

Cache Mediation Pattern

Fang Yan Rao, Ru Fang, Zhong Tian

IBM China Research Laboratory

Building 19 Zhongguancun Software Park, 8 Dongbeiwang West Road, Haidian District,

Beijing, P.R.C. 100094

{raofy, fangru, tianz}@cn.ibm.com

Eoin Lane, Harini Srinivasan, Tim Banks, Lei He

IBM Software Group

{eoinlane, harini}@us.ibm.com

tim_banks@uk.ibm.com

heleihl@cn.ibm.com

Abstract

Performance is one of the major concerns in a Service Oriented Architecture (SOA) environment because of the verbose XML message transmitted between the service providers and the service consumers over a network. Asynchronous messaging is widely recognized as an effective communication channel in an SOA environment where the service provider and the service consumer may be separated across the Internet. As an important communication channel in distributed computing environments like SOA, Message-Oriented Middleware (MOM) increases the interoperability, portability, and flexibility of the applications. MOM has the mediation subsystem to perform operations like logging and transforming upon messages routed by MOM. This paper discusses a Cache Mediation pattern as a reusable solution to accelerate the service response time where the MOM is employed as the communication channel. Cache mediation is a cache residing in the mediation subsystem in the MOM between the service providers and the service consumers. Unlike traditional design patterns, this pattern is more at the infrastructure level and leverages the existing caching capability. Moreover, the pattern is implemented using the pattern authoring and application support from current commercial tools, and takes care of not only design, but also instantiations and deployment.

1 Introduction

Service Oriented Architecture (SOA) raises a lot of interest in both researchers and practitioners. In an SOA environment, service providers and requesters are loosely coupled and distributed across a network, either within an organization or across organization boundaries. In such distributed environment one important issue is performance. For example, XML is a widely used message format for the service providers and the consumers. XML message packaging and parsing brings extra overhead to both ends. Therefore web service invocation costs more in terms of the response time than some other kinds of remote procedure invocation. Reducing the response time of web service invocations is a critical challenge in many real life cases.

Asynchronous messaging is widely recognized as an effective communication channel in an SOA environment where the service provider and the service consumer may be separated across the Internet [1]. Comparing with other communication channels like direct SOAP over HTTP, MOM provides controlled and mediated message delivery between the service providers and the service consumers. The mediation capabilities like message transformation, logging, routing, etc, are provided by the mediations in MOM. The performance of such mediated service invocation is further decreased by the intervening message mediations.

Caching, a widely used technology to improve performance, has been used in many solutions successfully to address the performance issue. This paper discusses a Cache Mediation pattern as a reusable solution to accelerate service responses in an SOA environment where the MOM is employed as the communication channel. MOM normally provides a message queue between the service provider and the service consumer. It is typically asynchronous, but it also supports synchronous message passing as well [2]. MOM is the foundation of Enterprise Service Bus (ESB) [3], which is widely believed to be a cornerstone of many SOA environments. Mediations within MOM perform tasks like logging and transforming upon messages routed by MOM. In this paper, we introduce a Cache Mediation pattern based on the caching pattern concept proposed in [4].

Unlike the cache on the server side or the client side, cache mediation resides in the middle between the service providers and the service consumers. Therefore providing a cache solution in MOM can serve the purpose of

accelerating service requests from multiple service consumers residing on different locations, and can also benefit multiple service providers.

Unlike traditional design patterns which usually deal with object oriented solution design at a class level, this pattern is designed as a solution at the infrastructure level and leverages existing capability provided by MOM. Such a pattern is especially useful when today's practitioners develop software based on existing middleware by using tools, i.e. if they want to build a cache for messaging, they can build the cache on top of an existing caching capability rather than from scratch. Furthermore, tools like Rational Software Architect (RSA) provide strong tooling support to define and apply patterns [5]. We implemented the pattern in RSA using its pattern authoring and transformation framework [6] to help practitioners apply the pattern.

The rest of the paper is organized as follows: related works are introduced in Section 2, Section 3 presents the Cache Mediation pattern, Section 4 illustrates the pattern implementation and usage in RSA for the problem proposed in Section 3, Section 5 introduces some known uses of the Cache Mediation pattern, and Section 6 concludes our work.

2 Related Works

There are different categories of patterns – from the design pattern dealing with code level design issue of software components [7], to system-level patterns providing solution at macro-design of system components [8], to patterns for e-business as described in [9] which extend the software patterns to earlier phases of the application development life cycle. The patterns for e-business have four major types of patterns: business patterns, integration patterns, application patterns, and runtime patterns. The Cache Mediation pattern is an integration pattern that provides a cache solution in an SOA environment at the middleware layer.

As introduced earlier, performance is one key issue in an SOA environment given the verbose XML messages transferred between web service providers and consumers. Caching is a widely used technology to improve the application performance. Various caches have been designed and implemented to improve the responsiveness of application in various software products, like JbossCache [10], SpiritCache [11], DynaCache [12]. Using cache to reduce the service performance cost is a problem of interest to cache technology practitioners. Some cache patterns are proposed to provide the reusable cache solution, like the Caching pattern proposed in [13].

[4] proposes an Enterprise Service Bus (ESB) cache pattern to accelerate the service response time in the ESB context (Figure 1). It transfers messages and provides various mediation capabilities like message transformation, logging, routing, etc. In the ESB context, both the service provider and the consumer are attached to the Enterprise Service Bus. Message mediations reside on the service bus to handle the request or the response messages. In this caching scenario, when the cache mediation finds a response to a service request in the cache, it will return the cached response directly to the message requester. If it does not find such items within the cache, it will continue to forward the request to the service provider which in turn generates the response. Meanwhile response messages will be cached by the cache mediation. Unlike various server side cache frameworks like JbossCache and SpiritCache, an ESB cache accelerates costly remote service invocations by providing a cache on the messaging middleware level, or in the communication channel, therefore it can benefit multiple service consumers and service providers in one solution.

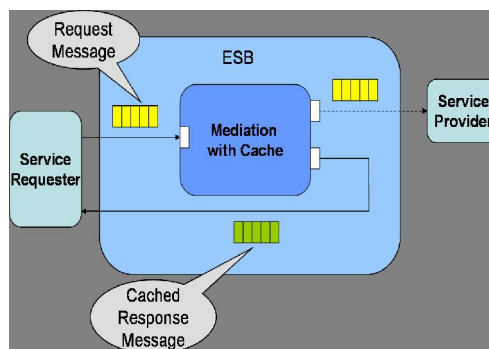


Figure 1 ESB Cache Pattern.

The ESB cache pattern is introduced as a concept in [4] without description of the pattern's structure. Based on this concept, we flesh out the idea and generalize it as a Cache Mediation pattern with detailed descriptions of its context, structure, consequences, etc. We also implement the pattern and illustrate the application of the pattern in the RSA tool in Section 4.

This work is also introduced in the 4th “Killer Example” for Design Patterns and Objects First workshop [14].

3 Cache Mediation pattern

This section provides the detailed description of the Cache Mediation pattern following the pattern description framework used in [13]. The Cache Mediation pattern described here is not targeted to provide the detailed design for a general purpose cache, but rather tackles the specific caching problems in the messaging middleware.

3.1 Example

Consider a weather forecast service provided as a value added service in wireless network, the mobile operator provides the connection channel from the end users to the service. The end users access the service using short messages. The mobile operator provides a MOM to transfer request messages to the provider and response messages back to the end users. When large amount of end users access the service at the same time, the workload on the service provider side increases, also does the workload on the mobile operator side. Consequently the overall service performance is decreased.

In addition to the weather forecast service, there are also other value added services facing the same problem, such as stock lookup service, ring tone download service and etc. The mobile operator wants to have one solution for these value added services.

The major challenge here is to improve multiple services’ performance in one solution with minimal change to the deployed services run by different value added service providers, while the services and their clients are connected via the MOM provided by the mobile operator.

3.2 Context

Multiple service providers and service consumers are connected via the messaging middleware, and the response time of multiple services needs to be improved without changing the provider or the consumer. The pattern can be applied under the following situations:

Communication channel

The communication channel between the service provider and the consumer needs to be MOM if we want to apply the pattern.

Invocation style

The invocation style should be request and response, where one response message is generated according to one request. Other invocation styles like publish/subscribe are not applicable to this pattern.

Service provider side tolerance

Service to be accelerated can tolerate that identical requests not being delivered to it and responses can be stored in the messaging middleware. This implies several characteristics: the service is not transactional; data change frequency is not high, or data expiration time is not short; data auditing policy allows that request data need not arrive at the service provider side; data privacy policy allows data to be stored in a communication channel.

3.3 Problem

When the MOM acts as the communication channel between the service provider and the service consumer, and large amount of repeated requests are submitted, the services’ response time is decreased. Repeated requests cause unnecessary overhead to the service provider, and also repeated processing of messages in intervening message mediations in the MOM.

When the message format is complex, the problem becomes more severe, like the widely used XML message in an SOA environment. The verbose XML message payload brings extra overhead of packaging and parsing XML in the message mediations.

To address the problem the following forces needs to be resolved:

Performance: Repeated requests cause unnecessary repeated processing to both the service provider and the intervening message mediations in the MOM. The processing time of the repeated requests needs to be minimized to improve the response time and the throughput of the service.

Benefiting multiple services: Multiple services’ performance need to be improved in one solution.

Implementation Cost: The solution’s implementation cost needs to be minimized. Therefore changes to the deployed services need to be minimized and reusing the existing middleware’s capability is preferred.

3.4 Solution

In general the Cache Mediation pattern shortens the service response time by caching service response messages in MOM. When an identical request message is issued, MOM retrieves the corresponding response message from the cache and returns it directly to the service consumer, instead of forwarding the request message to the service provider. By doing so, the costly remote service invocation is eliminated and the service response time is improved.

Unlike the server-side or the client-side cache, cache mediation sits between the service provider and the consumer, and therefore it can benefit multiple service providers and consumers in one solution. The MOM brings the challenge of identifying the request and the corresponding response - which is not a problem in server side cache or client side cache. Moreover, the Cache Mediation pattern tries to leverage existing caching capability instead of building cache from scratch. These characteristics of the Cache Mediation pattern decide its structure, participants as well as the collaboration between them.

3.5 Structure

Figure 2 depicts the structure of the Cache Mediation pattern. There are five roles involved in this pattern: service provider, service consume, cache request mediation, cache response mediation, and the underlying cache. The ordinary message flow is depicted by the black line, while the accelerated message flow is shown in red line.

- § *Service provider* produces responses according to the request from the *service consumer*. The communication channel between the service provider and the service consumer is MOM.
- § *Cache request mediation* caches messages routed to the service provider.
- § *Cache response mediation* caches messages routed to the service consumer. Both mediations reside in MOM.
- § *Cache* shared by the two mediations. It stores the cached data.

The structure can be characterized as two cache mediations coupled by one cache instance. As mentioned earlier, mediation refers to the message processor in a messaging system. It performs tasks like logging, routing, message format transformation and other tasks which need to be performed on messages. The two cache mediations coupled by a single cache instance can collaboratively identify the request and the correlated response message.

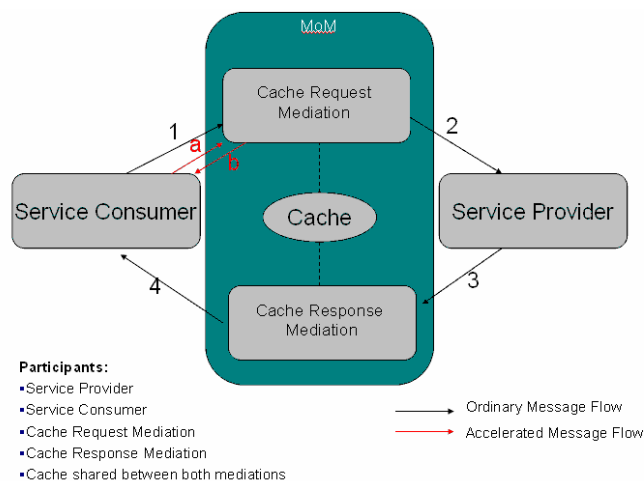


Figure 2 Cache Mediation Pattern Structure

3.6 Dynamics

Collaboration among the pattern participants is illustrated in Figure 3. The interaction between the service provider, cache mediations, and the service consumer is as follows. Firstly, the service consumer sends a request message to the service provider. MOM routes the message to the destination of the service provider. When the message arrives at the destination, the cache request mediation attached to the destination can access the message content and perform some actions on the message. The cache request mediation generates a cache key based on the request message, and searches for the response in the cache. If a matching response is found, the cache

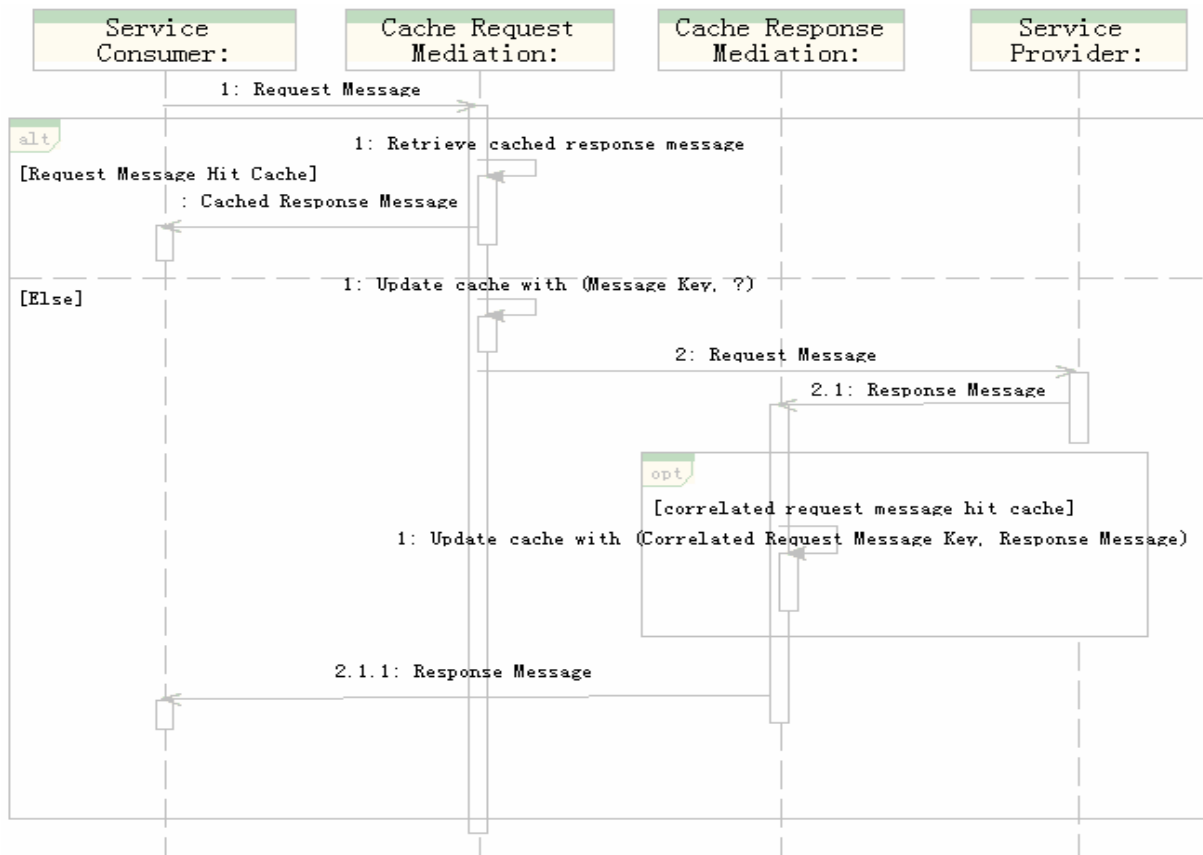


Figure 3 Behavior of Participants in Cache Mediation Pattern

request mediation returns the response message to the service consumer immediately. Otherwise, the request message will be sent to the service provider and a new cache entry, with the cache key generated, will be inserted into the cache.

When the service provider returns the response message and MOM routes the response to the service consumer, the cache response mediation is invoked. It completes the cache entry created by the cache request mediation by identifying the correlated request message's cache key in the cache, and completing the cache item with the response message.

3.7 Implementation

The implementation of the Cache Mediation pattern needs to follow the steps introduced below:

1. Choose the underlying cache and determine the cache interface

Cache Mediation pattern leverages existing caches rather than building cache from scratch to save the implementation cost. Different caches have different APIs to access its content. We need to determine what specific APIs to use. Usually these APIs includes method like *get()*, *put()* and *hitCache()*. These APIs provide basic caching capabilities: inserting a new cache entry into the cache, getting a cache item according to a specific cache key, and determining if the cache has the corresponding entry given a cache key.

For example, if we choose to use WebSphere's DynaCache as the underlying cache, it provides an interface class to access its contents:

```
class com.ibm.websphere.cache.DistributedMap implements java.util.Map
```

And its instance can be retrieved by a JDNI name specified by the cache configuration file. The *get()*, and *put()* operations should be used in the implementation.

2. Determine cache item identification policy

Cache policy specifies various configurable behaviors of the cache. Here we focus on the cache item identification policy. Other policies like data invalidation policy, synchronization policy, eviction policy are beyond the scope of this paper and not discussed here.

Cache item identification policy describes how a cache item is identified, or how the cache key is generated. Specifically for the Cache Mediation pattern, this policy determines how to decide the two requests are identical. The default policy could combine the information of the service endpoint, the requested operation, and the parameters in the request message to generate the cache item ID, as illustrated in the code snippet in step 4. Users can also specify other identification methods. For example, use the entire request message payload to generate a cache key; or only use the service endpoint and the requested operation to identify request messages.

3. Implement cache key generation method

In the cache entry (cache key, cache item), the cache key is used to identify the request message, and the cache item is usually the response message.

After the cache item identification policy is determined in step 2, developers need to implement the *generateCacheKey()* operation.

For example, if Service Integration Bus (SIBus) is used as the MOM, and in step 2 we use the combination of the service endpoint, the requested operation as well as the parameters as the cache item ID, the message destination / service endpoint address, the operation invoked, and the operation parameters need to be retrieved from the SIBus messages, which in turn to generate the cache key as shown in the following code snippet.

```
public Object generateCacheKey(SIMessage message){
    String serviceEndpoint = message.getSession().getBusName() +
        message.getSession().getDesinationName() +
        message.getSession().getMediationName();
    DataObject rootObject = message.getDataGraph().getRootObject();
    String operationName = OutputHelper.print(rootObject.getDataObject("info/body").
        getDataObject("parameters"));
    String reqPara = OutputHelper.print(rootObject.getString("parameter"));
    String cacheKeyContent = serviceEndpoint+ operationName+ reqPara;
    String cachekey = cacheKeyContent.hash();
    return cachekey;
}
```

4. Implement CacheRequestMediation class and CacheResponsetMediation class

Cache Request Mediation and *Cache Response Mediation* are two key participants in the pattern. These two mediations are coupled by one underlying cache instance at run time. And they collaboratively achieve the pattern's dynamic behavior described in section 3.6. They use the underlying cache selected in step 1 to store cached responses.

Cache Request Mediation resides on the request routing path. When a request message arrives, it is responsible to search the corresponding response from the underlying cache. If the corresponding response is found, *Cache Request Mediation* returns the cached response directly to the service consumer, otherwise the request message is sent to the service provider and a supportive cache entry (*correlationId*, cache key) is added to the cache. *CorrelationId* is the linkage between a pair of request and response message and it is used to help *Cache Response Mediation* to find the key of its correlated request message. Here *correlationId* needs to be retrieved from the messaging middleware.

Cache Response Mediation resides on the response routing path. When a response message arrives, it is responsible to refresh the cache entry of the corresponding request message. Two steps are needed. First it gets the cache key from the supportive cache entry (*correlationId*, cache key) according to the *correlationId*. Then it inserts a cache entry (cache key, cache item) where the cache item is generated from the response message.

In SIBus, the *correlationId* can be obtained from the SIBus API *getCorrelationId()*. And in both two mediation classes, the processing logic is implemented in the *handle()* operation as the following code snippet:

```
public class CacheRequestMediation {
    DistributedMap cache; // the interface class to access underlying DynaCache
    public void handle(SIMessage message){
        Object cacheKey = generateCacheKey(message);
        Object responseMsg = this.cache.get(cacheKey);
    }
}
```

```

        if (responseMsg != null){
            sendResponse(responseMsg);
        }else{
            Object correlationId = getCorrelationId();
            this.cache.put(correlationId, cacheKey);
            sendRequest(message);
        }
    }
    .....
}
public class CacheResponseMediation {
    DistributedMap cache; // the interface class to access underlying DynaCache
    public void handle(SIMessage message){
        Object correlationId = getCorrelationId();
        Object cacheKey = this.cache.get(correlationId);
        if (cacheKey != null)
            this.cache.put(cacheKey,message);
    }
    .....
}

```

3.8 Consequences

Behavior of MOM

The behavior of the MOM is changed. It actively changes the routing path of the request message and sends the cached response message back to the service consumer. The service provider will not receive every request message.

Actuality of responses

With the cache in the MOM, response messages may not be up to date. The data staleness introduced by the cache needs to be understood by the cache users, and can be alleviated by a cache data refresh policy.

Performance

The cache increases the responsiveness of the service by caching the request and the corresponding response messages. However, caching has its own cost. Message mediations have the extra cost like accessing the messages' payload to generate the cache key and storing cached messages.

4. Example Resolved

Consider the Weather Forecast service introduced in Section 3.1 that employs the MOM as the communication channel between the service and its consumers. Both the requests from the service consumers and the responses from the service are transferred by the MOM. Here we use WebSphere SIBus as the MOM, and the weather forecast service is deployed as a web service connected by SIBus. SIBus has a group of one or more interconnected servers or server clusters, which are called bus members. Weather Forecast web service connects to a SIBus at one of the messaging engines associated with its bus members. This messaging engine has an *Endpoint Listener*, which is used to receive the request message and return the response message.

We are trying to improve the performance of the weather forecast service when repeated requests are issued. The requirements described in the context section are satisfied in this example – communication channel is MOM; invocation style is request/response; and the weather forecast service allows responses to be stored in MOM. In addition, WebSphere has a built-in DynaCache which can reduce the solution implementation cost if it can be leveraged. Therefore we use the Cache Mediation pattern in the example solution and choose DynaCache as the underlying cache.

After the applying the Cache Mediation pattern, we add *Cache Request Mediation* and *Cache Response Mediation* to the two bus members, which reside on the request and the response routing path respectively. The two mediations share the same DynaCache instance to keep pairs of the request and the response. When a request arrives, the cache will be searched first. When an identical request message is found, SIBus retrieves the corresponding response message from DynaCache and returns it directly to the service consumer, instead of forwarding the request message to the service provider. By doing so, costly remote service invocation is eliminated and service response time is improved. When the response arrives at the cache response mediation, it is responsible to refresh the cache entry whose key is its correlated request message.

We implemented the Cache Mediation pattern in RSA using its pattern authoring and transformation technology, which are two key technologies enabling Model Driven Architecture (MDA) and Model Driven Development (MDD). Also we illustrate the usage of the Cache Mediation pattern in RSA for the problem in the Weather Forecast service.

In RSA a pattern is a parameterized representation of a solution to a problem. When different input parameter values are set, different specific solution, will be expanded from a general solution. Transformation is used to automatically generate various software artifacts from the UML models. As illustrated in Figure 4, when we apply the pattern in RSA, the first step is instantiating the Cache Mediation pattern, and applying the pattern to a UML model composed of a service client and a service, connected by an association class of “MOM Mediation”. This UML model is the service invocation model in the MOM context. Then we apply the UML model transformation to generate a deployment model; and in the last step, we apply UML to Java transformation to the deployment model to generate the runtime artifacts like JAVA code of the cache mediations and the deployment scripts.

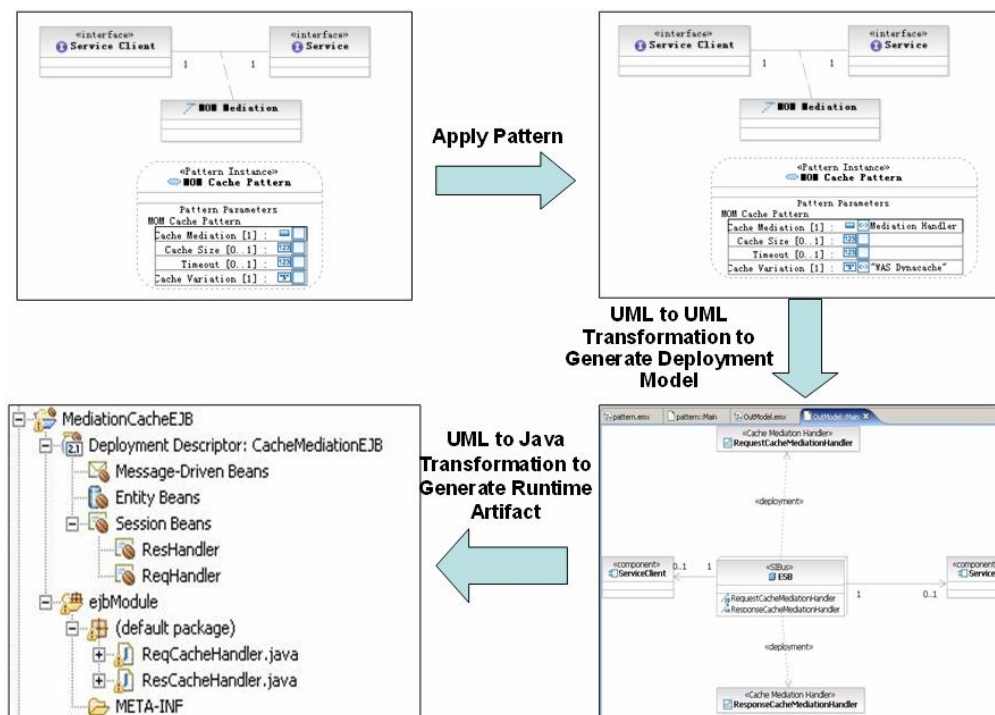


Figure 4 Pattern Application and Transformation in Rational Software Architect

5. Known Uses

Cache Mediation pattern is a specialized cache scheme in a middleware enabled environment. It sits between server applications and its clients, adding a mediation layer on top of the cache. The mediation builds intelligence into the cache scheme through custom definable correlation function between requests and responses and enables populating the underlying cache in two steps. The two mediations in the Cache Mediation pattern collectively achieve such effect. General Caching pattern like proposed in [13] can guide users on building the basic cache capability, while the Cache Mediation pattern can help users build the intelligent mediation layers on top of the underlying cache.

We also observe such intelligent cache mediations in some publications and products.

- 1 | **Pronto.** Pronto[15] is a middleware system for mobile applications with messaging as a basis. It is an intelligent MobileGateway providing a useful MOM intermediary between a server and mobile devices over a wireless network. Pronto uses SmartCaching as a plug-in component to improve the latency by reducing the network traffic. In Pronto, JMS messages are cached in SmartCaching. The cached data is stored for reuse without the need for revisiting the server and passing the data through the chain of reformatting and representation. In Pronto, CacheManager creates Cache objects and manages requests and

responses to the requesters. MobileGateway acts as MOM in Cache Mediation pattern, and CacheManager behaves as the cache mediations in the Cache Mediation pattern.

- | **Enterprise Service Bus.** Enterprise Service Bus (ESB) is the infrastructure which underpins many integrated and flexible SOA solutions [4]. Caching is one of the typical mediation capabilities in [4] to implement real SOAs. Service Integration Bus (SIBus), one member of IBM's suite of ESB product line, uses dynamic cache to mediate the pairs of request and response messages. Another product important to ESB, WebSphere MQ [16] can leverage dynamic cache or disk storages defined by users to cache the pairs of request and response messages. SIBus provides *getCorrelationId()* to get the correlation between request and response. WebSphere MQ has the same function by getting "message id" and "correlation id" in MQ messages with *getMQMDE()* operation. As practiced in the product experiment, the *CacheMediation* component in SIBus and the *channel* component in WebSphere MQ are used to access the cache as cache mediations in the Cache Mediation pattern, with the intelligence of caching correlated messages.
- | **reAgents.** reAgents [17] is a middleware solution for web applications. Both the web servers and clients are linked to reAgents, which is interposed in their communications. reAgents act like MOM in the Cache Mediation pattern. The *Cacher* component in reAgents is used to save commonly-accessed data at a location near the client to improve responsiveness when there is high network latency between the client and the server. The details of how to identify cache hitting are not found in published documents. But this can be done with intelligent programming by developers. Then reusable data can be retrieved from the cache directly without accessing the web server again. The *Cacher* component behaves as the cache request and response mediations in the Cache Mediation pattern.
- | **JBoss Messaging and JBoss Cache.** JBoss Messaging [18] is a distributed and reliable message transport systems, which is implemented on JBoss server. The component *channel* is the delivery mechanism that forward messages from clients to JMS providers synchronously or asynchronously. JBoss Cache [19] is deployed as an MBean inside JBoss server, which can be accessed and set caching policy by JBoss APIs. In some real cases, we observed that JBoss Cache was used in JBoss Messaging by customer programming, which help to save response data and replicate data from JMS providers regularly. In these cases, JBoss Cache provides cache object and the component *channel* in JBoss Messaging acts as the cache mediation in Cache Mediation Pattern.

6. Conclusions and Future Work

This paper proposes a Cache Mediation pattern as a new pattern especially useful in an SOA environment. While leveraging existing cache capabilities provided by the messaging middleware or other cache solutions, it is a specialized cache scheme in a messaging middleware enabled environment by adding a mediation layer on top of the underlying cache. Residing between the service providers and the service consumers, it can benefit multiple service providers and service consumers in one solution. Its implementation and usage in Rational Software Architect are also introduced. We found it is especially useful when today's practitioners usually develop software based on existing middleware and use tools extensively.

Following are the possible enhancements to the current Cache Mediation pattern and its implementation in RSA:

Cache Policy Guidance.

Besides cache item identification policy discussed in this paper, cache policy includes various aspects like invalidation policy, persistency policy, cache item eviction policy, etc. Guidance on these cache policies can be provided to help users choose a suitable cache policy in different specific environments.

Variation Accommodation.

To provide a general solution to a reoccurring problem, pattern authors need to consider the support of many variations in different contexts. In the case of the Cache Mediation pattern, style of access, item identification, cache populating mechanism, and other variations can be further designed.

7. Acknowledgement

Special thanks to our EuroPLoP 2006 shepherd Michael Kircher for his constructive comments and guidance.

References

1. Alan Brown, Simon Johnston, and Kevin Kelly. Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications. IBM developerWorks, 2003. <http://www-128.ibm.com/developerworks/rational/library/510.html>
2. Message-Oriented Middleware. http://www.sei.cmu.edu/str/descriptions/momt_body.html

3. Enterprise Service Bus.
<http://www-306.ibm.com/software/info1/websphere/index.jsp?tab=landings/esbbenefits>
4. M.-T. Schmidt, B. Hutchison, P. Lambros, and R. Phippen. The Enterprise Service Bus: Making service-oriented architecture real. IBM Systems Journal, VOL 44, NO 4, 2005.
5. Kunal Mittal. Introducing IBM Rational Software Architect - Gain an edge in design pattern-based development. IBM developerWorks, 2005.
http://www-128.ibm.com/developerworks/rational/library/05/524_rsa/
6. Peter Swithinbank, Mandy Chessell, Tracy Gardner, Catherine Griffin, Jessica Man, Helen Wylie, and Larry Yusuf. Patterns: Model-Driven Development Using IBM Rational Software Architect. IBM Redbook, 2005.
7. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.
8. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. Pattern-Oriented Software Architecture — A System of Patterns. John Wiley & Sons, 1996.
9. Jonathan Adams, Srinivas Koushik, Guru Vasudeva and George Galambos. Patterns for e-Business: A Strategy for Reuse. MC Press, 2001
10. JbossCache. <http://www.jboss.org/products/jboss-cache>
11. SpiritCache. <http://www.spiritsoft.com/products/cache/introducing.shtml>
12. R. Bakalova, A. Chow, C. Fricano, P. Jain, N. Kodali, D. Poirier, S. Sankaran, and D. Shupp. WebSphere Dynamic Cache: Improving J2EE application performance. IBM Systems Journal, Vol 43, No 2, 2004
13. Michael Kircher, Prashant Jain. Pattern-Oriented Software Architecture—Patterns for Resource Management, Volume 3. John Wiley & Sons, 2004.
14. Fangyan Rao, Ru Fang, Zhong Tian, Harini Srinivasan, Eoin Lane, Lei He and Tim Banks, Message Oriented Middleware Cache Pattern - a Pattern in an SOA Environment. 4th “Killer Example” for Design Patterns and Objects First workshop,
<http://www.cse.buffalo.edu/faculty/alphonse/KillerExamples/OOPSLA2005/>
15. Eiko Yoneki, Jean Bacon. Pronto: MobileGateway with publish-subscribe paradigm over wireless network. University of Cambridge Technical Report, UCAM-CL-TR-559
16. IBM WebSphere MQ. <http://www-306.ibm.com/software/integration/wmq/>
17. Eugene Hung and Joseph Pasquale. Web Customization Using Behavior-Based Remote Executing Agents. In Proceedings of WWW2004, New York, USA
18. JBoss Messaging. <http://www.jboss.com/products/messaging>
19. JBoss Cache. <http://www.jboss.com/products/jboss-cache>