STABILIT
Intelligente Informatik

SC

Service Connector

Requirements and Specification

SC_0_Specification_E (Version V1.4)

SC is a middleware component for new Eurex IT platform. This document summarizes the requirements, describes specifications and defines the scope of the project. It is the starting point for the design phase.

The creation of this software is sponsored by a customer. Other companies may develop similar software as a commercial product and destroy this effort in the mean time. In order to protect the investment of the sponsor and the idea of open source software, this and all subsequent documents are classified as "CONFIDENTIAL" during the development phase. They may be delivered only after signing an appropriate non-disclosure agreement. They will be published when the software development is finished and the software is made available as open source software.

During the development of this software this document is a spiritual ownership of STABILIT Informatik AG, and must not be used or copied without a written allowance of this company and signing a non-disclosure agreement. Unauthorized usage is illegal in terms of Chapter 23 / 5 UWG / Swiss law.

The information in this document is subject to change without notice and should not be construed as a commitment by STABILIT Informatik AG.

This Document has been created with Microsoft Word 2003 (11) with template file C:\STABILIT\STANDARD\TEMPLATES\S_REP_E.DOT and printed at 10 February 2010 10:57.

Identification

Project:	SC
Title:	Service Connector
Subtitle:	Requirements and Specification
Version:	V1.4
Reference:	SC_0_Specification_E
Classification:	Confidential
Keywords:	Requirements, Specification, Scope
Comment:	SC is a middleware component for new Eurex IT platform. This document summarizes the requirements, describes specifications and defines the scope of the project. It is the starting point for the design phase.
Author(s):	STABILIT Informatik AG Jan Trnka, Daniel Schmutz, Mauro Ricchetti, Joel Traber
Approval (Reviewed by):	Signature ..... Jan Trnka Signature ..... Francisco Gonzalez
Audience:	Project team, Eurex IT management, Review team
Distribution:	Project team, Eurex IT management, Review team
Filename	c:\stabilit\projects\eurex\sc\documents\sc_0_specification_e.doc

Revision History

Date	Version	Author	Description
14.06.2009	D1.0	Jan Trnka	Initial draft
09.08.2009	D1.1	Jan Trnka	Separate specifications and requirements into this document and move other parts to the architecture document
25.11.2009	V1.2	Jan Trnka	Corrections from talks with SIX
20.1.2010	V1.3	Jan Trnka	Finalizing requirements
31.1.2010	V1.4	Jan Trnka	Closing open issues

Table of Contents

Tables

Figures

1 Preface

1.1 Purpose & Scope of this Document

This document summarizes the known requirements, describes system specifications and defines the project scope.

The final and approved version of this document serves as base for the binding architecture and design, described in another document. It defines also the scope of the implementation project and is part of the contract between Eurex / SIX and Stabilit.

This document is particularly important to all project team members.

1.2 Classification

The creation of this software is sponsored by a customer. Other companies may develop similar software as a commercial product and destroy this effort in the mean time. In order to protect the investment of the sponsor and the idea of open source software, this and all subsequent documents are classified as "CONFIDENTIAL" during the development phase. They may be delivered only after signing an appropriate non-disclosure agreement. They will be published when the software development is finished and the software is made available as open source software.

1.3 Definitions & Abbreviations

Item / Term	Definition / Description
DMZ	Demilitarized Zone (Network segment between two firewalls)
DOS	Denial Of Service
EBS	Elektronische Börse Schweiz – older project name for ESY
ERM	Electronic Repo Market
ESY	Exchange System – SIX trading platform
GUI	Graphical User Interface
HTML	Hypertext Mark-up Language
HTTP	Hypertext Transport Protocol
HTTPS	HTTP over SSL, encrypted and authenticated transport protocol
IE	Internet Explorer – Microsoft Web Browser
J2EE	Java 2 Platform Enterprise Edition
Java	Programming language and run-time environment from SUN
JDK	Java Development Kit
Linux	Unix-like operating system, open source
Log4j	Standard logging tool used in Java
OpenVMS	HP Operating system, existing platform for ERM
OSI	Open System Interconnection reference model http://en.wikipedia.org/wiki/OSI_model
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
SSL	Secure Socket Layer – secure communication protocol with encryption and authentication
SIX Exchange	Member of SIX Group, Exchange platform provider
TCP/IP	Transmission Control Protocol / Internet Protocol
SIX Telekurs	Member of SIX Group, Financial information provider, and operational staff for ERM.

SC	Service Connector
USP	Universal Service Processor – existing middleware used by Eurex Repo
Web Service	software designed to support interoperable machine-to-machine interaction over a network
Web-GUI	Graphical User Interface running in Web-Browser built with Web components. Also called Thin-Client.
Windows	Microsoft operating system family
XML	Extensible Mark-up Language
XSD	XML Schema Definition
XSLT	Extensible Style Sheet Language Transformation

Table 1 Abbreviations & Definitions

1.4 External References

References	Item / Reference to other Document
[1]	USP Documentation http://www.stabilit.ch/index.php?option=com_content&task=view&id=27&Itemid=60
[2]	

Table 2 External references

1.5 Typographical Conventions

Convention	Meaning
text in italics	*USP feature or functionality or USP related comments*
`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 exist.
…	In syntax diagrams, indicates a repetition of the previous expression

Table 3 Typographical conventions

In formulation of requirements words "must" and "should" have the following meaning:

Formulation	Meaning
must, must not	Strong, mandatory requirement.
should, may, need not	Weak, optional requirement. (Nice to have)

Table 4 Requirement Formulations

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

1.6 Outstanding Issues

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

1. Must SC support OpenVMS Alpha? => only JDK 1.5 available!

2 Project Definition

2.1 Existing Situation

The new EUREX IT platform needs a middleware that allows exchange of messages between its components and applications. This middleware must also allow seamless replacement of the existing USP with a new forward looking solution. Old Eurex Repo services running on OpenVMS must be able to connect via SC to the trading GUI or to other components. However SC should not be treated as a plain USP substitution. It should open new opportunities for communication of the REPO platform with new internal or external services.

Researching the market and products we found no adequate software available. The SIX Exchange is looking for USP replacement by a standard product since 5 years without any visible progress. Our sound knowledge of USP and of the EUREX computing environment leads us to the idea of developing a new solution.

2.2 Goals

The following are the overall project goals (in priority order):

1. Create simple messaging middleware for all new components within the Eurex platform
2. Replace existing USP and keep the capability of access to existing Repo services.
3. Design a simple and open message transport protocol that can be adopted by the companies that want build custom solutions based on it.

The principle is: **“as simple as possible, serving only the real needs”.**

Other important goals are:

- Operating system independence (unix, linux, windows, OpenVMS Itanium)
- Use of Open Source components (all java)
- Universal API for message-based communication
- Use of standard underlying protocols (http, ftp, smtp, etc.)
- Publishing of SC source code in the Open Source Software community.
- Publically available documentation of the SC protocol
- API for applications written in Java (server and client) and C (server only)
- Availability equal or better then USP (> 99.90%)
- Performance better the USP (>1’000 msg./sec.)
- Scalability better then USP (>1’000 concurrent clients)
- Manageability better then USP (Web-GUI for administration)

2.3 Non-Goals

The SC will not have any persistence and thus will not provide any transactional concepts or reliable messaging.

The SC will not implement any security feature. The environment where SC is used must provide suitable security features like authentication, authorization, encryption, tunneling etc.

The SC will not provide any load balancing features. Established communication session will not be redirected to another server node initially or during its life time.

The SC will not implement any failovers features. Aborted communication must be re-established by the communicating partners. The client application must find out a service which is alive.

SC provides message transport and thus implements a message transport protocol. The transported message payload is not known to SC and will not be inspected or transformed by SC. The communicating parties must agree on the format and content of the message payload!

For the reason above, SC is not intended for direct communication to external systems like SIS, Telekurs VDF, SMF, Bloomberg or others. These systems use their own established protocols and must be connected with other middleware or can be connected to SC via a protocol-specific SC gateway that may be developed when necessary. It is the intention to publish the SC message transport protocol and allow third-parties to connect directly to SC.

2.4 Scope

The scope of the project is to design and develop the SC, including all interfaces to the existing USP applications, including the SC-Proxy.

Only http transport will be implemented in the first release for the underlying transport.

Only Java API will be implemented in the scope of this project. The SC client and server API for C will be developed in a separate joint project together with the customer (SIX Exchange) and is not scope of this project.

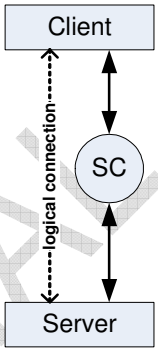
The migration from USP to SC is not scope of this development project. It will be planed and subsequently realized as a separate project.

2.5 Relationships & Dependencies

The USP migration to SC cannot begin before this project will finish and the acceptance test is passed. However it is possible to start developing the necessary adoptions of the connecting applications as soon as the SC interface has been published.

3 Requirements

The SC must support message exchange between requesting application (client) and another application providing a service (server). The client and the server are the logical communication end-points. The SC is always in between of these points and establishes the logical connection between the communicating partners. It never acts as a direct executor of a service.



The client must be able to communicate to multiple services. This must be possible in sequence (one connection at a time) or in parallel (two independent connections at the same time).

Server application must provide only one service. Multiple server applications must coexist on the same server node, each providing different service. Serving multiple services within one application will be handled at application level by an appropriate service dispatcher inside the application. All services are independent on each other.
Server application may request another service and so play the client role. Such cascading must be possible. Cascading of services must also be possible through the SC Proxy.

The client and server applications are started, stopped and managed by operational facilities that are not scope of the SC. The client may be started and stopped at any time. The server can be pre-started or not and must registers itself on SC as a service provider before the first client will need its service. In order to enable the communication, the SC must run all the time.

Messaging Services

The SC provides API and messaging for these two service types:

- Request/Response (client initiated communication)
  Former USP-RPC - remote procedure call
  See use case 0 and 4.1.2
- Subscribe/Publish (server initiated communication)
  Former USP-BCT – broadcast
  See use case 4.1.3

File Service

The SC provides API for these service types:

- Download file (from the server to the client)
  See use case 4.1.4
- Upload file (from the client to the server)
  See use case 4.1.5
- List files in a file repository (on the server).
  See use case 4.1.6

SC-Proxy Services

The SC-Proxy provides following services (See use case 4.2.1, 4.2.2 and 4.2.3):

- Cascading of all messaging and file services.
- Bundling of the network traffic for Request/Response services
- Message caching for Request/Response services
- Fan-out of the published messages for Subscribe/Publish services
- Redirecting of regular HTTP traffic to another server (acting like normal HTTP Proxy with no caching)

4 Specifications

The SC implements peer-to-peer messaging above OSI layer-7 (application) network model between client and server applications. The client and the server have a logical connection through the SC. The SC is always in between the communicating partners, establishing this logical connection and controlling the entire message flow.

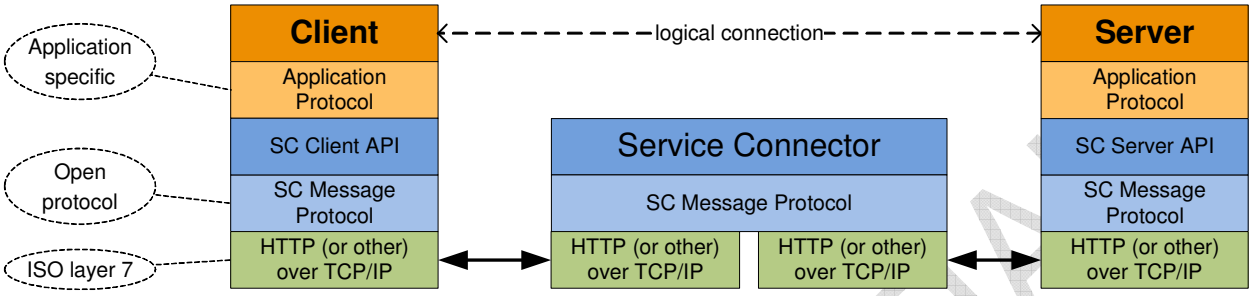


Figure 1 Communication Layers

The SC acts like a broker, passing messages between the client and the server. The communicating parties must agree on the application protocol i.e. format and content of the message payload.

4.1 Use cases

In the scope of the logical connection between the client and the server, the SC implements the use cases described here below.

4.1.1 Synchronous Request/Response

The client sends a request to a service that invokes an application code. Upon completion the service sends back a response message. The client waits for the arrival of the response message. The client may have only one outstanding request per service.

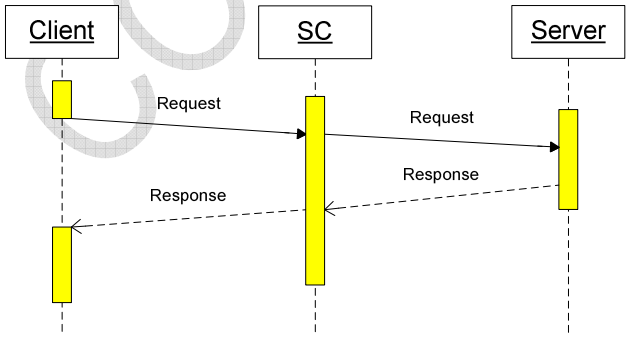


Figure 2 Synchronous Request/Response

Multiple clients may request the same service at the same time. In such case the service execution is in parallel, each one in a separate service context (thread or process). SC will choose a free server and pass subsequent requests from this client to the same server. The communication occurs in a scope of a logical session. The server decides how many sessions it can serve. Session information is always passed as a part of the message header.

This communication style is the most often used for getting data from the server or sending a message that triggers a transaction on the server.

USP maintains a session context. It starts and assigns a separate server process for each connected client and maintains a permanent TCP/IP connection between client, the USP and the allocated server until the client disconnects. This is very resource consuming and massively limits the scalability. Some session checks are in the server application (ICS, TXS, TXR) itself.

4.1.2 Asynchronous Request/Response

This is functionally equal to the synchronous case with the exception that the client does not wait for the arrival of the response message. The client must declare a notification method that is invoked when the response message arrives. The client may have only one outstanding request per service. When client issues a request before the previous one was completed, the method blocks until the no response is pending and the new request can be successfully sent.

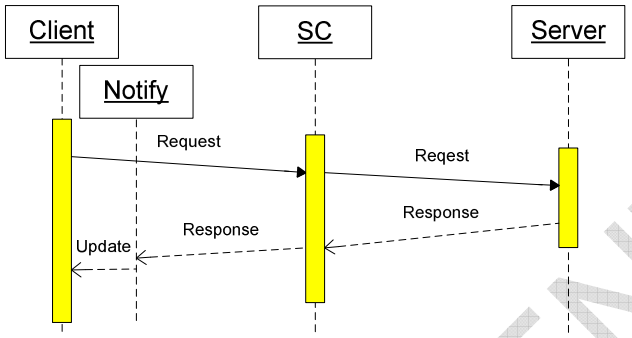


Figure 3 Asynchronous Request/Response

This communication style will be used to load data while other activities are in progress, e.g. to get large amount of static data at startup. It can be also used as fire-and-forget when the response is not meaningful.

USP does not support asynchronous request/response.

4.1.3 Asynchronous Subscribe/Publish

The client sends a subscription mask to the service, and so declares its interest on certain type of a message. The application service providing the message contents must designate the message with a type. When the message type matches the client subscription mask, the message will be sent to the client. Multiple clients may subscribe for the same service at the same time. In such case multiple clients can get the same copy of the message. Message that does not match any client subscription is discarded.

The client must declare a notification method that is invoked when the message arrives. The client may have only one outstanding subscription per service. The message delivery must occur in guaranteed sequence. Messages from the same service will arrive in the sequence in which they have been sent.

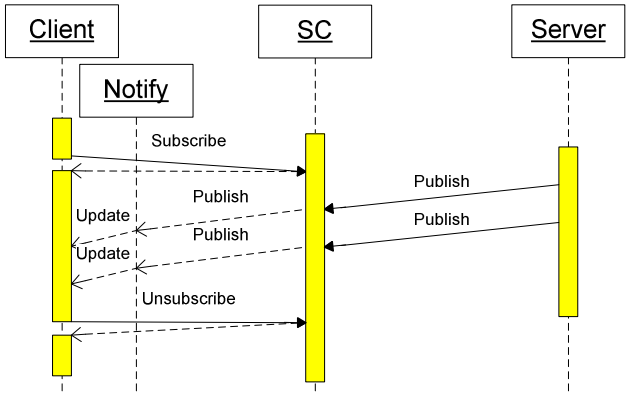


Figure 4 Asynchronous Subscribe / Publish

The client may change the subscription mask or unsubscribe. Initial subscription, subscription change or unsubscribe operation is always synchronous, even through a SC proxy.

Such communication style is used to get asynchronously events notifications or messages that are initiated on the server without an initial client action. It can be also used to distribute the same information to multiple clients.

USP implements a broadcasting service identical to the functionality described above. It queues messages that are later distributed to the clients. This can cause exhausting of memory when the client is not able to receive the messages and the connection is not aborted (e.g. blocking firewall in between)

4.1.4 Download File

The client sends a request to a download a file from via SC from the server. The downloaded file is stored locally on the node where the request was issued. Only download via HTTP will be supported. WebDAV protocol will not be supported. The communication style is synchronous. Standard Apache web server can be used.

This service will be used to download the actual SC-proxy configuration from a central server.

USP does not support file operations

4.1.5 Upload File

The client sends a request to upload a file via SC to the server. The uploaded file must exist on the node where the request was issued. Only download via HTTP will be supported. WebDAV protocol will not be supported. The communication style is synchronous. Standard Apache web server can be used.

This service will be used to upload log files from SC-proxy to a central server.

USP does not support file operations

4.1.6 List files

This service is indirectly supported by the standard web server (e.g. Apache) when directory browsing is allowed and enabled. The client sends an URL and gets back a HTML formatted output containing the file list. The client must parse this response in order to evaluate the exact file names. This communication style is synchronous. Standard Apache web server functionality is utilized.

This service will be used to get list of configuration files on the server that can be downloaded.

USP does not support file operations

4.2 SC-Proxy

For the client the SC-Proxy acts as a regular service, but passes all messages to another service of the same type. This "cascading" is possible for all service types. SC-proxy resides on a proxy node or on a server node. Unlike a server application, single SC-Proxy instance can serve and handle multiple services.

4.2.1 Request/Response

For Request/Response the SC-Proxy acts as a server. It bundles requests from multiple clients and passes them to another service like a single client. Received response messages are passed back to the appropriate clients. The purpose of bundling is to use only one communication channel and so save resources on the server.

Each time a client is started usually a high volume of static data must be loaded from the server. Most of this data is static i.e. valid for a long period of time and is accessible for all clients. In order to shorten the start-up time of the clients SC-Proxy should implement a message caching mechanism for dedicated server content. The purpose of the cache is to store shared data as close to the clients as possible.

The client request for a cached content will differ from regular request. The corresponding server response will be stored in the SC proxy and can be used by other clients. The cache content is refreshed when its time-to-live parameter expires and some client sends a request for it. The SC-Proxy cache is volatile and is discarded when the SC-Proxy is stopped.

USP-Proxy does not provide bundling of Request/Response traffic into a single connection and thus overloads the server with thousands of parallel connections. USP-Proxy does not implement message caching. Client start-up takes up to 10 minutes.

4.2.2 Subscribe/Publish

For Subscribe/Publish the SC-Proxy acts as a server, combines client subscriptions to a single subscription mask and provides fan-out (distribution) of response messages received from the server to the subscribed clients according to their subscription mask. In this ways the SC Proxy uses only one downstream connection per service.

USP-Proxy implements identical fan-out features.

4.2.3 HTTP Proxy

SC-Proxy node is often located in DMZ at the customer site and connected via VPN with the server. Therefore it must also provide simple HTTP proxy function, because it represents the only path to the server where a regular web server (apache) is running.
No file caching will be implemented in the proxy.

USP-Proxy implements http 1.0 proxy without caching.

4.3 Services

SC will support services implemented as servers. Servers are always pre-started and register themselves on SC. Two different server types are supported:

- Single threaded servers.
  One server process is dedicated for a single session. This is the case in actual ERM architecture.
- Multithreaded servers.
  One server has multiple threads and may serve multiple sessions. E.g. Apache Tomcat can be used to implement such services.

USP supports single threaded servers only

4.4 Service Naming

The network represents the namespace domain. The service names within this namespace must be unique. Therefore services residing on different server nodes within the same network must have different names.

USP behaves the same way.

4.5 Network

Client application, server application and the SC may reside on the same node or on separate nodes connected via TCP/IP network. No assumption about the physical network topology must be done. Multiple firewalls can be located on the path between the communicating applications.

UDP must not be used, because it will not pass firewalls and routers. FTP should not be used because it will not pass customer firewalls and may cause potential problems with customer security policy.

SC is not supported on network with nodes having mixed architectures (big and little endian).
SC is supported only on nodes with ASCII character set (no support for EBCDIC).

USP has the same limitations

The system must be able to handle network size defined in the following table

Item	Number	Comment
Clients	≥10'000	Currently 1'100 users are registered
Proxies	≥1000	Currently 250 USP-Proxies are in use
Services	≥100	Currently 2 nodes with 13 services each are in use

Table 5 Network Size

For technical reasons USP supports <1000 concurrent client.

4.6 Connection Topology

The SC supports following connection topology:

- Client ⇔ Service = Direct connection
- Client ⇔ Proxy (n-times) ⇔ Service = Cascaded connection via SC-Proxy.
  One or multiple SC-Proxies may be placed on the path between the client and service.

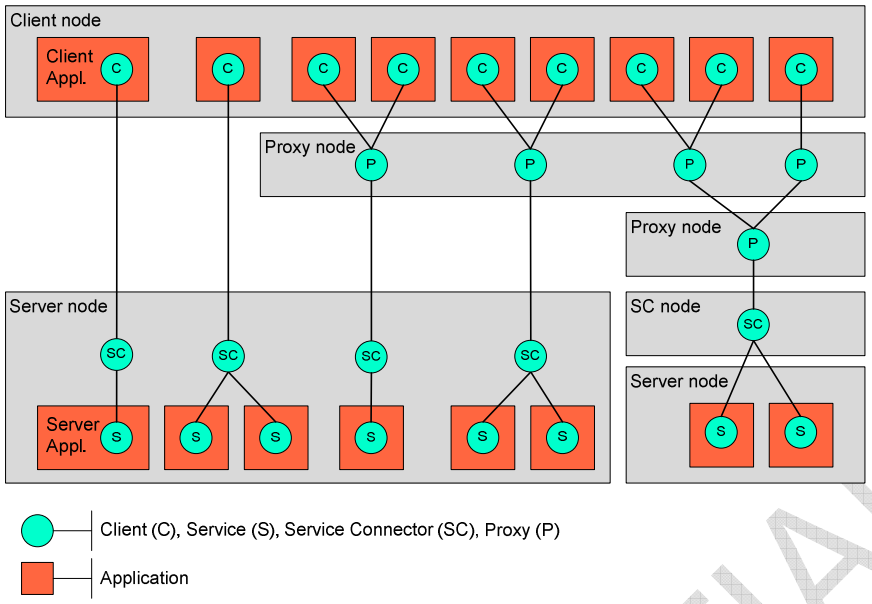


Figure 5 Connection Topology

Different connection topology types from left to right:

1. Client connected directly to a service
2. Client connected simultaneously to two services
3. Two clients connected to one service via proxy
4. Two clients connected to two services via single proxy
5. Three clients connected to two services via cascaded proxy on two separate nodes and SC offloaded to its own node.

Two different connection types can be configured between the Client ⇔ SC, Client ⇔ SC-Proxy, SC-Proxy ⇔ SC-Proxy or SC ⇔ Server.

1. Over HTTP
   Such connection may pass screening firewalls and is appropriate for communication within the customer organization e.g. Client ⇔ SC-Proxy or SC-Proxy ⇔ SC-Proxy.
2. Direct TCP/IP
   Such connection would not pass firewalls without an explicit security rules. It is useful for connection within the same node e.g. SC ⇔ Server.

Each connection can be configured individually, but these two possibilities are mutually exclusive.

4.7 Underlying Message Transport

All service types utilize only http protocol over TCP/IP. This guarantees that customer firewalls need not be reconfigured.

It should be possible to implement other message transport protocols later without changing the client or server API.

USP uses a proprietary binary communication protocol over TCP/IP and over DECnet.

4.8 Message Format

The message consists of a variable size header and variable size body = payload. The maximum total message body length is 64kB. (This limitation is due to C-API for OpenVMS)

Message header content belongs to SC. Its structure is published. Java API will be provided to get important attributes.

Messages are encoded in ISO-8859-1 (Latin 1) character set.

Message body content belongs not to SC and is transported as a block of bytes (8bit). It is the task of the communication partners to agree on its structure and format as well as to perform appropriate marshalling or de-marshalling of data types.

Message body compression and de-compression before and after the transport can be turned on or off.

USP does implement marshalling / de-marshalling of some standard data types (string, integer and float) but this is not used by SIX. Message body compression is implemented on application level and is rather cumbersome for troubleshooting, because the message content cannot be traced at the transport level.

4.9 Keep-alive Messages

When no real data is exchanged, the client and the service exchanges periodically keep-alive messages that allow the detection of the connection state. In case of connection loss a reconnect may automatically with the next request.

USP does not implement automatic reconnect, because the communication is statefull.

The keep-alive message should be used to exchange time information between client and server. This does slightly simplify troubleshooting and operation in different time zones. It also allows the client do measure the round trip performance.

USP does implement keep-alive for request/response in V3 protocol. This is not used by SIX Group. Instead the V2 protocol is used and keep-alive is implemented on application level. For subscribe/publish complex and cumbersome keep-alive messages are implemented by USP, but this does not work reliably.

4.10 Automatic Reconnection

For performance reasons connections over HTTP will not be established for each individual request, but kept over longer time. The automatic reconnection should automatically recover from the loss of such connection, as it is usual for any stateless communication. Parties communicating via HTTP may be physically disconnected for a defined period of time, and may seamless continue after reconnect. However a long breakdown of the connection will cause abortion of the connection.

Automatic reconnection is a feature that is considered in the design, but not implemented in the first release.

USP does not have an automatic reconnection or connection recovery and aborts both connections to the parties.

4.11 Message sequencing

The request - response message pair must have a sequence number. It is generated and steadily increased by the client. Upon arrival of the response message the sequence number is checked by the client to detect potential gaps (missed messages).

The subscribe/unsubscribe message must have a sequence number. It is generated and steadily increased by the client. It determines the subscription ID.

The publish message must have a sequence number. It is generated and steadily increased by the SC. It determines the sequence order in which the SC has received messages from the server. Upon arrival of the publish message the sequence number is checked by the client to detect potential gaps (missed messages).

USP does not implement message sequencing for request/response but only for subscribe/publish and in slightly different way.

4.12 Load balancing

The SC does not provide any load balancing features. Established communication session will not be redirected to another server node initially or during its life time.

USP does not implement load balancing. This is built into the DDC application.

4.13 Failover

The SC does not provide any failover features. Aborted communication must be re-established by the communicating partners. The client application must find out a service which is alive.

USP does not provide any failover features.

4.14 Security

The SC does not implement any security feature. The environment where SC is used must provide all required authentication, authorization, encryption, tunneling etc. features. Message transport over https will not be supported.

USP provides SSL communication in cooperation with Apache CSWS, however no customer is using this feature.

The IP address of the client and the IP of the incoming TCP/IP traffic must be stored in the message header and can be obtained via API within the service. This can be used to authorize the client when a VPN tunnel is used.

USP follows the same philosophy and has the same limitation.

4.15 Intrusion and Virus Protection

The entire network where SC is used is assumed to be safe and secure. No virus protection is embedded in SC. The customer may use screen firewall to protect the SC components. It is recommended to use SC Proxy within a DMZ.

SC is not designed to withstand network attacks like DOS or SYNC flood, or any other.

4.16 Deployment

The SC client and server APIs for java will be delivered as one jar-file.
The SC client and server API for C is not scope of this project.

USP provides client and server API for C and LEVEL4 and a client API for Java. C API is not used. C applications are wrapped with LEVEL4 code.

The SC and SC-Proxy are delivered as jar-file that can be started from the command level as a java application (runnable jar-file). The only difference between SC and SC-Proxy is the configuration. Each SC can play also the role of a SC-Proxy. Multiple instances of a SC or SC-Proxy must be able to run on a single node. Each instance must have its own configuration.

On Windows platform SC or SC-Proxy can be deployed as service.

USP is deployed as a OpenVMS kit, USP/Proxy is deployed as ZIP file. USP Java Client is available as ZIP file (source code).

4.17 Configuration

The SC components embedded in the application are configured through the corresponding API. The configuration of SC and SC-Proxy is file based. The format of the configuration file is ASCII. (XML is not friendly for editing!)

The configuration contains static (S) and dynamic (D) parameters. Change of static parameters cannot be changed at run time and requires SC restart and thus los of all connections. Dynamic parameters can be changed at run time.

The SC configuration defines:

- (S) Log file location
- (D) Log level
- (S) Nr of days to keep log files
- (S) Administration port (for admin GUI)
- (S) Administration username (plain)
- (S) Administration password (plain)
- For each implemented transport parameters like
    - (S) Transport type (http)
    - (S) Port number
    - (S) Other transport specific attributes

The SC configuration defines on service level

- (S) Service name
- (S) Service type
- (D) Log level
- (S) Log filename prefix
- (D) Flags that allows / disallows client connections
- (D) Maximal number of active clients (for Request/Response services)
- (S) Size of the subscription mask in bytes (for Subscribe/Publish services)
- (D) Keep-alive timer (in seconds)

The SC-Proxy configuration defines on service level additionally

- Redirected service name
    - (S) Server name or IP
    - (S) Server port

USP has a proprietary binary configuration file format. Some parameters are dynamic and can be changed at run time without USP restart. However this causes a connection loss to affected clients and services.

4.18 Administration

No administration features are available for the client or service. These components are embedded in the application and accessible via SCI API.

The SC and SC-Proxy administration is reduced to proper configuration.

4.19 Operation

The SC and SC-Proxy must provide reasonable tools for local operations. These are:

- Start SC or SC-Proxy
- Stop SC or SC-Proxy

- Enable or disable connection to a particular service
- Change dynamic configuration parameters
- Get information about the actual service state in order to pre-start more single threaded servers.
- Download new SC configuration from a server

Remote operations are not supported.

4.20 Monitoring and Troubleshooting

The SC and SC-Proxy must provide reasonable tools for local monitoring and troubleshooting. A monitoring Web-GUI should be implemented and should provide following features:

- Show state of all services or a particular service
- Show statistic of all services or one service
- Show the log file
- Upload selected log files to a server

The Web-GUI must not make any assumptions about the plug-ins used by the browser. IE V6.0 or later and Firefox 3.0 or later must be supported.

All components (Client API, Server API, SC and SC-Proxy) must provide:

- Multilevel logging facility (log4j) with the following levels:
  - Connection (connects, disconnects, abort, loss of connection, re-connect, etc.)
  - Message header

SC and SC-Proxy must additionally provide:

- Statistical information
  - Actual version number
  - Configuration file loaded
  - Start-up timestamp
- Statistical information per service
  - Total nr. of connections
  - Total nr. of re-connects
  - Total nr. of messages sent
  - Total nr. of messages received
  - Average message size sent
  - Average message size received
  - Maximal message size sent
  - Maximal message size received
  - Actual subscription mask (Subscribing client only)
  - Actual number of connected clients (Publishing server only)
  - Actual number of executing clients (Responding server only)

USP has a utility to get limited run-time information.

4.21 Operating environment

The SC client, server and proxy components should run on any operating system supporting Java 6. Typically the client and proxy components will run on Windows or Unix platforms. The SC and server components can run on OpenVMS Itanium at the first stage and later on Linux.

SC will not run on OpenVMS - Alpha because this platform does not support Java 6 (but only JDK 1.5)

4.22 Visibility

The Service Connector will be published to SourceForge as Open Source Software. Also the source code will be publically available.
Client and proxy components will be used at Eurex and can be also distributed to selected EUREX customers.

4.23 Licensing

The SC will be available under Apache 2.0 software license.
http://www.apache.org/licenses/LICENSE-2.0.html

4.24 Availability

This software is the connection between the banks and the exchange and is classified as business critical. The required yearly availability is 99.9 % on regular business days (Monday to Friday), during extended business hours 6:00 – 22:00.

4.25 Performance

SC must be capable to transport >1000 msg./sec. (request/response, client and server on the same node, message length 128 bytes)

USP/RPC reference benchmark can handle 600 msg./sec, each 128 byte long on AlphaServer 800 under OpenVMS 8.1

4.26 Language

English (U.S.) language must be generally used for all project documentation and software components including user interface, configuration and administration. Other languages will not be supported. Documents in German language are not allowed.

4.27 Flexibility Constraints

It must be possible to add new transport methods. This must not have impact on the API.

4.28 Documentation

SC package must be delivered together with the following documentation:

- Programming Guide
- JavaDOC for Client and Server API
- Operation Guide for SC and SC-Proxy containing the chapters:
  - Concepts
  - Configuration
  - Operation
  - Troubleshooting

5 Glossary

Service

The term service is used as synonym for a piece of application code that implements a certain function.

Client

The term client designates the application requesting a certain service.

Client node

Client node is physical network node (machine) on which the client application resides.

Server

The term server designates the application providing a certain service.

Server node

Server node is physical network node (machine) on which the server application resides.

Message payload

Message payload is part of the transported message containing the application data. SC does not inspect, transform or interpret the payload. It is the responsibility of the communicating partners to define the format, syntax and semantic of the payload.

SCMP => SC Message Protocol

Is a publically presented protocol that transports the messages between client, SC and the server. In the OSI layer model, message transport protocol is immediately above the application protocol layer 7.

Message transport format

Format of messages transported messages between client, SC and the server. It consists of a header and a payload.

RR => Request / Response

This is basic communication style. The client requests an execution of a (remote) service and gets back the results in form of a response. The execution may be blocking - clients waits for the response, or non-blocking (asynchronous) – the clients does not wait for the response. Sometimes it is also called Remote Procedure Call = RPC.

SP => Subscribe / Publish

This is another popular communication style. The client tells the server what he is interested in and gets the data from the server later when some event arise. The execution is always non-blocking (asynchronous) – the client does not wait for the data receipt. This is because the event occurs on the server.

Appendix

Appendix text

Index