SC\_CC

Service Connector Cache Coherency

SC Cache Coherency Model

SC\_CC-V1.0\_E (Version V1.0)

This document describes the SC Cache Coherency Model.

**Copyright © 2010 STABILIT Informatik AG, Switzerland**

**Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at**

**http://www.apache.org/licenses/LICENSE-2.0**

**Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.**

**See the License for the specific language governing permissions and limitations under the License.**

All other logos, product names are trademarks of their respective owners.

This Document has been created with Microsoft Word 2003 (11) with template file and printed at 14 May 2012 14:40.

# Identification

|  |  |
| --- | --- |
| Project: | SC\_CC |
| Title: | Service Connector Cache Coherency |
| Subtitle: | SC Cache Coherency Model |
| Version: | V1.0 |
| Reference: | SC\_CC-V1.0\_E |
| Classification: | Public |
| Keywords: | Architecture, Cache Concept, Coherency Model |
| Comment: | This document describes the SC Cache Coherency Model. |
| Author(s): | STABILIT Informatik AG  Joël Traber |
| Approval (Reviewed by): | Signature Jan Trnka |
| Audience: | Project team, Review team |
| Distribution: | Public |
| Filename |  |

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Author | Description |
| 10.05.2012 | D1.0 | Joël Traber | Initial draft |
| 12.06.2012 | V1.0 | Joël Traber | Change cache structure, appendix messages (SIX requirements). One cache per SC! |

# Table of Contents

1 Preface 3

1.1 Purpose & Scope of this Document 3

1.2 Definitions & Abbreviations 3

1.3 External References 3

1.4 Typographical Conventions 3

1.5 Outstanding Issues 3

2 Introduction 4

2.1 Caching concept 4

2.2 Cache coherence problem 4

3 Cache Coherence Model 6

3.1 Cache Coherence 6

3.1.1 Fundamental caching concept 6

3.1.2 Cache Manager 7

3.1.3 Large SCMP Messages 8

3.1.4 Deletion of a cached message 8

3.1.5 Cache message update rules 8

3.1.6 Clear managed data in all caches 8

3.1.7 Broken or inactive Cache Subscription Service on SC Proxy 9

3.1.8 Limitations 9

4 Cache Coherence - SC Client API 10

4.1 SC Client API and cache coherence 10

4.2 SC Client API – Loading of large messages 10

4.3 SC Client API – Deletion of cached Messages 10

5 Monitoring and Troubleshooting 11

5.1 Cache Coherence Logging 11

5.2 Cache Coherence Monitor 11

6 Configuration of Cache Subscription Service 12

6.1 Cache Subscription Service configuration 12

7 Best Practice of Cache Coherence Model 13

7.1 Proper separation of updatable and non-updatable data 13

7.2 Use of the subscription mask 13

7.3 Understanding the Topology effects 13

8 Glossary 14

Appendix 15

Index 16

# Tables

Table 1 Abbreviations & Definitions 3

Table 2 External references 3

Table 3 Typographical conventions 3

# Figures

Figure 1 Topology & Cache Concept of Service Connector 4

Figure 2 Cache Coherence Model 6

Figure 3 SC Caching – Concept of CacheId 7

# Preface

## Purpose & Scope of this Document

This document describes the SC-CC **S**ervice **C**onnector **C**ache **C**oherency Model.

The final and approved version of this document serves as base for publication as Open Source.

This document is particularly important to all project team members and serves as communication medium between them.

## Definitions & Abbreviations

|  |  |
| --- | --- |
| Item / Term | Definition / Description |
| HTTP | Hypertext Transport Protocol |
| HTTPS | HTTP over SSL, encrypted and authenticated transport protocol |

Table Abbreviations & Definitions

## External References

|  |  |
| --- | --- |
| References | Item / Reference to other Document |
| [1] | SC\_0\_Specification\_E – Requirement and Specifications for Service Connector |
| [2] | SC\_0\_SCMP\_E – SC Message Protocol V1.2 |
| [3] | SC\_4\_Operation\_E – Configuration and Operation Guide |
| [4] |  |

Table External references

## Typographical Conventions

|  |  |
| --- | --- |
| Convention | Meaning |
| *text in italics* | features not implemented in the actual release |
| text in Courier font | code example |
| [ phrase ] | In syntax diagrams, indicates that the enclosed values are optional |
| { phrase1 | phrase2 } | In syntax diagrams, indicates that multiple possibilities exists. |
| … | In syntax diagrams, indicates a repetition of the previous expression |

Table Typographical conventions

The terminology used in this document may be somewhat different from other sources. The chapter Glossary includes a list of often used terms with the explanation of their meaning in this document.

## Outstanding Issues

Following issues are outstanding at the time of the document release:

* none

# Introduction

The SC supports message exchange between requesting application (client) and another application providing a service (server). Caching of messages can be activated on every SC.

Cache coherency refers to the consistency of data stored in local caches of every SC node. The coherency protocol described in following sections addresses the problem of maintaining the consistency between all the caches in a system of distributed shared memory. Therefore a coherence model has been specified.



Figure Topology & Caching in Service Connector

## Caching concept

Caching or not caching of a message (response) is an agreement of client and server. A cached message carries an “expire date time”. The cache module in SC is self maintaining and removes expired data itself. The expiration is given to the message by the responding server. Only the server can take this decision. Expiration of a cached message is driven by the semantic of the message content.

## Cache coherence problem

The duration until expire date of a cached message describes its **most common lifecycle** which is the timeframe it is valid for clients.

Lots of messages don’t really have a known lifecycle or it is dependent on other client actions. Caching of this type of messages is only possible if we are having a model to keep data consistent. This simply means at the time cached data changes on server node, caches immediately need to get the new message content and stop satisfying request by handing out obsolete data. Main target of the concept is to keep time between “data change” and “cache update” as small as possible. The client’s actions shouldn’t base on obsolete data.

It is impossible to avoid the coherence problem completely. For a short time period it will always exist.

# Cache Coherence Model

To assure cache consistency Cache Managers are supported by the SC. The main strategy of communication works like a publish service. Any changed data on a server node and known as cached content needs to be published to a Cache Manager. By using the fan out mechanism data update gets populated to the Clients. Updating the cache content is done inside the SC and works according to [Cache replacing rules](#_Replacing_rules).



Figure Cache Coherence Model

## Cache Coherence

### Fundamental caching concept

As mentioned earlier, caching or not caching of data is an agreement of client and server. A client indication (cache id) for caching of a message is needed. Afterwards the server confirms caching by returning the same cache id. Indication of the client allows blocking other clients with the same request. If the server disclaims returning the cache id **nothing** gets cached.

The granularity of cached messages exactly correlates to the SCMP messages sent over the wire. Neither restructuring of messages nor modification of bodies will be done!



Figure Fundamental Caching Concept

Caching is completely based on the cache id. Structuring the cached data it’s up to client and server. The concept of used identifiers must be agreed.

### Cache Manager

**Cache Structure**

The cache is divided into static and managed data.

Static data stays in cache until “expiration time” times out or a remove is received. Neither publishing appendixes nor replacements are allowed for static data.

For managed data the SC is responsible for handling the coherency problem. Appendixes or replacements may be applied.

**SC Client API**

The SC Client API subscribes to a Cache Manager. Any changed data published to this Cache Manager by the publish server will be populated up to the client. As long as an API user keeps the cache manager active updates will automatically be received over a callback. At the time the Cache Manager is inactive requested data is still consistent but client is not informed about updates anymore.

**SC**

Cache Managers are defined in the configuration file (sc.properties) of the SCs. The cache is managed by Cache Managers. On a cascaded SC more than one Cache Manage might be active. Messages get loaded by a session request of a client. The first Cache Manager applying an update to a cached message is responsible to keep data consistency. Different messages may be assigned to different Cache Managers.

A Cache Manager stops in following cases:

* No client online (nobody interested in updates)
* Connection loss between client and SC (potential lack of updates)
* Connection loss between SC and SC (potential lack of updates)
* Connection loss between SC and Publish Server (potential lack of updates)

Stopping of a Cache Manager triggers a cleaning procedure in the cache module. Any message the inactive Cache Manager has treated will be removed to avoid the coherence problem. Next client requesting a deleted message causes a new load process**. In clever topology the request will never end up on server level!**

**Publish Server**

…. To be done! Rules of updating

A publish server sends updates to a Cache Manager. Messages are populated up to the clients.

The same update might be published of more than one Publish Server to different Cache Managers. Cached messages are only updated of one Cache Manager. Other updates are ignored.

As long as no large messages are publish it’s possible to have more than one server sending updates to the same Cache Manager. In this case sending the same update twice over each server is nonsense and invalids cached message!

### Large SCMP Messages

For updating a large message in the cache the header attribute “bodyHash” must be set. The publishing server is responsible to set a hash code in every part message of the complete body. Two services might update the same large message in the same cache at the same time! So a part message can arrive two times or from different services. Knowing to body of the complete message allows the cache consuming the large message of several services.

### Deletion of a cached message

Publishing a message with an already expired “expire date” causes a deletion of a cached message. Message will be published to the client any clear the caches on its paths. The SC Client API needs to recognize and inform the user about the deletion.

### Cache message update rules

Updating obsolete data is based on the identifier “serviceName\_cacheId”. At the time a “CacheSubscription” service receives a publish message, the SC looks up the message in cache. Below mentioned rules take place.

Cache updatable message is found

* Gets updated completely
* Even the expire date gets overwritten. If new expire date is in past whole message gets removed. That’s the way **a deletion of cached** **message** can be processed, see [Deletion of a cached message](#_Deletion_of_a).

Cache updatable message is found in loading (not complete yet) mode

* Loading actions aborts (partial data gets removed).
* Received message gets discarded after forwarding.
* The client needs to request data again.

No message with the given “cacheId” found in cache

* Nothing happens in cache.
* Received message gets discarded after forwarding.

Message is found without the “cacheUpdatableMessage” attribute set

* Existing message in cache gets updated completely.
* Current message in cache is modified to an updatable message.
* Received publish message will be discarded after forwarding.

**Update of Large Messages**

(Has to be done!) consuming large messages of two services.

### Clear managed data in all caches

The new service topology brings a new feature, which the maintainer of the publish server needs to be aware of. Shutting down or abortion of the publish service clears any updatable data in all caches. Unsubscription of the publish service causes to abort any subscription existing to this service in the whole network topology. This might be a good or bad thing depending on the situation.

### Broken or inactive Cache Subscription Service on SC Proxy

Cache Subscription service might break on SC proxy node. Cached data is not able to get updated and proxy may miss a publish message. To avoid running into coherence problem and handing out obsolete data every cache updatable message will be deleted. Non updatable messages stay in cache.

Cache Subscription service might be/get inactive on SC proxy node. This is case at the time clients are not or stopped (no more subscriptions) communicating over the proxy. The “broken service” procedure takes place as described above.

### Limitations

To assure cache consistency it is necessary to publish large messages to the service. This feature is basically working. To run properly only one publish server is allowed to publish. Publishing is working with a queue. The data parts of a large message have to queue in sequence. This condition can’t be guaranteed if more than one server is publishing data to the same service.

For routing and for handling the failover problem it is allowed to define two cache subscriptions services in the property file of an SC instance. A cache might be updated by two services at the same time.

# Cache Coherence - SC Client API

The SC Client API supports the proper use of the cache coherence model by the interface described in following section.

## SC Client API and cache coherence

**SCClient**

In order to support cache coherence model SCClient introduces new methods. Method calls are synchronous and can be done after attaching SCClient successfully.

*startCacheSubscription(String serviceName, int operationTimeoutSeconds, SCSubscribeMessage scSubscribeMessage, SCMessageCallback scMessageCallback, int receivePublicationTimeoutSeconds) throws SCServiceException, SCMPValidatorException*

* *serviceName*: Identifies the CacheSubscriptionService
* *operationTimeoutSeconds*: Time until starting of CacheSubscriptionService aborts. (Operation timed out)
* *scSubscribeMessage*: Service subscribe message (see publish service for more details)
* *scMessageCallback*: Callback to receive cache updates.
* *receivePublicationTimeoutSeconds*: Time to wait for completion of a receive publication request.

It’s up to the SC Client API user to activate cache subscription service or not. There are updateable and non updatable messages in the cache. Having the cache managing updatable messages starting of the cache subscription service is necessary. If service is inactive only non updatable messages are provided by the cache.

Following method stops the cache subscription service in a shutdown scenario.

*stopCacheSubscription()*

## SC Client API – Loading of large messages

Having an active cache subscription slightly changes loading of large messages. During the time a client loads a complete cached message a CACHE\_LOADING exception can be thrown.

**Given Scenario**

Client reads a large message from cache. During this process the cache receives an update for exact this large message. Message state will be changed to loading. Cache stops satisfying requests on this “cacheId”. Client gets a CACHE\_LOADING exception, needs to discard already loaded parts and to retry.

## SC Client API – Deletion of cached Messages

(has to be done!)

SC Client API gets informed some way??... how? Special exception, special message

To be discussed with KEEL.

What about the expiration of a message. Has the cache to publish a message to the API??..Does user need to be informed of this event?

# Monitoring and Troubleshooting

Base concepts of SC monitoring and troubleshooting are described in <SC_4_Operation_E.pdf>. Subsequent section covers monitoring and troubleshooting for the context of cache coherence.

## Cache Coherence Logging

The **cacheLogger** is responsible for logging the events in context of caching.

Following events will be logged:

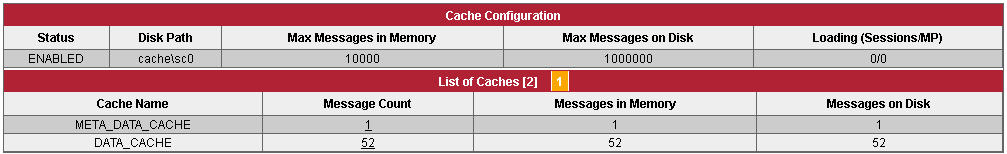
* New subscription on cache subscription service.
* New updatable message received.
* New part of updatable message received.
* Message part successfully updated.
* Large Messages completely updated.
* Message successfully updated.
* Message gets updated and deleted because of expired date.
* Message can’t be updated because no corresponding “cacheId” found.
* Message can’t be updated because corresponding message is in loading mode -> message deletion.
* Broken cache subscription service -> Updatable messages removed.
* No more subscriptions on cache subscription service -> Updatable messages removed.

## Cache Coherence Monitor

WebUI has to show updatable messages, last update, and state of subscription service. Number of updated messages since ?. Number of Hits for specific messages. Number of loading messages because of published updates.

Cache

The cache form shows the state of the cache.



# Configuration of Cache Subscription Service

The SC is configured with a single configuration file sc.properties. More details are described in the [SC Operation Guide](SC_4_Operation_E.doc). Configuration of cache subscription service is also done in sc.properties.

## Cache Subscription Service configuration

The configuration of a cache subscription service basically looks like a publish service definition. Only service type “cacheSubscription” differs and indicates SC to treat it different.

…...

sc1-cacheSubscription.type=cacheSubscription

sc1-cacheSubscription.enabled=true

sc1-cacheSubscription.remoteNode=sc1

sc1-cacheSubscription.noDataIntervalSeconds=10

sc2-cacheSubscription.type=cacheSubscription

sc2-cacheSubscription.enabled=true

sc2-cacheSubscription.remoteNode=sc2

sc2-cacheSubscription.noDataIntervalSeconds=10

…...

# Best Practice of Cache Coherence Model

## Proper separation of updatable and non-updatable data

## Use of the subscription mask

Like a publish service does a cache subscription service support the usage of the subscription mask. Published updates of data will be broadcasted according to the subscription mask. It’s up to server and clients agreement to wisely use it.

For example if they agree an ordered list of all “cacheIds” and use the particular positions to identify the mask bit. Client may precisely subscribe for the updates he needs. This also reduces network traffic between proxies.

## Understanding the Topology effects

(Has to be done!) Keeping at least one client on RSC direct connection.

# Glossary

Cache Coherency

fgdfgdfg

Cache Consistency

dfgdfgdfg.

Cache Subscription Service

1. Appendix

C

Cache • 11