

# The requester side caching pattern specification, Part 2: The requester side caching pattern implementation specification

Skill Level: Advanced

[Dr. Harini Srinivasan \(harini@us.ibm.com\)](mailto:harini@us.ibm.com)

Senior Solution Engineer

IBM

[James Conallen \(jconallen@us.ibm.com\)](mailto:jconallen@us.ibm.com)

Senior Solution Engineer

IBM

[Dr. Eoin Lane \(eoinlane@us.ibm.com\)](mailto:eoinlane@us.ibm.com)

Senior Solution Engineer

IBM

13 Mar 2008

[Part 1](#) of this article [series](#) provided an overview of the requester side caching (RSC) pattern specification, which can help you make and document design decisions around the cache and policies. In this second installment in the series, examine the requester side caching pattern implementation specification, a bridge between the human readable pattern specification from the Gang of Four and the pattern implementation that can be used in a development environment to automate the application of the pattern. From this implementation specification, you have the freedom to create numerous implementations. Find out how in this article.

## Introduction

The [first article in this series](#) detailed the pattern specification of the RSC pattern. You can now create a number of implementations of the RSC pattern based on the pattern specification. The following three examples showcase these different ways:

- A manual implementation (hand coding) that uniquely tailors the design and implementation for each situation.
- An executable program that automates the application of the pattern in the context of a larger model-driven-based development project (for example, in the context of IBM® Rational® Software Architect).
- An aspect-based implementation. You can use aspect-oriented programming to implement the caching as a cross-cutting concern and as a simple and useful mechanism for developing applications. This approach works if aspect-oriented expertise is already available in the organization.

However, before authoring a pattern, you must create a pattern implementation specification. First take a look at some background information.

## Background

The RSC pattern mediates the interaction between one or more clients and one or more data providers. The mediation consists of holding data items that have been produced by the provider(s) and using them to support requests from the client(s). The purpose of the mediation is to speed up the data, reduce the cost of access to the data, or both. This is a very general pattern that has many variations to meet different design goals and issues. The pattern should make it easy for you to make design decisions and to provide a basis for documentation about design decisions made around the cache and policies.

There are several issues that must be addressed in any pattern variation, including:

- Style of access (visible to the client; hidden from the client).
- Item identification (client-provided domain key; explicit, well-defined key space; computed from data items; and so on).
- Populating the cache with data items (on cache misses; on cache misses but with some anticipatory population; fully prepopulated)
- Cache capacity and capacity management (unlimited with respect to the set of data items; easily exceeds to working set of the clients; marginal with respect to the working set of clients).
- Tolerance for data staleness (must be exact; can be slightly out of date; accuracy of the data is a performance concern, not a logical concern).
- Management of data volatility (unchanging data; rarely changing data; rapidly changing data).

- Topology of deployment (single instance; multiple coordinated instances; multiple independent instances).

The RSC pattern facilitates accelerating service operations via caching. The cache used can be an in-memory, custom cache provided by the pattern itself or the IBM WebSphere® run time's caching mechanism (that is, Dynacache). The runtime cache can be in memory or distributed. One of the goals of this pattern is to make it easy for you to programmatically use the WebSphere runtime caching mechanisms.

This pattern currently assumes that the service operations exposed to the requester include methods for:

- Retrieving objects from the service provider based on a selection criteria, `getItems(criteria)`.
- Retrieving the keys that uniquely identify the objects based on selection criteria, `getItemKeys(criteria)`.
- Retrieving a single object based on a key value, `getItem(key)`.
- Making changes to objects based on key value, `changeItem(..., key, ...)`.

If operations such as `getItemKeys(criteria)` and `changeItem(key, ...)` aren't available, one suggestion is to have some kind of a facade that sits between this caching pattern and the service interface where these operations are inserted.

## Requirements

Here's a list of design requirement for the RSC pattern:

- The implementation generates UML model elements as a result of applying the pattern.
- The generated source code should be error free and capable of being executed. Exceptions may be thrown if code marked up with a mandatory `TODO` hasn't been updated accordingly by the user.
- This pattern must allow for at least two choices in the caching implementation.
- The pattern should also allow for cache configurations that include cache population, management of data volatility, cache capacity management, and item identification.
- The pattern should make it easy for you to make design decisions and to provide a basis for documentation about design decisions made around

the cache and policies. The transformation is also constrained by the caching implementations provided by the WebSphere platform, but you also provide custom caching skeleton code.

## Design constraints

The design and implementation of this pattern is constrained by the pattern and transformation authoring functions of Rational Software Architect 6.0 and later and 7.0 and later.

## Design artifacts

Note the following design artifacts:

- **Pattern name:** Requester side cache.
- **Overview diagram:** The pattern UI represents this pattern with an overview diagram (see the [previous article](#) in this series for a pattern specification for overview diagram).
- **Group:** The pattern is defined in the SOA Patterns group.
- **Pattern type:** An instance of this pattern is of the UML2 element type collaboration.
- **Short description:** The RSC pattern provides an accelerated implementation of operations that retrieve information from a service provider. The pattern provides options for cache implementations and caching policies.
- **Search keywords:** The following keywords are defined for the pattern to assist the client with locating the desired pattern:
  - SOA Patterns
  - Caching
  - Service requester

### Pattern parameters

Table 1 shows the pattern parameters that may need to be bound at run time.

**Table 1. Pattern parameters that may need to be bound at run time**

	Short description	Type	Multiplicity	Parameter dependency	Default value	Keyword
<b>Service</b>	The	Interface/Class		None	None	None

	interface/class that contains the operation you want to accelerate.					
<b>getItem</b>	The operation on the service interface/class that's used to get a single item given an item key.	Operation	1	Depends on service	None	None
<b>getItemKeys</b>	The operation on the service interface/class that's used to get keys given a set of criteria. Note that this operation is essential to be able to implement the accelerated version of the service.	Operation	1	Depends on Service	None	None
<b>getItems</b>	The operation on the service interface/class that's used to get items given a set of criteria.	Operation	1	Depends on service, getItemKeys, and getItem	None	None
<b>changeItemKey</b>	The parameter in the change item operation that corresponds to the key of the item.	Parameter	0..* (If change item is specified, then this parameter is required.)	None	None	None
<b>Cache size</b>	The size of the cache.	LiteralInteger	0..1	None	500	None

	The size of the cache can be specified as an integer or the character * to denote an unlimited cache.				
<b>Clustering</b>	Boolean value: True implies the underlying topology is clustered; false implies the underlying topology is not clustered.	LiteralBoolean1	None	True	None
<b>timeOut</b>	The timeout value (measured in milliseconds) after which an item has to be evicted from the cache. If this parameter isn't specified a time out, eviction policy isn't used.	LiteralInteger 0..1	None	-1	None

The derived parameters from the pattern parameters in Table 1 are the:

- `ItemKey` class that can be derived as the argument of the `getItem` operation.
- `Item` class that can be derived from the return parameter of the `getItem` operation.
- `changeItem( )` method that can be inferred from the `changeItemKey` (this is also its owning operation).

## Use cases

The actions denoted below occur only at the time the argument is supplied to the pattern parameter. After that the argument may change irrespective of the constraints of the pattern definition.

### Pattern interactions

The following lists parameters that can be bound when the pattern is applied.

Service parameter:

- Supply an argument to the parameter:
  - Add the *Accelerated* keyword to the argument if the keyword doesn't exist.
  - Create a new class in the same package as the argument.
  - The name of the generated class is going to be the name of the argument prepended with the string *Accelerated*.
  - The generated class realizes the argument if the parameter is an interface.
  - The generated class provides implementations for all of the argument's operations if the parameter is an interface. If the parameter is a class, the generated class simply inherits the operations.
  - The generated class provides implementations for the constructor (where *Service* is the name of the argument),  
`AcceleratedService ( service : Service )`.
  - The generated class has a directed private association to the argument called *service*.
  - The generated class has a public getter method for the instance member *service*.
  - The generated class has a directed private association to a `Map` interface.
  - The generated class has a public getter method for the instance member `Map`.
  - If the clustering parameter is set to false, the constructor creates a cache object of type `Cache`, which is an implementation of the `Map` interface, which supports `get`, `put`, `remove`, and `clear` operations.

- If the clustering parameter is set to true, the constructor (`AcceleratedService`) gets a cache object of type `DistributedCache`, which is an implementation of the `Map` interface. The operations of this class are again the same as declared in `Map`.
- Supply the default argument to the parameter:
  - N/A
- Remove an argument from the parameter:
  - Undo what you did before; in particular, remove the newly created class and the association to the `IService` parameter. But don't remove the `Map` interface if it's being referenced by any other modeling element.
- Replace the parameter argument:
  - Default to the framework semantics of removing the current argument and supplying the new argument.
- Pattern reapply:
  - Default to the framework semantics of removing the current argument and supplying the new argument.
- `getItem` parameter:
  - This argument is only referenced during the UML-to-Java™ code transformation.

`getItems` parameter:

- This argument is only referenced during the UML-to-Java code transformation.

`changeItemKey` parameter:

- This argument is only referenced during the UML-to-Java code transformation.

Cache size parameter:

- This argument is only referenced during the UML-to-Java code transformation.

Clustering parameter:

- This argument is only referenced during the UML-to-Java code transformation.

timeOut parameter:

- This argument is only referenced during the UML-to-Java code transformation.

## UML-to-Java transformation

The transformation implemented for this pattern extends or reuses the UML to Java transformation.

### Service

The `AcceleratedService` UML class created during pattern instantiation expands to a Java class that implements the `Service` interface. The constructor of this class takes a `service` of type `Service` as the argument. The class also has a private member `service` of type `Service` that's instantiated by the constructor to the slow original service implementation.

Another member of the `AcceleratedService` class is the `cache` of type `Map`. The operations generated in `AcceleratedService` are operations defined in `IService`. In addition, you have a method to retrieve the cache, `getCache()`, that returns the cache implementation object.

### getItem

The `getItem` pattern parameter expands into an operation in the `AcceleratedService` class that mirrors its name and signature. This operation takes a `key` as input and returns an `Item` as output. The implementation of this method is fully specified during the UML-to-Java transformation phase. In Listing 1, the method body contains the following variable values:

- **Item:** The return type of the original `getItem` method
- **ItemKey:** The argument (that corresponds to the key value) of the original `getItem` method in the service, which looks like this:

### Listing 1. getItem Java implementation code2

```
public Item getItem(ItemKey itemKey) {
    Item item = (Item)this.cache.get(itemKey);
    if (item == null)
        item = this.service.getItem(itemKey);
    return item;
}
```

## getItems

The `getItems` pattern parameter expands into an operation in the `AcceleratedService` class that mirrors its name and signature. The signature of this operation is arbitrary, and this can contain any number of parameters and parameter types. The return type of this operation must be a List JDK collection type. The implementation of this method is fully specified during the UML-to-Java transformation phase. In Listing 2, the method body contains the following variable values:

- **List:** The return type class.
- **arguments::** The argument list of the original `getItems` method in the service.
- **ItemKey:** The key class/type of the `Item` class. This value is identified in the argument list of the `getItem()` pattern parameter.
- **Item:** The `item` class/type. This value is identified as the return type of the `getItem()` pattern parameter.

### Listing 2. getItems Java implementation code

```
public List getItems(arguments) {
    List keys = this.getItemKeys(arguments);
    Iterator iterator = keys.iterator();
    List list = new ArrayList();
    while (iterator.hasNext()) {
        ItemKey key = (ItemKey) iterator.next();
        Item item = this.getItem(key);
        list.add(item);
    }
    return list;
}
```

## getItemKeys

The `getItemKeys` pattern parameter expands into an operation in the `AcceleratedService` class that mirrors its name and signature. The signature of this operation is arbitrary, and this can contain any number of parameters and parameter types. The return type of this operation must be a List JDK collection type. The implementation of this method is fully specified during the UML-to-Java transformation phase. In Listing 3, the method body contains the following variable

values:

- **List:** The return type class
- **arguments:** The argument list of the original `getItems` method in the service

### Listing 3. getItemKeys Java implementation code

```
public List getItemKeys(arguments) {  
    List keys =  
    this.service.getItemKeys(arguments);  
    return keys;  
}
```

Users are expected to properly select qualifiers of `getItemKeys`, and it's recommended that the `Ordered` Qualifier is selected to generate a method that expresses multiplicity of the return parameter in terms of `java.util.List`.

## changeItemKey parameter

The `changeItemKey` pattern parameter expands into a specific argument (corresponding to the key) of an operation `changeItem` in the `AcceleratedService` class. The `changeItem` operation mirrors the `changeItem` method declared in the `Service` interface. The implementation of this method is fully specified during the UML-to-Java transformation phase. Listing 4 contains the following variables:

- **Arguments1:** Zero or more arguments of the original `changeItem` method that precede the `changeItemKey` argument
- **changeItemKey:** The argument that corresponds to the `changeItemKey` parameter
- **Arguments2:** Zero or more arguments of the original `changeItem` method that follow the `changeItemKey` argument

### Listing 4. changeItem Java implementation code

```
public changeItem(Arguments1, changeItemKey,  
Arguments2) {  
    this.service.changeItem(Arguments1,  
changeItemKey, Arguments2);  
    this.cache.remove(changeItemKey);  
}
```

## Clustering parameter

As explained earlier, the clustering parameter effects the kind of cache implementation used by the `AcceleratedService` class. Based on the value of clustering, the implementation of `Map` is either an in-memory cache, (that is, class `Cache` if clustering is false or class `DistributedCache` if clustering is true). If clustering is true, the `getCache()` method of the `AcceleratedService` class looks like Listing 5.

### Listing 5. `getCache` implementation code

```
public Map getCache() {
    com.ibm.websphere.cache.DistributedMap map =
    null;
    try {
        javax.naming.InitialContext ic = new
        javax.naming.InitialContext();
        Object obj =
        ic.lookup("services/cache/EmployeeCache");
        if (obj != null) {
            map =
            (com.ibm.websphere.cache.DistributedMap)
            javax.rmi.PortableRemoteObject
            .narrow(obj,
            com.ibm.websphere.cache.DistributedMap.class);
            System.out.println("successfully retrieved a
            map");
        }
        catch (Exception e) {
            System.out.println("failed to retrieve a
            map");
            e.printStackTrace();
        }
        return map;
    }
}
```

If clustering is false, the `Cache` class, which implements the `Map` interface, is generated with a prepackaged code template. In this case, the `getCache()` method of the `AcceleratedService` class looks like Listing 6.

### Listing 6. `getCache` implementation code

```
public Map getCache() {
    Map map = new Cache(size, timeout);
    return map;
}
```

## Cache size Parameter

If clustering is true, the `cacheSize` parameter expands into the `cacheSize` value in the properties file used by the underlying WebSphere caching implementation (`cacheinstances.properties`). If clustering is false, the `cacheSize` parameter expands into the `workingSetSize` value in the custom cache implementation, `InMemoryCache`.

## timeOut Parameter

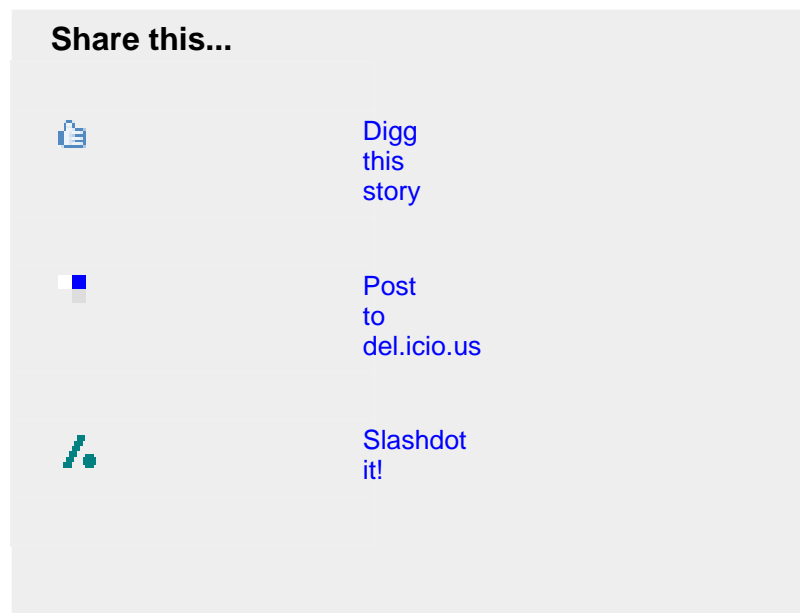
If clustering is true, the `cacheSize` parameter expands into the `cacheSize` value in the properties file used by the underlying WebSphere caching implementation (`cacheinstances.properties`).

## Implementing the specification

Armed with the pattern implementation specification, you can now author a pattern implementation using the Rational Software Architect pattern engine. The output of this authoring process is an Eclipse plug-in that can be used in RSA to automate the application of the RSC pattern to a model.

The approach to using this pattern follows the model-driven development approach of instantiating the pattern parameters to specific UML model elements of the service or interface. After the pattern parameters are bound, additional UML elements, such as the cache and the cache-aware service proxy, are automatically created. A UML-to-Java transformation, invoked on the resulting model, generates the resulting Java implementation artifacts. This implementation also allows the user to choose between a custom (in-memory) user-defined cache or the WebSphere platform dynamic cache. If the user chooses the latter, the pattern transformation automatically generates configuration files to be used with Dynacache.

## Conclusion



In this article, you examined the RSC pattern implementation specification. This implementation specification is a bridge between the human readable pattern specification (for example, pattern specifications originated by the Gang of Four) and

pattern implementation that can be used in a development environment to automate the application of the pattern. From this implementation specification, you have the freedom to create numerous implementations.

# Resources

## Learn

- Read the first article on this topic, "[The requester side caching pattern specification, Part 1: Overview of the requester side caching pattern](#)" (developerWorks, Oct 2005).
- Check out "[Enable C++ applications for Web services using XML-RPC](#)" (developerWorks, Jun 2006) for a step-by-step guide to exposing C++ methods as services.
- The [SOA and Web services zone](#) on IBM developerWorks hosts hundreds of informative articles and introductory, intermediate, and advanced tutorials on how to develop Web services applications.
- Play in the [IBM SOA Sandbox!](#) Increase your SOA skills through practical, hands-on experience with the IBM SOA entry points.
- The [IBM SOA Web site](#) offers an overview of SOA and how IBM can help you get there.
- Stay current with [developerWorks technical events and webcasts](#).
- Browse for books on these and other technical topics at the [Safari bookstore](#).
- Check out a quick [Web services on demand demo](#).
- Get an [RSS feed for this series](#). (Find out more about [RSS](#).)

## Get products and technologies

- Innovate your next development project with [IBM trial software](#), available for download or on DVD.

## Discuss

- [Participate in the discussion forum for this content](#).
- Get involved in the developerWorks community by participating in [developerWorks blogs](#), including the following SOA and Web services-related blogs:
  - [Service Oriented Architecture -- Off the Record](#) with Sandy Carter
  - [Best Practices in Service-Oriented Architecture](#) with Ali Arsanjani
  - [WebSphere SOA and J2EE in Practice](#) with Bobby Woolf
  - [Building SOA applications with patterns and reusable assets](#) with Dr. Eoin Lane
  - [Client Insights, Concerns and Perspectives on SOA](#) with Kerrie Holley

- [Service-Oriented Architecture and Business-Level Tooling](#) with Simon Johnston
- [SOA, ESB and Beyond](#) with Sanjay Bose

## About the authors

### Dr. Harini Srinivasan

Harini Srinivasan is in IBM's Enterprise Integration group. Her research and technology experiences include compiler and runtime analysis of data parallel and control parallel applications for massively parallel computers; compiler and runtime optimizations of object-oriented programs, JVM run times and optimization; and performance analysis, program understanding, and debugging tools for enterprise real-world applications. More recently, she has been working in the area of model-driven approaches to building SOAs. She has been an employee of IBM for 11 years and has been a Research Staff Member at the IBM T.J. Watson Research Center.

---

### James Conallen

Jim Conallen is a software engineer in IBM Software Group's Rational Model-Driven Development Strategy team, where he's actively involved in applying the Object Management Group's (OMG) Model-Driven Architecture (MDA) initiative to the IBM Rational model tooling. Jim is a frequent conference speaker and article writer. His areas of expertise include Web application development, where he developed the Web Application Extension for UML (WAE), an extension to the UML that lets developers model Web-centric architectures with UML at appropriate levels of abstraction and detail. This work served as the basis for IBM Rational Rose and IBM Rational XDE Web Modeling functionality.

---

### Dr. Eoin Lane

Dr. Eoin Lane, senior solution engineer, is the lead for harvesting and developing application patterns from key IBM SOA engagements and driving those patterns through the IBM pattern governance process to accelerate adoption. Eoin also specializes in using model-driven development (MDD), asset-based development, and the Reusable Asset Specification (RAS) to facilitate SOA development.

## Trademarks

IBM, the IBM logo, Rational, and WebSphere are registered trademarks of IBM in the United States, other countries or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.