Improving Service Availability of Singleton Software Component: Challenges and Strategies

Master's thesis in Computer Systems and Networks

RETA SHIFERAW
YOHANES KUMA

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2016

Master's thesis 2016

Improving Service Availability of Singleton Software Component: Challenges and Strategies

RETA SHIFERAW
YOHANES KUMA

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2016

Improving Service Availability of Singleton Software Component:
Challenges and Strategies

RETTA SHIFERAW
YOHANS KUMA

Supervisor: Roger Johansson, Computer Science and Engineering
Examiner: Jan Jonsson, Computer Science and Engineering

Master's Thesis 2016:NN
Department of Computer Science and Engineering
Computer Systems and Networks
Chalmers University of Technology
University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Improving Service Availability of Singleton Software Component: Challenges and Strategies
Comuter Science and Egineering
University of Gothenburg
Chalmers University of Technology

Abstract

This study is intended to contribute for higher availability property of stateful singleton components with user sessions lasting significantly long duration. As availability tactic, the study focuses on replication. Management of state synchronization of long lasting stateful user sessions during switchover is among the main challenges of this study.
Design research is employed to device a suitable prototype architecture. Via the design research, the challenges are thoroughly analysed, and prototype is designed to incorporate strategies for the challenges. Qualitative research is also undertaken to propose suitable design patterns and object serialization technologies. The prototype is evaluated to observe correctness of long lasting stateful user sessions after switchover is performed among the replicas. In addition, further stress analysis is performed on the prototype to show points of strength and weakest points.

Keywords: synchronization, task migration, availability, replication, stateful connection, session replication.

Acknowledgements

First and for most, we would thank God for the duration of this project. The support and encouragement from Peter Eriksson, Mikeal Krekola, and Björn Östlund is really significant. We also gratitude Roger Johansson and Jan Jonsson for believing in our study effort, and support in the process. Finally, we are thankful for our families and friends for chilling us when we get stuck.

Reta Shiferaw, Gothenburg, 2016
Yohanes Kuma, Gothenburg, 2016

Contents

List of Figures

List of Tables

List of Abbreviations

CAT	Common Architecture Tier
CDMA	Code Division Multiple Access
DCI	Data Context Interaction
DOD	Data Object Library
EIP	Ericsson In-house Pattern
GCM	Global System for Mobile Communications
LTE	Long-Term Evolution
RAN	Radio Access Network
RBS	Radio Base Station
SSC	Stateful Singleton components
WCDMA	Wideband Code Division Multiple Acces

1
Introduction

In modern software systems engineering, analysis of non-functional requirements, such as availability, as part of the design and implementation is important. Availability is a way of describing the responsiveness of software systems by measuring how fast a software system can correctly recover (downtime), and how frequent failure (system outage) occurs [1, 8].
To secure higher availability, most availability tactics focus on two strategies. As first strategy, they employ tactics which could help for faster recovery in order to reduce the downtime. Among tactics for this strategy, replication of system components is the common one [2,3,4,9,10]. This redundant system components are usually referred as replicas [3,4,6,10]. This tactic mainly targets on reducing the downtime, by switching over the system services to another replica, so that the system will be responsive while the failed replica is being recovered. The second strategy involves tactics which are used to make system service outage transparent from service users side [3, 4, 6]. This means that a system service outage will not be noticed from the user side while the system is undertaking tactics for the first strategy. The second strategy is particularly important when the system has stateful components.
A stateful component refers to a component which creates and retains active sessions for each interacting user which could be used for a couple of method invocation for certain period of time [11]. The stateful component will need to track the state of each session as long as the sessions are alive. The opposite of such components are stateless components whereby each invocations from the system user side are completely independent. An example of such stateful components is Ericsson's Radio Base Station (RBS) singleton components. A stateful singleton component (SSC) is a single threaded stateful component responsible to handle certain functionalities of Ericsson RBS architecture. This functionalities will only be served by exactly one SSC executing thread.

1.1 Problem Statement

In this thesis study, the challenges and strategies of replicating SSC in Ericsson AB Department that develop control software for RBS is studied. As detailed in section 2.2, Ericsson RBS architecture is organised in so called 'environments' whereby each 'environment' contains many system components, among which is SSC. The main purpose of the replication is to switchover the services of SSC to backup SSC replica in another 'environment', so that a failed system component in the 'environment' where the SSC is executing could easily be maintained by a technician. This means

that the switchover of SSC services is triggered by a technician, unlike the usual case whereby switchover of a component is triggered by a failure in the component. The replication of SSC is used as a tactic to increase availability of an 'environment' in RBS Architecture.
In order to increase availability, the services of SSC will need to be available during the switchover to SSC replica , but with degraded responsiveness (services may take more than the usual response time but still below the specified time-out for the service). As mentioned before, although there could be two replicas of SSC, only one of them could serve and manage states of users. This means, in order to stay being responsive during or after the switchover, it will be important to make sure sessions and states are synchronized in between the SSC replicas.
Synchronization of sessions and states of SSC in Ericsson RBS is more complex than other similar strategies used for SSC such as [6]. The complexities are the following:

1. The SSC of Ericsson RBS have significantly longer session duration. This implies that for each session there will be lots of data stored in physical memory. This data is kept in heap memory that is allocated for an object instance which encapsulates each sessions. Hence, serialization of object instances, encapsulating the sessions, in strategic manner is important to undertake a correct switchover in between SSC replicas.
2. In addition to a kind of interaction where the SSC will act as a server, SSC will also act as a client to communicate with other system components inside or outside its 'environment'. Such interactions are mainly triggered from object instances which encapsulate the sessions when SSC acts as a server. Therefore, a serialized object instance encapsulating a session will need to continue any communication which are initiated acting as a client towards other components.
3. In order to make SSC responsive during switchover in between SSC replicas, there need to have strategy to immediately serve or queue invocations (messages) which are received by SSC during switchover.

From this point forward, we call the SSC replica which is currently handling sessions and states as primary SSC. The other SSC replica will be called backup SSC.

1.2 Research Question

The main contribution of this study is to design strategies which could efficiently synchronize (and serialize) run-time information in physical memory, so that objects in backup SSC would behave correctly after switchover. It contributes also strategies which are used to keep SSC available with degraded responsiveness during switchover, and to present design patterns which would ease switchover from primary to backup SSC The following are the research questions this study addresses:

1. Is it possible to have controlled switchover functionality for SSC in RBS to preserve component functionality and state, and thus increase system availability?
2. What changes are needed compared to the current way of implementing SSC to better support switchover?

3. What extra functionality needs to be added in SSC to handle switchover?

1.3 Project Limitation

Due to time limitation, designing strategies to increase availability for situations such as failure of a primary SSC is not in this study's scope. In addition, when technician replaces a failed component in an 'environment' whereby the primary SSC has been communicating with the failed component, it is expected that a strategy needs to be employed to update the replaced component to the state of the failed component. However, such situation won't be in this project's scope, rather the project will assume all components which are interacting with SSC are functional when undertaking the switchover.
On the other hand, as mentioned by [12], the reliability (failure rate) and the time it takes to detect the failure, of switchover mechanism software, itself, will also contribute in service unavailability of components. Although, an appropriate software reliability and fault detection analysis will need to be undertaken for the switchover mechanism, it is assumed to be 100 % successful.

1.4 Report outline

The rest of the paper is organized as follows. In the next section (Section 2) we explain the theoretical background with emphasis on replication for availability, and technical background about the responsible department in Ericsson. In section 3, we discuss previous studies that are relevant to this paper. Then, we move on and describe methodology used to approach our research question (Section 4). We explain the results of the study on Section 5, and then discuss the implementation results in section 6. Finally, we conclude as a summary and direction for the future work in section 7 and 8.

2
Background

2.1 Theoretical Background

This section gives the reader theoretical background about availability, replication, and Stateful software components.

2.1.1 Availability

Service availability represents the extent of readiness of the system to deliver correct services to the user and it is often measured in downtime minutes, outage duration and frequency [1] . The formal definition of service availability by TL900 [1] is "the ability of a unit to be in a state ready to perform a required function at a given instance in time or in any period within a given time interval , assuming the external resources, if required, are provided".
Depending upon the critically of the application, requirement of service availability may differ for different software industries. For example, for non critical application availability of 95% may be perfectly acceptable, whereas even 99.5% may not be acceptable for others.
There are different way of increasing service availability; among others duplicating or replicating software component for availability is a common practice in industries. The duplicated software component can take over the operation and continue providing service to the users in case the primary (parent) component fails.

2.1.2 Replication

Replication is a way of maintaining multiple copies of data/components at a replicated server. It is a common technique used to improve the availability of software services. Replication has advantage of performance and data availability, and durability over failures. However, it introduces the issue of consistency.
The two most common replication techniques are active and passive replication. The key concept of active replication is, when a client invokes an operation, each server replicas receive and process the same sequences of the client request. In this type of replication technique the server process the request in deterministic way. Since the client address the server in a group, atomic broadcast protocol is used in active

[1] TL 9000 is a quality management practice designed by the QuEST Forum in 1998. It was created to focus on supply chain directives throughout the international telecommunications industry, including the USA

replication to ensure the same inputs are received in the same order in all servers. The main advantage of active replication is its simplicity and failure transparency whereas deterministic constraint is the major drawback[2].

In passive replication, when a client invokes an operation, only the primary server replica receives and process the operation. Then, the primary replica replicate the operation to the backups in order to update the state changes. In this replication technique, reliable total ordering multicast protocol or other mechanism must be used to ensure either the non-primary replica have the update state or none of them has the update state [2, 3].

The choice of replication technique, active or passive, depends on a number of factors. Some of the factor that should be consider while choosing the replication techniques includes communication and computational cost of operation, and size of the data that needs to be updated for each operation. When we consider passive replication, since only the primary replica performs the operation there will be less computational cost. There may also be less communication cost because only the primary replica responded to each client request. However, the cost of multicasting the state of primary replica in order to update the non primary replica can be costly especially if the state information data is large[3].

With Active replication, since each operation is performed in all of the replicas and all replicas respond to the client, there will be increased computational and communication cost. However, active replication does not require extensive state transfer to update the state change at each replica; therefore there is no cost in this regard unlike passive replication technique [3].

2.1.3 Stateful Software Component Replication

Stateful software components are components that keep state information or status about each connections for the lifetime of the connection. When the user sends a request to a stateful software component, connection session information/object in the form of session Id will be created that trucks the information requested. When the user sends another request, the request operates on the state from the previous request. One of the advantage of stateful component replication is to provide reliable connections.

In passive replication, to provide reliable connectivity, state information of existing connections from the primary replica should be replicated continuously to the backup using different stateful replication algorithms[3, 4, 7, 6]. In case the primary fails, since states are replicated, the backup can easily reconstruct the state and continue the client connection without service interruptions.

2.2 Technical Background

This section describes about the responsible department in Ericsson whereby this study is undertaken and the role of SSC in Ericsson RBS architecture.

2.2.1 Ericsson RBS

Ericsson RBS is one of the core product of Ericsson AB. RBS is a system which plays important role in between the antennas which receive signals from user ends and other products which deal with user service rights. Figure 2.1 shows high level description of Long-Term Evolution (LTE) Radio Access Network(RAN) [14]. The user ends are shown as UE. The user service rights are handled by Evolved Packet Core.

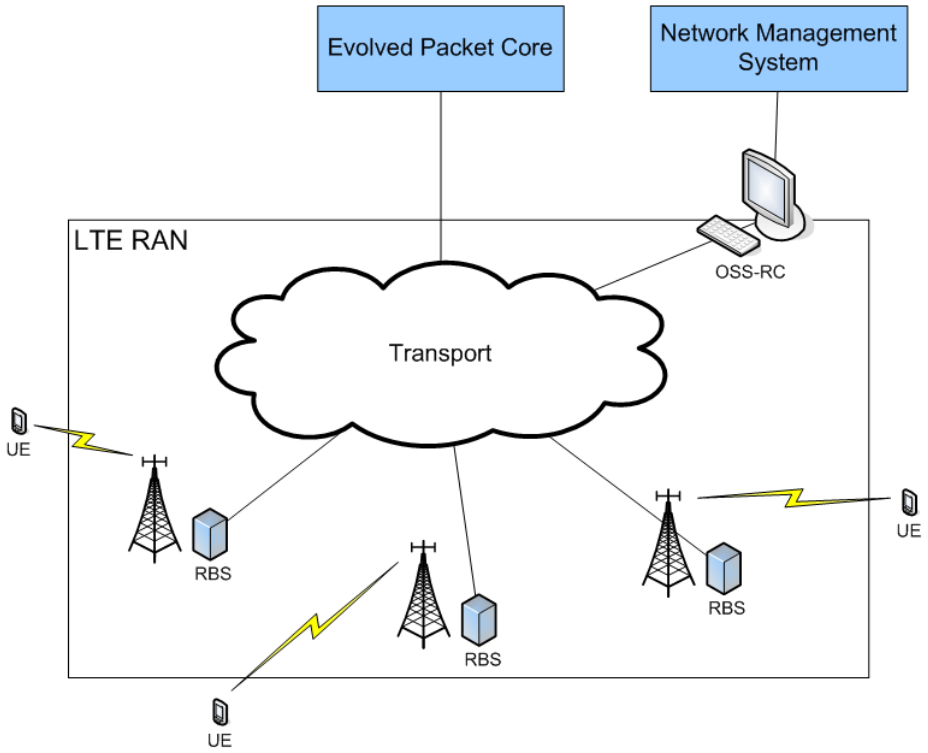

Figure 2.1: Position of RBS [13].

2.2.2 Ericsson RBS CAT

Common Architecture Tier (CAT) is a function module in RBS which is used to group common functionalities used by various implementations of standards for wireless communications such as Code Division Multiple Access (CDMA), Wideband CDMA (WCDMA), Global System for Mobile Communications (GSM), and LTE. There are around 60 teams involved with CAT. Each team has 6-8 people working as cross-functional teams.

Ericsson RBS CAT is composed of various software components. A software component is a package that encapsulates a set of related services. The services are provided over (a set of) interfaces. A software component consumes functions/services from other components over required interfaces. Interoperability is secured by compatibility of the provided and required interfaces[13].

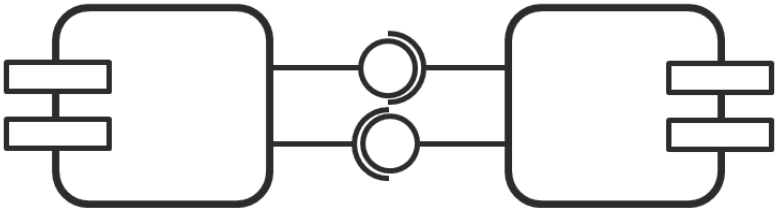

Figure 2.2: CAT software components interaction [13]

2.2.3 Ericsson RBS CAT SSC

A group of related software components in Ericsson RBS CAT are hosted by a hardware system which is known as 'environment'. Among software components in as 'environment', SSC is one of them. Figure 2.3 shows a scenario of an 'environment' in CAT. An 'environment' is shown as green boxes. As shown in the Figure, the red line shows the communication channel toward the core network (which is indicated as Evolved Packet Core on Figure 2.2). The black lines shows operational status, accounting, and maintenance channel. This channel is mainly used by Operational center and technicians (which is shown as Network Management System in Figure 2.3). In this scenario, OAM & Conf component is a SSC.

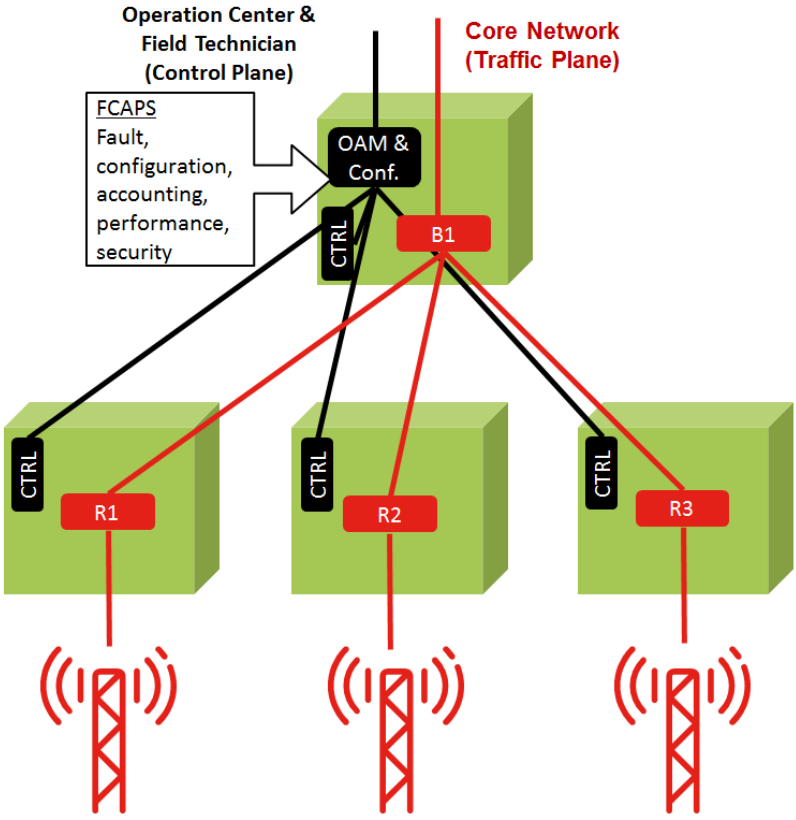

Figure 2.3: CAT 'environment' [13]

As indicated on Figure 2.3, the future goal of the 'environment' shown on Figure 2.4 will be to have another replica of an 'environment'. The OAM & Conf SSC in

the primary 'environment' replica could be switched over to backup 'environment' replica.

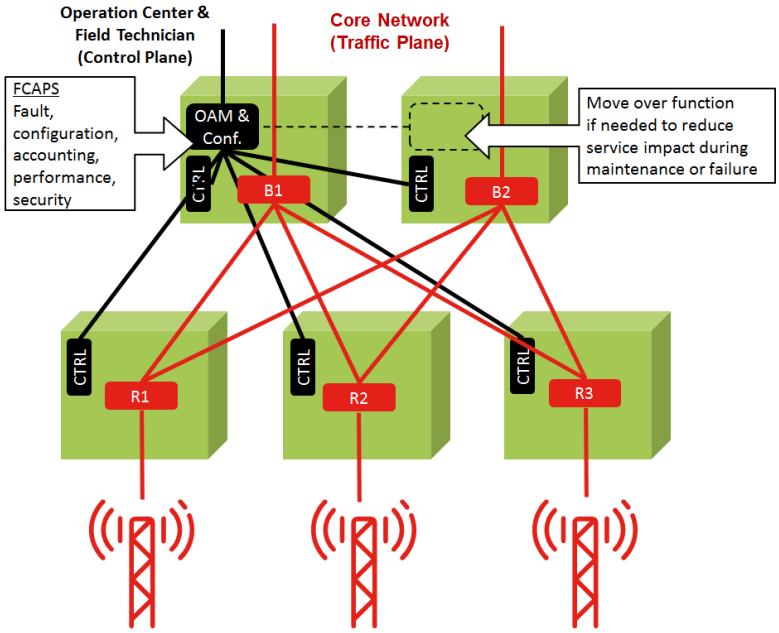

Figure 2.4: Future plan for CAT 'environment'

An SSC will interact in various manner (acting as client or server) with various components(singleton components or non singleton components). An example scenario of a sample SSC is depicted in Figure 2.5. Component A is the sample SSC which will act as a server for interactions with singleton component D, and non singleton components B; while in case of C, A will act as client. In addition, Component A will act as a client, using services provided by non-singleton Components E and F.

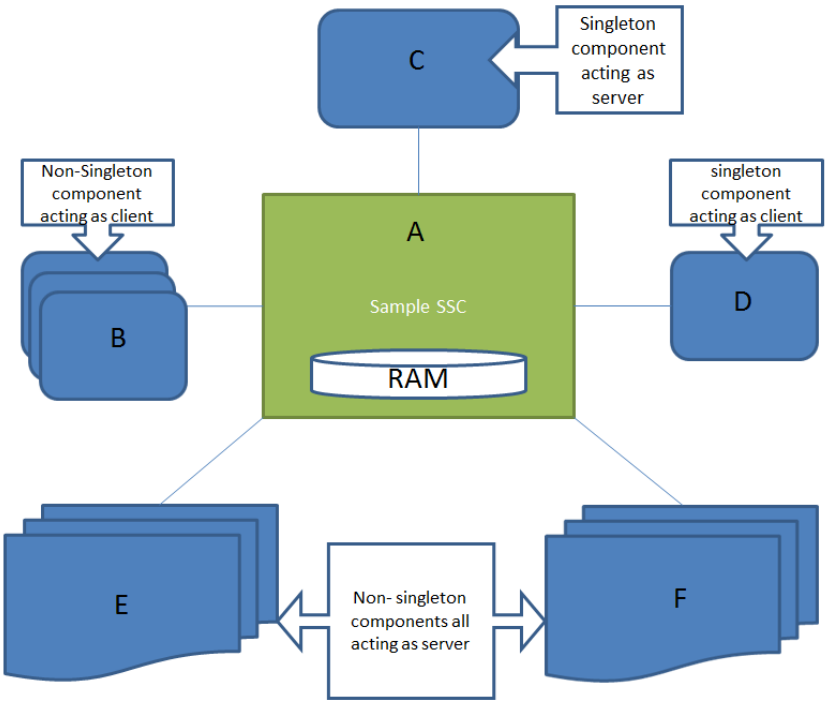

Figure 2.5: Sample SSC

All connections to/from Component A in Figure 2.5 are stateful, which means a connection session will be required to be established before any kind of collaboration is intended. Under normal circumstances, connection sessions are kept alive as long as Component A is functioning. In duration of one minute, Component A will process 5000 messages with a size of 100 bytes per message. Component A saves or track session and configuration information related to each connection as well as processed coordination information related to several connections in physical (RAM) memory. This makes Component A to manage 5 to 50 MB of RAM.

3
Related Work

One of the most used methods of achieving availability requirement is replication of software components. While replication is a choice for availability, handling stateful sessions and requests at the time of switchover for SSC is an important challenge. This thesis project is related to Wu, H. & Kemme, B. 2005 [6] and Huaigu Wu 2008 [7] which propose a replication tool that is able to handle stateful sessions and transaction by the time the primary stateful application server (AS) replica crashes. The proposed algorithm is based on the Enterprise JavaBean (EJB) in J2EE architecture.The algorithm by [6,7] has client, primary, backup and fail-over parts. Each request submitted from client to primary (server) is intercepted by client replication algorithm and executed at current primary. State changes are recorded at primary and propagated to the backup once transactions are committed. Upon a failure exception, the client replication algorithm re-sends client request to the new primary. When the primary crashes, a backup will take over and becomes the new primary. After switchover, a client request to the primary will be checked whether the request has recorded a corresponding response/request pair. If the client request is recorded the new primary will respond immediately, if not the request will be processed accordingly.

Even if the algorithm proposed by [6] address similar problem in terms of handling stateful sessions during switchover, the assumption made by [6] and the platform in which the tool is based is completely different with this thesis project. The algorithm by [6] is designed mainly for transaction execution that has external database system in which the volatile data such as session information are maintained by AS. Once the transaction is committed the data will be saved in the external database and the session will be disregarded. But in this thesis project, the singleton component has no database related transaction execution. Once the connections are established, the session information and data are maintained by RAM, and this run time information at the RAM need to be serialized at the primary and deserialized at the backup during switchover process. On the other hand, singleton components in RBS are designed by using C++ based platform, unlike the tool proposed by [6] which is based on J2EE architecture.

This thesis project is also related to [3] that proposes a schemes for both passive and active replications that allows operations to be performed on an object while a state is being transferred from primary to backup. In passive replication, while the states are being transferred the primary will not stop further processing operation. Instead, the scheme uses Replication Manager that logs a “postimage”, the value of the updated parts of the state after update, of each update that it performs. Since updates can be performed at the primary while states are transformed to the backup,

the Replication manager's first transfers the existing state and then it transfers the postimage of the update to the new primary. The new primary will reconstruct the state by applying the postimage to have a consistent states between the replicas. This state transfer mechanism is conceptually similar to the thesis project except that instead of logging the state, the primary will forward the state to the backup and states will be queued until switchover is completed. Once switchover phase is completed the states will be dequeued and served at the new primary.

4

Methodology

This section describes the methodology used, qualitative and design research, to approach the research questions, and the motivation behind choosing this methodology.

4.1 Qualitative methodology

This methodology is employed for two purposes. The first is to discuss design patterns which could help answer research question 1. The authors have reviewed literature that provides insight into design patterns to ease switchover process. The results are discussed on section 5.1.

The second purpose is to provide the design research. For this purpose, the authors reviewed literature that provides insight mainly into session and state replication techniques, and object serialization. The main objectives are to investigate the current state-of-the-art of designing a replicated SSC and switchover mechanisms, and to investigate open source solutions which potentially will contribute in synchronization and serialization of sessions and states in C++. Section 5.2 presents the details about the results in C++ serialization.

The authors has also conducted an interview with a senior architect within Ericsson in order to gain some insights about pattern and technologies, which are used for availability that can contribute to the research questions of this study.

The outcome of the study is used as an input for design research.

4.2 Design Research

The authors follow the work of [3] as a guideline to the design research. The purpose of the design research is investigate answers to research questions 2 and 3. During this methodology, the authors designed and implemented a best suited replication and switchover strategy considering Ericsson's RBS SSC, and evaluate them.

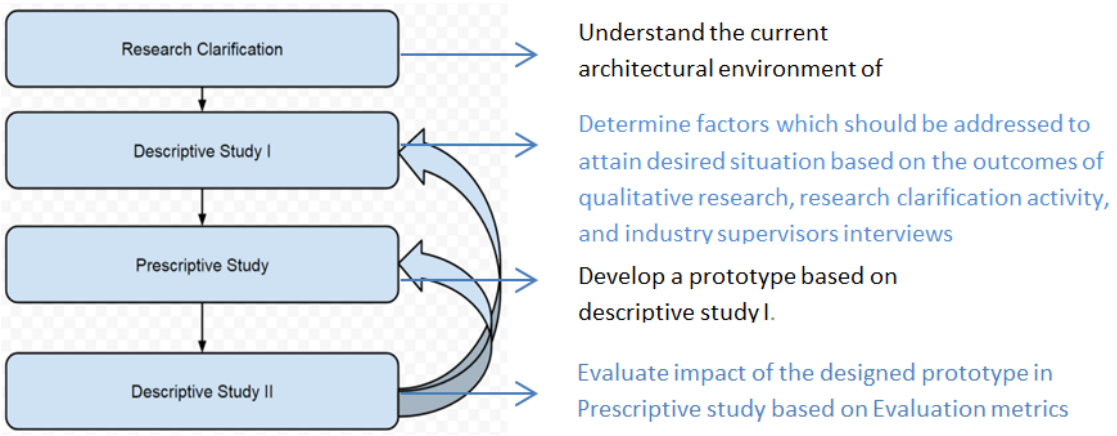

Figure 4.1: Design research method

As can be seen in Figure 4.1 the design research consists of 4 stages.

4.2.1 Stage I (Research Clarification)

The design research starts with research clarification. The main purpose of this activity is to understand the current architecture of Ericsson RBS SSC. The authors accomplished this activities as follows:

- Walking through one sample SSC product and documentation, and understand high level structure of connection sessions and component interactions.
- reviewing communication frameworks, and data communication standards to/from SSC

4.2.2 Stage II (Descriptive study I)

In Descriptive study I activity, the authors used the outcomes of qualitative research, research clarification activity, and an empirical analysis from interview and industry supervisors discussion to understand the existing situation and determine the factors which should be addressed to attain the desired situation specific to Ericsson RBS SSC. After undertaking this activity, the authors have determined three factors that needs to be addressed: (1) name serving of SSCs, (2) handling incoming service requests during switchover, and (3) handling outgoing service requests (incoming service response) during switchover.

4.2.3 Stage III(Prescriptive study)

In Prescriptive study , the authors designed detailed specific strategies, and developed a prototype based on a reference architecture (Figure 4.2). The detailed design strategies are presented in section 5.3. Before the strategies are implemented, the reference architecture in Figure 4.2 is implemented. EqCoord is the SSC that needs to be "moved" from one execution environment to another during switchover. The main responsibility of this component is to configure available equipment resources (EqResourceR and EqResourceB) into equipment groups and report its status. Each EqUser request for creation of its own equipment group (this is the operation which

will trigger creation of new session). Each sessions are encapsulated by using eqGroupInstance object instance. The equipment resources assigned to a specific group report their status (FAULTY, OFFLINE, OPERATIONAL) to the owning session (eqGroupInstance). eqGroupInstance will, then, compute its group state (FAULTY, DEGRADED, OPERATIONAL) based on the aggregated states of its equipment resources. The eqGroupAcceptor is responsible for accepting EqUser group creation request and creation of unique eqGroupInstance for each request (session). Once the eqGroupInstance is created, EqUser will subscribe to group status indication messages by directly communicating eqGroupInstance. Each eqGroupInstance are uniquely identified in EqCoord, and are responsible for creating EqCoordBasebandR and EqCoordRadioR. These two types of objects, EqCoordBasebandR and EqCoordRadioR, are responsible in receiving and reporting the status of EqResourceB and EqResourceR, respectively, back to the eqGroupInstance. There is one executable for EqCoord. For each EqUser, EqResourceB, and EqResourceR, there is one executable per instance. Each executable runs in its own process memory. For communication between executables, Enea LINX Interprocess Communication [24] is used. This communication platform guarantees FIFO (first-in-first-out) ordering of messages (messages from one sender will be delivered in the order they are sent. However, the order between different senders is not preserved). There is however a filtering mechanism allowing the receiver to select only specific messages received.

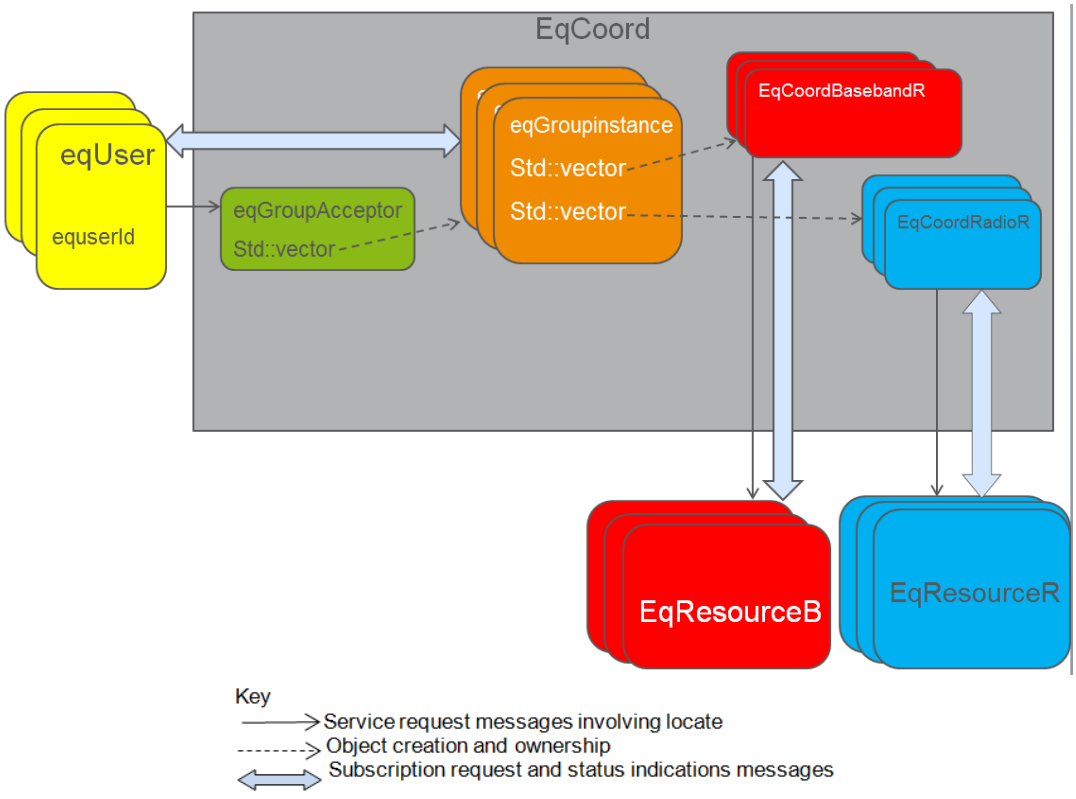

Figure 4.2: reference architecture for prototype

4.2.4 Stage IV (Descriptive study II)

In order to investigate the impact and evaluate the prototype, a Descriptive study II is performed as next activity. Here, the main evaluation metric was correctness of whether the new primary is correctly behaving as the old primary after switchover. This is evaluated by regular test via predetermining the expected behavior during and after switchover, and confirming if the expected behavior is observed. The expected behavior are perceived by reading logs from the EqUser and resources. In addition, the authors has conducted a stress test. The actions involved during stress test were, mainly, sending a lot of resource intensive messages and observe the behaviour.
The outcome of these evaluations lead to have iteration from Descriptive study I with additional empirical analysis; or to have iteration from Prescriptive study where the factors identified from the evaluation will be used to revise the prototype. Parallel execution of the activities were performed for more efficient activity execution.

5

Results

This section present the results of qualitative and design research.

5.1 Design Patterns

This section describes the result of qualitative research of design patterns which will best suit in replication and switchover of SSC replicas. The study has mainly studied the suitability of an Ericsson In-house Pattern(EIP). EIP is hugely inspired by Data Context Interaction (DCI) pattern [26, 27].

DCI pattern is designed for easing the complexity of object-oriented languages, such as C++. Traditional object-oriented paradigms have a usual goal of encapsulating end user mental models with objects, while leaving algorithm implementations spread out in different objects [26]. This leads to undesirable object cohesion and complex object interactions. DCI's main impact would be separation of data models encapsulating the user mental models from algorithm implementing code. Data models are designed as simple object that will only focus on what the model is. At run-time, the system will create a context for certain algorithm implementation (Interaction). The context will collect the required Data objects from memory and will assign them roles, which will interact depending on the behavior of the algorithm [26, 27]. Figure 5.1 shows simplified version of DCI.

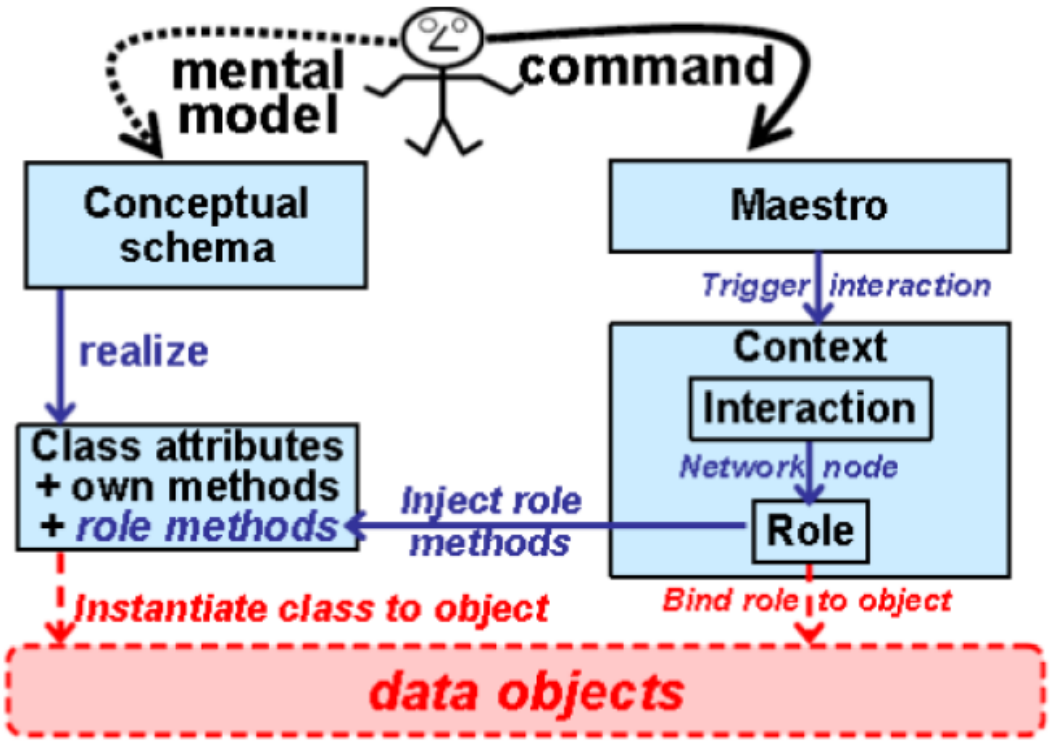

Figure 5.1: DCI paradagim [28]

Inspired by DCI, EIP has added some more features. As shown in Figure 5.2, it has four main conceptual components. The service broker follows broker pattern [29]. The broker will be responsible to publish service request/response callbacks or endpoints. When a service request is invoked, it will define a run-time context (similar to DCI's context) by which service realization will be executed. The repository component will act as a service abstraction layer for accessing data store. This is inspired by repository pattern [30], which will enable the system to have flexibility in terms of how the domain data entities could be stored(for instance, as list of objects, or compressed serialized binary data of objects). Service handler component (class or object) will be a short living run-time context execution for executing service behavior executions. It will get the data entities required for performing behavioral executions (DCI's roles) from repository via service broker. For better organisation, service handlers sharing a particular data entity object are grouped together. This will enable repository to hold a cache (for example, object in memory) of the data entity object, or perform blocking or queueing for a resource limited data entity object.

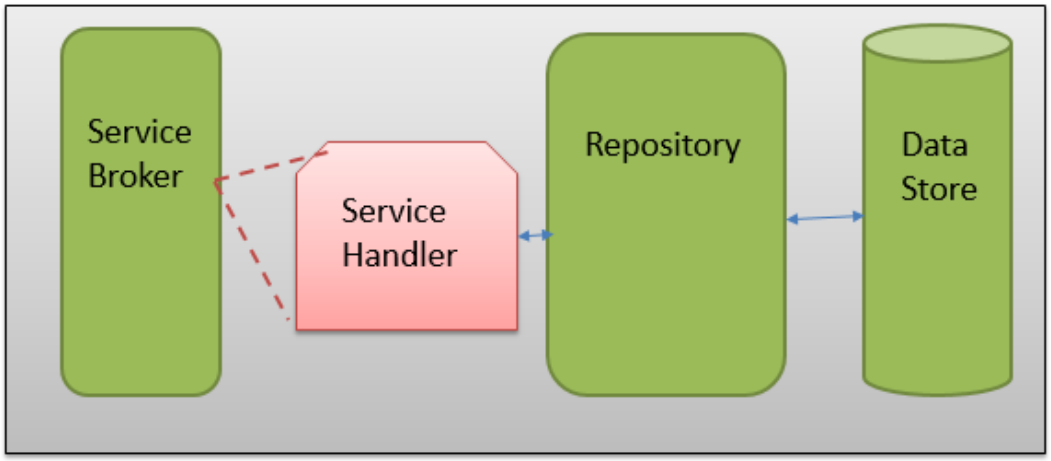

Figure 5.2: EIP paradagim

The authors thought EIP will ease switchover among replicas of SSC. As could be noted, the relatively long-living objects (data entities) are not spread out in the system, rather are kept well structured in data store. This will make easy transfer of state information (object) among SSC replicas during switchover. In addition, it will ease strategy design for incoming or ongoing services during switching over to another replica.

5.2 Object Serialization Patterns

This section details object serialization technologies in C++ which are suitable for this study. Serialization is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer, or transmitted across a network connection link) and reconstructed later in the same or another computer environment [15]. The formats which are currently available in C++ serialization alternatives are binary, ASCII, XML, and JSON.

Serialization in Java and C# is relatively easier since there is a built in support for reflection. Reflection is the mechanism for encoding execution states as data [16]. A programming language which supports reflection will have the ability to manipulate this data which represents the state of the program during its own execution. In terms of serialization, reflection allows a program to traverse members of a serialized object and, for each member, it helps identify whether the member is a reference and what is the type of its target [17]. Standard C++ library does not support reflection. Therefore, serialization technologies in C++ will need to have strategies to identify the types of pointer members in application classes during serialization.

There are four technologies which are used in C++ serialization. The technologies are categorized into two patterns: Data structure focused and object collection focused serialization. Each are discussed in next sub sections. The chosen technology for prototyping and its rationale is discussed in section 5.2.3.

5.2.1 Data structure focused serialization pattern

This pattern uses a generic data structure as library which could be used by application programmers. The main objective is to abstract out complexities which are arised due to not having reflection from application programming. Serialization technologies will be the one which provide generic data structure and track objects and references, so that serialization and deserialization code in application programs could be smooth and simple. Application programs will need to specify and generate code by using provided interfaces from the serialization technologies.
There are two serialization technologies in C++ which use this pattern. The following subsections review the technologies.

5.2.1.1 Data Object Library

C++ Data Object Library (DOL) is a relatively old C++ library which is based on intrusive data structures [18]. Intrusive data structures have support for bidirectional, many-to-many related object entities [17]. The library has a code generator called zzprep which will generate the required data structures during preprocessing stage of code compilation process.
It has a support for binary and ASCII serialization formats. During binary serialization, saving to disk (encoding objects to binary) is relatively fast, because there is no formatting or massaging of data. Objects are stored as blocks of bytes. When deserializing the objects from binary data, objects are allocated in new places in memory, while keeping a table of their old and new address. After all objects are read to memory, DOL walks through them, and resets the internal pointers to new values. Although the serializing part is faster in both ASCII and binary formats, the deserialization takes significantly longer time[17]. The library is licenced under no-nonsense free-source licence.

5.2.1.2 Protocol Buffers

Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data [19]. Application programmers define data structure in so called .proto file. A code generator will then compile .proto file to generate code.
Serialization is done via binary wire format. Textual protocol buffer message will be encoded to binary wire format, where a message just uses the field's number as the key. The name and declared type for each field can only be determined on the decoding end by referencing the message type's definition. This space efficiency is possible due to the huge fact that Protocol Buffers enables C++ reflection. Protocol Buffers is licenced under Apache 2.0 License.

5.2.2 Object collection focused serialization pattern

In this pattern, active object collection will be done, and the objects will be structured as Graph (where one or more root object(s) will be assigned) [17]. A java

style Breads-first or depth-first search could be done to access the objects. Serialization technologies which use this pattern usually provide an API to indicate the root objects. The main challenge in this pattern is to make sure identical objects are not serialized more than once, which could happen when two object referencing pointers are pointing towards the same object.

There are two serialization technologies in C++ which use this pattern. The following subsections review the technologies.

5.2.2.1 Boost

Boost is a collection which provides free peer-reviewed portable C++ source libraries, among which is Boost serialization library [17]. It supports binary, ASCII, and XML formats. It is distributed under the Boost Software License, Version 1.0.

Application programs needs to provide serialize() function which will list members to be stored, and invoke serialize() function of base classes. In order to make sure objects are not serialized multiple times, Boost uses object tracking using memory address [20]. Performing object tracking requires extra booking, since every object need to be tracked whether it has already been serialized or not. This comes at the cost of runtime performance and increases the code complexity of the library considerably. It also complicates application code since it adds extra considerations about tracking types.

5.2.2.2 Cereal

Cereal [21] is C++11 header only serialization library. Header only means it will not add complexity during linking process [22]. It supports compact binary encoding (binary archives will ignore name-value pairs and only serialize the values), XML, or JSON. It also has an interface for hybrid solution of saving base64 binary data in XML format. The binary format makes no attempt to ensure endianness across different architectures. If data will be read on both little and big-endian machines, a special format called portable binary should be used, which tracks the endianness of the saving and loading machines and transforms data appropriately. It has slightly more overhead than the regular binary format [21].

Likewise as Boost, application object will need to implement a template serialize() function to list object members which are required to be serialized. However, unlike Boost, Cereal fully supports serializing classes that use inheritance, via utilizing cereal:: base-class method to cast the derived class to the base class instead of calling serialize function directly.

Cereal doesn't support traditional C style pointer references; rather it only supports pointer references which are encapsulated by C++11 smart pointers. This feature enables Cereal to not worry about object tracking techniques.

Based on experiments done by [23], Cereal has a 100% performance advantage over Boost and Protocol Buffers. Cereal is licensed under the BSD license.

5.2.3 Serialization technology choice and its rationale

The object structure in RBS SSC mostly use intensive compositional pointers. This makes this study to mainly focus on utilizing Object collection focused serialization pattern. The compositional pointers are created by using smart pointers. Due to its performance advantage over Boost, Cereal is decided to be used as a serialization technology in prototype implementation.

5.3 Prototype Design

This section describes selected replication and switchover strategies for addressing factors identified by descriptive study I (section 4.2.2). The duration from the point in time where the system starts preparing the current primary replica to point in time where the used to be backup replica has become primary replica (started serving old and new sessions) will be referred as switchover process. In general, the strategies are designed with an assumption that there will be three phases during switchover process: preparation, synchronization, and dequeueing phase. The strategies are designed with the following assumptions:

- The long living sessions are expressed with the presence of the respective eqgroupinstance for each EqUser. We can say that 'EqUserId + EqCoordId' is our session identifier. It is expected that the switchover activity will result in change of session identifier. However, this do not implicate that the session is restarted since the respective eqgroupinstance object for the session is kept alive.
- LINX guarantees FIFO ordering of messages (messages from one sender will be delivered in the order they are sent. However, the order between different senders is not preserved). There is however a filtering mechanism allowing the receiver to select only specific messages received.

5.3.1 Architectural Design Overview

The strategies are designed and implemented by considering two additional software components to the reference architecture: a second SSC (EqCoord) replica, and HAController. HAController component is responsible to control the phases of switchover process. It has a dedicated interface towards both replicas (Figure 5.3).

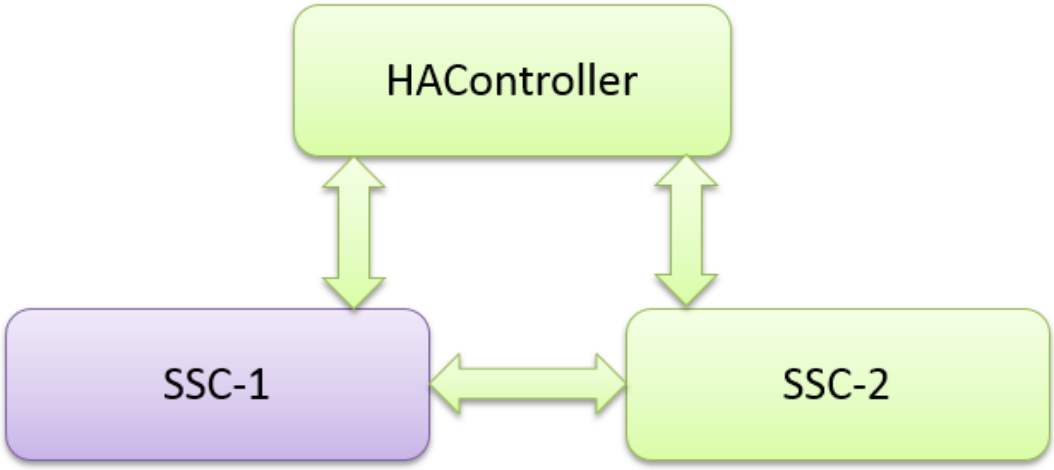

Figure 5.3: replication architecture

At any point in time one of the replicas will be in primary replica state, while the other become in backup state. Until switchover process is initiated by HAController, the backup replica will not receive service requests. In other words, the interface between the replicas will be used only during switchover process.
Figure 5.4 shows a simplified run-time view of stateful sessions in normal situation, where no switchover process is active. As pointed before, eqGroupInstance is an object which encapsulates each stateful session with the user. Each eqGroupInstance holds a collection of eqCoordR or eqCoordB type object, which each encapsulates the stateful connections towards each of the configured eqResourceR or eqResourceB resources for each user.

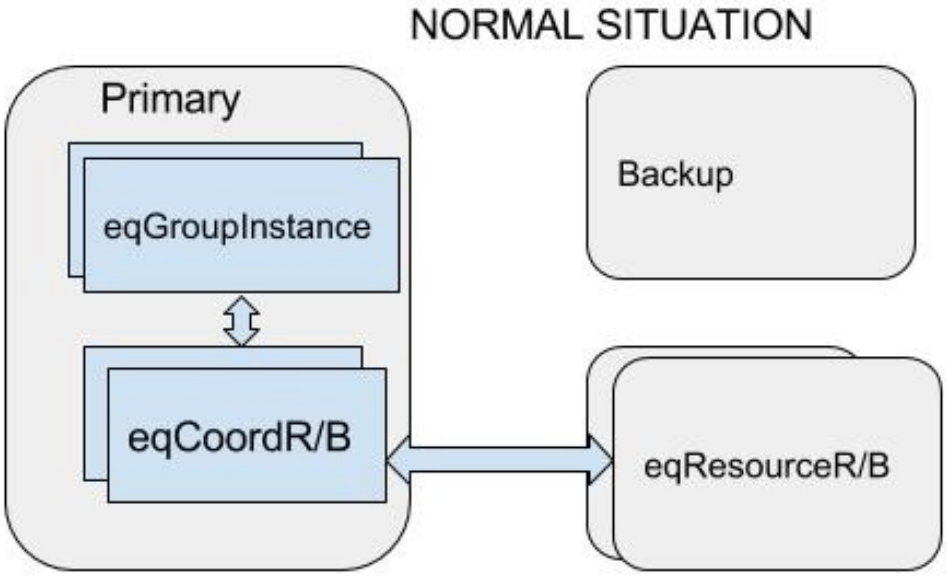

Figure 5.4: Run-time situation of replication architecture when the system is not in switchover process

In such scenario, design of the switchover process needs to guarantee that all session abstracting objects and configurations are synchronised. Such guarantees makes

sure the backup replica to correctly behave and serve current active users. However, the switchover process will need to balance two important trade-offs:

- network bandwidth usage: serialization of objects in primary replica, and sending serialized binary data to backup replica. The backup replica will deserialize and obtain objects.
- computational cost: Instead of serialization, make the backup replica to freshly create objects by reading some configuration file.

The prototype is designed considering this trade offs. Objects, such as eqGroupInstance, which are computationally costly to be recreated are serialized. On the other hand, eqCoordR or eqCoordB type objects are designed to be recreated in the backup replica. This effectively means, after a particular eqGroupInstance object is successfully deserialized on backup replica, both the objects in the primary and backup replica will subscribe for their configured resources (Figure 5.5).

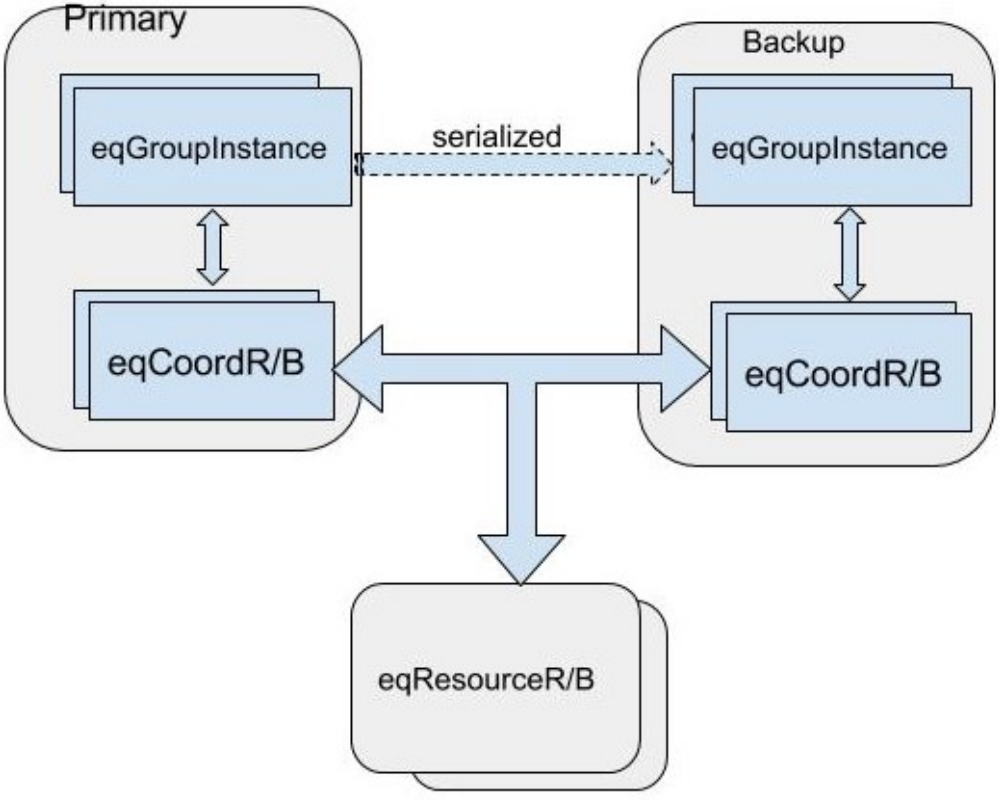

Figure 5.5: Run-time situation of replication architecture when the system is performing serialization in switchover process

However, such duplicated connections will endure higher network bandwidth usage. Therefore, as depicted in Figure 5.6, as soon as backup replica has guaranteed any synchronization of required data for each eqGroupInstance, stateful connections from primary replica towards resources will be terminated.

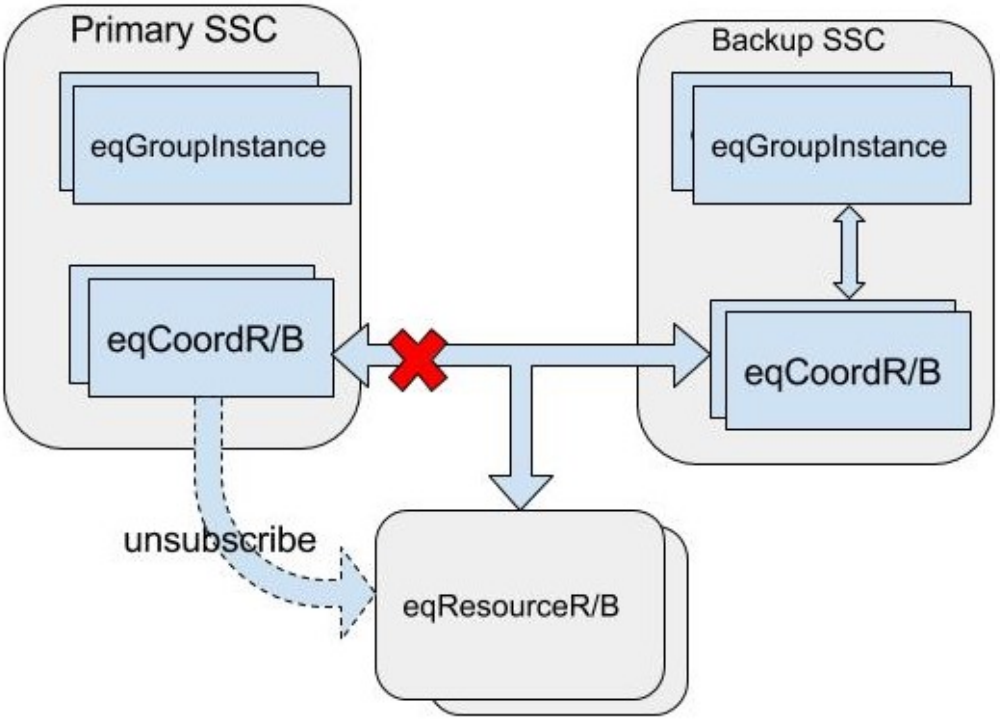

Figure 5.6: Run-time situation of replication architecture when Primary SSC terminate session towards resources in switchover process

5.3.2 Switchover Process Phases

For simplification and management reasons, the activities in switchover process are categorized into three phases: preparation, synchronization, and dequeueing. Figure 5.7 explains phase transitions during and after switchover process. The main goals of each phase is indicated below:

- Preparation Phase: system state view consistency between replicas is guaranteed during this phase. This means, both replicas will have a consistent view of current active stateful sessions in primary replica, including sessions abstracting connections towards resources.
- Synchronization Phase: message consistency is guaranteed during this phase. Please refer section 5.3.3.3 for details.
- Dequeuing Phase: This phase starts servicing queued messages from resources while guaranteeing service consistency of the system. Please refer section 5.3.3.3 for details

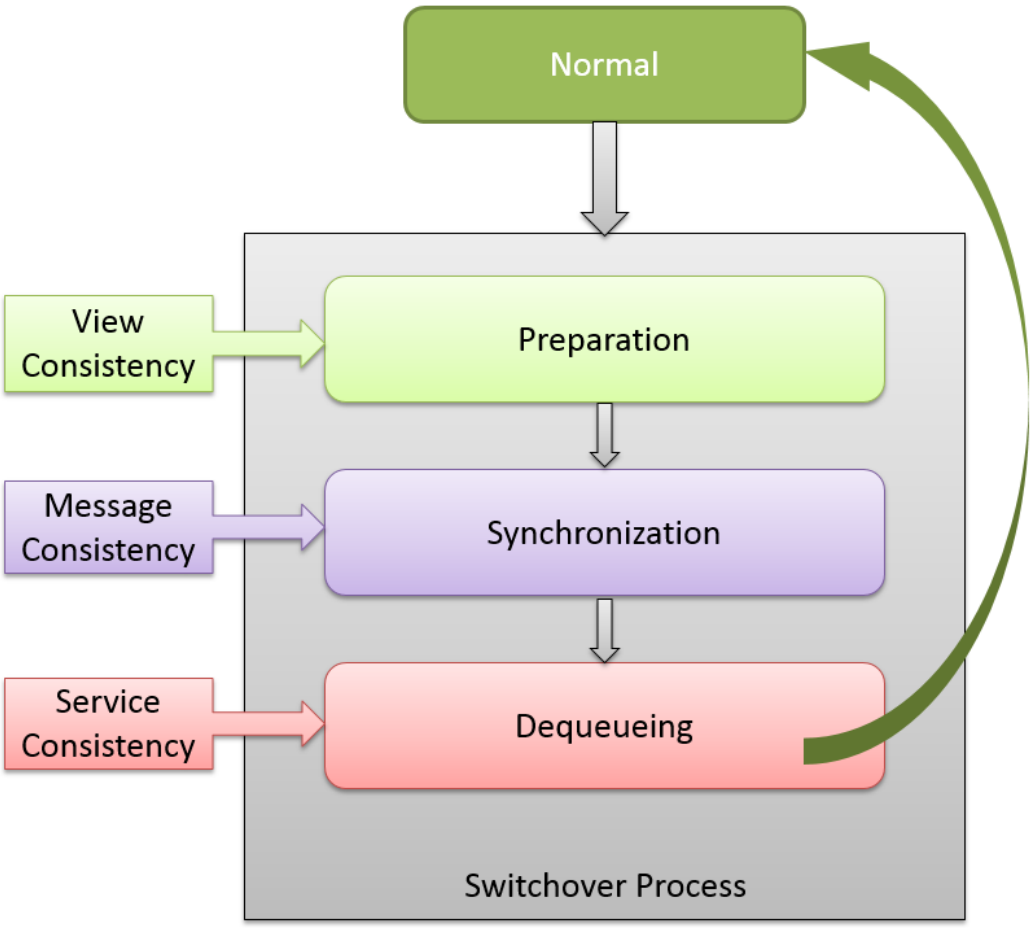

Figure 5.7: switchover process

The switchover process phases are controlled by control messages owned by HAController component. A sequence diagram showing general overview of the switchover process is shown in Figure 5.8.The HAController get confirmation/rejection for the three phases. Each proceeding phase is started after all replicas has reported about finishing preceding phase to HAController .

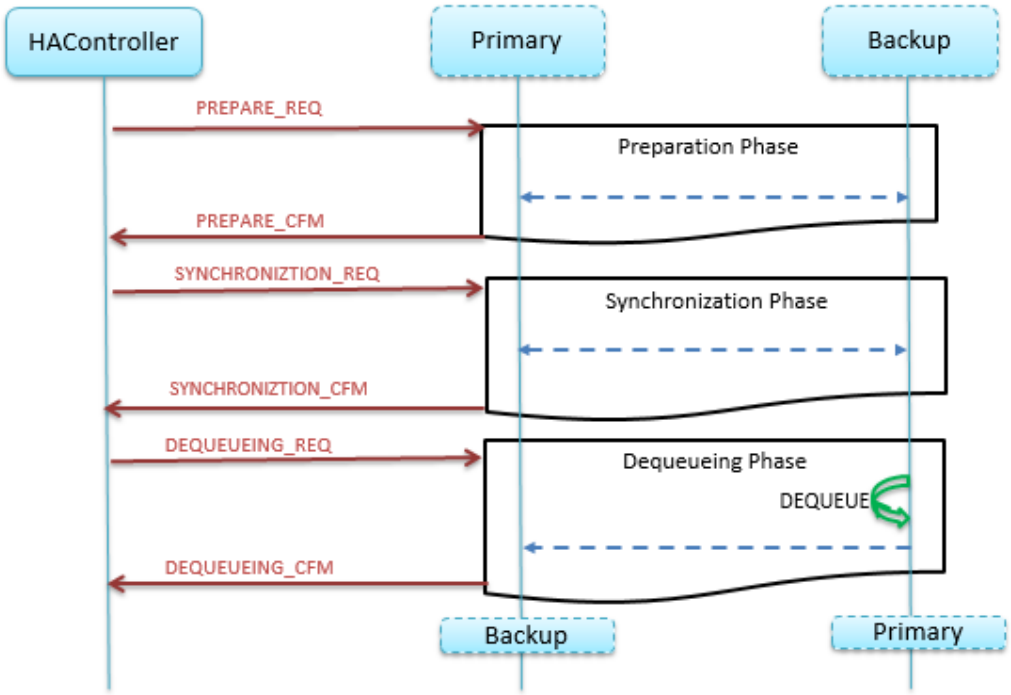

Figure 5.8: Switchover activity overview

5.3.3 Prototype Design Decisions

As pointed out in section 4.2.2, the prototype design during prescriptive study stage of the design research is performed after determining the factors which need to be addressed by the prototype. Three main factors were determined after careful qualitative research and non-structured interviews with industry supervisors. The following subsections clarifies each factor, and presents the design solution implemented on the prototype.

5.3.3.1 Name Serving of Replicas

It is clear from the current reference architecture that, name or address of the SSC (EqCoord) can be easily hard-coded to user process (EqUser) configurations. This is possible since there is just one SSC process in the reference architecture. However, as discussed in section 5.3.1, this solution can't be used anymore since there will be two SSC (EqCoord) replicas, whereby at any given time one of them could be the primary.
The prototype has devised two solutions depending on the state of the system:

- At system initialization state of the reference prototype, users use the name (address) of the primary SSC replica, which is configured on system configuration file.
- Once the prototype system has initialized, it will be up to SSC replicas to inform users having stateful sessions about change of address. During synchronization phase of the switchover process, all users which have stateful sessions in primary SSC replica will be informed by backup SSC replica to change SSC addressing.

Figure 5.9 shows the solutions in sequence diagram. The user in the prototype (eqUser) reads configured primary SSC (EqCoord) replica name (address). Using that address it sends service request. The user process will cache primary SSC replica address for future communications. The cache will be updated to the soon to be primary SSC replica address when it receives a 'change address' information message from backup SSC replica, which will be primary after switchover process.

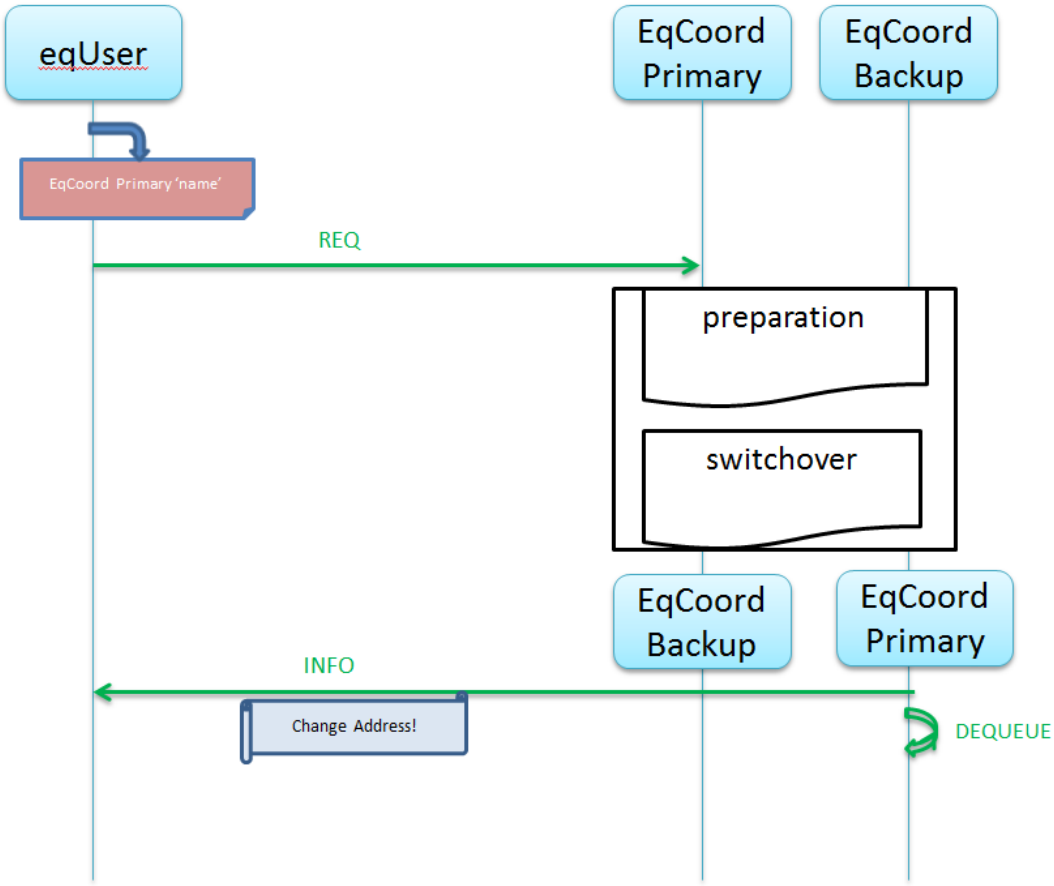

Figure 5.9: Name serving strategy

5.3.3.2 Service Request Handling

At any point in time, user processes which have stateful sessions with primary SSC replica will request for services. The main service request operations in the reference architecture are group creation requests and group state subscription requests. Such requests have soft real-time properties, meaning that they shouldn't be rejected or take longer time than threshold service response time. The service requests can be categorized into two:

- Non-state changing requests: service request which doesn't cause state changes on current sessions (eqGroupInstance objects) on primary SSC replica. For instance, group creation requests [which requests for creation of a new object (eqGroupInstance) for encapsulating allocated resources for the requesting user], will only result in creation of new eqGroupInstance object.

- State changing requests: service request which result in state changes on current sessions (eqGroupInstance objects) on primary SSC replica. Group state subscription requests, for example, is made to inform the object abstracting the session of the requesting user process that, it (the requesting user) would like to receive push notifications whenever the aggregated allocated resources state changes.

Enabling the primary SSC to continue serving service requests during switchover process will have various disadvantages. One could disagree, specially for non-state changing requests, since such requests can be allowed to be serviced by primary SSC during preparation phase. However, this could result in non-deterministic duration of preparation phase, which could affect the response time of state changing requests. On the other hand, serving state changing requests during switchover process will result in inconsistency of state (for group subscription request, knowledge of whether or not the respective user has subscribed) between eqGroupInstance object in primary and, serialized eqGroupInstance object in backup SSC.
To avoid non-deterministic duration of preparation phase and session objects inconsistencies, service requests will not immediately be served if they are received during switchover process. Instead, as shown on Figure 5.10, the strategy for service requests of both types during switchover process is to seamlessly forward to the backup SSC. During forwarding, the current primary SSC will keep the sender address to the requesting user, so that the backup(soon to be primary) SSC will see the service request message is sent directly to its address.

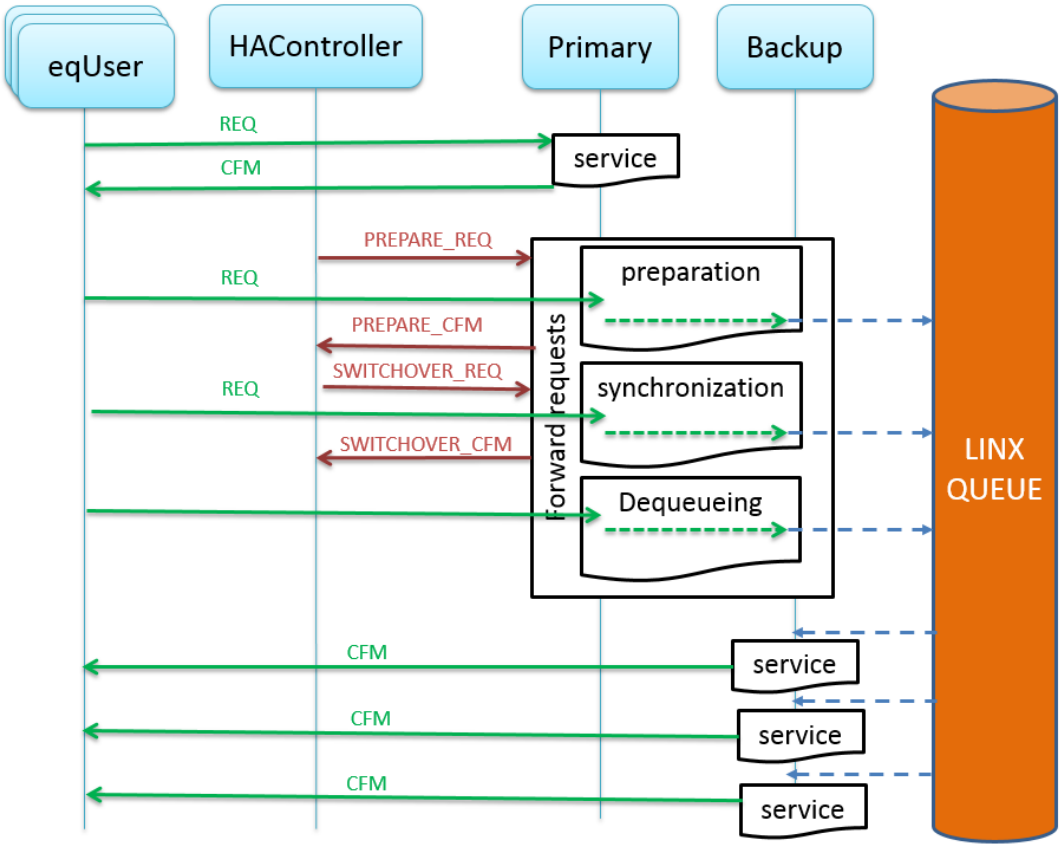

Figure 5.10: Service request handling strategy

As could be seen in Figure 5.10, the backup SSC is equipped with a mechanism which will detect and queue forwarded service requests, while actively responding to control messages from HAController. The LINX queue guarantees service requests are serviced in the correct order as they are received in old primary SSC, since there is a guaranteed FIFO order during forwarding of requests from primary to backup SSC during switchover process.

5.3.3.3 Service Response Handling

As discussed in section 4.2.3, the main use case in the reference architecture is that each user with a configured stateful session with SSC will get allocated resources. The SSC object (G) encapsulating a user's session will act as a client and configure a stateful session towards each resource allocated for a user. Each of such sessions will be encapsulated by an object instance (R1 ... Rn). The state of each user will, thus, be the aggregated group state, which will be computed by considering the state of all R type object instances encapsulating stateful sessions towards configured resources for that user. Whenever their state is changed, each configured resource will send its current state as a service response message to configure R type object in G type object. Figure 5.11 shows a scenario for one user with 3 allocated resources.

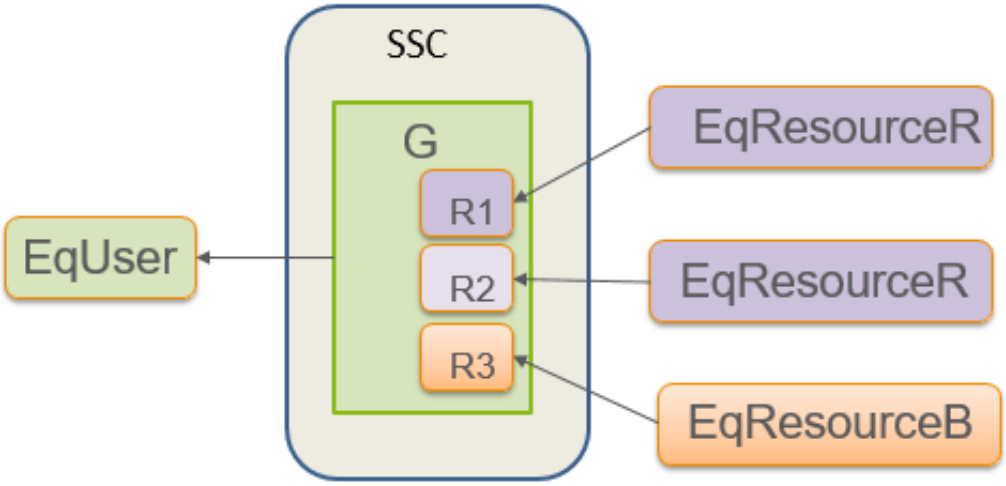

Figure 5.11: Object structure of one user in SSC

One could notice that the computed group state of G will depend on the arrival time of service response messages (state update information) from the resources (which means that there is a probability to get different G state, when we consider R1 getting update before R2 and R2 getting update before R1). As presented in section 5.3.1 and 5.3.2, object G will be serialized to backup SSC replica during preparation phase of the switchover process. After successful deserialization, G will create a fresh stateful connections towards resources to create its R type objects (the rationale is discussed in section 5.3.1). This make each resource to have double stateful session to each replica.
Our study has found that, such double stateful sessions will cause group state inconsistency for G object. This is mainly attributed to missing of total order guarantee

from LINX. Total ordering guarantee that sets of service response messages from all resources to must be delivered in the same order by both replicas [25]. Figure 5.12 explains how FIFO order from LINX would be problematic during preparation phase. FIFO order only guarantee ordering of message between two processes. As shown by numbers in Figure 5.12, service response messages could arrive (deliver) in different sequence in time by the replicas. This will result in two inconsistent group state for G in the replicas.

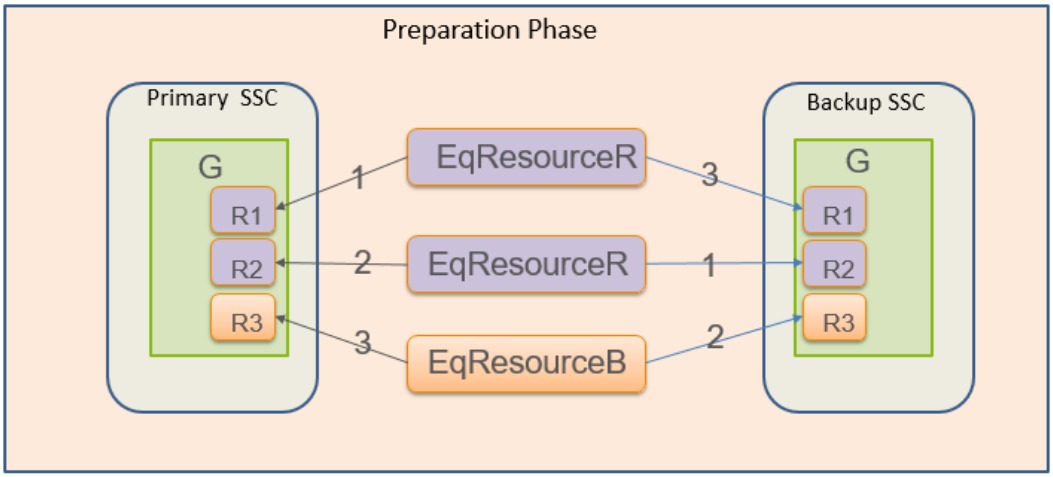

Figure 5.12: Inconsistency of user group state due to different message arrival on a time unit by the replicas

In addition, even if we assume having total ordering, it is possible that before G is successfully deserialized in the backup SSC during preparation phase, one or more service response messages could be received on primary SSC. As shown on Figure 5.13, at time unit 1, R1 in primary SSC have received service response message before G completed deserialization. This will cause inconsistency between group state of G, since G in backup SSC will analyse message at time 2 relative to state of G just before preparation phase, while G in primary SSC analyse message at time 2 relative to the resulting state of G after analysing message at time 1.

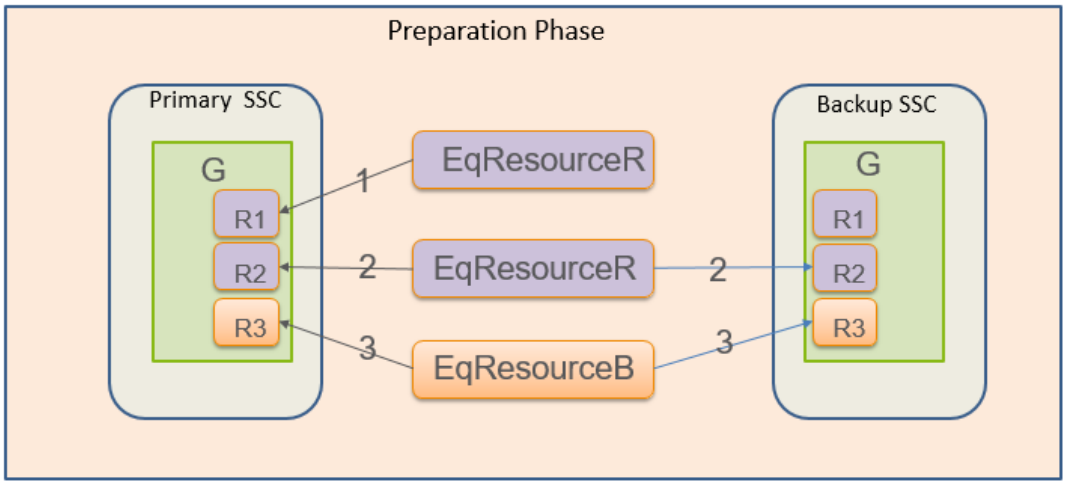

Figure 5.13: Inconsistency of user group state due to missing of messages in preparation phase

Due to time limitation, this study has mainly focused on designing strategies for securing consistency considering the challenge shown in Figure 5.13. As a solution for guaranteeing delivery of missing messages to G on backup SSC, sequence numbers are used. Each service response message from each resource is tagged with sequence numbers. This helps to uniquely identify each service response message by combining unique identifier of each resource with sequence numbers. Via implementing a protocol to synchronize missing service response messages from primary to backup SSC, consistent group state could be guaranteed.

As could be seen in Figure 5.14, when switchover process is started, primary SSC begin to store received service response messages in a way that they could be identified. It could be seen that G on backup SSC has missed sequence number 2 from resource X. During synchronization phase, the backup SSC informs primary the earliest sequence number it has received for each resource X of each G. The primary SSC analyses the information and synchronizes any missing service response messages to the backup SSC. When the synchronization phase is completed and dequeueing is started, the backup SSC pops service response messages based on their sequence numbers. As discussed in section 5.3.1, G in primary SSC unsubscribes (terminates stateful session) towards resources during synchronization phase.

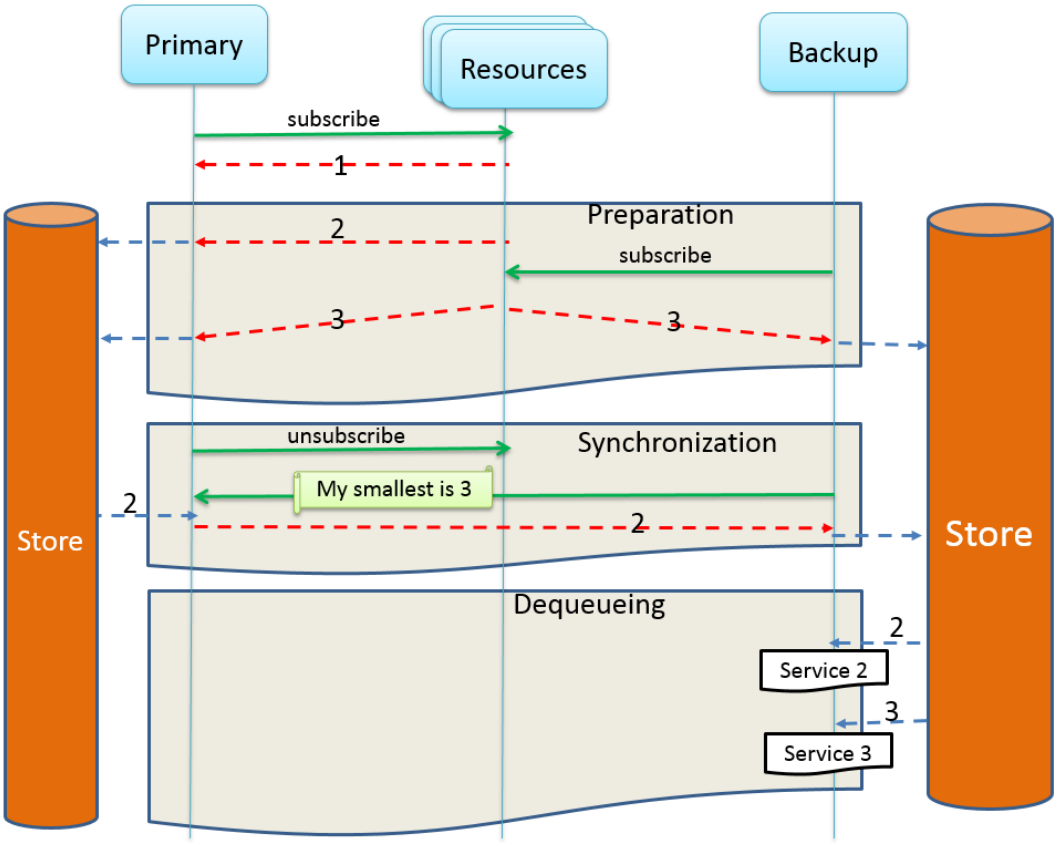

Figure 5.14: A sequence diagram to show strategies for handling missing service response messages

5.4 Switchover Process Summary

Table 5.1 summarizes all the activities discussed in section 5.3. The backup SSC will become officially primary SSC with NORMAL state, when all stored service response messages are analysed. However, forwarded service requests will be served when the backup SSC become officially primary SSC.

5.5 Prototype Evaluation Results

This section presents evaluation result of the prototype mainly in the following areas: prototype correctness, effectiveness of object serialization, memory utilization and scalability issues.

5.5.1 Correctness analysis

Correctness analysis is performed by predetermining the expected behaviour during and after switchover process. In order to observe the expected behavior on both replicas, the old primary will not terminate sessions towards resources during synchronization phase.

Table 5.1: Switchover process summary

Switchover process phase	Primary SSC	Backup SSC
Preparation	Serialization of user session encapsulating objects	Deserialization of user session encapsulating objects
	Store service response messages	Configure sessions with resources
	Forward service request messages	
Synchronization	Synching of missing service response messages	Synching of missing service response messages
	Store service response messages	Store service response messages
	Terminate sessions to resources	Inform each user about change of primary address
	Forward service request messages	
Dequeuing	Store service response messages	Store service response messages
		Analyse stored service response messages Considering sequence numbers

For this analysis 5 groups, each having 2 EqResourceR and 2 EqResourceB resources, are created. The primary SSC is simulated to receive 2 service response messages from each resources, which are simulated by the backup SSC to be missed. Other than that, a couple of service request and service response messages are performed before, during and after switchover process. As shown in Figure 5.16, both replicas has proved to result similar states for each group during dequeueing phase and after switchover process is completed.

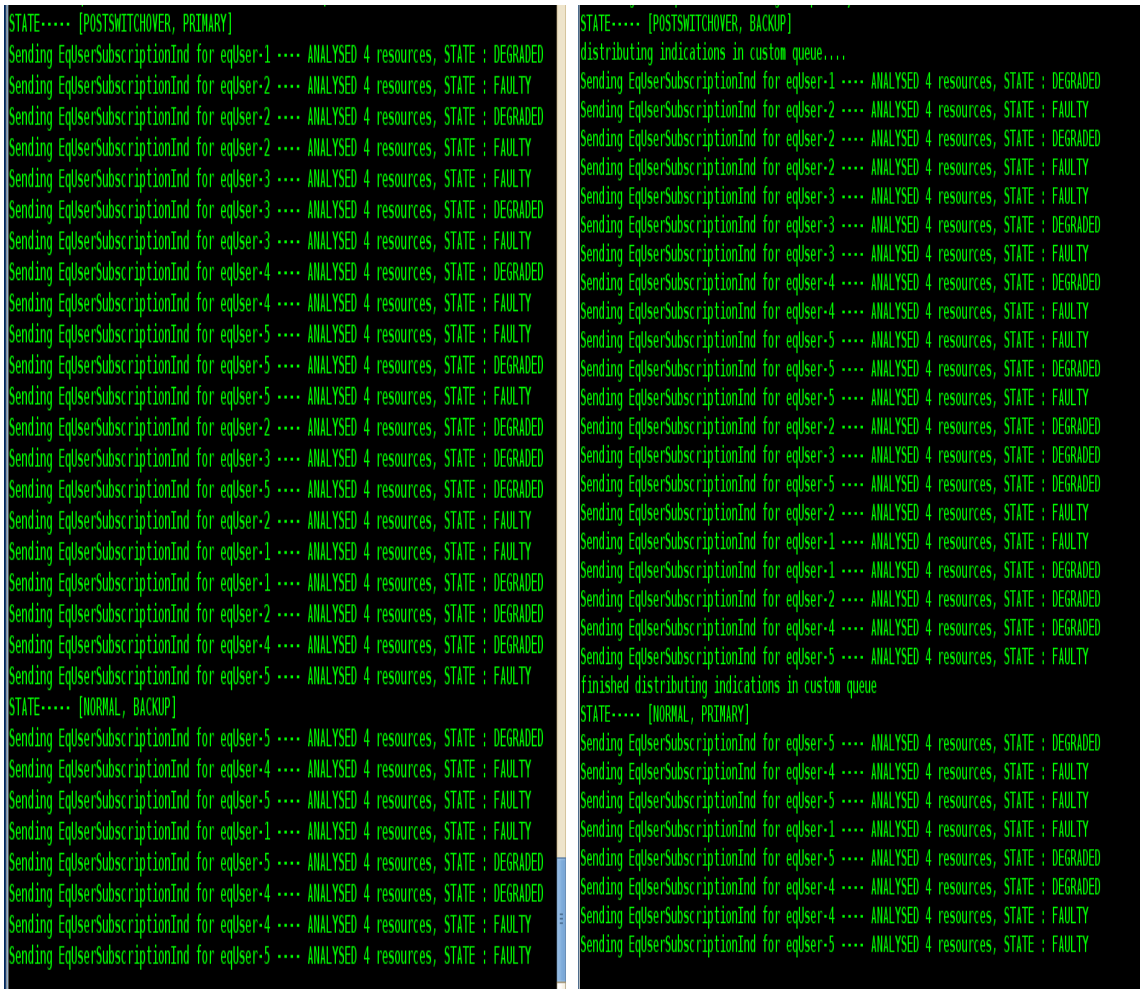

Figure 5.15: correctness test

5.5.2 Time Effectiveness of serialization and deserialization

Efficient serialization at the primary and deserialization at the backup has significant contribution on the switchover process duration; and hence it is important to measure the time effectiveness of object serialization of the prototype. Fig 5.17 shows the relationship between objects to be serialized and deserialized versus the time consumption. In this test the authors has increased the numbers of users i.e. numbers of group instances connected to SSC up to 300 and observe duration of time for object serialization and deserialization.

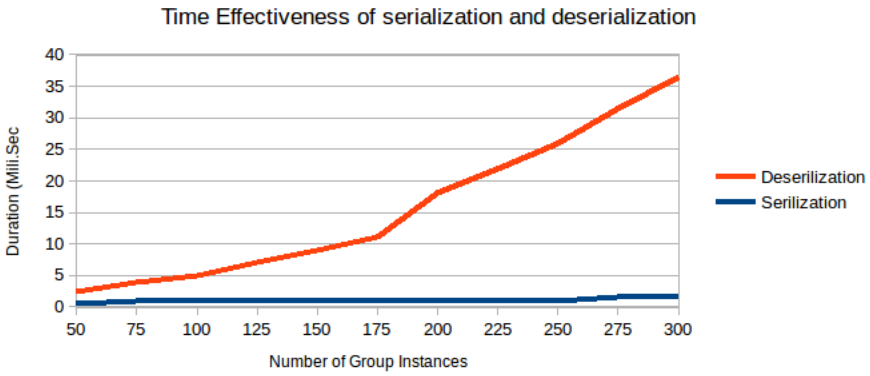

Figure 5.16: Time Effectiveness of serialization and deserialization

5.5.3 Scalability analysis

This is the performance testing to investigate the scalability of the prototype in terms of switchover process duration considering number of groups(user session), memory size of each group (user session), and number of missing service response messages.

5.5.3.1 Effect of number and size of stateful sessions

This analysis shows the relation between numbers of group versus size of each group versus switchover duration.The analysis is done with assumption that no missing messages are required to be synchronized during synchronization phase. By size of each group, we mean added payload string on eqGroupAcceptor and eqGroupInstance. For each group, 2 EqResouceR and 2 EqResouceB resources was assigned. Each resource sends service response messages in an interval of 5 seconds
Fig 5.17 shows increase of the switchover process duration as the number of groups and payload increased. There is a observable significant performance effect when the number of groups are increased but the effect of payload within the group is not significantly observed. Generally, the author found it difficult to determine the pattern of payload effect on switchover duration because of unpredictable test result. For example, when the number of group instances to be serialized are 50 and the payload is 25 byte the switchover duration is around 4100 milliseconds; but when the payload is increased to 3000 byte for the same number of group instances, the switchover duration decrease to $\sim$ 3100 milliseconds (but as to the author it is supposed to increased).

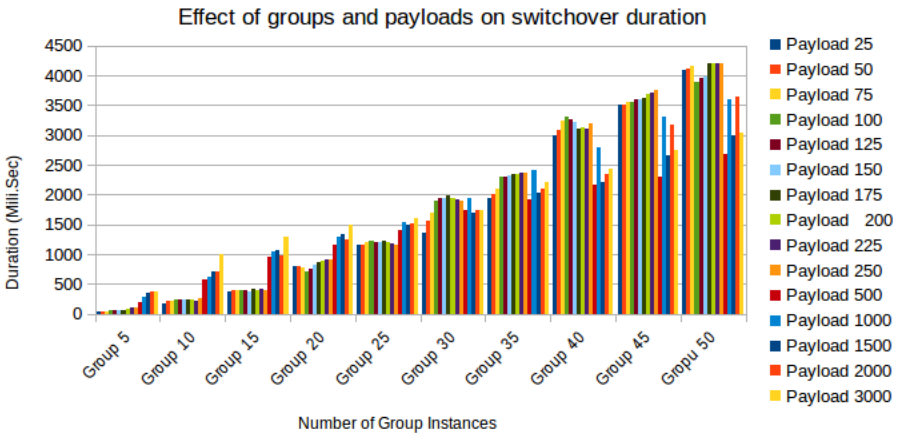

Figure 5.17: Effect of Groups and Payloads on Switchover durations

5.5.3.2 Effect of synchronization of missed service response messages

This analysis shows the relationship between synchronization of missing service response messages versus switchover duration. For this analysis 5 groups are created, each having 2 EqResouceR and 2 EqResouceB resources. To simulate the analysis, preparation phase is started without doing serialization. By letting the Primary to receive the required amount of service response messages (to be sent as missing later). The switchover process will then continue with serialization, which makes the backup SSC to request for missing messages during synchronization phase. Figure 5.19 shows the result based on the sample data i.e number of missed message up to 100 and payload up to 100 byte. Based on the result synchronizing missed message doesn't have significant performance effect on the prototype.

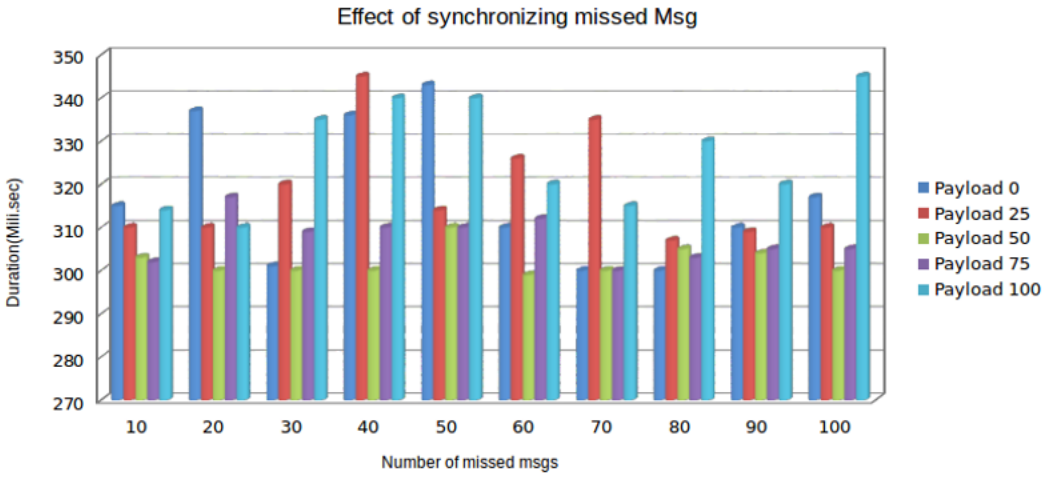

Figure 5.18: Effect of Synchronizing missed message on switchover duration

5.5.4 Memory Utilization

The serialization and deserialization of objects is a computational and memory intensive activity. As can be referred in Figure 5.16, the deserialization computation will require a relatively longer computational time.

To measure the memory utilization efficiency of the prototype, the authors undertook a test on memory utilization during serialization of objects instance. Object can be serialized one by one or all group all-together. The result in Figure 5.20 shows that, there is a high memory consumption when serializing groups separately, while on the contrary serializing groups all together (for example collecting groups in a vector) reduce the memory consumption.

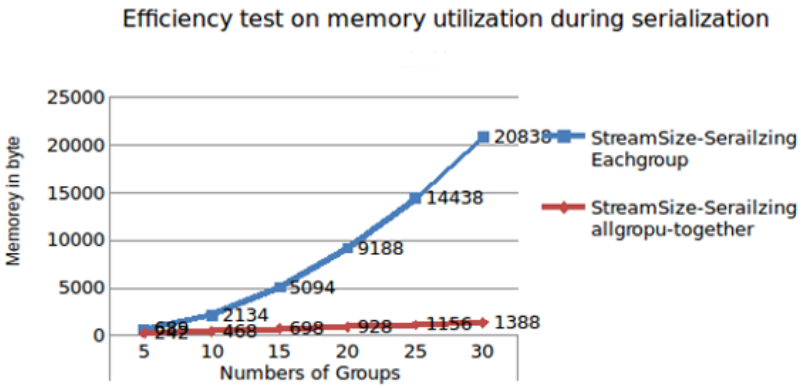

Figure 5.19: Effect of serializing groups individually and all-together on memory

6
Discussion

6.1 Performance

Performing the switchover process as fast as possible in order to reduce the service impact on SSC is one of the issue considered in design strategies of this paper.
As the number of users connected to the SSC increases, the object instances of the corresponding user with its own data that needs to move from primary replica to backup replica will also increases. This has its own effect on switchover process duration, and hence we need efficient way of object serialization. As discussed in Object serialization pattern, (section 5.2), among the other object serialization technologies, Cereal is chosen and implemented for performance and simplicity reasons.
As seen in the time effectiveness of serialization in Figure 5.16, even if the graph grows linearly as the number of group increased for deserialization, still the overall duration of serialization duration is reasonably acceptable . On the other had, serializing groups (user session objects) all-together has significant improvement on memory utilization, which indirectly improves the duration of serialization and deserialization.

6.2 Added Functionality

It is inevitable that extending the reference architecture is required component-wise and code-wide to implement this study's design strategies. Component-wise, the prototype has added two extra components (C++ 11 Executable processes), namely backup SSC replica and HAcontroller (Figure 5.3). This two added components are C++ 11 executable processes in which the environment will need to allocate heap and stack memory (among others) to function correctly. Three interface units are required also for interactions among the replicas and HAController.
To have a controlled switchover process, SSC replicas and HAController will need to have computation and management codes. However, considering extendability and decoupling is important in the code implementation, so that it could be easier to modify and/or plug the design strategies in other SSCs. To satisfy such properties, code-wise changes in SSC are performed in the following manner:

- The decision of SSC replicas to act as primary or backup is left to the run-time system. This will help reduce code by requiring to only add state control structures.
- Code implementations related to run-time SSC state and switchover process management are decouple and encapsulate in separate management shared

library. This library will be the owner of published interfaces used to communicate from/to HAController and partner replica. All computational actions during switchover process is encapsulated in this library. In addition, it is also home to data structures used to queue or track messages. ∼ 1000 LOC is implemented in this library.

- An effort is made to reduce code related to Cereal on object classes. Hence, for serealization and deserialization purposes only (∼ 5 LOC) will need to be included in all object class headers which are planned to be serialized.

As shown on Figure 6.1, each replica (EQCOORD) will only be informed about its state (primary or backup) at system start-up. Its state will then be managed by the added library. This enables to add only ∼ 100 LOC on the executable main class.

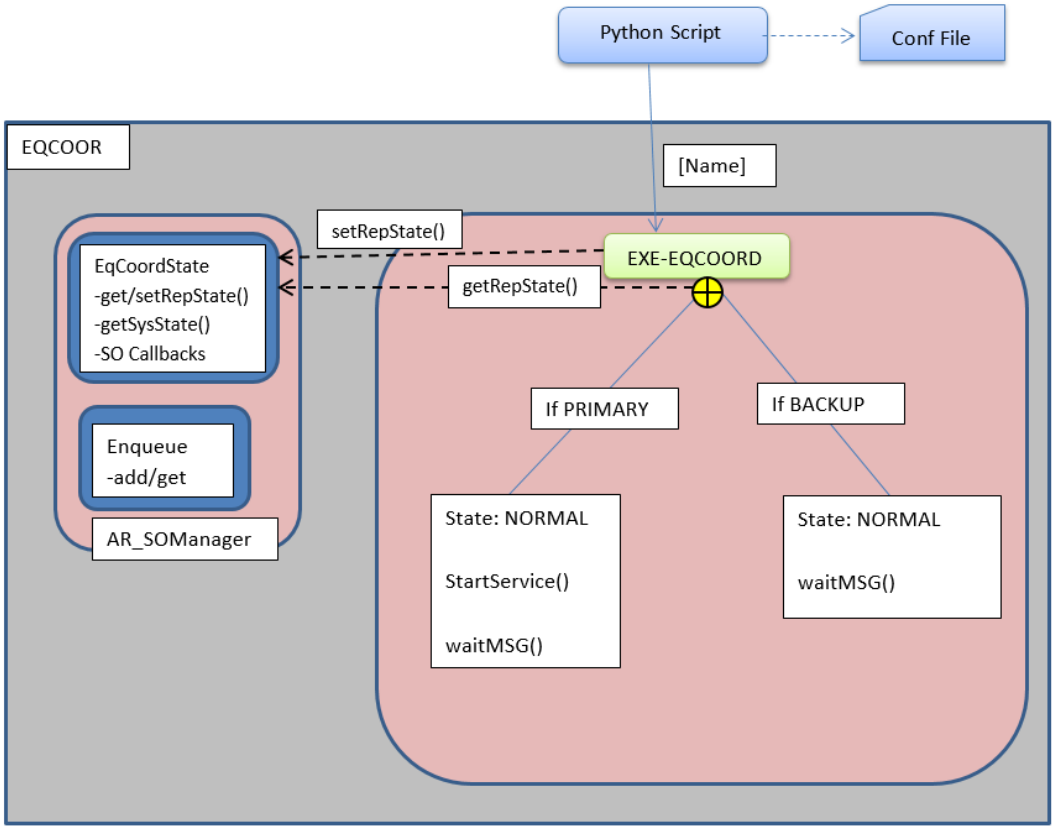

Figure 6.1: Start-up state management of SSC replicas

The state oriented design has simplified the flow of execution when any message arrive at the executable process of each SSC replica(EQCOORD) during normal or switchover process (not normal) situation. The full state flow during run-time when any message arrives is shown on Figure 6.2.

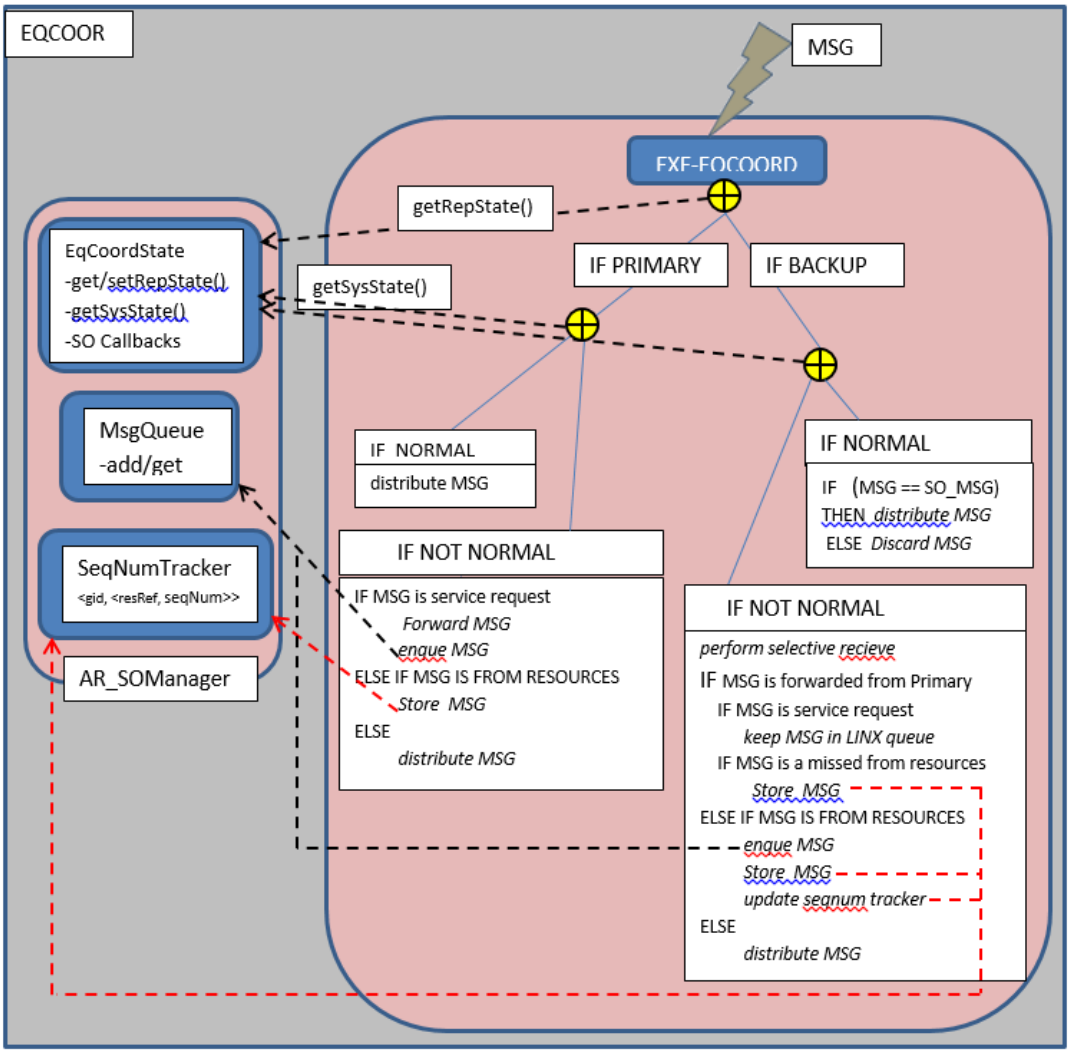

Figure 6.2: Run-time state structure flow by SSC replicas

6.3 Fault Tolerance

During switchover process, as discussed in section 5.3.1, a service request message arriving at the primary replica is directly forwarded to the backup replica. For fault tolerant purpose, each forwarded message to the backup will also be stored in a custom queue at the primary replica. If the backup fails to take over the duty of primary replica during switchover process, the "old" primary will continue as a primary by reconstructing(rollback) those message in the custom queue. Except that, fault tolerance related implementations are not performed in the prototype since the switchover process is assumed to be 100% successful(section 1.3).

However, extendability for fault tolerance purposes are considered during the design. As shown on Figure 6.3, the primary SSC replica can terminate the switchover process and start serving queued service request messages when HAController rejects the continuation of switchover process.

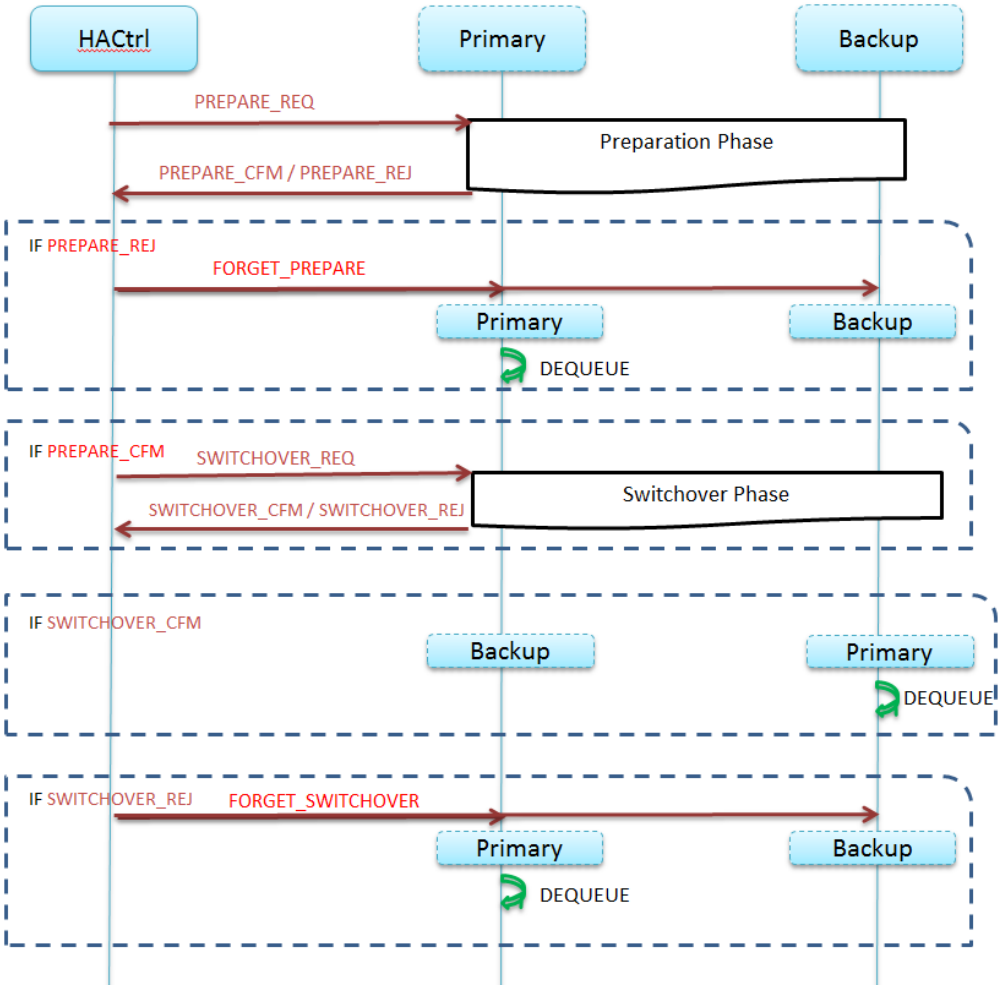

Figure 6.3: Fault management in switchover process

6.4 Memory Utilization

As mentioned in section 5.5.4, serialization and deserialization of objects are memory intensive activities. These activities (referred on Figure 5.17 and 5.20) require extra memory to be allocated for storing serialized data. For example, for an SSC Replica handling 3 groups (user sessions) without any added payload, the replica process will require approximately 130 bytes for serialized data of the groups. In addition, depending on the maximum size allowed for a message, one or more messages will be required to send this data to partner replica.
In addition to messages to transport serialized data, if there are X number of resources, it will need 2X messages for configuration, and X number of messages to be used for informing about sequence numbers. An extra memory and network resources will also be required to synchronise any missing service response messages by the backup SSC replica. For managing the switchover process, about a couple of messages will be utilized.
On the other hand, all of extra added functionalities to support the switchover process will, in some way, have extra burden on the environment, which increase or complicate the memory utilization.

7
Future Work

Given the complexities of SSCs and long-living stateful sessions, this study is a begining (door opening) effort for increasing availability of such systems. The authors strongly believe more studies will be needed in the future to attain high availability requirements.
A main focus has been given for correctness during the prototype design, since there is no point of making a super fast switchover as long as the replicas do not behave correctly with consistent states. The study has proposed an architecture which has proven correct behavior after switchover among SSC replicas. The next step with this regard, would thus be to consider the assumptions(limitations) scoped out from the study (section 1.3). This will require to widen the scope in 'environment' level considering failure of SSC, failure of other non-singleton components, and failure of activities in proposed switchover process. Once considering such failures, it would be possible to measure annual system outage and availability percentile metrics. In addition, as pointed out in section 5.3.3.3, the prototype has not considered situations of service responses having different total order in the replicas during preparation phase of the switchover process. Suitable design strategies would need to be incorporated with the proposed prototype architectures if total order differences has higher probability of happening, or if it is a critical business requirement. On the other hand, it will be important to undertake serious efforts to find suitable patterns to ease proposed switchover process strategies. An exemplary pattern is researched and presented in section 5.1. The authors believe revising the study's proposed architecture based on EIP would greatly improve the switchover process duration and other performance metrics.

8

Conclusion

Availability of systems such as Ericsson's RBS software components are becoming increasingly vital to have higher customer satisfaction. More specifically, availability of SSC is very important for higher availability of the superset component, which we call in this study 'environment'. This is attributed to the fact that state information and configurations of the 'environment' users are by design only kept in memory of SSC process or thread. Among couple of tactics, replication is a common strategy for attaining higher availability of software systems. This study has studied the challenges which arose in implementing replication of SSC for high availability purposes. A design research was employed to factor out critical problems which need to be addressed via designing strategies in form of prototype. The strategies include replication fashion, and switchover process strategies among the replicas. In addition, a qualitative research was, also, performed to find suitable patterns in easing replication and switchover process, and to find suitable object serialization technologies. The prototype architecture resulted from the design research was found to be successful. The replicas has performed correctly as expected after switchover, which is very important requirement considering the nature of SSC. In addition, a range of stress testing of the prototype has resulted in pointing out area for improvement for more faster switchover duration. A reasonable design pattern for easing switchover process activities were also presented as part of the study. In general, the authors believe the prototype architecture will be an important cornerstone in securing high availability of system containers having SSC as one component, such as Ericsson's RBS 'environment'.

Bibliography

[1] Malkawi, M.I. 2013, "The art of software systems development: Reliability, Availability, Maintainability, Performance (RAMP)", Human-centric Computing and Information Sciences, vol. 3, no. 1, pp. 1-17.

[2] Wiesmann, M., Pedone, F., Schiper, A., Kemme, B. & Alonso, G. 2000, "Understanding replication in databases and distributed systems", , pp. 464.

[3] Moser, L. E., Melliar-Smith, P. M. and Narasimhan, P. (1998), Consistent object replication in the Eternal system. Theory Pract. Obj. Syst., 4: 81–92. doi: 10.1002/(SICI)1096-9942(1998)4:2<81::AID-TAPO3>3.0.CO;2-A

[4] Kistijantoro, A.I., Morgan, G., Shrivastava, S.K. & Little, M.C. 2003, "Component replication in distributed systems: a case study using Enterprise Java Beans", IEEE, , pp. 89.

[5] Feng, Y., Huang, N., Liu, R. & Wu, M. 2007, "Flow Digest: A State Replication Scheme for Stateful High Availability Cluster", IEEE, , pp. 1298.

[6] Wu, H. & Kemme, B. 2005, "Fault-tolerance for stateful application servers in the presence of advanced transactions patterns", IEEE, , pp. 95.

[7] Wu, H. 2008, Adaptable stateful application server replication, ProQuest Dissertations Publishing.

[8] Bauer, E. & Adams, R. 2012, "Service Reliability and Service Availability" in , 1st edn, John Wiley & Sons, Hoboken, NJ, USA, pp. 29-62.

[9] Wang, Z. & Wang, D. 2013, "NCluster: Using Multiple Active Name Nodes to Achieve High Availability for HDFS", IEEE, pp. 2291.

[10] Guerraoui, R. & Schiper, A. 1997, "Software-based replication for fault tolerance", Computer, vol. 30, no. 4, pp. 68-74.

[11] Infocenter.sybase.com. (2016). SyBooks Online. [online] Available at: http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.infocenter.dc00547.0600/html/ [Accessed 25 Apr. 2016].

[12] Taylor, Z.& Ranganathan, S. 2013, "Fault Management Architectures" in , 1st edn, Wiley, Hoboken, NJ, USA, pp. 377-396.

[13] Ericsson, RBS and CAT Quick Introduction, 2016.

[14] Wikipedia. (2016). LTE (telecommunication). [online] Available at: https://en.wikipedia.org/wiki/LTE(telecommunication) [Accessed 25 Apr. 2016].

[15] Wikipedia. (2016). Serialization. [online] Available at: https://en.wikipedia.org/wiki/Serialization [Accessed 25 Apr. 2016].

[16] Malenfant, J., Jacques, M. and Demers, F.N., 1996, April. A tutorial on behavioral reflection and its implementation. In Proceedings of the Reflection (Vol. 96, pp. 1-20).

[17] Soukup, J., Macháček, P., SpringerLink (Online service), Books24x7 (e-book collection), SpringerLink (e-book collection) & Books24x7, I. 2014;2013;, Serialization and Persistent Objects: Turning Data Structures into Efficient Databases, 2013;1;2014; edn, Springer Berlin Heidelberg, Berlin, Heidelberg.
[18] Codefarms.com. (2016). Data Object Library | New Ideas On How To Design Better Software. [online] Available at: http://www.codefarms.com/dol [Accessed 25 Apr. 2016].
[19] Google Developers. (2016). Protocol Buffers | Google Developers. [online] Available at: https://developers.google.com/protocol-buffers/ [Accessed 25 Apr. 2016].
[20] Boost.org. (2016). Serialization. [online] Available at: http://www.boost.org/doc/libs/1-52-0/libs/serialization/doc/index.html [Accessed 25 Apr. 2016].
[21] Uscilab.github.io. (2016). cereal Docs - Main. [online] Available at: http://uscilab.github.io/cereal/index.html [Accessed 25 Apr. 2016].
[22] Rubén Torres Bonet. (2014). An overview of data serialization techniques in C++. [online] Available at: https://rubentorresbonet.wordpress.com/2014/08/25/an-overview-of-data-serialization-techniques-in-c/ [Accessed 25 Apr. 2016].
[23] GitHub. (2016). thekvs/cpp-serializers. [online] Available at: https://github.com/thekvs/cpp-serializers [Accessed 25 Apr. 2016].
[24] Enea.com. (2016). Interprocess communication - Enea Software. [online] Available at: http://www.enea.com/solutions/middleware/interprocess-communication/ [Accessed 25 Apr. 2016].
[25] Défago, X., Schiper, A. & Urbán, P. 2004, "Total order broadcast and multicast algorithms: Taxonomy and survey", ACM Computing Surveys (CSUR), vol. 36, no. 4, pp. 372-421.
[26] Reenskaug, T. and Coplien, J.O. (1996) The DCI architecture: A new vision of object-oriented programming. Available at: http://www.artima.com/articles/dci_vision.html (Accessed: 14 April 2016).
[27] Bluemke, I. & Stepień, A. 2015, "Experiences with DCI pattern", pp. 87.
[28] Reenskaug, T., 2008. The Common Sense of Object Orientated Programming.
[29] Broker pattern (2016) in Wikipedia. Available at: https://en.wikipedia.org/wiki/Broker_Pattern (Accessed: 17 April 2016).
[30] Microsoft (2016) The repository pattern. Available at: https://msdn.microsoft.com/en-us/library/ff649690.aspx (Accessed: 16 April 2016).

A

Appendix 1