) also uses a lease construct. DHCP uses a lease to manage IP addresses by allowing a network node to use the IP address only for the period of time specified by a lease.

Web Services (Champion, et al., 2002; Orth, 2002) are a new and rapidly evolving set of standards and technologies. They are desirable because they allow distributed systems to communicate with each other in an interoperable, loosely coupled manner. Web Services are a popular set of technologies that are used to implement SOAs. Web Services also have no standard means for specifying a service description lease.

While there is no prescribed approach for implementing a lease in Web Services, the WS-Policy (IBM, 2004) framework developed by the Web Service Interoperability Organization (WS-I) provides a model and syntax for expressing a broad range of service requirements, preferences and capabilities. We suggest using the WS-Policy framework to express a lease in a machine-readable form. The service requestor may use a service proxy object to reload the service description when the lease expires.

### **Benefits of a Lease Construct in SOAs**

We have identified several benefits that may be realized from the employment of a lease in a SOA. These benefits will be discussed in proceeding sections. Benefits include support for:

- 1) Simpler interface description changes
- 2) More efficient service provider resource management
- 3) More robust fail-over capability

### **SERVICE DESCRIPTION CHANGE**

In order to estimate the utility of using a lease in SOAs, we must identify the types of contract changes that can take place without requiring a change in the service requestor. It is assumed that the service provider is trusted to make changes in its implementation without a manual notification to the service requestor owners.

The interface description for a service may change over time. For instance, a service provider might need to update the service interface description should an operation need an additional type. The service implementation description might be updated if the network address needs to change or if an additional transport type is added or retired. A service policy may change should a quality of service level or an application level constraint change. A composite service might change the way that the finer-grained services are composed and consequently need to update the service interface or quality of service descriptions.

Generally, a

ny change in a service description that has the same or is less constraining may be handled dynamically at run-time. For example, if a text field size is increased, then this change may occur dynamically without changing the service requestor. However, if a new required data type is added, then the service requestor will need to be updated to handle this change. The service lease provides value in the case of less constraining service interface descriptions. Some examples of service description changes that may be handled dynamically include:

- 8) Reduction in required data size
- 9) Broader range checking
- 10) Additional lookup list values
- 11) Data type change from checked to unchecked
- 12) Additional non-required data elements
- 13) Change in network endpoint
- 14) Less stringent QoS policy change

There are several components of a service provider that might be updated. In this section, we will describe the various elements of a service description and then outline some of the possible scenarios that might cause a service description to be updated and demonstrating the utility of a lease construct in these scenarios. In the subsections to follow, we discuss the information that may be included in a service description in a machine-readable format (W3C, 2004).

### **Service Interface Descriptions**

Service interface descriptions contain the interface contracts for the services including the data types, message descriptions, and interaction types (e.g. synchronous, asynchronous, one-way, etc.), and the operations and descriptions of the operations that a service supports. Web Services use Web Services Description Language (WSDL) (W3C, 2004) documents to describe

the service interfaces.

A service provider updates the interface description when message types change or the methods of interaction between the service requestor and the service provider changes. When a change in the service interface description is needed, the service provider will maintain two versions of the service interface at two different endpoints. When the lease expires on the previous version, the service requestor will obtain the new service interface description and the new endpoint address from the discovery agency. When the duration of the lease expires, the service provider will retire the old service interface.

### **Service Implementation Descriptions**

Service implementation descriptions specify the service transport types that are available and the network addresses for each transport type. The WSDL document in Web Services requires the specification of ports. A port associates a set of operations and a transport type with a network address.

A service provider may change the network address for a given service endpoint. If such a change is necessary, then the service requestor will obtain the new endpoint network address when the lease expires. This capability gives the service provider the ability to update the network locations of services known by service requestors dynamically, without human intervention. Furthermore, in the case of failure of a network endpoint, the service provider may change the endpoint in the discovery agency so that the service requestor may retry failed transactions on that endpoint.

### **Policies**

Policies contain the constraints that a service requestor must understand and abide by when invoking a service. The service policies contain quality of service (QoS) (Chatterjee, Sydir, Lawrence, 1997) levels for the service as application level constraints. Service requestors use QoS levels to identify services that meet non-functional requirements such as security and performance. Application level constraints contained in the service policy documents describe the business terms and conditions of usage of the service.

A service provider might update a quality of service characteristic of a service such as performance or security that might not meet the currently bound service requestor requirements. When the lease expires, the service requestor will identify a new service that meets its non-functional requirements. For example, an application level constraint such as the time of day that a service is available may be updated dynamically. If the time of day that the service is available is increased, then the service requestor may dynamically send additional requests to that service during the newly stated hours of availability.

### **Relationships Between Services**

Another component of service descriptions that may change are the relationships between services. Fine-grained services may be composed into coarser grained services in several ways. A software agent may programmatically invoke several Web Services and publish a composite result set. Web Services may also be orchestrated whereby services interact and provide a composite service by using a standard markup language that describes the interactions. Markup languages such as the business process execution language for Web Services (BPEL4WS) (IBM 2004) provide these capabilities. Generic business process agents use documents that describe a business process to execute multiple services and assemble the specified results. These orchestrated business processes are services. They have full service descriptions that are published in discovery agencies.

Composite services are dependent on the service descriptions of the services they compose. When the description of a service changes, the composite service description may also change. Therefore, the lease value of the composite service description must be the same or less than the smallest lease value of the service that it composes. In other words, if the service description of the composed service changes more frequently than the composite service, then the service requestors that are bound to the composite service may be bound to a service that is in an invalid or changed state. If the lease on the composite service is the same or smaller than the composed services, and they are synchronized, then the composite service can manage changes and report those changes to the discovery agency to satisfy the service requestors that are bound to it. For example, consider a composite service that summarizes a customer's stock account and bank account. If the lease on the stock account service description is ten minutes and the lease on the bank account service is five minutes, then the lease value for the composite service must be five minutes or less and synchronized.

## **SERVICE LEASE**

In this section, we define the concept of a *lease*. A service lease is a constraint on the binding of a service requestor to a service provider. It limits the potential for an infinite binding between these two entities. A lease is granted by the service provider and is used by the service requestor (Jain, Kirchner, 2000). The lease specifies the conditions under which a service requestor may use the service. When a lease elapses, the lease is said to have expired, and the service requestor may not

access the service. The service requestor must obtain new lease information to continue to access the service provider.

The implementation of a service lease in a SOA involves the interaction of the participants shown in the Unified Modeling Language (UML) class diagram in Figure 2.

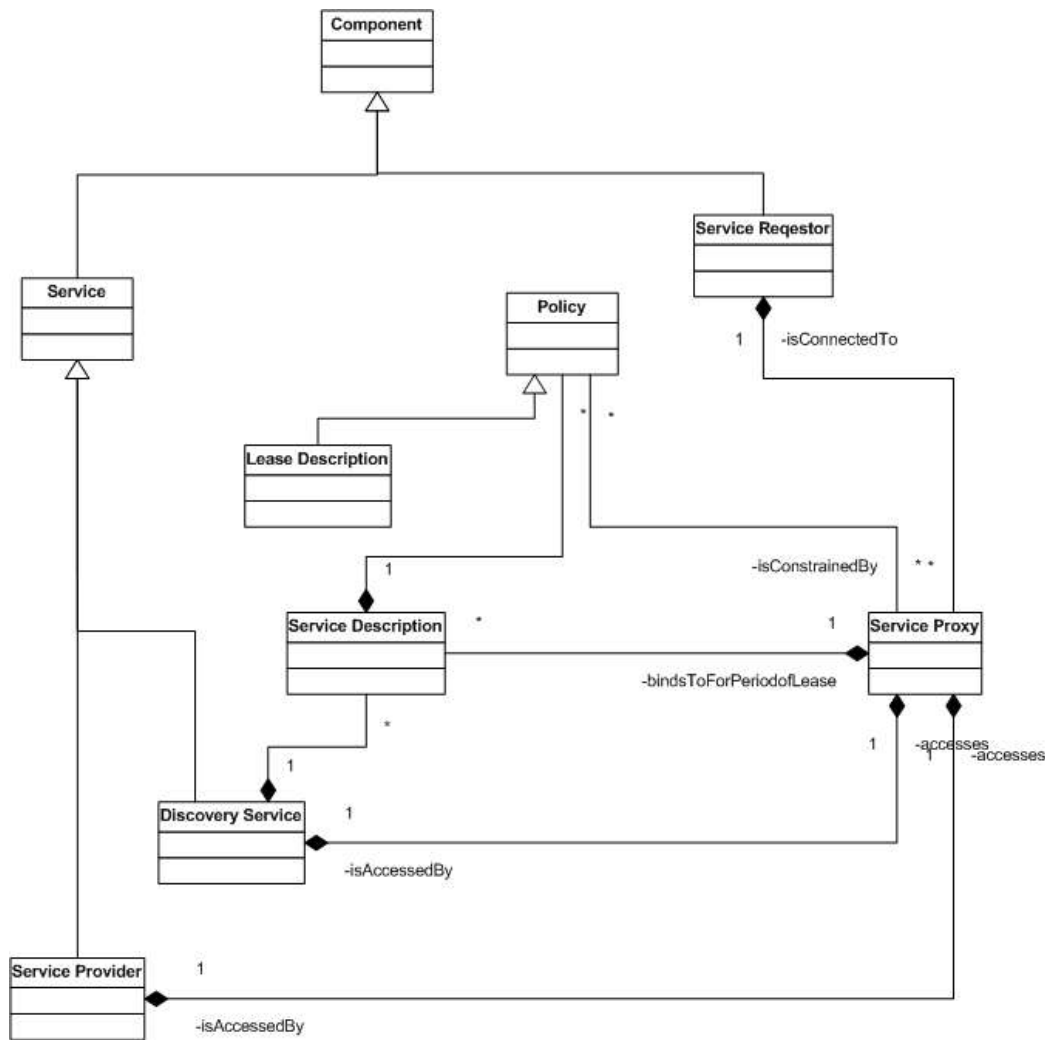


Figure 2. – SOA static model with lease participants

A service requestor typically uses a service proxy (Gamma, Helm, Johnson, Vlissidies, 2003) to access a service. The service proxy encapsulates the logic for discovering the service, binding to the service and executing service functions. The service proxy is also responsible for adhering to the constraints described in the service policy documents including the lease description.

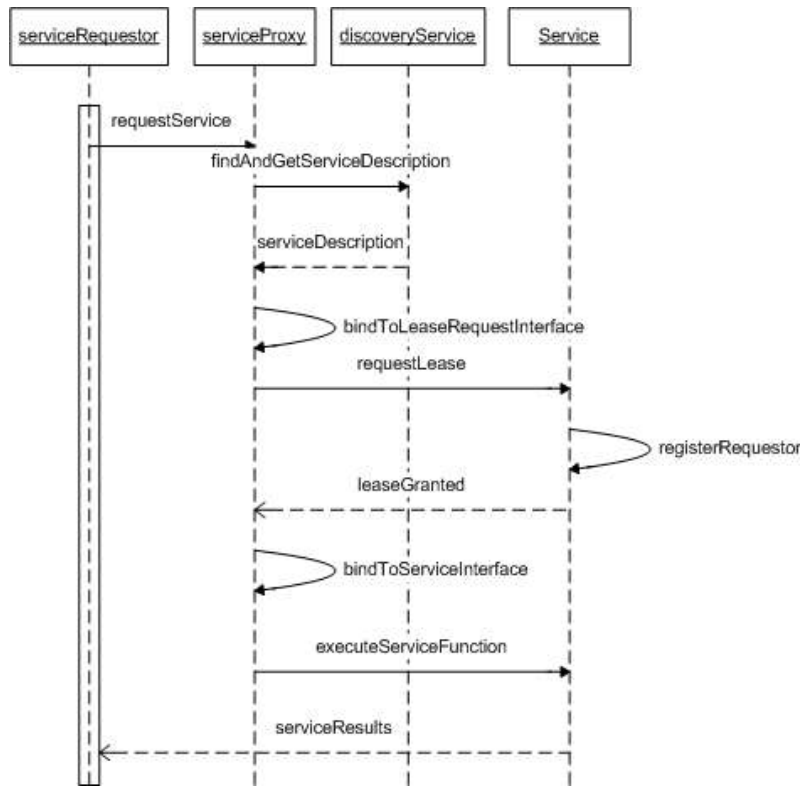
We propose two primary patterns of implementation for constructing a lease in a SOA:

- 1) Active Lease – an active lease is the classic leasing concept whereby a service requestor is allowed to access a service provider for a length of time. This length of time uses the lease request time as the starting point for the length of the lease. When the amount of time has expired, then the service requestor must request another lease if they desire to continue to access the service. Jini and DHCP are examples of an active lease scheme. Alternate implementations may specify the number of executions that the service requestor may make to the service provider. In addition, a lease may combine a time period with a number of executions to constrain a service requestor to a specified number of service executions within a period of time.
- 2) Passive Lease – a passive lease is a new concept introduced by this paper. It is also a length of time during which the service requestor is allowed to access a service provider. However, this length of time is specified in the service description for all service requestors as an expiry time.

An active lease requires the maintenance of conversational state. This paper introduces the passive lease scheme to eliminate the need for the service provider to maintain conversational state. A property of an SOA is that all conversations between service requestors and service providers are stateless (Champion, et al., 2002). This means that all of the information necessary to process a given request is in the request. Due to the stateless nature of SOAs, a passive lease scheme is an attractive option.

**Active Lease**

With an active lease, it is the responsibility of the service provider to explicitly grant a lease to each service requestor. As shown in the sequence diagram in Figure 3., the service requestor must explicitly request a lease from the service provider by calling a lease request service interface on the provider prior to executing the service function. Access to the service provider is granted from the time that the lease was requested until the lease time has expired. A service requestor will be denied access to the service provider if a lease has not been explicitly granted to the service requestor or if the lease has expired. If the service requestor has been denied access, then the service requestor must explicitly request another lease from the service provider. The service requestor will request a new lease prior to lease expiration. An alternative implementation for handling lease expiration is for the service provider to notify the service requestor via a callback that the lease has expired. The WS-Callback protocol (BEA, 2004), specified by BEA Systems, provides the SOAP and WSDL 1.1 mechanisms that enable an asynchronous callback from the service provider to a service requestor. This protocol is attractive because it could provide a standard means of notifying a service requestor of a lease expiry.



**Figure 3. – Active Lease Sequence Diagram**

With an active lease, the service provider uses the lease description to manage access to services. The service requestor uses the lease description to request another lease prior to lease expiration.

Active leases have the following benefits:

- 1) Active leases offer better service provider resource management (Jain, Kirchner, 2000). Since the service provider has information about all of the service requestors, it can allocate resources more efficiently. If the service provider allocates resources for individual service requestors, then those resources may be released and used for other tasks. In addition, if there are a limited number of resources that may be used by service requestors, access to those resources can be controlled through the use of load balancing.
- 2) Active leases allow service provider implementations to be changed more easily. A service provider

implementation may be changed whenever there are no service requestors that are currently bound to the old service implementation. Unlike the passive lease scheme, the service provider change does not necessarily need to wait until the lease time has expired if it is known that there are no currently bound service requestors.

When a service description changes, the service provider cannot immediately change a service implementation. The new service implementation and the old service implementation must be available for a period of time. For active leases, the old service implementation must be maintained until all service requestors have unbound from the old implementation and bound to the new service implementation.

### Passive Lease

In a passive lease scheme, the service provider does not grant access to the resource explicitly. The lease value specifies the timeframe in which the service description is valid. The lease contains the date and time at which the service description will become invalid. The service requestor is required to obtain a new service description and rebind to the service prior to the old service description becoming invalid. This gives the service provider the flexibility to change service descriptions. The service provider does not know what service requestors are bound to the service. This means that the service provider does not know when the service requestors read the lease or if there are current service requestors that are using expired service descriptions. It is the responsibility of the service proxy on behalf of the service requestor to check for lease expiration and obtain a new service description when the lease expires for both active and passive leases.

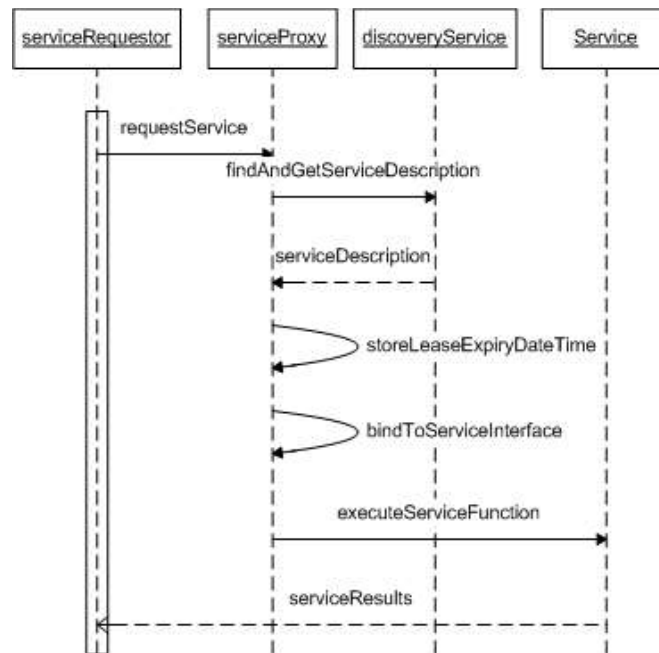


Figure 4. – Passive Lease Sequence Diagram

The sequence diagram in Figure 4. shows the service requestor executing the service function but not requesting a lease or registering a session with the service provider.

The benefits to the passive lease scheme over an active lease scheme include:

- 1) Simpler service provider implementation - the service provider does not have to hold information about the leases that have been granted to service requestors.
- 2) Better scalability - the service provider requires fewer resources because the service provider is not required to maintain state for each service requestor.
- 3) Better performance - an explicit request to the service provider to request a lease is not necessary, thus improving performance over an active lease scheme.
- 4) Simpler change implementation - when a service description change is necessary, the service provider does not necessarily need to maintain two versions of the service. If the clocks of the service requestor and the service provider are synchronized, then the service provider can change implementations without maintaining the old implementation. However, if the clocks of the service provider and the service requestor are not synchronized,

then the service provider will need to maintain both the old service provider and the new service provider for a period of time.

Applications that require better scalability and performance should implement a passive lease scheme. However, applications that require stateful services and undergo a high degree of change would benefit from an active lease scheme. Stateful services will necessitate resource management capabilities that are best satisfied through the use of an active lease.

### **Implementing A Lease With Web Services**

The current Web Services specifications do not contain a protocol or a language for specifying leases. However, the WS-Policy (IBM, 2004) framework may be used to specify a lease. A smart proxy may use the lease specification in WS-Policy to refresh the service description and rebind to the service when the lease expires. A smart proxy dynamically changes its interface at run-time based on the current interface contract. When the interface contract changes, the interface on the smart service proxy is updated. If an interface is removed or a data type in the interface has changed, then the service requestor will handle an exception at run-time instead of a compile error at design-time. The smart proxy can also gracefully handle network protocol or endpoint changes specified in the service description without affecting the service requestor. Smart proxies are able to cache service descriptions and handle service description changes and improve performance by caching results for subsequent service requests. The smart service proxy also can support a standard callback mechanism to implement a callback notification from the service provider that the lease has expired. A service proxy is a valuable reusable component that insulates service requestors from the intricacies of managing service provider interaction.

The WS-Policy framework provides a general-purpose model and syntax for specifying the policies for a Web Service. WS-Policy has a base set of constructs that can be extended by other Web Services specifications to describe other service requirements, preferences and capabilities. WS-Policy is used by other specifications such as WS-Security to specify the requirements and constraints of a service provider.

### **Impact On Quality Attributes**

A service lease improves some quality attributes of SOAs and has a negative impact on others. Using a service lease positively impacts modifiability. A lease improves the modifiability of a SOA by providing a simple approach to changing a service provider interface. Modifiability is particularly important in applications that span multiple organizations. Without a facility to provide automated interface versioning support, inter-organization services are extremely difficult to modify.

However, performance is decreased when using a service lease due to the additional messages that are passed from the service consumer to the service provider. In a passive lease scheme, there is less of an impact on performance because there is no lease protocol between the service requestor and the service provider. The service requestor must only abide by the lease retrieved from the discovery agency. However, there is an impact on performance because the service requestor must retrieve a new service description and rebind to the service provider when the lease duration expires. An active lease has a greater impact on performance because a separate and distinct request is made from the service requestor to the service provider to request a lease whenever the lease expires.

Using an active service lease can negatively impact scalability. An active lease introduces a requirement to store state in the service to manage the service requestors that are currently bound to the service. Adding additional service implementations for a single service would require that the implementations share information about the state of leases that have been issued to service requestors. This additional complexity reduces the scalability potential of an active lease. A passive lease does not impact the scalability potential of a SOA because it does not add a requirement to store additional state in the service provider.

By using a passive lease scheme in a stateless SOA, availability and reliability could be improved. A stateless SOA may use a passive lease to increase availability by introducing multiple endpoints into the service description. If an endpoint has failed, then the service requestor, via the proxy, may send the requests to the backup endpoint(s). In addition, if some of the endpoints are experiencing sporadic failures, then those endpoints may be removed from the registry. When the lease expires, those problematic endpoints will not receive any additional requests from service providers because they have retrieved service descriptions from the discovery agency with the problem endpoints removed. By using these techniques with a passive lease and a stateless SOA, overall availability and reliability can be improved.

There is a potential to use these techniques in an active lease also. However, active leases require the transfer of the state of the leases between service provider implementations in order for these techniques to be effective. The implementation and management of the state transfer may make this implementation prohibitively expensive, especially if service provider implementations exist on multiple non-clustered servers or even in different organizations.

## FUTURE WORK

There are lease implementation variants beyond the active and passive lease implementations described in this paper. Additional work would include the development of lease schemes that are effective in different deployment contexts. These different contexts may include clustered service provider implementations, intra-organizational service provider implementations and inter-organizational service provider implementations.

In addition, a new WS-Lease specification should be developed and considered by standards bodies for inclusion in Web Services. The new specification would extend the WS-Policy and perhaps include WS-Callback specifications to implement lease protocols.

## CONCLUSION

Web Services and SOAs provide a standards-based, loosely coupled and dynamic set of technologies and design principles for systems development. However, other distributed technologies and protocols such as Jini and DHCP provide for the use of a lease to more easily manage service provider resources and to manage change in the environment more easily. In a distributed environment where service providers and service requestors are in different organizations and in many cases might not have any knowledge of each other, an automated method for handling change is a requirement. Change can be effectively managed by using a lease in SOA based systems.

## REFERENCES

1. Baresi, L., Heckel, R., Thone, S. and Varro, S., "Modeling and Validation of Service-Oriented Architectures: Application vs. Style," In Proc. ESEC/FSE 2003, Helsinki, Finland, September 2003.
2. BEA, "WS-Callback Protocol (WS-Callback) 0.91", [http://dev2dev.bea.com/technologies/webservices/WS-Callback\\_Intro.jsp](http://dev2dev.bea.com/technologies/webservices/WS-Callback_Intro.jsp), accessed February 2004.
3. Bennet, K., Xu, J., Monro, M., Hong, Z., Layzell, P., Gold, N., Budgen D., Brereton, P., "An Architectural Model for Service-Based Flexible Software," In Proc. COMPSAC'01 2001, IEEE, 2002.
4. Champion, M., Ferris, C., Newcomer, E., Orchard, D. "Web Services Architecture, W3C Working Draft 14," 2002, <http://www.w3.org/TR/2002/WD-ws-arch-20021114/>, accessed January 2004.
5. Chatterjee, B., Sydir, M., Lawrence, T., "Taxonomy for QoS Specifications," In Proc. WORDS'97, Newport Beach, CA, 1997.
6. Droms, R., "RFC1541 Dynamic Host Configuration Protocol PROPOSED STANDARD," October 1993, <ftp://ftp.rfc-editor.org/in-notes/rfc1541.txt>, accessed January 2004.
7. Gamma, E., Helm, R., Johnson, R., Vlissidies, J., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley (2003).
8. IBM, "Specification: Business Process Execution Language for Web Services Version 1.1," <http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/>, accessed January 2004.
9. IBM, "Specification: Web Services Policy Framework," <http://www-106.ibm.com/developerworks/library/ws-polfram/>, accessed January 2004.
10. Jain, P., Kirchner, M., "Leasing Pattern," PLoP 2000 conference, Allerton Park, Illinois, USA, August 13-16, 2000. <http://www.cs.wustl.edu/%7Emk1/Leasing.pdf>, current October 2003.
11. McGovern, J., Tyagi, S., Stevens, M., Mathew, S., Java Web Services Architecture, Morgan-Kaufmann, 2003.
12. Orth, G., "The Web Services Framework: A Survey of WSDL, SOAP, and UDDI," Master's Thesis, Vienna University of Technology, 2002.
13. Stevens, M., Service-Oriented Architecture Introduction, Part 1, Developer.com, <http://www.developer.com/net/article.php/1010451>, current February 2004.
14. Stevens, M., Service-Oriented Architecture Introduction, Part 2, Developer.com, <http://www.developer.com/net/article.php/1014371>, current February 2004.
15. Sun, "Jini™ Technology Architectural Overview", <http://www.sun.com/software/jini/whitepapers/architecture.html>, 2001, accessed January 2004.
16. W3C, "Web Services Description Language (WSDL) 1.1", <http://www.w3.org/TR/wSDL>, accessed January 2004.