

Message Oriented Middleware Cache Pattern – a Pattern in a SOA Environment

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

Harini Srinivasan, Eoin Lane, Lei He,

Tim Banks

IBM Software Group

{harini, eoinlane}@us.ibm.com

heleihl@cn.ibm.com

tim_banks@uk.ibm.com

Abstract

Caching is a widely used concept in many applications and systems to improve performance. This paper discusses a Message-Oriented Middleware (MOM) Cache pattern as a reusable solution to accelerate service response in a Service Oriented Architecture (SOA) environment. As an important communication channel in distributed computing environments like SOA, Message-Oriented Middleware (MOM) increases the interoperability, portability, and flexibility of the applications [Rao]. A MOM cache is a cache residing on the MOM between the service providers and the service consumers. Unlike traditional design patterns, this pattern is more on the infrastructure level and leverages existing capability provided by MOM. Moreover, the pattern is implemented using 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. One major concern for SOA is performance. Reducing the response time of costly remote service invocations in such environments is a critical challenge in many real life cases. Caching, a widely used technology to improve performance, has been used in many solutions successfully to address the issue. This paper discusses a Message-Oriented Middleware (MOM) Cache pattern as a reusable solution to accelerate service response in a SOA environment. As a communication channel in the distributed computing environments like SOA, MOM normally provides a message queue between the service provider and the service consumer to transfer messages. MOM is typically asynchronous, but it also supports synchronous message passing as well [MOM]. MOM is the foundation of Enterprise Service Bus (ESB) [ESB], which is widely believed to be a cornerstone of many SOA environments. In this paper, we introduce a MOM Cache pattern based on Enterprise Service Bus (ESB) cache pattern proposed in [Hutchison et al.]. The cache concepts here are easy to understand for a senior college student. Getting exposed to a hot architecture style (SOA) and a hot family of products (MOM) should provide enough motivation to encourage the students to get through the implementation and deployment details.

Asynchronous messaging is widely recognized as an effective communication channel in an SOA environment where service provider and consumer may be separated across the Internet [Brown et al.]. Unlike traditional cache on server side or client side, MOM Cache resides in the middle between service consumers and service providers. Therefore providing a cache solution in MOM can serve the purpose of accelerating service requests from multiple service consumers residing on different locations, and also benefit multiple service providers. Meanwhile it is an under-utilized technology because Web Services is a relatively new technology. Using the MOM Cache pattern in the messaging middleware to address performance issue in SOA environment is also a relatively new practice.

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 in giving the students a realistic setting where 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 cache capability rather than from scratch. Furthermore, tools like Rational Software Architect (RSA) provide strong tooling support to define and apply patterns [Mittal]. We implemented the pattern in RSA using its pattern authoring and transformation framework 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 MOM Cache pattern, Section 4 illustrates how the pattern implementation and application in Rational Software Architect, and Section 5 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 [Gamma et al.], to system-level patterns providing solution at macro-design of system components [Buchmann et al.], to patterns for e-business as described in [Adams et al.] 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 MOM Cache pattern is an integration pattern that provides a cache solution in a SOA environment at the middleware layer.

In an SOA environment, service providers and requesters are loosely coupled and distributed across network, either within an organization or across organization boundaries. The performance issue of services raises more concerns than traditional distributed systems as SOA solutions are designed more for interoperability than for performance. For example, XML is a widely used message format for service providers and consumers in SOA. XML message packaging and parsing brings extra overhead to both ends. Therefore a web service call costs more in terms of response time than some other kinds of remote procedure invocation. Caching is a widely used technology to improve application performance. Various caches have been designed and implemented to improve responsiveness of application in various software products, like JbossCache [JBoss], SpiritCache [Spirit], Websphere Application Server DynaCache [Bakalova et al.]. Using cache to reduce service performance cost is a problem of interest to cache technology practitioners. Some cache patterns are proposed to provide reusable cache solution.

[Hutchison et al.] propose an Enterprise Service Bus (ESB) cache pattern to accelerate service response time in the ESB context (错误! 未找到引用源。). The ESB is the messaging backbone in the SOA environment. It handles message transfer and provides various mediation capabilities like message transformation, logging, routing, etc. In the ESB context, both service provider and consumer are attached to the Enterprise Service Bus. Message mediation resides on the service bus to handle request or response messages. In this caching scenario, when 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 forward the request to the service provider which in turn generates response which is routed back to the message requester. Meanwhile response messages will be cached by the cache mediation. Unlike various server side cache frameworks like JbossCache [JBoss] and SpiritCache [Spirit], an ESB cache accelerates the 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.

In this paper, we flesh out the ESB pattern and generalize it as the MOM Cache pattern with detailed descriptions of its context, structure, consequences, etc. We also illustrate the application of the pattern in the RSA tool.

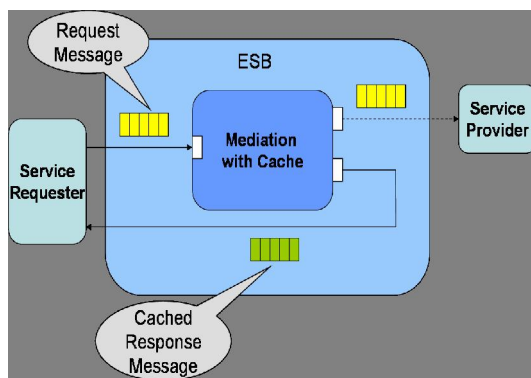


Figure 1 ESB Cache Pattern. Revised from [Hutchison et al]

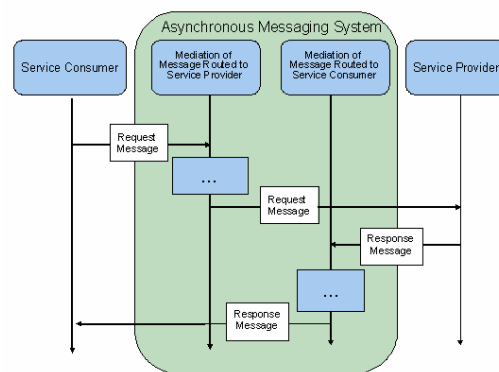


Figure 2 Behavior of Mediations in Asynchronous Messaging System

3 MOM Cache pattern

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

3.1 Motivating Problem

In an SOA environment, MOM provides an integration layer between multiple service providers and multiple service consumers. The MOM, in addition to routing the message requests and responses to and from service providers, also support mediation functionality such as data transformations, enrichment, caching etc. The MOM Cache pattern targets to help developers build caching in mediation subsystem.

3.2 Context

In a messaging environment, the request message and response message are delivered in the messaging middleware independently – the processors of request and response message, or message mediations work independently and asynchronously. Figure 2 depicts the behavior of mediations in messaging system. In this environment, message mediations work in a message-driven way. Request messages and response messages trigger request message mediation and response message mediation respectively. For example, in the SIBus implementation of ESB in the Websphere Application Server [Whyley], message mediation handlers are implemented as EJB beans [SUN]. Individual message mediation can not handle both request message and response message during one execution lifecycle. This particular behavior needs special consideration for cache design.

3.3 Solution to the Problem

The solution is to shorten service response time by caching service response messages in messaging middleware. When an identical request message is issued, the message system retrieves the corresponding response message from cache and returns it directly to 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.

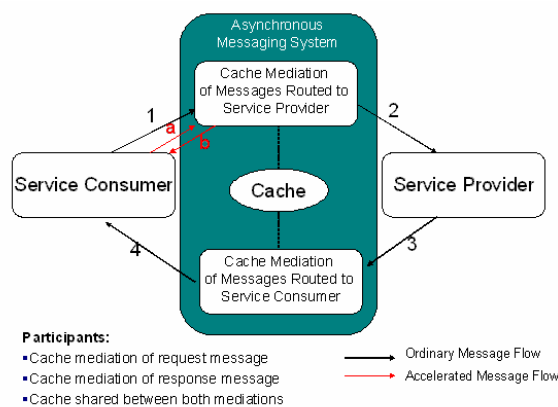


Figure 3 MOM Cache pattern Structure

3.4 Structure – Two Cache Mediations Coupled by One Cache

Figure 3 depicts the structure of the MOM Cache pattern. 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 each message. The rationale behind such a design is based on the behavior of mediations described earlier.

3.5 Collaboration

Collaboration among the pattern participants is described in Figure 4. The interaction between service provider, cache mediations, and service consumer is as follows. Firstly, a service consumer sends a request message to a service provider. MOM routes the message to the destination of 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 a response in the cache.

If a matching response is found, the cache 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 the 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 updating the cache item with response message.

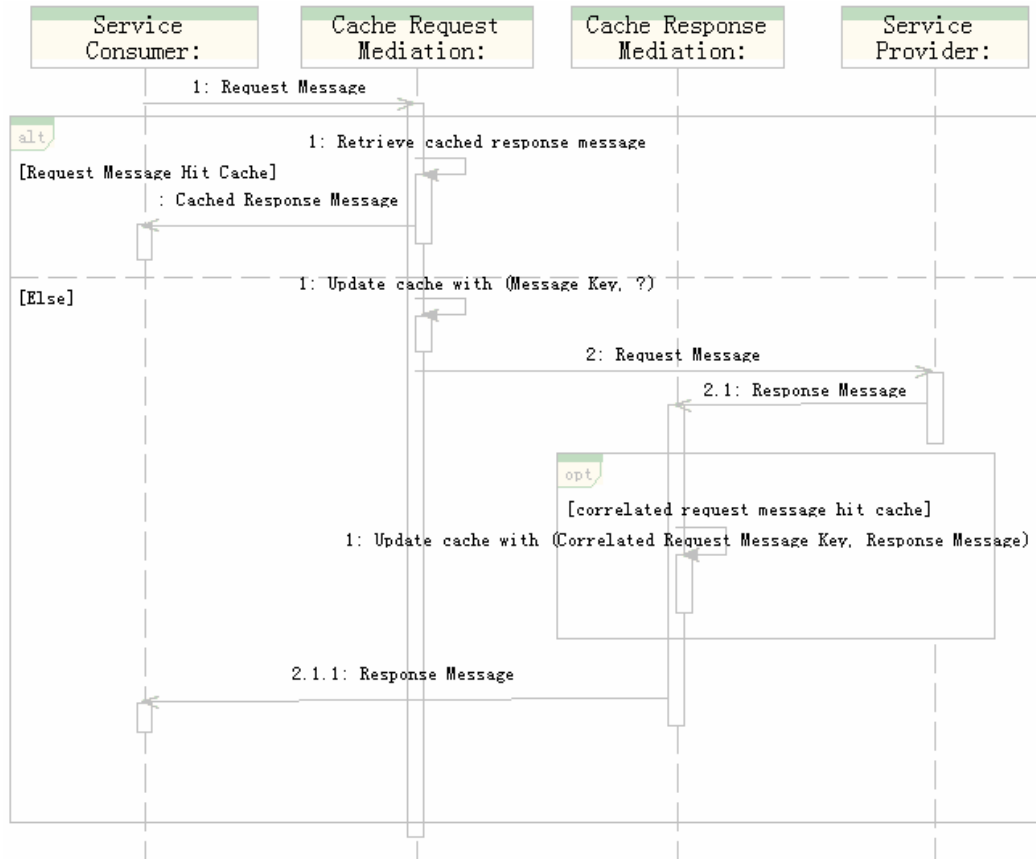


Figure 4 Behavior of Participants in MOM Cache pattern

3.6 Cache Contents

A cache entry is organized as a duple of <cache key, cache item>. Cache key is utilized to identify request message. Here, identical messages mean that two request messages are equal from the perspective of the cache user. For example, the cache key can be composed from the requester message destination / service endpoint address, the service operation, and the operation parameters. It is desirable to use the underlying message identification method provided by the messaging system, if it provides such capability. Fast cache key generation is critical for cache performance.

The cache item is composed from the response message.

Another supportive cache entry is created as <message ID, cache key>. It is used by the cache response mediation to find a cache key of its correlated request message.

3.7 Cache Policy

Cache policy specifies various configurable behaviors of cache. Here we focus on two kinds of cache policies – the cache item identification policy, and the invalidation policy.

Cache item identification policy describes how a cache item is identified, or how the cache key is generated. The default policy for cache item identification is the combination of service endpoint, operation, and parameters to identify the request message. Users should also be able to specify other identification methods. For example, user can specify using the entire request message payload to generate cache key; or only using service endpoint and operation to identify cache item.

Invalidation policy describes how the data in cache is invalidated. Cached data gets stale as the data source change over time. Therefore, we need to invalidate the cache data so that the cache can refresh its contents. For example, two kinds of invalidation policy are supported in Websphere Application Server's DynaCache – time based invalidation and programmatic active invalidation. For time based invalidation, a timeout value is set for each cache entry. When the cache entry's lifetime exceeds the timeout value, cache will automatically invalidate the cache entry. The timeout value should be set as less than average data change time. For programmatic active invalidation, applications being accelerated can actively invalidate the cache data when data changes are detected.

Cache policy heavily depends on the underlying cache mechanism used in the cache pattern. For different caches, a different cache policy may be provided.

3.8 Applicability

Communication channel

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

Cache location

The cache is associated with the cache mediation that resides in MOM. Caches on the service provider side or the service requester side are not applicable for this pattern.

Invocation style

To apply the pattern, 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

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

3.9 Consequences

Behavior

The behavior of the messaging middleware is changed. It actively changes the routing path of request message and sends cached response message back to the service consumer.

The service provider will not receive every request messages.

Data

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

Performance

The cache increases the responsiveness of the service by caching request and corresponding response messages. However, caching has its cost. Message mediations need extra cost to access the payload of messages and store cached messages.

3.10 Dependency

Message correlation

MOM needs to provide message correlation mechanism to correlate request and corresponding response messages

Correlation can be either automatically done by messaging middleware, or by the service provider with correlation APIs provided by MOM.

Message identification

Message middleware can provide faster message identification options so that pattern users can choose appropriate message identification for cache item identification key generation and comparison.

Underlying cache

The MOM Cache pattern is not a design pattern for general purpose cache design, it leverages existing basic cache capability provided in MOM or other cache system. In our referencing implementation, we are using the Websphere Application Server's DynaCache as the underlying cache for this pattern.

4 Pattern Implementation and Application in Rational Software Architect

The pattern application in RSA is illustrated in Figure 5. We implemented the MOM Cache pattern in RSA using its pattern authoring and transformation technology. The pattern is a parameterized representation of a solution to a problem in RSA. When different input parameter's value is set, different specific solution is expanded from general solution. Transformation is the automatic generation of various software artifacts from UML models. Patterns and transformations are two key technologies enabling Model Driven Architecture (MDA) and Model Driven Development (MDD). When users need to apply the pattern, it needs to be instantiated first. A UML model is then created with service client and service, as well as an association class of mediation handler. To apply the pattern on the model, we need to associate mediation handler with one cache pattern parameter to specify the application target. After instantiating the pattern parameters, we can apply the UML model transformation to generate a deployment model for the MOM Cache, here specifically Websphere Application Server(WAS); and transform from the deployment model to runtime artifacts like EJB beans and deployment scripts. After these transformations, the cache code can be automatically generated and deployed on WAS, which saves a lot of effort for cache users.

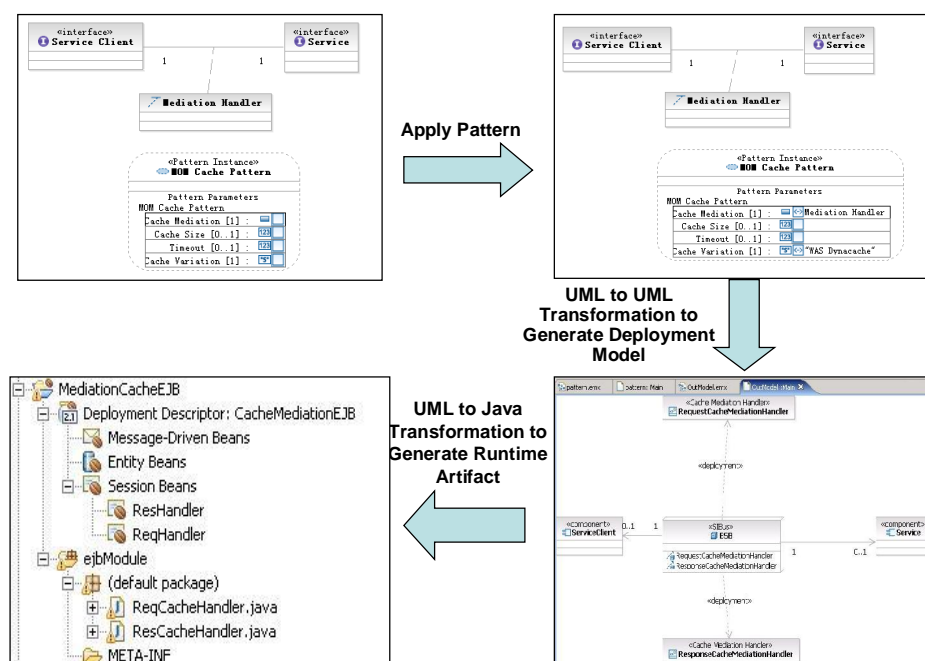


Figure 5 Pattern Application and Transformation in Rational Software Architect

5 Conclusions and Future Work

A pattern is a solution at a certain level of abstraction intended to solve reoccurring problems in different contexts. This paper proposes the MOM Cache pattern as a killer example for patterns in an SOA environment. Also it introduces its implementation in RSA. This pattern can be used as a challenge to senior college student, or term assignment for 2nd year graduate students to set a realistic pattern design, implementation and application.

The following are ideas for further enhancements to the current MOM Cache pattern and its implementation:

Cache Policy Guidance.

Besides cache invalidation policy and cache item identification policy discussed earlier, cache policy includes various aspects like 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.

As a general solution to a reoccurring problem, how to design a pattern to support many variations in different contexts needs to be considered for pattern authors. In the case of the MOM Cache pattern, style of access, item identification, cache populating mechanism, and other variations can be further be designed. Accommodation of other customer cache implementations like JBossCache can also be considered.

References

- [Adams et al.] Jonathan Adams, Srinivas Koushik, Guru Vasudeva and George Galambos, Patterns for e-Business: A Strategy for Reuse, MC Press, 2001
- [Bakalova et al.] R. Bakalova et al, WebSphere Dynamic Cache: Improving J2EE application performance, IBM Systems Journal, Vol 43, No 2, 2004
- [Buchmann et al.] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. Pattern-Oriented Software Architecture — A System of Patterns, New York: John Wiley & Sons, 1996
- [Brown et al.] Alan Brown, Simon Johnston, and Kevin Kelly Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications, IBM, June 2003
- [ESB] Enterprise Service Bus: <http://www-306.ibm.com/software/info1/websphere/index.jsp?tab=landings/esbbenefits>
- [Gamma et al.] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, Addison-Wesley, 1995.
- [Hutchison et al.] Beth Hutchison, Peter Lambros, Rob Phippen, Marc-Thomas Schmidt, Enterprise Service Bus – Making SOA Real, to appear in IBM System Journal, Nov 2005
- [Jboss] JBossCache: <http://www.jboss.org/products/jboss-cache>
- [JCache] JCache, the Java Temporary Caching API (JSR-107) <http://www.jcp.org/en/jsr/detail?id=107>
- [Spirit] SpiritCache: <http://www.spiritsoft.com/products/cache/introducing.shtml>
- [Keen et al.] Martin Keen, Amit Acharya, Susan Bishop, Alan Hopkins, Sven Milinski, Chris Nott, Rick Robinson, Jonathan Adams, and Paul Verschueren, Patterns: Implementing an SOA Using an Enterprise Service Bus, IBM Redbooks, 2004
- [Mittal] Kunal Mittal, Introducing IBM Rational Software Architect - Gain an edge in design pattern-based development, http://www-128.ibm.com/developerworks/rational/library/05/524_rsa/
- [MOM] Message-Oriented Middleware: <http://www.sei.cmu.edu/str/descriptions/momt.html>
- [Rao] Rao, B.R. "Making the Most of Middleware." Data Communications International 24, 12 (September 1995).
- [SUN] Enterprise JavaBeans Technology: <http://java.sun.com/products/ejb/index.jsp>
- [Whyley] Chris Whyley, Manipulate Web services messages using the Service Integration Bus, <http://www-128.ibm.com/developerworks/webservices/library/ws-mmsib1/>