11-20-2008

Proxy Module for System on Mobile Devices (SyD) Middleware

Joseph Gunawan

PROXY MODULE FOR SYSTEM ON MOBILE DEVICES (SyD) MIDDLEWARE

by

JOSEPH GUNAWAN

Under the Direction of Sushil K Prasad

ABSTRACT

Nowadays, users of mobile devices are growing. The users expect that they could communicate constantly using their mobile devices while they are also constantly moving. Therefore, there is a need to provide disconnection tolerance of transactions in the mobile devices' platforms and its synchronization management. System on Mobile Devices (SyD) is taken as one of the examples of mobile devices' platforms. The thesis studies the existing SyD architecture, from its framework into its kernel, and introduces the proxy module enhancement in SyD to handle disconnection tolerance, including its synchronization. SyD kernel has been extended for the purpose of enabling proxy module. SyDSync has been constructed for synchronization with the proxy. The timeout has been studied for seamless proxy invocation. A Camera application that tries to catch a stolen vehicle has been simulated for the practical purpose of using the proxy module extension.

INDEX WORDS: Proxy, System on Mobile Devices, SyD, SyDSync, SyD API, Client, Server, Mobile, Synchronization, Fault-tolerant, Transactions, Timeout time, Disconnection, Tolerance

PROXY MODULE FOR SYSTEM ON MOBILE DEVICES (SyD) MIDDLEWARE

by

JOSEPH GUNAWAN

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in the College of Arts and Sciences

Georgia State University

2008

PROXY MODULE FOR SYSTEM ON MOBILE DEVICES (SyD) MIDDLEWARE

by

JOSEPH GUNAWAN

Committee Chair:	Sushil K Prasad
Committee:	Raj Sunderraman
	Saeid Belkasim

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
December 2008

To my father, Johan Gunawan,

for his advice, encouragement, support, and sacrifice.

To my mother, Kwa Minawati,

for her prayer, dedication, and encouragement.

To my brother and my sister,

thank you for allowing me to be a good example.

ACKNOWLEDGEMENTS

Two years is a quite long time. But, I thank God for letting me to get through all those years strongly. God has been so good to me for placing me into Georgia State University as a place to further my knowledge until what I am now, writing my thesis for my Master of Science degree requirement in Computer Science. He is good and always good to my unprecedented life and career. I know that He is with me and guides me on all challenges that I face.

My next gratitude goes to my advisor, Dr. Sushil K Prasad, for his time, guidance, encouragement, and constant support. His knowledge, perceptiveness, and innovative ideas have guided me throughout my graduate study. I also would like to express my sincere gratitude to my committees, Dr. Raj Sunderraman and Dr. Saeid Belkasim, for their help and their interest on my work. It has been a pleasure for me to meet and to work with them for the past few years. They are always there to support and to lead me to the right direction. Without their guidance and encouragements, my thesis would not have been possible.

In addition, I want to thank students in DiMoS group, Srilaxmi Maladi who introduced SyD to me, Sunetri Priyanka who first helped me setting up SyD on my work environment, Chad Frederick who always calls me "Junior," and Akshaye Dhawan who shares his thought to me. I thank you guys for all of your support and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Nowadays, there are many interesting features being offered by mobile devices' (cell phones, PDA, smart phones, etc) software vendors. The reason behind it is because there are increasing demands of the users on mobile devices. Microsoft, as an example, offers Microsoft Office for mobile on top of Windows Mobile OS 6 such as Word, Presentation, Outlook, etc to let its users to get their work done while they are travelling on the wireless network connection.

On the other side, Yahoo offers Yahoo onePlace service, which is based on a familiar bookmarking process, to let its users to easily link into practically any pieces of contents (news feeds, websites, videos, etc) from anywhere while they are still maintaining Internet connection. Google, partnered with T-Mobile and HTC, lately introduces Google Android, which offers OS and middleware platform for mobile devices.

Different than Android, System on Mobile Devices (SyD), developed by Distributed and Mobile Systems (DIMOS) research team, introduces its proxy module extension to provide disconnection tolerance of transactions not only for mobile devices, but also for any devices either on wired or wireless network connection. In other word, a proxy module has been constructed on SyD to let its users to seamlessly getting the requested of the services within the network connection.

1.1 Purpose of the proxy

The present of wireless network makes all mobile devices' features become possible, which lets the mobility communication and continuous access to the services and resources to happen. The combination of wireless network and mobility will [1] "engender new applications and services, such as collaborative software to support impromptu meetings, electronic bulletin boards whose contents adapt to the current viewers, lighting and heating that adjust to the needs of those present, and navigation software to guide users in unfamiliar places and on tours."

Since wireless communication normally interacts with the signals, there are some obstacles that are faced within the communications, such as lower bandwidths, higher error rates, frequent spurious disconnections, which in turn, is going to increase communication latency. Mobility that becomes the main purpose of people having mobile devices could also cause wireless connections to be lost or degraded. In some point, the network bandwidth could be overloaded since there are large numbers of users using the network.

Therefore, mobile devices providing users' services (normally mobile device server application) may become unavailable due to the following reasons:

- First, involuntary failure of mobile devices.

  Ex.: Loss of battery power, mobile devices are overloaded by requested service computations.
- Second, voluntary disconnection of the mobile devices by their users.

  Ex.: Users shutdown the system.
- Third, it might be caused by the wireless link failure or overloaded.

If the scenario above happens, there will be no responses on the users' requested services. The proxy comes as one of the solutions to tolerate the failures on the users' requests (fault-tolerant). It means that the proxy works as a replication of the actual failure of a mobile device providing services. Proxy could also mean a "substitution". Therefore, the proxy keeps of all the state transactions of the users' requested services. When the mobile device is not in the failure mode, it, upon synchronization with its proxy, gathers and processes all of the users' requests transactions.

1.2 Motivation

There are many technologies that could be used to develop collaborative applications that are running on heterogeneous, possible mobile, devices such as JXTA, .Net, J2ME, etc, that could tolerate the failures. But, developing applications using those technologies requires too many details and also consumes too much time. Therefore, there is a need to provide a mobile platform or framework that is very systematic and streamlined for rapid development and deployment of collaborative applications. This is where System on Mobile Devices (SyD) comes as a solution (Chapter 3 describes about the existing SyD, including its framework and its kernel).

Even though it allows less time to develop mobile applications using SyD, the current SyD architecture is unable to handle the failures, which result in the discontinuity of the services of the users' requests. Therefore, as being mentioned above, a proxy comes as one of the solutions toward the failures. With the enhanced proxy on SyD, users' requests could be handled properly even though the registered mobile device server providing services is in the failure mode.

Camera application has been designed and constructed to introduce the practical use of proxy module for SyD platform. The camera application is used to look for a specified stolen vehicle. This is done by capturing a vehicle license plate numbers on every vehicle passing through the camera. The simulation model (described more on chapter 6) represents a camera as a mobile device (laptop) that runs SyD client and server. Since it runs SyD client and server, a camera could act both as a server and a client. The old, non portable desktop computer is used as a proxy of a camera. A proxy only runs SyD server. A proxy is intended to be a smart proxy, which not only gives a receipt of every requested user’s transactions, but also saves the current user’s state invocation upon the failure of a camera. Another mobile device (laptop), running also SyD client and server, acts as a client. It is used by the user to request transaction remotely into the camera and a place for the camera to tell the client that a specified car has been found or not. A client, in the context of camera application, is the user of camera application. Otherwise, a client is the one who initiates the requests of services to the server.

1.3 Thesis outline

In the chapter 1, we give an introduction about the purpose of having the proxy module on the existing SyD platform to allow disconnection tolerance of transactions possible. In chapter 2, we introduce the background of choosing proxy, “substitution”, as the best fit for SyD platform. Overview of the existing SyD platform is introduced in chapter 3. Chapter 4 is going to describe the extended version of SyD platform implementing the proxy module. The performance analysis of our SyD proxy module is introduced in Chapter 5. A case study presenting a developed camera application is presented in chapter 6. Chapter 7 summarizes the thesis and presents our direction for the future work.

2. BACKGROUND

This chapter lays out the road map of our chosen design proxy module to work as a "substitution" of a mobile device (normally a mobile server application) for the System on Mobile Devices (SyD) platform. We have gathered all possible related works being done in the area by many researchers. Among our top three categories: intelligent mobile agent, checkpoint and message log, and replication technique, we come to the conclusion to design and to implement our SyD proxy module based on the replication technique. The camera application, described in chapter 6, is designed using a product of our proxy module extension on the existing SyD platform.

There are many ways to handle the failures that happen on the transactions within the mobile device system. In this thesis, we have categorized into three ways to handle such failures: using intelligent mobile agent, using checkpointing and message logging, and using replication technique. The proxy module design for SyD is almost similar to the way of replication technique being used to handle mobile device's failures. Our proxy design not only replicates a mobile device server's functionalities, but also works "smarter." It is a smart proxy in the way of storing the current user's transactions when there is a failure in a mobile device server. Upon synchronization, the current state transactions, which were saved before, are processed.

2.1 Intelligent mobile agent

Intelligent mobile agent is introduced as a technique to tolerate the disconnection of transactions in the mobile environment. Chess et al [2] defined mobile agents as "programs, typically written in a script language, which may be dispatched from a client computer and transported to a remote server computer for execution." In their paper, Wong et al [3] also mention that the mobile agent concept grows on the influence of the previous technologies: process migration [4], remote evaluation [5], and mobile objects [6]. All are developed in order to improve the remote procedure calling (RPC) for the distributed programming.

Process migration is introduced [3] by allowing an entire address space to be moved from one computer into another. The network bandwidth is overloaded when multiple RPC calls are needed to execute an application. Then, remote evaluation programming comes to allow [3] "one computer to send another computer a request in the form of a program (rather than an entire process address space)." After the remote computer executes the received program referenced in the request within its own local address space, it returns the results to the sending computer. Mobile objects (based on formal OOP techniques) come as the extension of the remote evaluation. Mobile objects encapsulate more program behaviors or states to be sent to do more computation remotely. Finally, mobile agents come up with the much improvement from the mobile objects. The emerald system [6] is said to be the first mobile object that leads to the development of mobile agent.

Mobile agents indeed reduce the network bandwidth for applications processing large quantities of data. Not only that, compare to the client/server model, mobile agents extend the model by allowing the program module to be sent to do the computation into the server and come back to the sending client after finishing the computation as shown in Figure 1 [3] below.

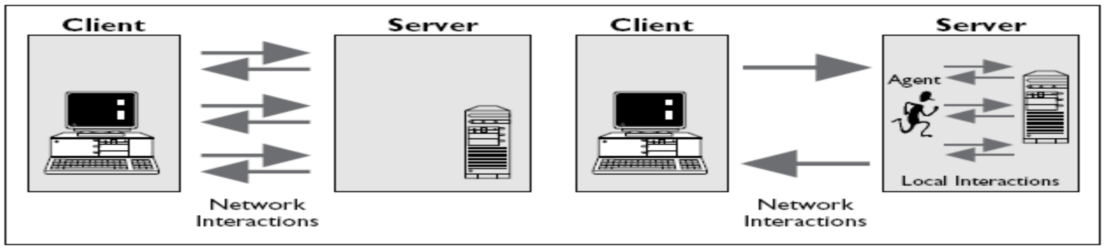


Figure 1. Client/Server model and agent model to do the computation [3].

Mobile agents provide some autonomy which let them to dynamically decide when and where to travel to a particular destination to perform some computation. Mobile agents also provide a way for executable code, program state information, and other data to be transferred to whichever devices the agents need necessarily to carry out the actions specified in the applications. Mobile agents are also ready to adapt to the changes in both the program state and the network environment (such as network partitioning and disconnected devices) to modify their routing behavior. With these abilities of the mobile agents, it allows them to be used as the fault-tolerant technique in the mobile environment to do transient transactions. Upon the failure, mobile agent that is sent by the mobile device can finish what needed to be done, then, return back with the result to the sending mobile device whenever it is recovered.

Wong [3] also introduces the generic architecture of mobile agent based on Java. The architecture consists of six major components: an agent server, an agent manager, an inter-agent communications manager, a security manager, a reliability manager and application gateway, and a directory manager. Figure 2 [3] shows the complete architecture of Java based mobile agent.

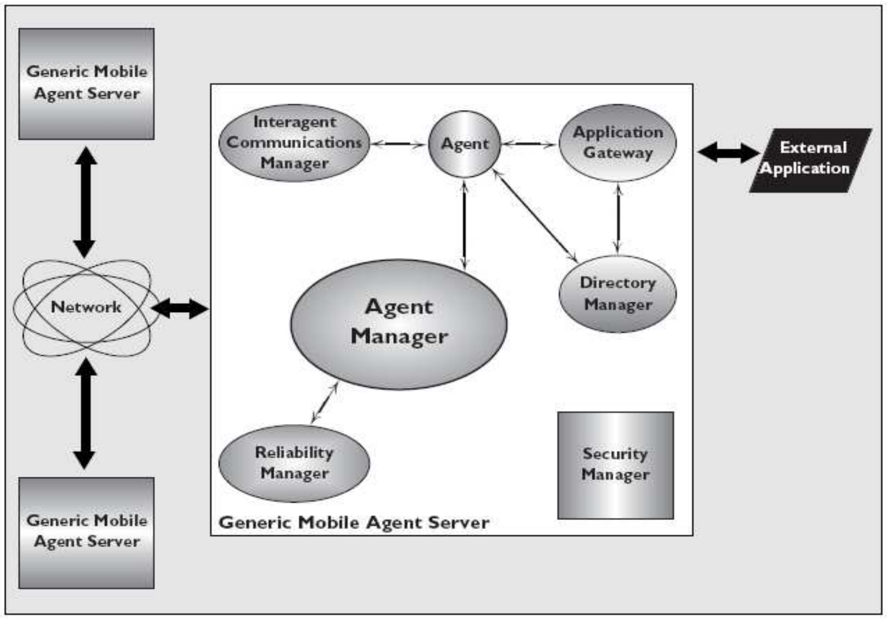


Figure 2. A generic Java based mobile agent [3].

The agent server has the purpose to create the agent. On the other side, the agent manager has the following purposes:

- To send agent to the remote host.
- To receive agents for execution on the local host.
- To serialize the agent and its state before migrate it.
- To pass the agent to the reliability manager.
- To reconstruct the agent and the objects it references.

The reliability manager is used to ensure that the agent, passed by the agent manager, is received by the agent manager on the remote device and to guarantee the persistence of the state information whenever the host fails. The security manger has the job to make sure that only the authorized agent could extract and do the computation within the mobile device. The inter-agent communication provides the layer for the agents to do communication throughout the network. Finally, the application gateway serves as security entrance through which the agent can interact with the host.

Gong-ping et al [7] introduce the mobile agent life cycle and its life states. They define an agent life cycle as "a series of stages through which an agent passes during its lifetime." It is normally used to monitor and to control the transition of the state agent (Figure 3).

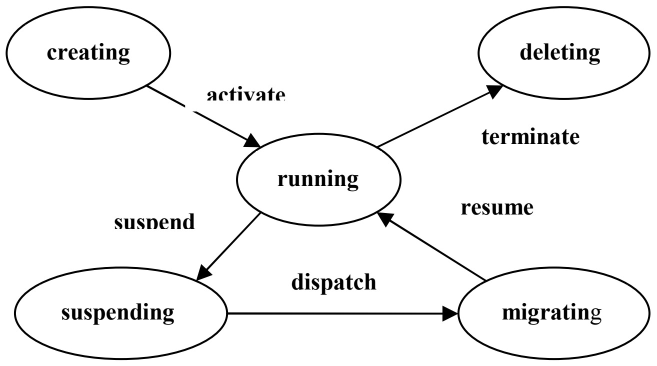


Figure 3. The five states of mobile agent live cycle [7].

Their mobile agents' life cycle consists of five states as describing below:

- The creating state. The state when the mobile agent begins its life cycle (not activated).
- The running state. The agent is activated and is able to perform the actions to accomplish its goals.
- The deleting state. The state when the agent is terminated.
- The suspending state. The state when the agent is in the halt position and stay in the agent server.
- The migrating state. The state when the agent is travelling between two server instances.

Madiraju et al [8] introduces mobile agent technique used in the existing SyD architecture. They claim that mobile agent approach inherently has advantages when compared to the original SyD, which is implemented using Java RMI (Remote Method Invocation). The mobile agent once is transported to a destination host can perform the computation even in the case of the connection failure. The mobile agent returns the result of computation to the host, which initiates requests, whenever the connection is alive. The model of mobile computing used in the scenario is most likely based on client/server model. Mobile devices can work as clients or servers and form ad hoc mobile network. There might be base station (directory server) within the network system. Figure 4 shows their mobile agent architecture using µCode [9].

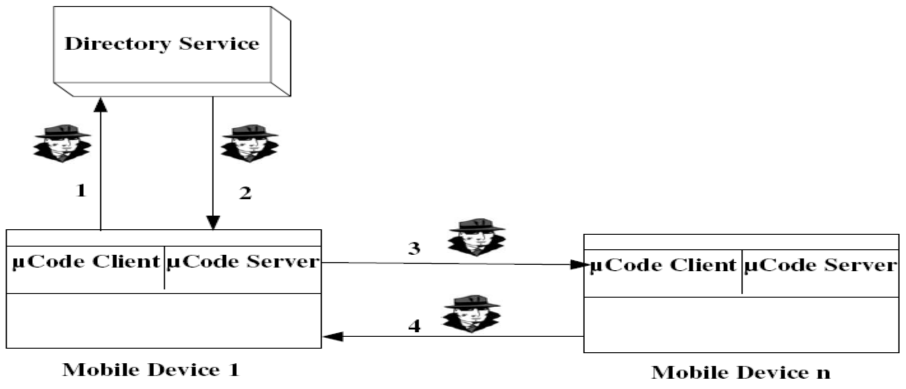


Figure 4. The internal architecture of SyD mobile agent [8].

On the process of execution, based on the SyD platform, the mobile device 1 first of all sends an agent to the directory service or base station to get the physical location information (in the form of IP address) of device *n*. Upon receiving the IP address of the mobile device *n*, the mobile device 1 dispatches mobile agent to mobile devices *n* to perform the computation. In the case, every mobile device has a listener to listen for incoming agents. Finishing performing the computation, mobile device *n* returns the result back through an agent to mobile device 1. In case, if the connection fails between mobile device 1 and *n*, mobile agent sent by mobile device 1 is going to wait and perform its computation on mobile device *n*. It will return back with the result whenever the connection between mobile device 1 and mobile device *n* is established again. The technique provides fault-tolerance in a mobile environment.

2.2 Checkpointing and message logging

Checkpointing [10] is the process of saving the program state, normally into stable storage, so that it can be used for reconstruction later in time prior to failure. The primary purpose of the checkpointing is to provide the backbone for rollback recovery. The combination of checkpointing and rollback recovery allows fault tolerance on failures.

On the other hand, message logging [11] is a technique, which requires that the state information of the mobile device needs to be recorded periodically and the received messages upon successful record transaction is logged. Strom and Yemini [21] explains PWD (piecewise deterministic) within the message logging. PWD ensures the recorded log information by requiring that all nondeterministic events that a mobile device executes could be identified and the information needed for recovery is logged into the event's determinant. Message logging is normally good to be used to interact with the outside world (consist of all input and output devices that cannot rollback).

On the simple scenario, message logging normally consists of fixed numbers of mobile devices that only communicate by transferring messages. Figure 5 shows a simple message logging scenario. m1 and m2 in the Figure 5 describe state intervals, which initially from the nondeterministic events and stored in the deterministic events for the purpose of the consistency. Most of the time, state interval can be recovered if there is sufficient information to replay the execution up to that state interval prior future failures in the system.

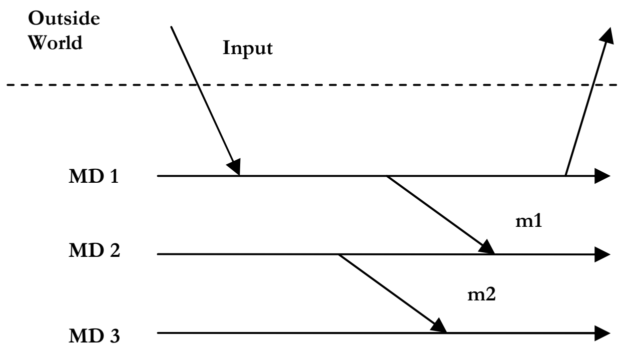


Figure 5. A simple message logging with three mobile devices.

Whenever the failure of the mobile device appears, the mobile device is given the appropriate recorded local state (checkpoint) and the logged messages in the order they were originally received so that it can recover. The recovered mobile device needs to make sure that its state is consistent with others (no orphan, the surviving mobile devices whose states are consistent with the recovered state of a failed mobile device). [12 - 20] are examples of message logging protocols.

What is called by no orphan sometimes produces problem called as rollback propagation. Rollback propagation might enforce the surviving mobile device to rollback to its previous state prior failure with the purpose to maintain consistency, which is normally up to the maximum recoverable state [18], which is the most recent recoverable consistent system state (shown in Figure 6 below).

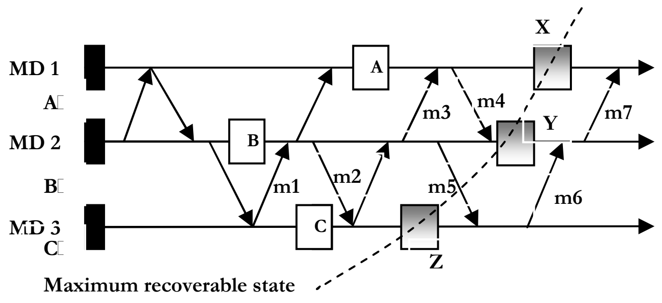


Figure 6. The maximum recoverable state [18].

Based on Figure 6, suppose mobile device 2 and mobile device 3 fail before logging the message m5 and m6, the message m7 becomes orphan message since mobile device 3 cannot regenerate the existence of m6 and mobile device 2 cannot regenerate the existence of m7 without the original m6. Because of the case, mobile device 1 becomes orphan device and is forced to roll back (rollback propagation).

With the formation of state X, Y, and Z to be the most recent recoverable consistent state, consistent recovery could be achieved. Mobile device 1, 2, and 3 will be rolled back to the state A, B, and C respectively since those states are considered to be consistent up to the maximum recoverable state (X, Y, and Z). Upon rollback to the previous state (state B), mobile device 2 needs to replay m1. The similar process is also done by mobile device 1 and 3 by replaying the m3 and m2 respectively.

In the other scenario, rollback propagation might cause the function of the whole system in the network to roll back to the initial state (A', B', and C' shown in Figure 6). The saved state information or the saved work might be gone. The situation is well known as the domino effect

[24], which normally appears as the result of the independent or uncoordinated checkpoint technique.

Chandy and Lamport [23] introduce the coordinated checkpointing technique to hinder the domino effect problem. In the coordinated checkpointing, the mobile devices try to coordinate their state in order to save a system-wide consistent state, which could be used to bound the rollback propagation. A stable storage such as base station normally stores a system-wide consistent state.

A stable storage has to ensure that the recovery state information persists upon the tolerated failures and upon the recovery process. The stable storage such as volatile memory can be used if it is used to tolerate a single failure [25]. If the transient failure within a cell needed to be tolerated, stable storage such as a local disk could be used. But, if the purpose is used to tolerate non-transient failures, there are needs of stable storages such as local disks to be put outside the cell (into another cell) by using replication.

Alvisi and Marzullo [22] bring in three strategies for the message logging: pessimistic, optimistic, and causal. Pessimistic approach tends to log the events periodically to the stable storage. Pessimistic approach helps much on the recovery process, but it hurts on the normal performance. Optimistic, on the other hand, reduces the failure-free performance, but hurts the recovery. And, the casual approach tends to strike the balances between pessimistic and optimistic approach.

2.3 System replication

The replication techniques have been developed since the past, especially in the traditional hardware implementation, which known as N concepts (N-versions). Replication, within the mobile host, such as mobile device, is most likely done by replicating the existing mobile system (redundancy). This method [26] is done with the purpose of retrying the same operation in hope that the failure state could be resolved on the other chances of trying. The concept of redundancy is based on the ad hoc method of recovery block founded by Randell [27], which work mostly in transient faults.

The system replication techniques applied to the software to do fault-tolerant could also be slightly applied as the fault-tolerant techniques on mobile devices. As we already discussed that the nature of the mobile devices are constantly moving, there are problems that could appear within the mobile devices communication, especially the involuntary failure of mobile devices and the wireless link disconnection problem. Not only the movement of mobile devices might cause the problems, but their designs, with the limitation of storages, and the purpose of saving the energy also cause the problems.

Normally, there is one base station, stable storage stores mobile devices information, in each cell of the network. The connection between the mobile devices (MDs) and the base station (BS) within a cell are normally happen through wireless medium. Beacon protocol [28] is normally used as one of the protocols which a mobile device establishes its contact with the new base station following by informing the id from its previous base station. The following Figure 7 below is the normal scheme of the mobile devices networks.

It is more likely that mobile devices are highly dependent on the base station. The scenario model makes the base station to be highly fault-tolerant or persistent. If not, the failure of the base station will make big troubles on the mobile devices. All the important state information stored within the base station is gone. Mobile devices are forced to wait or freeze until the base station is being recovered.

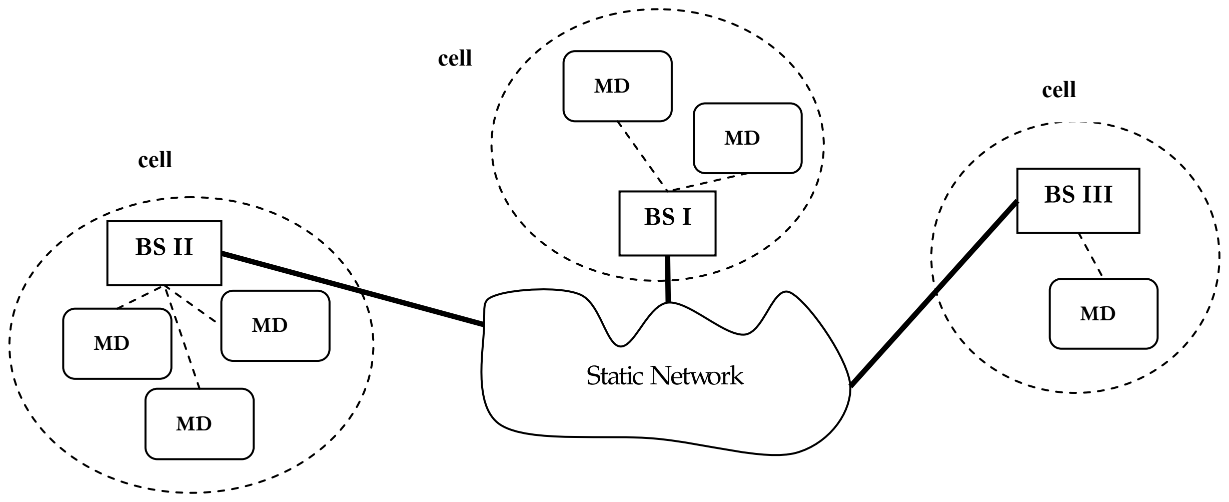


Figure 7. The mobile device network model.

On replicating the base station, Alagar et al [38] in their paper propose two schemes to tolerate the simultaneous failures of base station up to k numbers of base stations. They do it by replicating the information stored at the primary base station into several secondary base stations. If the base station fails, the mobile devices within its cell could switch into one of their secondary base stations to continue their computation. The switching might cause the movement of the mobile devices to the new cells, which carry their state information.

In their model, they assume that a logical communication channel exists between every pair of base stations, which is done only by message passing. Communication channels are FIFO with the finite amount of time to deliver messages. Fail-stop failure model [29] is used as the model for the base station with the purpose of hiding the visibility of the error by responding to the internal failure. Mechanism which the failure of a base station can be detected by its neighbors and the mobile devices in the cells is assumed to be there.

Mobile devices store their state information in their base station. All communication among the mobile devices happens through and with the control of the base stations. As in Figure

7 as an example, in order for the MD in BS I to communicate with MD in BS II, MD in BS I needs to send the package/message into BS I first. Receiving package from MD, BS I sends it to BS II. Finally, BS II sends it to the appropriate MD. Within a cell, BS also works as a medium to send the package among MDs themselves. This way, the BS holds all the state information of its MDs. Its failure causes the MDs to halt till it recover.

Alagar et al [38] overcome the problem by replicating the state information of mobile device into several secondary base stations. Each of base stations has its sets of selected secondary BSs. The secondary base stations must be at least k in order to tolerate k numbers of BS failures.

They call their schemes as pessimistic and optimistic replication. In pessimistic replication method, the primary/original base station needs to ensure that all selected secondary base stations maintain the same state of the mobile device. The only delay that happens here is in delivering the messages/packages to base station or mobile devices. Once the primary base station fails, mobile device can switch to one of the secondary base stations (there is some delay happens in the process) and continues its computation without any delay.

On the other hand, optimistic replication replicates the state information of the mobile device asynchronously. In the optimistic method, there is no delay on delivering the messages/packages to mobile devices or base stations since the state information is transferred into one of its secondary base stations whenever its primary base station fails. But, the technique makes recovery process costly whenever mobile device switches to one of the base stations upon failure of its primary base station.

Two strategies on selecting the secondary base stations are also being introduced. The first strategy considers a certain localities of the mobile device's movements (mobility pattern of a mobile device is known). The candidates for the secondary base stations are a fixed set within the locality. The state information of the mobile device within primary base station is maintained to its secondary base stations. The technique causes the mobile device's movement easily and does not require additional handoff procedure.

On the second strategy, the selection of the secondary base stations is the neighbors of the primary base station. This technique is based on the assumption of the dynamic base stations. Therefore, the neighbor of base station might not be the same all the time. Movement of the mobile device from its primary base station's cell into other cells might require the copy of the state information of the mobile device in the original base station and the costly handoff procedures.

Using the same model as Alagar, Rangarajan et al [30] also introduces some replication technique. The state information of the mobile devices, which is stored at the primary base station, is replicated into other base stations that their coverage overlaps the primary base station's coverage. With the technique, if the primary base station fails, the mobile device could retrieve its state information from other base stations that overlaps its primary base station. If there are no other base stations that overlap its primary base station, then the mobile device might lose its information upon the failure of its primary base station.

Gifford [31], in his paper, introduces the replication technique known as weighted voting. The technique introduced is mainly on the algorithm for the maintenance of the replicated state information in mobile environment (or replicated files in the distributed system). Although his technique is more into the fault-tolerant of distributed computing system, it can be categorized as the replicating technique for mobile environment since the mobile system environment is more likely similar to the distributed system environment. The rest of the replication techniques is more likely on the client-server approach on mobile environment.

The replicated information is stored in the stable system such as base station, which each copy of the replications is assigned some version numbers. Copies of the replications are shared into others group base stations within the network. Votes are assigned to the base stations carry the copies.

A serial transaction is used when the mobile device tries to access its information in the base station. In the situation which the base station fails, the technique will create several requests of information to other closed base stations in parallel. Numbers of votes responded by the base stations are weighted in order to get the correct decision requests.

Herlihy [32] explores an alternative approach to managing replication information by presenting two replication methods in which concurrency control and replica management are handled by a single integrated control. Remember that replication technique tends to copy the information and stores them at the multiple base stations to enhance the availability whenever one of the base stations fails. Concurrency and replica management become so much important in the case.

If the mobile device, upon failure the mobile host, has method to retrieve its state information from many other base stations that store it, concurrency is needed so that it ensures the incorrect behavior cannot occur as the result of concurrent access by mobile device. On the other side, replica management is used to ensure that mobile device can gather its state information back from other sources of base stations whenever its base station fails. A single integrated technique introduced to manage the performance between concurrency and replica management in order they can be traded off: constraint on concurrency may be relaxed by tightening constraints on availability, and vice versa.

Upon the failure of the base station, mobile device retrieve its state information to others of its base stations using what is called as a schedule. A schedule maintains the concurrency of the transaction that could appear when the mobile device retrieving its information from many sources of base stations. Others [33 - 35] also use similar techniques of replication to maintain the availability of the state information of the mobile device.

Satyanarayanan [36], based on the CODA file system, also introduces the replication technique that are optimistic than the one introduced by [31 - 35]. The optimistic approach is used to ensure the consistency of the replicated state information on many base stations (problem in pessimistic approach) and to restrict the placement of unacceptable limits of replication information.

Disconnected operation and replication on base station are introduced with the purpose of handling transient transaction (fault - tolerant) within mobile devices system. Sometimes, a cache is also implemented into a mobile device with the purpose of reducing the dependency of a mobile device with its base station. In case of all base stations, hold the replicated state information, fail, a mobile device can still retain its state information through its cache storage to continue its computation.

Hara's [37] method of replication can also be applied to a mobile device by replicating the state information in the mobile device, including itself, within the same network. This is based on the assumption that if the base station fails, mobile devices in the cell can form ad hoc network and still maintaining the connection with other cells.

2.4 SyD proxy module using replication technique

Looking at the techniques on handling the failures done by the researches above, we come to the conclusion to use the replication technique as the base on doing disconnection tolerance of transactions for System on Mobile Devices (SyD) platform. The one, which is going to be replicated, is not the base station, but a mobile device, which provides services (normally mobile device server). The replication itself is done by implementing the SyD proxy module in another mobile device. It means that a mobile device, which implements SyD proxy module, acts as a proxy. It replicates and also provides similar services applications with different functionalities. Therefore, we could say that proxy is a "substitution." Figure 8 below displays the replication technique on SyD implementing SyD proxy.

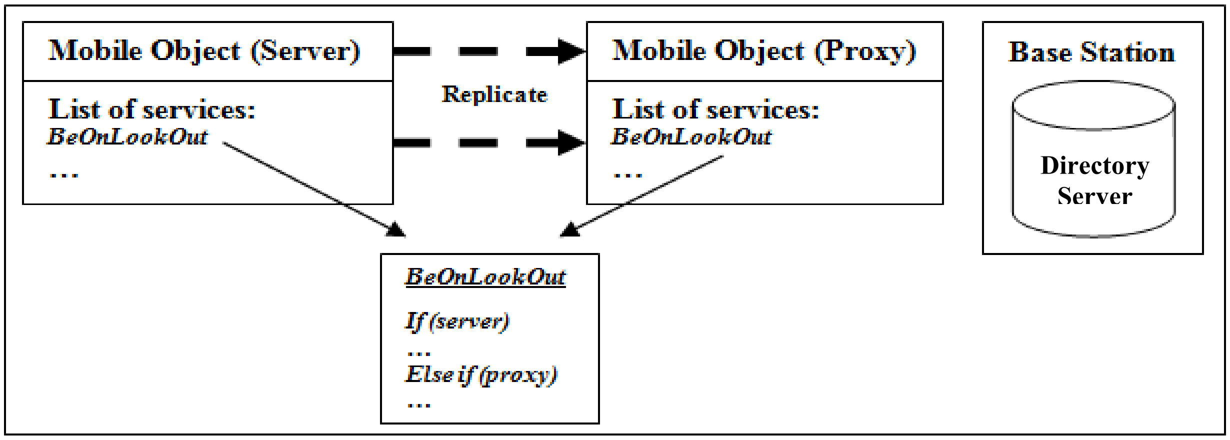


Figure 8. The replication technique on SyD.

In case there is a failure on the mobile device providing services, the proxy, implementing the proxy module, comes as a solution to provide a "substitution." The proxy recovers the failure by providing similar services with "smart" solution. The proxy informs the client, requesting services, that the requests have been received. Then, it stores the state transaction requests of the clients. Upon synchronization with the mobile device, proxy lets the mobile device to gather all stored state of transactions. Finally, mobile device transacts the state transaction requests.

Functionality, design, and implementation of our proxy module extension on SyD could be seen on chapter 4. Performance analysis of our SyD proxy and an application running SyD proxy are introduced in chapter 5 and 6 respectively. Chapter 3 tells about the existing SyD platform.

3. OVERVIEW OF SYSTEM ON MOBILE DEVICES (SyD)

SyD was developed by Yamacraw Embedded System research team with the goal of providing such kind of middleware platform for mobile devices that allows: uniform connected view of device, data and network; ease of development and deployment of distributed server applications hosted on mobile devices; high level development and deployment environment. Therefore, SyD claims to be [39] "a new platform technology that addresses the key problems of heterogeneity of device, data format and network, and mobility." To achieve its goal, SyD models a mobile device running as an object (based on Object-Oriented approach). Each object of mobile devices could run as a client or as a server. Sometimes, it could run both as a server and a client within a mobile device object.

3.1 SyD framework

Each of the mobile devices runs SyD middleware platform is an object. As an object, the device is assumed to be independent of each other and does not share a global schema. Together, all object devices cooperate to perform interesting tasks. To achieve its tasks, SyD has the following framework, as described in the Figure 9 [39] below.

At the lowest layer, SyD Deviceware consists of a listener module and an engine module. A listener module is normally used to register objects and to execute local methods in response to remote invocations. On the other hand, an engine module is used to invoke methods on remote objects. This layer also contains individual data stores, represented by device objects, and methods or operations for access and manipulation on the data. Simply, at this layer, a mobile

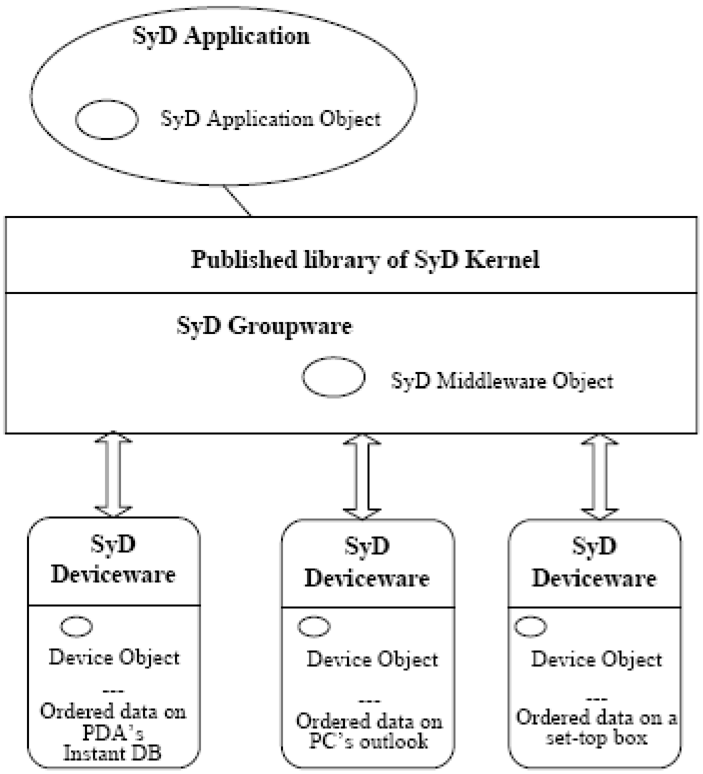


Figure 9. The System on Mobile Devices (SyD) framework [39].

device is an object that could act as a server, registers and provides services, as a client, invokes services on remote objects, or as both client and server.

At the middle layer, there is SyD Groupware, which is a logically coherent collection of services, APIs, and objects to facilitate the execution of application programs. The layer consists of a directory service module, group transactions and global event support, with application-level Quality of Service (QoS). Simply, the middle layer is where SyD middleware platform is located. At the highest level, we could find SyD applications, which rely only on these groupware and deviceware SyD services, and are independent of device, data and network. An instantiation of server object, which contains an aggregation of the device object and SyD middleware object, is included in the layer. As the conclusion, each of mobile devices runs SyD applications is an object, which is following Object-Oriented approach.

Those three layers architectures of SyD enable applications to be developed without knowledge of device, database and network details. SyD groupware holds the most important roles in the framework since SyD middleware platform is located in this layer. It provides a directory service module to interact with the Directory Server, a storage storing information for all registered SyD object devices. Therefore, the layer is responsible to make the object device applications (anywhere) aware of the named objects and their methods or services, to execute these methods on behalf of applications, to allow the construction of SyD Application Objects (SyDAppOs) that are built on the device objects. It provides only a named device object for use by the SyDApps, applications written for the end users that operate on the SyDAppOs alone and are able to define their own services that utilize the SyDAppOs, without revealing the physical address, type or location of the information store.

3.2 SyD kernel and its modules

SyD uses simple yet powerful idea of kernel or core system to develop applications within mobile devices. SyD kernel captures the essential features of the overall SyD framework and several SyD based applications. Figure 10 [39] below describes the kernel application of the existing SyD middleware platform.

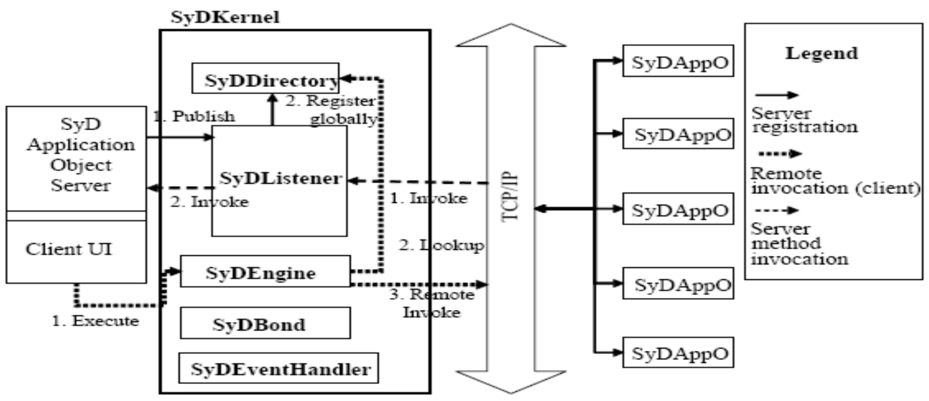


Figure 10. The architecture of SyD kernel and its application [39].

SyDDirectory, SyDListener, SyDEngine, SyDBond, and SyDEventHandler are modules developed within the SyD kernel architecture. Those are the main core of the existing SyD platform.

- **SyDDirectory**

The module is located in the middle layer of SyD framework (SyD Groupware). It provides users' objects publishing, management, and lookup services to SyD device objects. Basically, it contains all information of every registered object devices (normally server application objects), including their locations, their methods of services, and their important information. Figure 11 below is the state transition of existing SyDDirectory.

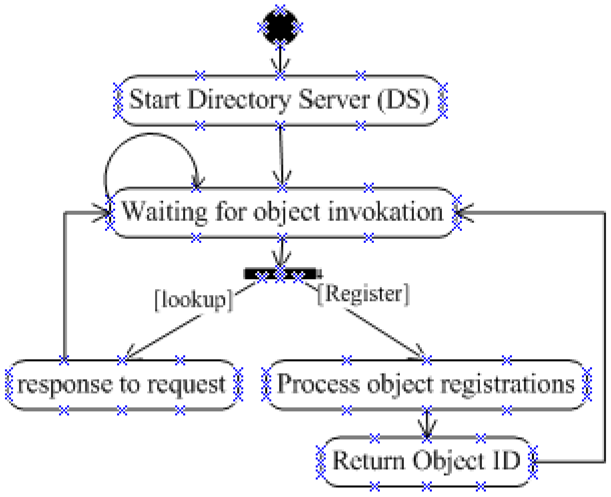

Start Directory Server (DS)
Waiting for object invokation
[lookup]
[Register]
response to request
Process object registrations
Return Object ID

Figure 11. The state transition of SyDDirectory.

SyDDirectory module is quite important module and has to be started first of all so that all other SyD applications could do their jobs, such as providing services, requesting services, registering services, etc. As simple as we could say, it works as a mediator, opening the connection to Directory Server. After it is started, it waits for the objects' requests.

- **SyDListener**

It is normally located at the bottom layer of SyD framework. SyD object devices use it to publish their services (server applications) locally in the device and globally via Directory Service. SyDRegistrar module in SyDListener is used for that purpose. Figure 12 explains the state transition of the existing SyDRegistrar in SyDListener module.

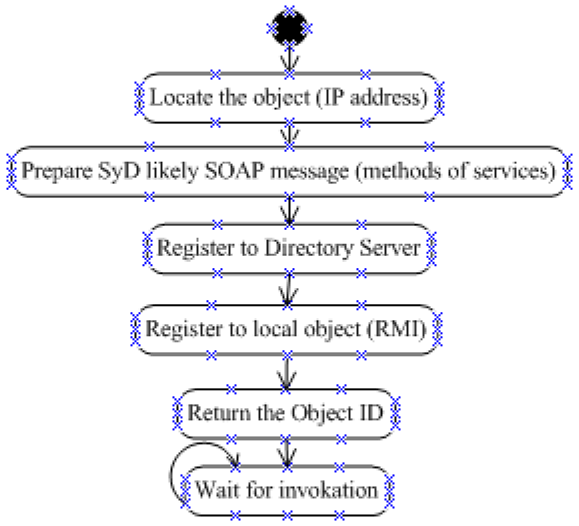

Locate the object (IP address)
Prepare SyD likely SOAP message (methods of services)
Register to Directory Server
Register to local object (RMI)
Return the Object ID
Wait for invokation

Figure 12. The state transition of SyDRegistrar.

When the object device (normally server applications) registers its information, including its identities and services, it is going to find where it is located (normally IP address) first. Then, it collects all documents, in XML, containing the methods of services, address location, all ports either local RMI port or remote listener port, and etc. If the application object has not been registered yet, it then gets registered golabally in the Directory Server and locally in the device object. The object now waits for remote invocation.

It also works as a "listener" to listen and to ivoke local registered services in the device upon requests. SyDListener module in SyDListener is used for the purpose. Simply to say, it opens the port communication between the object server and the object client. It also allows for the object client and server to communicate each other remotely by exchanging documents, in XML. The object client uses SyDListenerDelegate module for this purpose.

- **SyDEngine**

It is located at the bottom layer of SyD framework. SyDEngine allows the client to dispatch, using SyDDispatcher module, services remotely and to aggregate the results. The module will make the invocation on the remote object transparently. Figure 13 pictures the state transition within the existing SyDEngine.

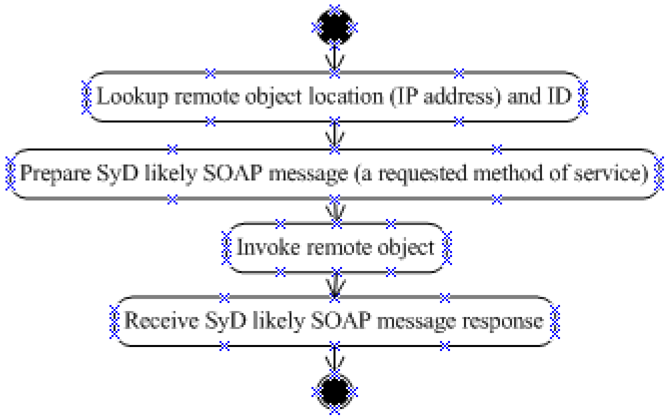

Lookup remote object location (IP address) and ID
Prepare SyD likely SOAP message (a requested method of service)
Invoke remote object
Receive SyD likely SOAP message response

Figure 13. The state transition of SyDEngine.

In order for the user's client to invoke services remotely, it needs to get the remote object's url, which is listed as IP address. A remote object is usually the object server. The user's client, then, bundles all of the needed information into a XML document. The information normally contains a method of a service and its parameters values. The client sends the XML message document to the remote object server. The object server un-bundles the message document and processes the client's request. It, then, sends the result back to the user's client in the form of XML message documents also.

- **SyDBond**

  It enables a SyD object device to link to other devices for automatic updates and to create and enforce interdependencies.

- **SyDEventHandler**

  It handles local and global event registration, monitoring, and triggering.

4. PROXY MODULE EXTENSION FOR SYSTEM ON MOBILE DEVICES (SyD)

Mobile device may become unavailable due to the following reasons. First, involuntary failure of mobile device, such as loss of battery power, mobile device becomes overloaded by requested service computations, etc. Second, voluntary disconnection of the mobile device by their users, such as users shutdown the system. Third reason might be caused by the wireless link failure or overloaded.

Therefore, there is a need to build a platform that could handle those failures on mobile device system. The existing SyD platform, taken as one of the mobile device's platforms and has been described in chapter 3 above, could not handle those failures. As a result, the existing SyD platform has been extended for the purpose of implementing disconnection tolerance of transactions in the mobile device system. The proxy comes as a solution and its module has been constructed in the existing SyD platform.

The extension of SyD not only considers the disconnection tolerance of transactions, but it also allows the synchronization to happen between mobile devices (normally between the mobile object server and its proxy). The existing SyD framework is still used for this implementation, but the existing SyD kernel has been modified and extended. SyDDirectory, SyDListener, and SyDEngine are the main SyD kernel that has been extended. SyDSync has been added for the purpose of server-proxy synchronization.

4.1 SyDProxy functionalities

The main reasons to build the proxy module as the extension module in the existing SyD platform (SyDProxy) are as following:

- To handle failures within mobile device system by letting the disconnection tolerance of transactions to happen (the mobile object client's request could be handled by the mobile object proxy temporarily upon the failure of the mobile object server).
- Since disconnection tolerance of transaction needs to happen, there is also a need for allowing synchronization of the transaction to happen within mobile device system (upon the time for the mobile object server to be in active mode, it allows the object server to synchronize its state of data with its object proxy to process the mobile object client's transaction request and to give a response).

To let those functionalities happen, the SyDProxy module has been built to extend the main existing SyD kernel, such as SyDDirectory, SyDEngine, and SyDListener. The purpose is to allow disconnection tolerance of transaction to happen. In addition to that, SyDSync has been constructed for the purpose of synchronization. Figure 14 displays the SyDProxy module.

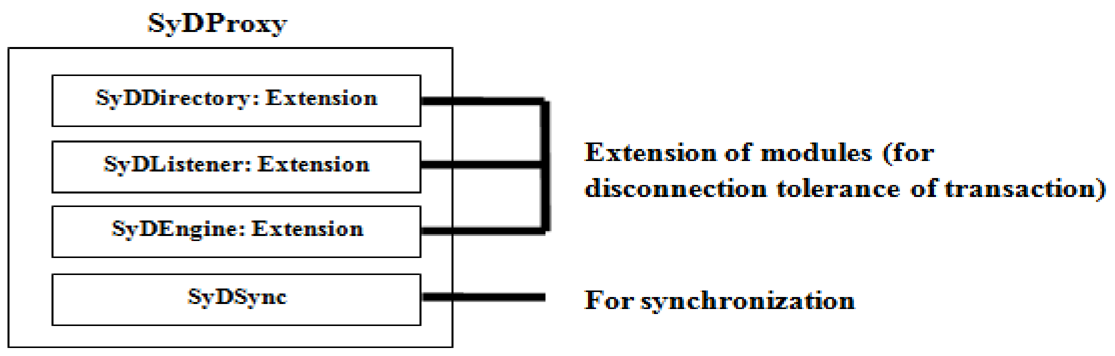


Figure 14. SyDProxy module.

In the design of proxy module for SyD, we consider the following failures conditions:

- Voluntary disconnection.

The status of an object, which provides services (an object server), is in off mode. It means that a mobile device (a server) has been shut down. The device at that time could not provide services.

- Involuntary disconnection.

An object, which provides services (an object server), is in busy mode. That means an object does too many workloads for a request. So, it could not handle the other received requests. Or, an object could not listen to a client's request eventhough the Internet network is still connected. This might happen because of an object server's listener is in off mode.

- Wired or wireless link is overloaded.

An object server's listener is getting too busy handling too many clients' requests. Thefore, it becomes overloaded.

The new design of SyD, implementing proxy module, now could handle those failures (allowing disconnection tolerance of transactions and synchronization). Figure 15 displays the simple scenario. Condition 1 allows normal invocation of a client to a server. If listener of a server is in off or busy mode, client's invocation is handled by a proxy (condition 2). The similar situation also appears when the server itself is in off or busy mode. Synchronization appears between the server and its proxy.

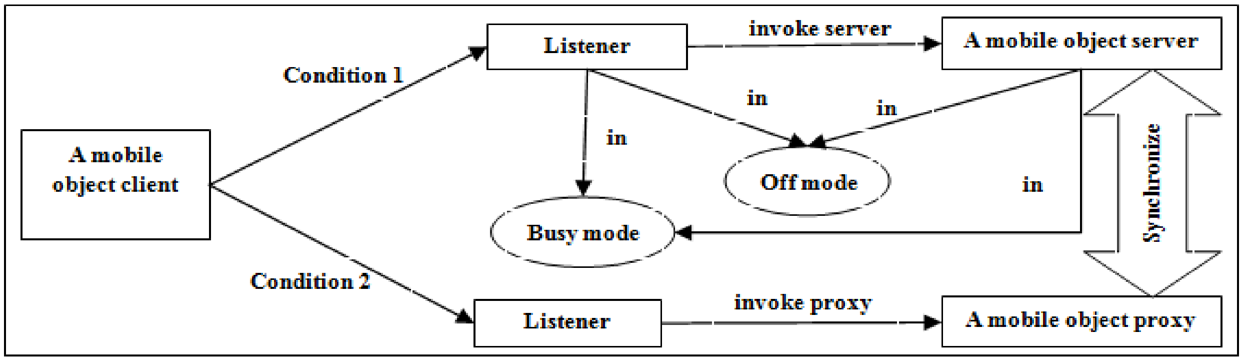


Figure 15. The simple scenario where the proxy comes as a solution to handle failures.

4.2 SyDProxy design architecture and its description

With the proxy module that has been enabled for SyD, SyDDirectory could provide users' objects publishing, management, and lookup services not only to SyD device objects but also their proxies now. SyDListener, using SyDRegistrar module, also allows the registration process of SyD device objects with their proxies. SyDEngine lets seamlessly remote transactions to occur either to the SyD device objects or to their proxies. Finally, SyDSync allows the synchronization of the data between the objects and their proxies to happen. Figure 16 below explains the higher level view of SyDProxy module extension (disconnection tolerance and synchronization) on top of existing SyD platform. The details of the use of the proxy module extensions for SyD, including the SyDSync module, are explained below.

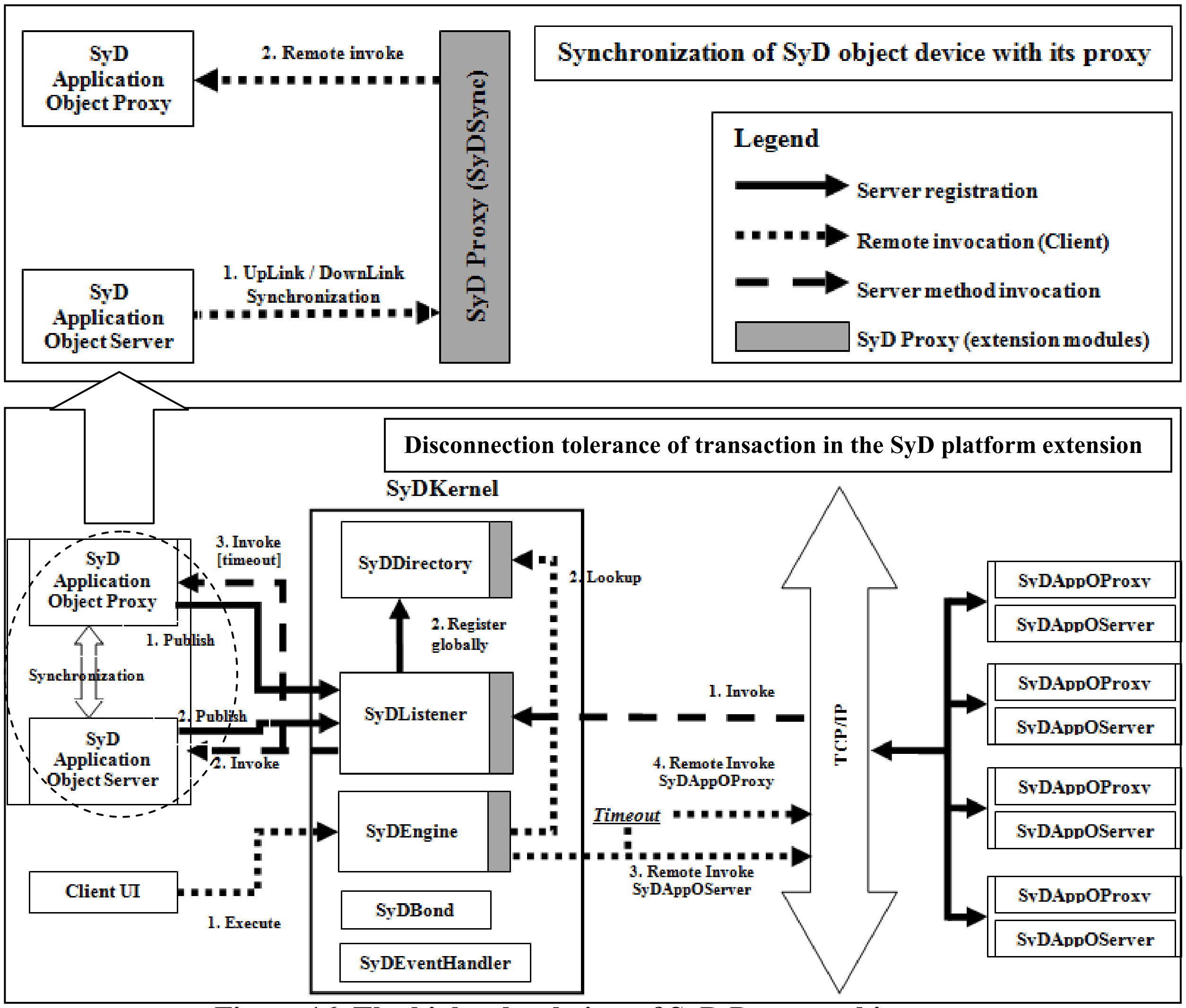

Synchronization of SyD object device with its proxy
SyD Application Object Proxy
2. Remote invoke
SyD Proxy (SyDSync)
1. UpLink / DownLink Synchronization
SyD Application Object Server
Legend
Server registration
Remote invocation (Client)
Server method invocation
SyD Proxy (extension modules)
Disconnection tolerance of transaction in the SyD platform extension
SyDKernel
SyDDirectory
2. Register globally
2. Lookup
SyDListener
SyDEngine
SyDBond
SyDEventHandler
3. Invoke [timeout]
1. Publish
Synchronization
2. Publish
2. Invoke
Client UI
1. Execute
1. Invoke
4. Remote Invoke SyDAppOProxy
Timeout
3. Remote Invoke SyDAppOServer
TCP/IP
SyDAppOProxy
SyDAppOServer

Figure 16. The higher level view of SyD Proxy architecture.

Looking into Figure 16 above, all SyD objects, includes SyD object proxy, SyD object server, not Client, before it can be located and can provide methods of services, need to be registered first. Registration is done using SyDRegistrar module in SyDListener, which is going to be registered globally in the Directory Server and is going to be registered locally into the RMI registry of the object afterward. After registration, every registered SyD object is ready to provide services and is considered as a SyD object server. In the case, SyD object server and SyD object proxy are normally servers. They also need to listen to every single request of services from clients.

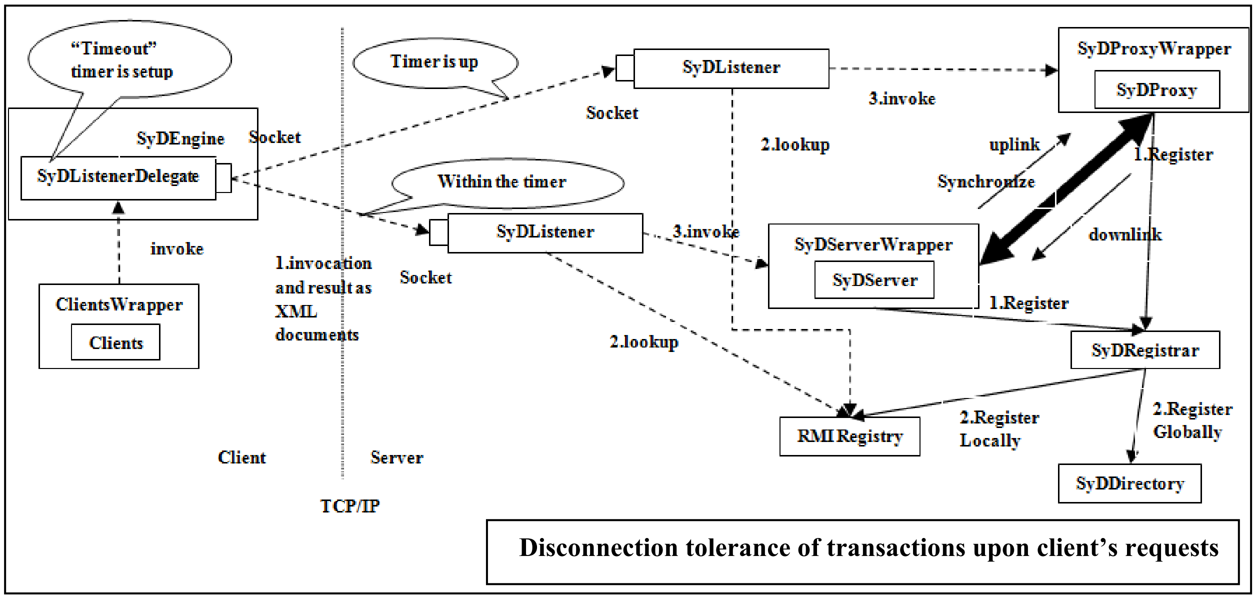
For that reason, SyDListener is provided for each of the SyD object servers. Practically, SyDListener keeps the assigned port and uses it exclusively only to listen and to execute local method of a service based on a remote request. The request is normally in the XML documents. SyDListener gets the request, extracts and executes it locally, and sends a response on behalf of the SyD object server. Since SyD objects are independent of each others, each of the SyD object servers is required to have one SyDListener to listen and to execute.

The one, which requests SyD server's services, is normally SyD object client (Clients). To request a service, a client needs to use SyDEngine. In the process of invoking a request to a SyD object server, a request is bundled into SyD documents, using SyD standard XML, which is similar to SOAP message, within SyDEngine. SyDEngine uses SyDListenerDelegate to send the request of document afterward. The communication between the object client and the object server using SyDListenerDelegate and SyDListener is done through TCP/IP.

There is a time, a "timeout", to allow a client to wait for a response from a server after invocation. The reason behind implementing a "timeout" here is for not letting a client to wait a response from a server too long. A server might be overloaded with works or might be off. If a client does not get a response at the end period of a "timeout" time, SyDEngine, used by a client, seamlessly invokes a request to a proxy.

The proxy works smartly in the essence of keeping of all requests, which could not be processed by a server at the time of client's requests of invocation. The proxy also acts similarly as a server by sending a response to a client that a request of a service is accepted. In this case, client does not know if a server is in a failure mode since a proxy also sends a similar receipt of an acceptance of a request. At this case, a client only thinks that its request is accepted and is going to be processed. A client then waits for a response.

All requests kept by a proxy could be processed later by a server if a server knows that there are requests of services for it. To let a server knows if there are requests for it, synchronization, using downlink module, between a server and a proxy is needed here. Upon the synchronization (SyDSync) process, a server gets all stored requests from a proxy. A server extracts requests and processes each of the stored requests. The chapter below will explain the details of the process of synchronization. Figure 17 below shows the overall SyDProxy architecture, which is approached from lower level design.



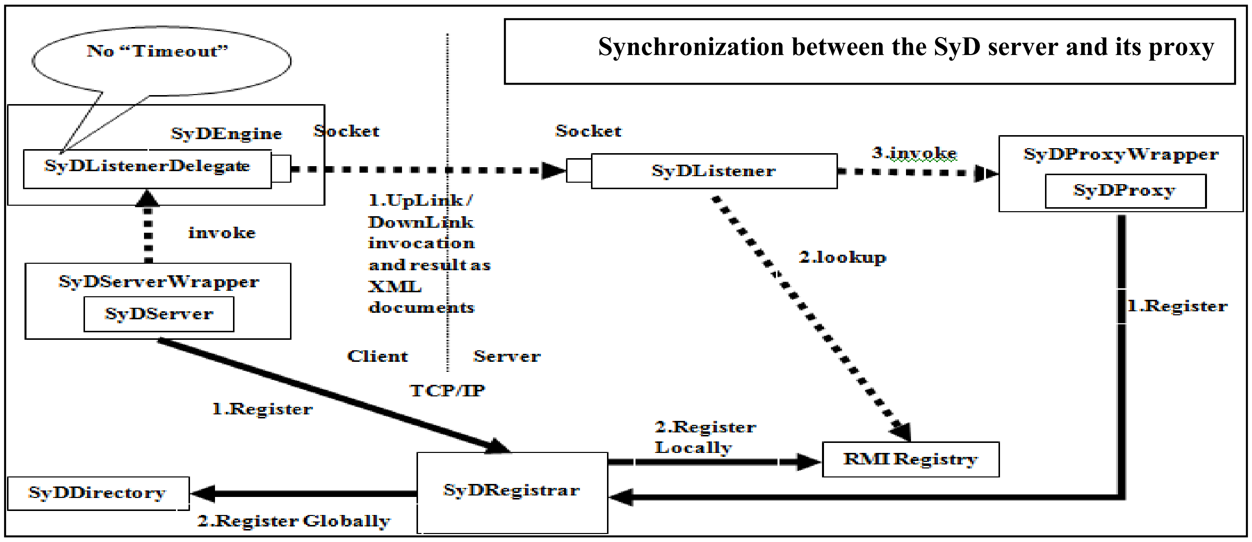


Figure 17. The lower level design of SyDProxy architecture and its application modules.

SyDDirectory

As it has been mentioned, SyDDirectory is used to provide users' objects publishing, management, and lookup services to SyD device objects and their proxies. Its module provides interaction with all information of every registered object devices, including their locations and their methods of services. It is an interface to connect into Directory Server (could also be called as a base station). Directory Server's function is used as storage of all device objects' information. Figure 18 shows the storage (database) schema of Directory Server.

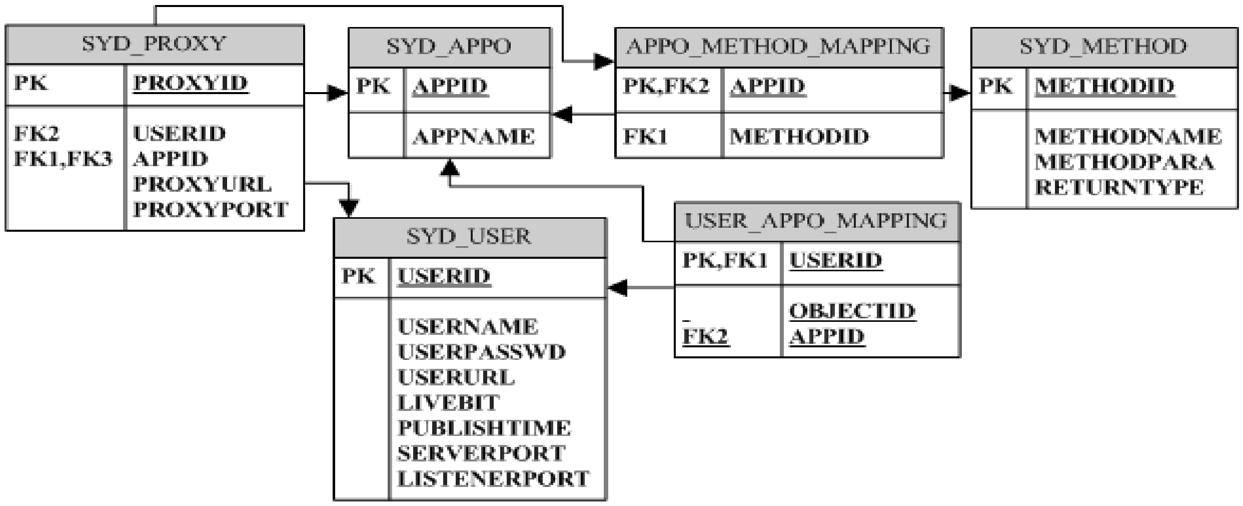


Figure 18. The storage (database) schema of Directory Server.

There are 6 objects in the Directory Server schema as shown in Figure 18: SYD_USER, SYD_APPO, SYD_PROXY, SYD_METHOD, APPO_METHOD_MAPPING, and USER_APPO_MAPPING.

- **SYD_USER**

It is an object, which contains all information of the registered users' applications, normally server applications. It stores server application name and password for identification purpose. Where the application is published (userurl) and its published time (publishtime) are also recorded in the object. Application location is normally shown as IP address. At the end, it also contains the condition of the application (livebit – on/off), the local RMI device port, which contains registered methods of services (serverport), and the object listener port for remote invocation (listenerport).

- **SYD_APPO**

It contains the identity of the registered application, such as application name.

- **SYD_METHOD**

It lists all registered methods of services, such as method names, parameters of the methods, and the return types of the methods.

- **APPO_METHOD_MAPPING**

It holds the relationship of the applications with their registered methods of services.

- **USER_APPO_MAPPING**

  It maps the registered users with their registered applications.

- **SYD_PROXY**

  It contains the relationship of the registered users with its published proxies and its published applications.

The way SyDDirectory works is still similar to the existing one, but with the extension of allowing proxy publication or registration to happen. Figure 19 is the state transition that happens in Directory Service of SyDDirectory.

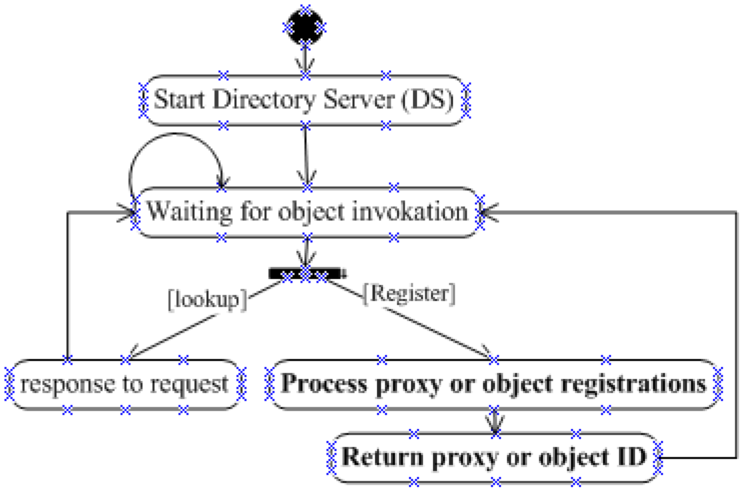

Start Directory Server (DS)
Waiting for object invokation
[lookup]
[Register]
response to request
Process proxy or object registrations
Return proxy or object ID

Figure 19. The new state transition of the SyDDirectory.

Directory Server in SyDDirectory needs to be started first in order that object server applications could register and setup their methods of services or object client applications could fetch information or requests. Since the extension of SyD has proxy module enabled, proxy needs to be published or registered first in the Directory Server.

Once it is registered, a proxy object ID is listed(remember that SyD makes a device as an object and provides only a named device object). Object application, normally object server application, which will be registered after the proxy, needs to provide proxy object ID for the registration process. That way, object application will have its object proxy. At the current development, one registered object application, which is an object server application, only has one object proxy. It is likely to say that one server has one proxy.

SyDListener

Located in the bottom layer of SyD framework, it has three main modules (described in chapter 3): SyDRegistrar, SyDListener itself, and SyDListenerDelegate. SyDRegistrar is used by SyD object devices to publish their services (server applications) locally in the device and globally via Directory Server. Figure 20 describes the state transition of proxy and server object registration in the extended SyDRegistrar.

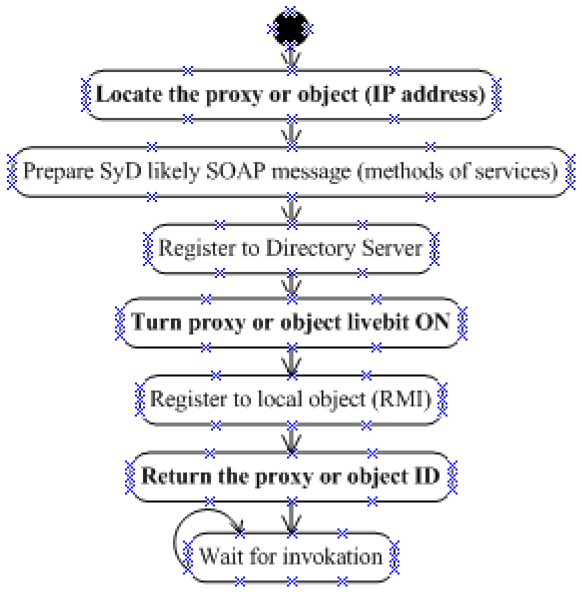

Locate the proxy or object (IP address)
Prepare SyD likely SOAP message (methods of services)
Register to Directory Server
Turn proxy or object livebit ON
Register to local object (RMI)
Return the proxy or object ID
Wait for invokation

Figure 20. The state transition of proxy and object registration in SyDRegistrar extension.

As you can see the difference of SyDRegistrar from the previous one, every user's object application, either already registered or not registered, is going to turn its status livebit ON. This is a signal that the application is up and running.

SyDListener is a "listener," which is used to listen users or other applications objects' methods invocations. Since it is the only way to do the communication between the remote objects, it opens the communication port for the registered server applications. The object client's requests of services are received by the object server through the open communication port. Once the requests are received, it locally accesses and executes the active services from the object's local registry.

On the other hand, SyDListenerDelegate allows the SyD object device (object client) to communicate with other devices (normally server applications and proxy application) remotely to exchange the data. A "timeout" has been implemented in the SyDListenerDelegate with the purpose of controlling client's request of service to get seamlessly process either to the server or to its proxy. SyDListenerDelegate and SyDListener is a pair of TCP/IP socket communication within SyD. The only difference is that SyDListenerDelegate is in the client applications side and SyDListener is in the server applications.

SyDEngine

It is located at the bottom layer of SyD framework. It allows users to dispatch (SyDDispatcher) services remotely and to aggregate the results. SyDEngine allows transparent services invocation. Figure 21 below shows the state transition process of the extended SyDEngine using SyDDispatcher module to allow seamlessly remote method invocation.

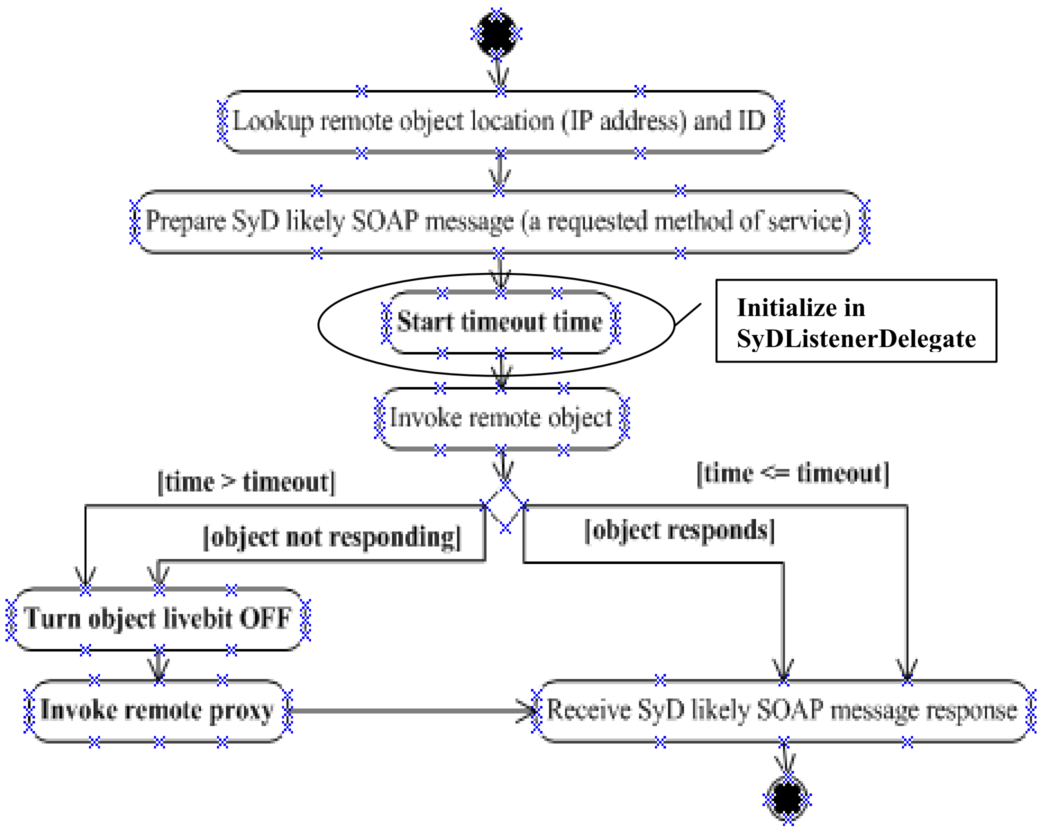

Lookup remote object location (IP address) and ID
Prepare SyD likely SOAP message (a requested method of service)
Start timeout time
Initialize in SyDListenerDelegate
Invoke remote object
[time > timeout]
[time <= timeout]
[object not responding]
[object responds]
Turn object livebit OFF
Invoke remote proxy
Receive SyD likely SOAP message response

Figure 21. The SyDEngine extension for seamlessly remote method invocation.

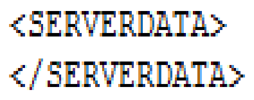
The extension of SyDEngine is different than the previous one, shown in the chapter 3. The extension of SyDEngine holds the important roles of controlling the flow of client's requested services, either to get responses from the server or from server's proxy. There is a "timeout" that has been introduced on SyDListenerDelegate that carries through SyDEngine.

SyDListenerDelegate is the one, which initiates the "timeout" time. SyDEngine then determines whether it should invoke the server or the proxy within or after the "timeout" time. Within the allowed time (time <= timeout), the client is going to get the services' responses from the server. But, if it is over the allowed time (time > timeout), the client is going to get the responses of services from the proxy. Proxy is a "substitution" of the actual server application.

SyDSync

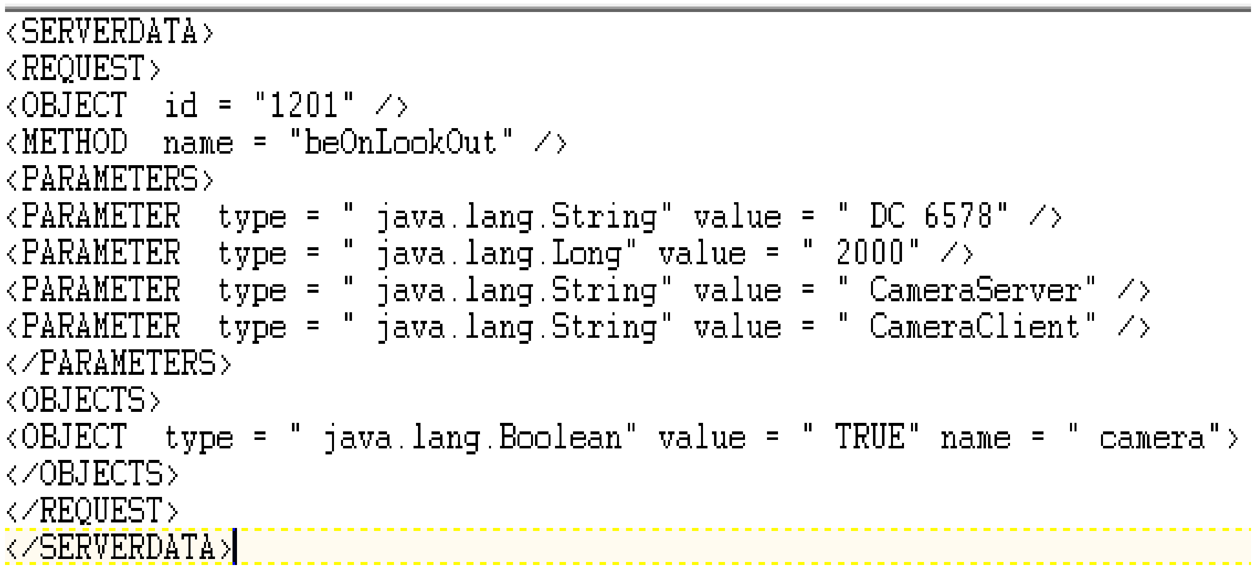
In the case that a server application is busy or not alive, the proxy comes to substitute the server. The proxy substitution has similar functionalities to its server application. It is a “smart” proxy in the essence of keeping all of the states of the transactions of the requested services that are missed. Synchronization between the server and the proxy could happen upon user’s request within the user’s sync time. To allow that happen, we have designed and implemented SyDSync module to ease the SyD server developer’s job.

We try to make it as uniform as possible by following the existing SyD architecture. If there is a failure in the object server, requested services will be stored as a state of XML data by the object proxy. Figure 22 below shows the XML format data of stored state request.



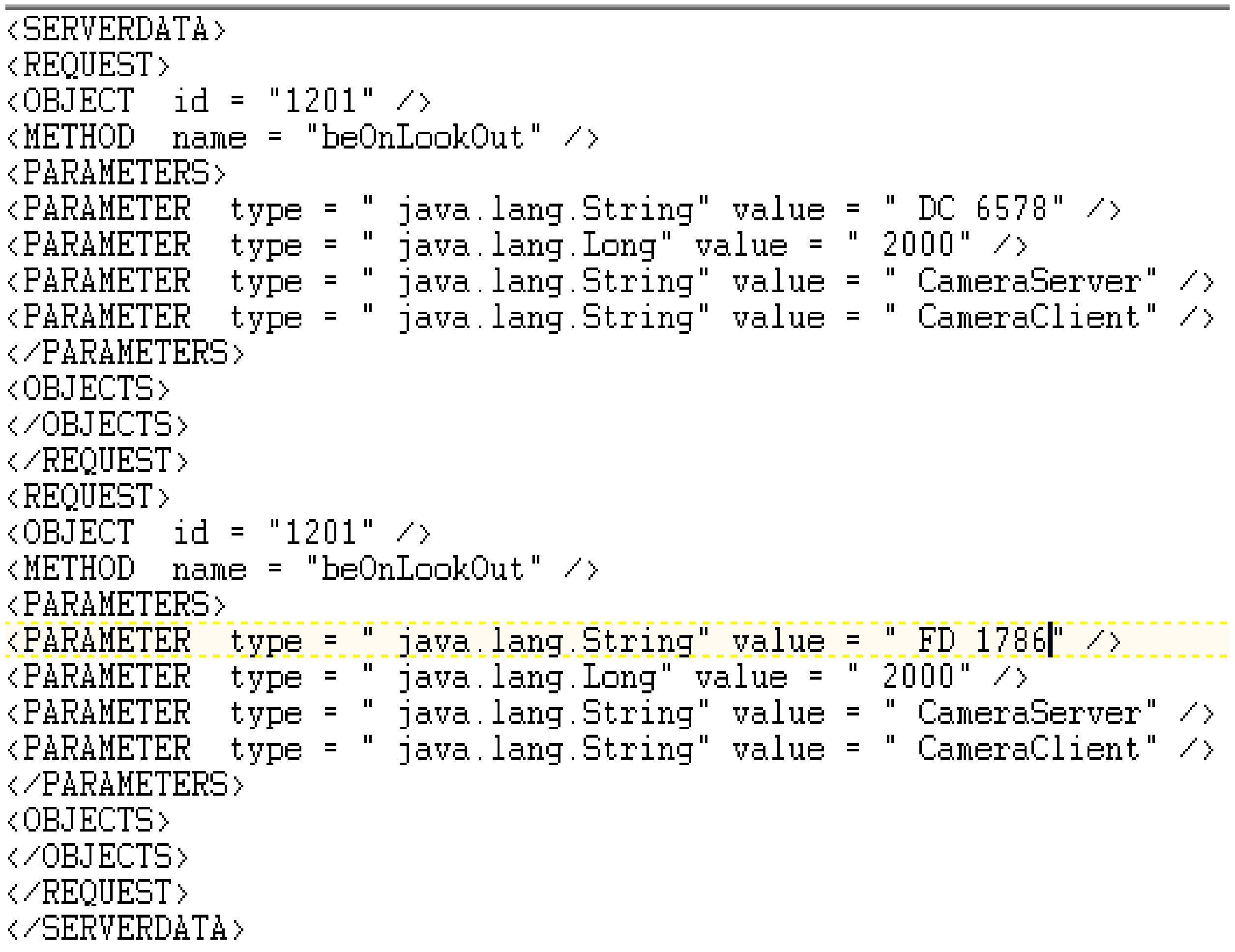
```
<SERVERDATA>
</SERVERDATA>
```

Figure 22a. The empty stored request data.



```
<SERVERDATA>
<REQUEST>
<OBJECT  id = "1201" />
<METHOD  name = "beOnLookOut" />
<PARAMETERS>
<PARAMETER  type = " java.lang.String" value = " DC 6578" />
<PARAMETER  type = " java.lang.Long" value = " 2000" />
<PARAMETER  type = " java.lang.String" value = " CameraServer" />
<PARAMETER  type = " java.lang.String" value = " CameraClient" />
</PARAMETERS>
<OBJECTS>
<OBJECT  type = " java.lang.Boolean" value = " TRUE" name = " camera">
</OBJECTS>
</REQUEST>
</SERVERDATA>
```

Figure 22b. The stored state request data contains the methods and the objects.



```
<SERVERDATA>
<REQUEST>
<OBJECT  id = "1201" />
<METHOD  name = "beOnLookOut" />
<PARAMETERS>
<PARAMETER  type = " java.lang.String" value = " DC 6578" />
<PARAMETER  type = " java.lang.Long" value = " 2000" />
<PARAMETER  type = " java.lang.String" value = " CameraServer" />
<PARAMETER  type = " java.lang.String" value = " CameraClient" />
</PARAMETERS>
<OBJECTS>
</OBJECTS>
</REQUEST>
<REQUEST>
<OBJECT  id = "1201" />
<METHOD  name = "beOnLookOut" />
<PARAMETERS>
<PARAMETER  type = " java.lang.String" value = " FD 1786" />
<PARAMETER  type = " java.lang.Long" value = " 2000" />
<PARAMETER  type = " java.lang.String" value = " CameraServer" />
<PARAMETER  type = " java.lang.String" value = " CameraClient" />
</PARAMETERS>
<OBJECTS>
</OBJECTS>
</REQUEST>
</SERVERDATA>
```

Figure 22c. The stored state request data. It contains two saved state requests of client.

As it is shown in Figure 22a, b, and c, all the data will be bounded by <SERVERDATA> tag. Within the <SERVERDATA> tag, there is <REQUEST> tag to store all client state requests. To count manually of how many requests that have been saved by object proxy is by counting numbers of <REQUEST> tag. <OBJECTS> tag is used to store the live variable objects of the application, either server or proxy (shown in Figure 22b). At the Figure 22b, camera is the live object parameter within the application, either server or proxy. It has a Boolean variable type and the true value.

<OBJECT id = "…"> tag is the attribute to know the object server's ID (object server application's ID) that was requested by the object client. <METHOD name = "…"> tag is also

the attribute for the method of service that was requested by the object client. All parameters values of a requested method of service are stored within <PARAMETERS> tag.

Within the SyDSync module, there are APIs to ease the synchronization process between the proxy and the server. Table 1 below lists all of the SyDSync APIs.

Table 1. The SyDSync API with its description.

SYDSYNC CONSTRUCTOR

SyDSync	Constructor.

SYDSYNC DATA MANIPULATION

createAFile	To create a data file.
readData	To read a data in a file.
writeData	To write a data into a file.
formatData	To format a data. It eliminates '<', '>', '\', '/'.
cleanUpServerResData	To cleanup a response data.

SYDSYNC DATA REQUESTS MANIPULATION

writeEmptyRequest	To write an empty request.
createRequest	To create requests.
updateRequest	To update requests.
readDeleteRequest	To read and delete requests.

SYDSYNC MAIN METHODS (UPLINK AND DOWNLINK)

uplink	To setup uplink. Allowing the client's server to setup variables in the proxy server.
contDownLink	To do continuous downlink synchronization between the server and proxy.
runDownLink	To run continuous downlink.
downlink	To setup downlink synchronization.

SYDSYNC GET METHODS

getCurrentDirectory	To get the current directory where the SyD file will be stored.
getFormatObjectId	To get the object ID after the data being formatted.
getFormatMethodName	To get the method name after the data being formatted.
getFormatParamT	To get the types of parameters after the data being formatted.
getFormatParamV	To get the values of parameters after the data being formatted.
getFormatObjectT	To get the types of objects after the data being formatted.
getFormatObjectV	To get the values of objects after the data being formatted.
getFormatObjectName	To get the name of the objects after the data being formatted.
getNumOfSavedRequest	To get numbers of stored state requests.

getObjectId	To get the object ID.
getMethodName	To get the method of service name.
getParamtT	To get the types of parameters.
getParamV	To get the values of parameters.
getObjectT	To get the types of the objects.
getObjectV	To get the values of the objects.
getObjectName	To get the name of the objects.
fillData	To fill the data into the vector.

Two main important APIs within SyDSync are uplink and downlink. Uplink API is normally used by the object server's client to setup data in the proxy server. We expect that the developer of SyD server, knowing all information about its proxy also, uses the uplink API to set the proxy data. On the other hand, the downlink API is normally used for synchronization purpose between the server and its proxy. Condownlink is a thread module that wraps the synchronous downlink module. The detail of SyDSync API is provided in appendix A. Figure 23a and b below picture the scenario before and after server synchronizes with its proxy.

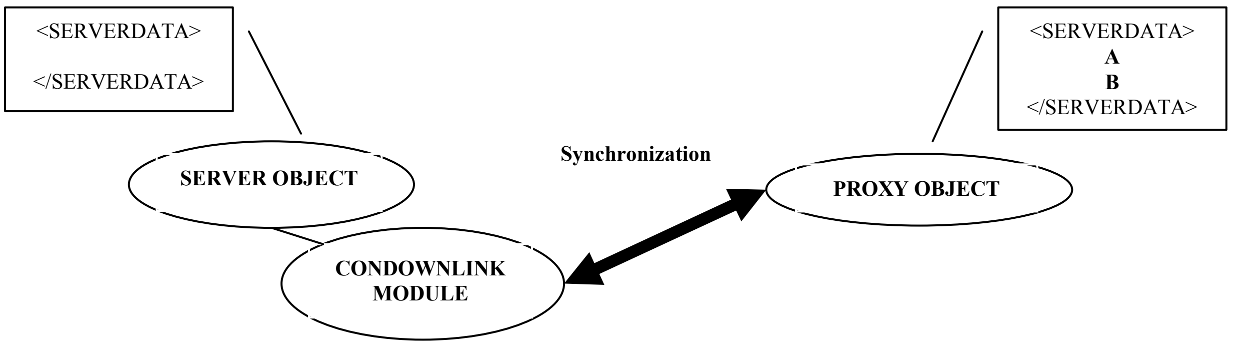


Figure 23a. Server tries to synchronize with its proxy using downlink thread module.

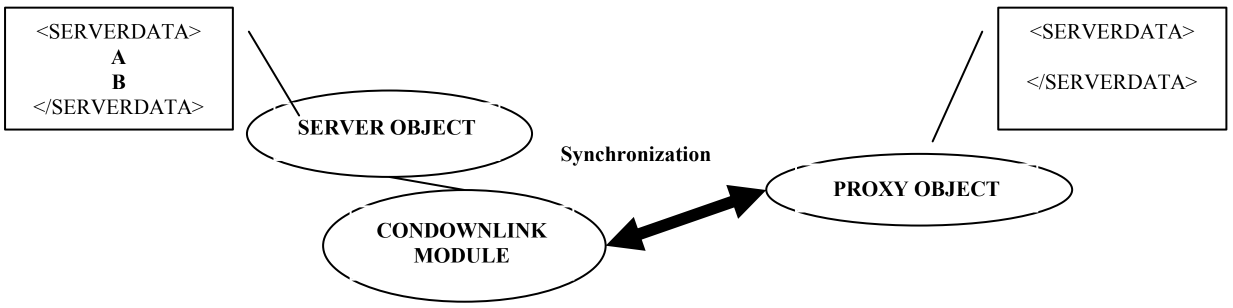


Figure 23b. Server finishes its synchronization with its proxy.

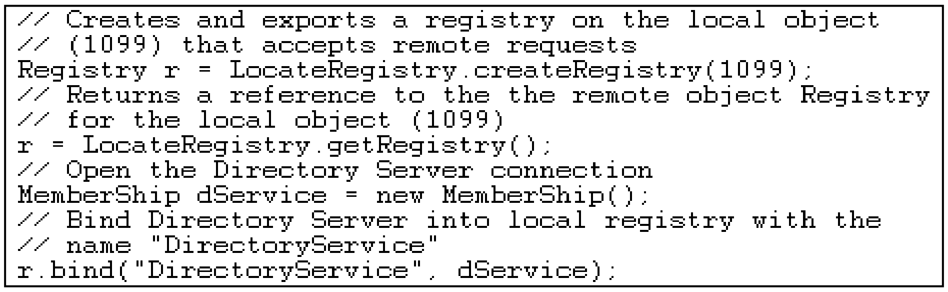
4.3 SyDProxy implementation

In section 4.1 and 4.2 above, we present the proxy module design architecture and its descriptions for SyD. In this section, we would like to present the core implementation of the proxy module design for System on Mobile Devices (SyD). The following below explains the list of files used as the core of proxy module in SyD.

SyDDirectory

- **DirectoryServer.java**

Basically, it is used to get a reference to a bootstrap local object registry for remote invocation. Then, it binds the reference with the Directory Server, which contains all information of registered object applications, in the local object device. The implementation is shown in Figure 24 below.



```
// Creates and exports a registry on the local object
// (1099) that accepts remote requests
Registry r = LocateRegistry.createRegistry(1099);
// Returns a reference to the the remote object Registry
// for the local object (1099)
r = LocateRegistry.getRegistry();
// Open the Directory Server connection
MemberShip dService = new MemberShip();
// Bind Directory Server into local registry with the
// name "DirectoryService"
r.bind("DirectoryService", dService);
```

Figure 24. The Directory Server code snippet.

- **MemberShip.java**

The implementation of the membership file here is used by a remote object to be a part of the members within the network, connected all SyD device objects. The main methods used in the module are "publish", "lookUp", "advanceLookUp", "lookUpObject", "turnOff", and "setUp". "publish" method is used for the object's registration purposes. In the implementation of proxy module, it is suggested that every object, which will have the proxy, needs to register its proxy first then the object itself. If the object will not have a proxy, then the object must registered as a proxy object.

“lookUp”, “advanceLookUp”, and “lookUpObject” are methods used for lookup a specified attribute for an object, more attributes for an object, and an object or a proxy information, such as object url, object ID, respectively. “turnOff” method is used to turn off the status of an object, either livebit is on or is off, and bring the proxy object information. “setUp” method is used to set the attribute of an object.

SyDListener

- **SyDRegistrar.java**

This is where the registration process of an object or a proxy takes place. In the module, the object or the proxy is registered globally into Directory Server and locally into the object into RMI registry afterward.

- **SyDListenerDelegate.java**

Client, which requests services from an object server or an object proxy (if an object server is in failure mode), uses SyDDispatcher module in SyDEngine, to do the invocation. Communication between the client and the object server or the object proxy, which is done through SyDDispatcher by exchanging SyD XML document, appears to be happened in this module. Basically, the SyDDispatcher is used to invoke remote object server by using the SyDListenerDelgate to send SyD likely XML SOAP document message.

Therefore, SyDListenerDelegate is normally used in the client side. A “timeout” time has also been implemented in the module. The purpose of a “timeout” time is to give an enough time for a client to get a response from a server. If the client has not received a response for a given “timeout” time, it allows SyDDispatcher to invoke the object proxy since it will assume that the object server is in busy or in fail mode.

- **SyDListener.java**

The implementation of the module resides in the server side. Together with SyDListenerDelegate module in the client side, it lets the communication of the client-server to happen through socket listener on certain port numbers (8888 is normally used as the convention for listener port of an object server). It also invokes the local services of the object server or the object proxy if the object server fails.

SyDEngine

- **SyDDispatcher.java**

The module works as an engine module used by a client to do remote invocation. It works together with SyDListenerDelegate module to determine whether to invoke an object server or an object proxy.

SyDUtil

- **SyDDoc.java**

A client, requesting a service, has to create a request document in the form of SyD likely SOAP message to be sent remotely. The module contains a method to enable the default document message creation. It has the methods to parse the message documents and also to create a message response of a requested service.

- **SyDSync.java**

Synchronization between an object server and an object proxy is needed for the purpose of allowing disconnection tolerance of transactions on the client-server invocation. The module provides an easy implementation of synchronization process. Appendix A explains the details of the module.

SyDObject

- **ObjProxy.java**

The module is a template for an object proxy. The main methods are publishProxyDoc and registerProxy, which are to publish proxy services in the form of SyD XML message document and to register those services in the proxy respectively.

```
ObjProxy myProxy = new ObjProxy();
myProxy.publishProxyDoc(…);
myProxy.registerProxy(…);
```

- **ObjAppo.java**

The module is also a template for an object server to create a SyD XML message containing of all services and to publish those services in the object server. The main methods are publishAppoDoc and registerAppo, which are to publish services in the form of SyD likely SOAP message and to register those services respectively.

```
ObjAppo myServer = new ObjAppo();
myServer.publishAppoDoc(…);
myServer.registerAppo(…);
```

- **ObjClient.java**

The module is also a template for a client. To invoke a remote object server, a client will only need to provide the parameter values needed for a service, a server name, and a method of a service to be requested.

```
ObjClient myClient = new ObjClient (ServerName, parameter values, a method name);
myClient.run();
```

5. PERFORMANCE ANALYSIS

The experiment has been conducted to get the idea of what the performances of the extended SyD are. What is the relationship between the timeout and the percentages of servers' failures, the average responses vs. numbers of clients requested services, numbers of requested clients responded by the server or the proxy server, and the best time used to be given for the timeout are analyzed in the section.

To analyze the performance of the SyD extension, we run the experiment using the following instruments (all machines run different OS and are placed in different locations):

- SunOS machine with 900 MHz Ultra SPARC III Cu processor.
- Windows XP machine with Intel Pentium IV 1.7 GHz mobile.
- Windows Vista Business with 1.67 GHz of Intel Centrino Duo machine.

The following below are the scenarios of the experiment:

- Oracle 10g database server runs on SunOS box.
- The proxy is an object that runs in SunOS machine. Proxy runs as SyD object server.
- The server application is an object that runs in Windows XP machine. The server runs as SyD object server.
- The users' client is also another object that runs in Windows Vista machine. The users' client also acts as a SyD object client.
- There is no beOnLookOut method of service that is being requested by users' client (service time is equal to 0ms) for section 5.1 to 5.3. For section 5.4, we apply service time.
- There are no network failures. All network connections are always up and running.
- SyDDirectory module in each object is used to fetch the data directly from the database.

5.1 Relationship between the timeout and failure rate

Based on our experiment, we found that there is no relationship between the timeout and the failure rate on the server. As long as the server fails, the proxy takes in charge and responses the client's request. But, with the increasing percentage of numbers of failures on the actual server, the clients' requests get response from proxy increasingly. As long as proxy is responding the request, the average response time is linearly increasing. It is shown in Figure 25 below.

Failure rate (%) - X axis	AVG response time (ms) - Y axis
0	231.9
25	6917.8
50	9083.1
75	17926.8
100	22221.2

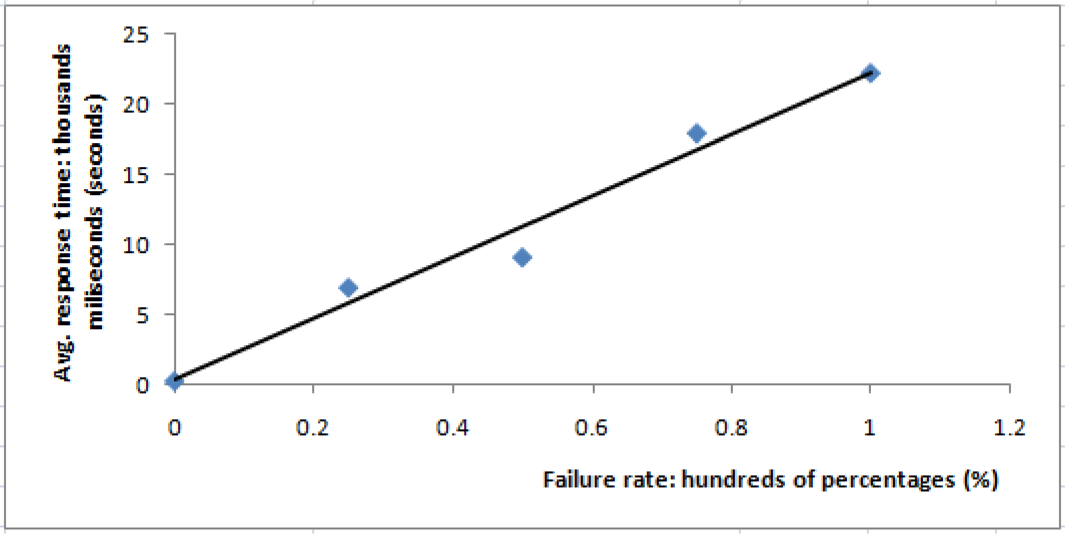


Figure 25. Increasing number of failure rate increases the average response time.

5.2 The numbers of clients requesting services vs. response time

On the 5.1, we have concluded that as long as the server fails, proxy takes in charge of server's responsibility. Since proxy takes the control, there is another trip for the request to go to the proxy. Therefore, there is a delay for the client to get its response back.

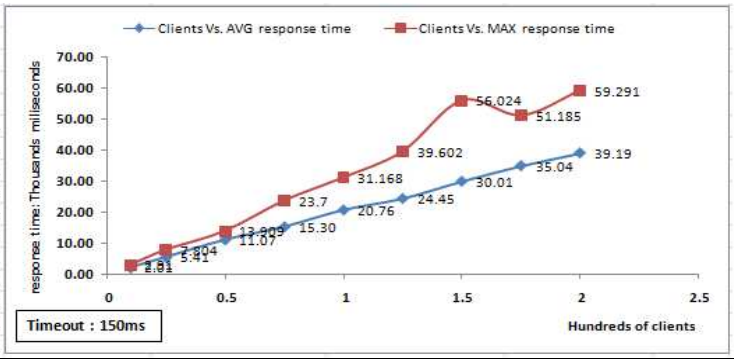
Knowing that there will be a delay of time for the client to get its response back if the server fails, we would like to know what the best timeout time is needed to be implemented so that the server could handle all the clients' requests without going into the proxy. As we notice, we have implemented the timeout time within the SyD proxy module. The timeout time is used to control the client's request to get the response from the server or the proxy. The implementation of timeout time is very critical in the SyD proxy module since we do not want the clients to wait too long to get the response from the server while the server is in failure mode. Other problems will rise as long as the client waits too long: communication between the client and the server is occupied for only that specified client (synchronous invocation), it will not give other client's chance to request services on the server, etc.

Table 2 below shows our experiment using increasing timeout time to see the relationship of numbers of clients requested services versus the average response time.

Table 2. The numbers of clients vs. the average response time on certain timeout time.

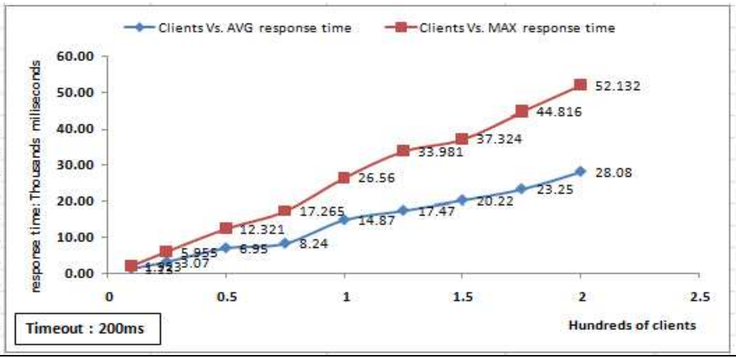
Timeout time : 150 ms

# of Clients - X axis	AVG time (ms) - Y axis	Max time (ms) - Y axis
200	39189.85	59291
175	35043.22	51185
150	30013.92	56024
125	24447.98	39602
100	20763.49	31168
75	15299.13	23700
50	11070.16	13909
25	5407.48	7804
10	2005.80	2910



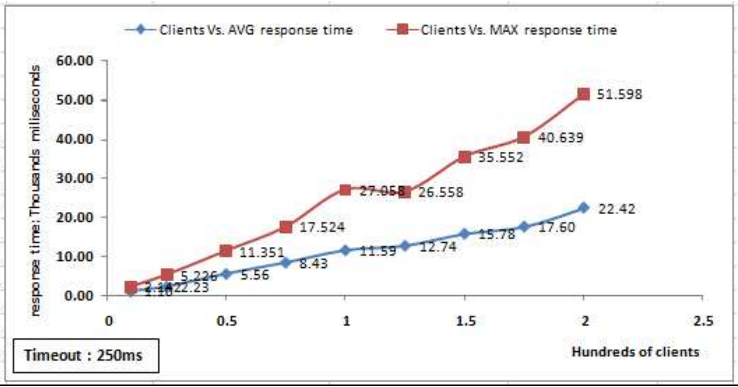
Timeout time : 200 ms

# of Clients - X axis	AVG time (ms) - Y axis	Max time (ms) - Y axis
200	28078.91	52132
175	23250.66	44816
150	20224.18	37324
125	17466.45	33981
100	14871.42	26560
75	8240.10	17265
50	6946.22	12321
25	3071.16	5955
10	1125.30	1923



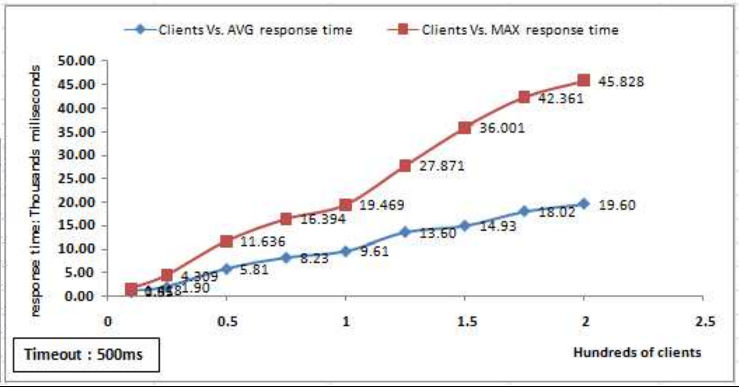
Timeout time : 250 ms

# of Clients - X axis	AVG time (ms) - Y axis	Max time (ms) - Y axis
200	22423.75	51598
175	17597.94	40639
150	15782.54	35552
125	12739.72	26558
100	11588.47	27058
75	8432.49	17524
50	5562.78	11351
25	2231.88	5226
10	1096.70	2142



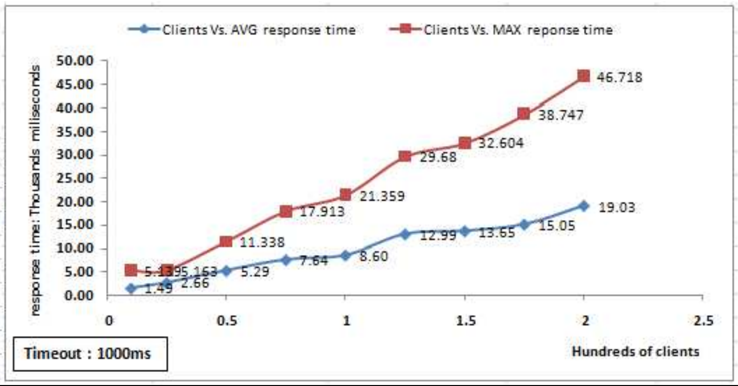
Timeout time : 500 ms

# of Clients - X axis	AVG time (ms) - Y axis	Max time (ms) - Y axis
200	19595.00	45828
175	18016.37	42361
150	14932.19	36001
125	13596.12	27871
100	9606.97	19469
75	8231.36	16394
50	5814.68	11636
25	1902.60	4309
10	848.90	1418



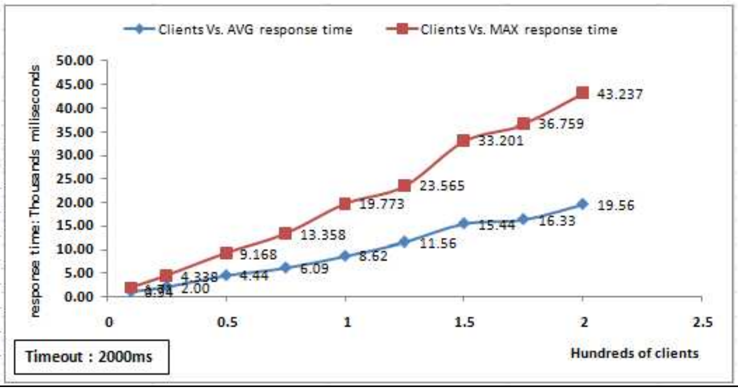
Timeout time : 1000 ms

# of Clients - X axis	AVG time (ms) - Y axis	Max time (ms) - Y axis
200	19032.26	46718
175	15050.89	38747
150	13645.14	32604
125	12989.39	29680
100	8595.25	21359
75	7636.38	17913
50	5290.26	11338
25	2659.12	5163
10	1490.20	5139



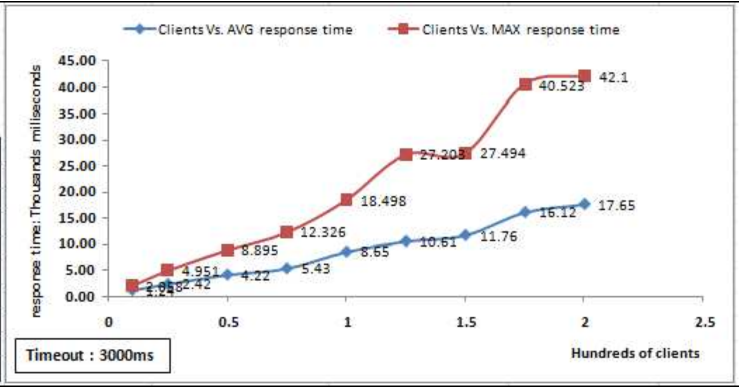
Timeout time : 2000 ms

# of Clients - X axis	AVG time (ms) - Y axis	Max time (ms) - Y axis
200	19557.06	43237
175	16327.60	36759
150	15438.02	33201
125	11556.21	23565
100	8622.06	19773
75	6087.31	13358
50	4439.68	9168
25	2003.52	4338
10	937.10	1710



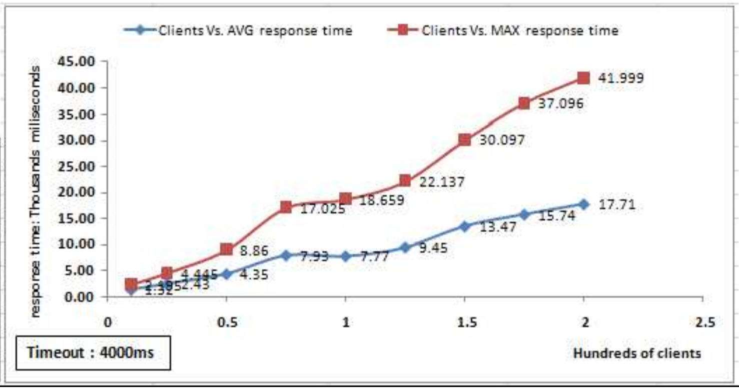
Timeout time : 3000 ms

# of Clients - X axis	AVG time (ms) - Y axis	Max time (ms) - Y axis
200	17647.65	42100
175	16122.38	40523
150	11761.49	27494
125	10614.32	27203
100	8654.91	18498
75	5432.36	12326
50	4220.06	8895
25	2418.24	4951
10	1241.50	2058



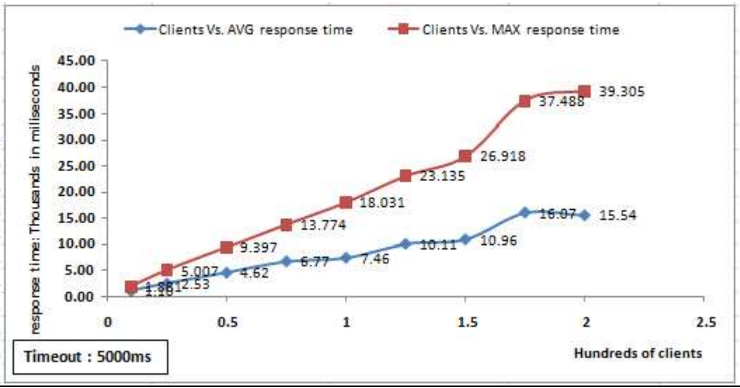
Timeout time : 4000 ms

# of Clients - X axis	AVG time (ms) - Y axis	Max time (ms) - Y axis
200	17706.67	41999
175	15738.24	37096
150	13474.65	30097
125	9452.98	22137
100	7773.39	18659
75	7925.47	17025
50	4351.14	8860
25	2432.16	4445
10	1315.20	2195



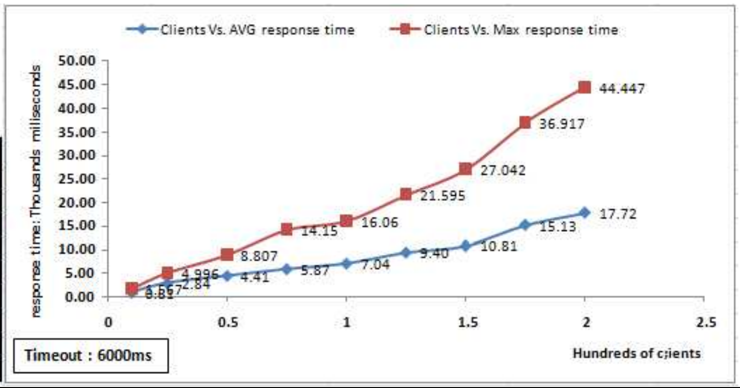
Timeout time : 5000 ms

# of Clients - X axis	AVG time (ms) - Y axis	Max time (ms) - Y axis
200	15544.94	39305
175	16074.59	37488
150	10958.05	26918
125	10108.80	23135
100	7461.73	18031
75	6769.40	13774
50	4624.70	9397
25	2534.08	5007
10	1095.80	1881



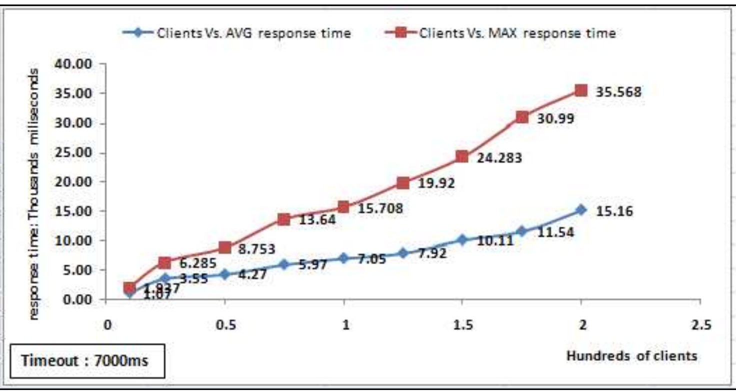
Timeout time : 6000 ms

# of Clients - X axis	AVG time (ms) - Y axis	Max time (ms) - Y axis
200	17724.53	44447
175	15133.79	36917
150	10807.13	27042
125	9403.81	21595
100	7044.31	16060
75	5868.79	14150
50	4408.60	8807
25	2836.88	4996
10	806.10	1567



Timeout time : 7000 ms

# of Clients - X axis	AVG time (ms) - Y axis	Max time (ms) - Y axis
200	15161.61	35568
175	11542.81	30990
150	10110.66	24283
125	7920.11	19920
100	7047.36	15708
75	5970.44	13640
50	4271.40	8753
25	3548.56	6285
10	1074.20	1937



On our experiment above, there is linearly incrementation toward the average response time on many numbers of clients requesting server's services. We also could see clearly that as long as the given timeout time is higher, the average response time and the maximum response time is getting smaller for the clients to get their response. For 200 numbers of clients using timeout time of 150 ms, the average response time and the maximum resopnse time for the clients to get their response is about 39.19 seconds and is about 59.29 seconds. But, for 200 numbers of clients using timeout time of 7000 ms, the average response time and the maximum response time needed for the clients to get their response is about 15.16 seconds and is about 35.57 seconds. The reason behind it is what we believe that server could handle all the clients' requests if we put the appropiate timeout time (section 5.3 clearly tells you that the server could handle more request on the higher timeout time).

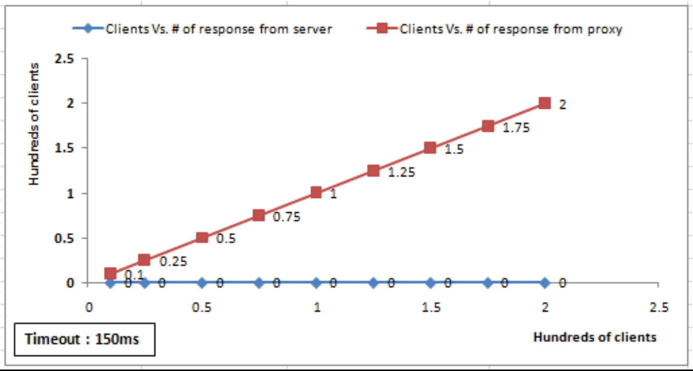
5.3 The numbers of services responded by either server or proxy

In the section 5.3 here, based on our experiment, we study that there is the influence of the given timeout time toward the numbers of clients requesting services. If we let the timeout time to be smaller, then we could see that the proxy most of the time handles the client's request. The server could handle the client's request as long as the appropriate timeout time is given. As we notice in our experiment, if we let the timeout time on 150 ms, for 200 numbers of clients requesting services, proxy takes in charge of responding 200 clients' requests. But, on the higher timeout time, such as 7000 ms, for 200 numbers of clients requesting service, proxy only responses 94 requests. 106 requests are responded by the server. Table 3 below shows the details of the numbers of services that could be handle either by the server or by the proxy for the given timeout time.

Table 3. The numbers of clients vs. the numbers of responses by the server and proxy.

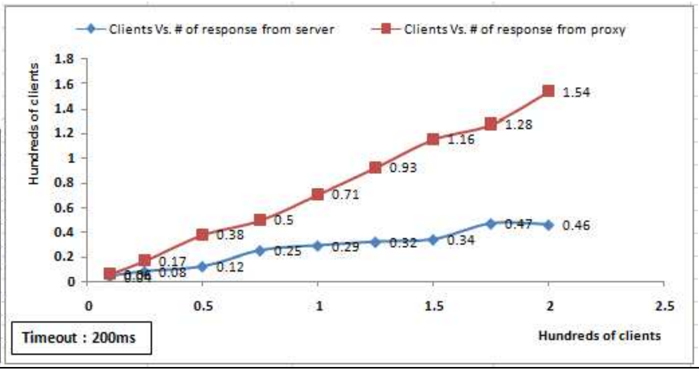
Timeout time : 150 ms

# of Clients - X axis	# of Server Response - Y axis	# of Proxy Response - Y axis
200	0	200
175	0	175
150	0	150
125	0	125
100	0	100
75	0	75
50	0	50
25	0	25
10	0	10



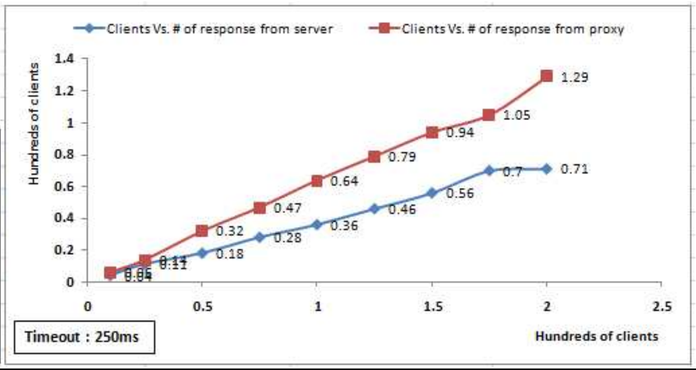
Timeout time : 200 ms

# of Clients - X axis	# of Server Response - Y axis	# of Proxy Response - Y axis
200	46	154
175	47	128
150	34	116
125	32	93
100	29	71
75	25	50
50	12	38
25	8	17
10	4	6



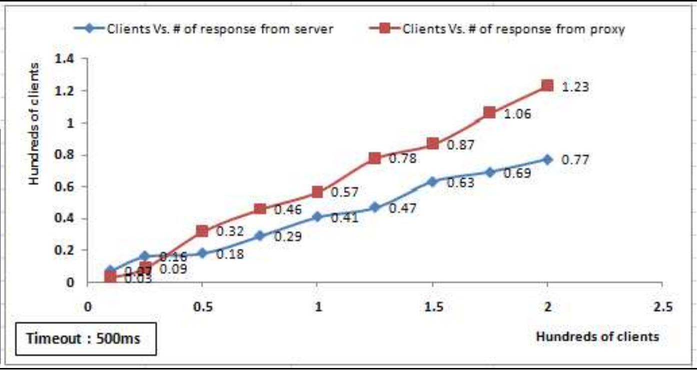
Timeout time : 250 ms

# of Clients - X axis	# of Server Response - Y axis	# of Proxy Response - Y axis
200	71	129
175	70	105
150	56	94
125	46	79
100	36	64
75	28	47
50	18	32
25	11	14
10	4	6



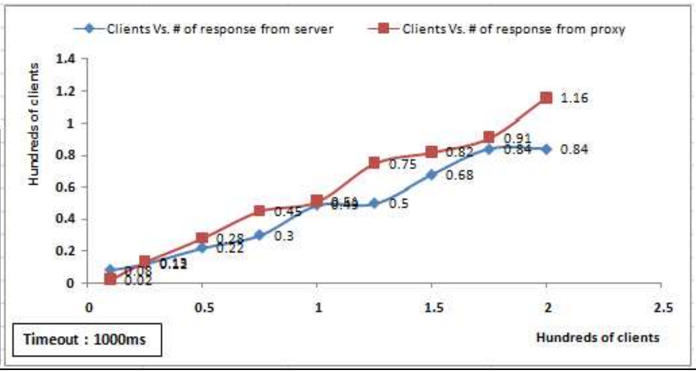
Timeout time : 500 ms

# of Clients - X axis	# of Server Response - Y axis	# of Proxy Response - Y axis
200	77	123
175	69	106
150	63	87
125	47	78
100	41	57
75	29	46
50	18	32
25	16	9
10	7	3



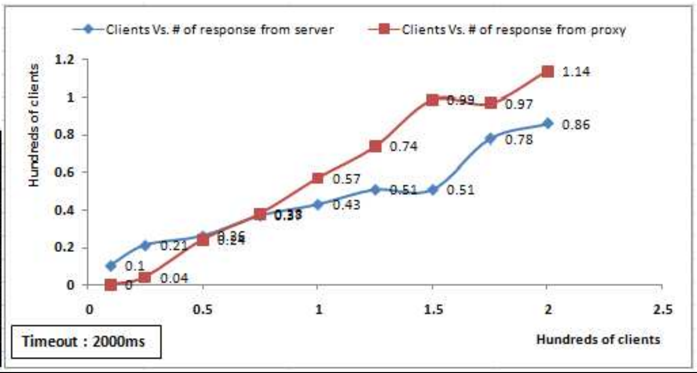
Timeout time : 1000 ms

# of Clients - X axis	# of Server Response - Y axis	# of Proxy Response - Y axis
200	84	116
175	84	91
150	68	82
125	50	75
100	49	51
75	30	45
50	22	28
25	12	13
10	8	2



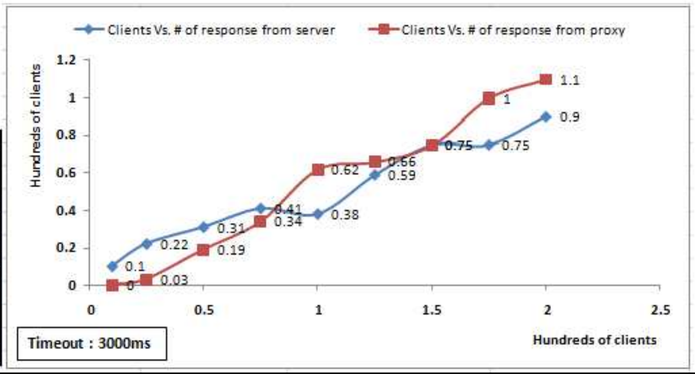
Timeout time : 2000 ms

# of Clients - X axis	# of Server Response - Y axis	# of Proxy Response - Y axis
200	86	114
175	78	97
150	51	99
125	51	74
100	43	57
75	37	38
50	26	24
25	21	4
10	10	0



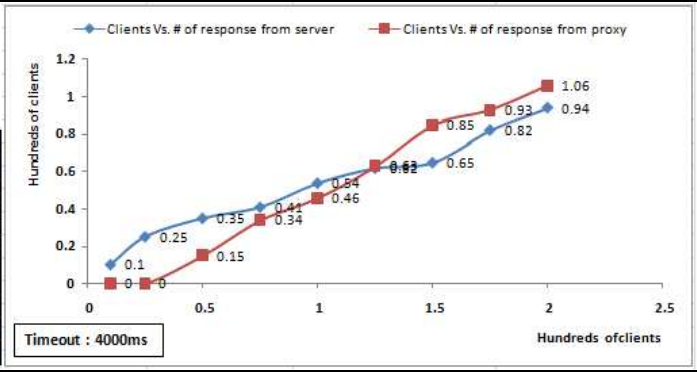
Timeout time : 3000 ms

# of Clients - X axis	# of Server Response - Y axis	# of Proxy Response - Y axis
200	90	110
175	75	100
150	75	75
125	59	66
100	38	62
75	41	34
50	31	19
25	22	3
10	10	0



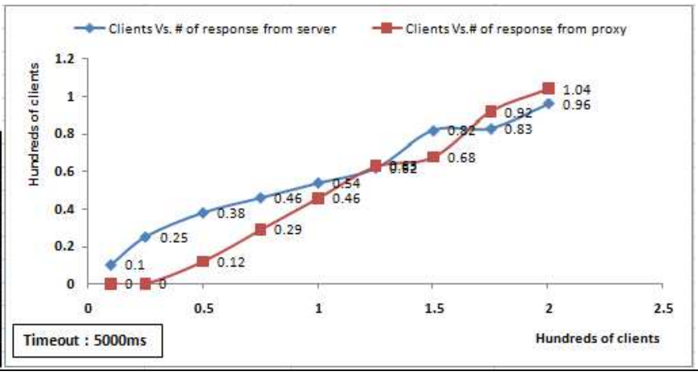
Timeout time : 4000 ms

# of Clients - X axis	# of Server Response - Y axis	# of Proxy Response - Y axis
200	94	106
175	82	93
150	65	85
125	62	63
100	54	46
75	41	34
50	35	15
25	25	0
10	10	0



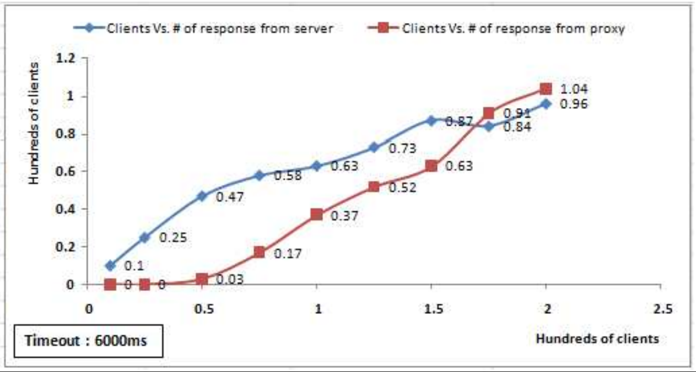
Timeout time : 5000 ms

# of Clients - X axis	# of Server Response - Y axis	# of Proxy Response - Y axis
200	96	104
175	83	92
150	82	68
125	62	63
100	54	46
75	46	29
50	38	12
25	25	0
10	10	0



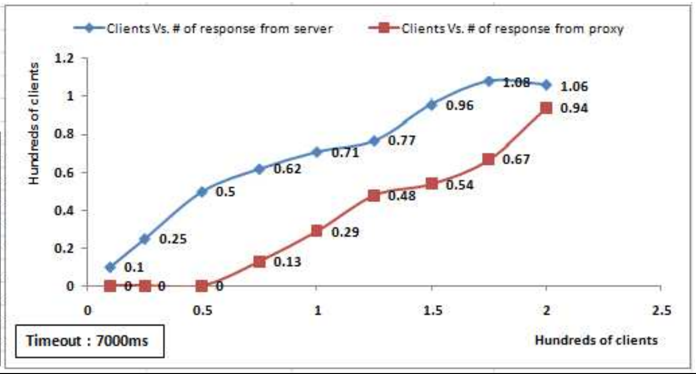
Timeout time : 6000 ms

# of Clients - X axis	# of Server Response - Y axis	# of Proxy Response - Y axis
200	96	104
175	84	91
150	87	63
125	73	52
100	63	37
75	58	17
50	47	3
25	25	0
10	10	0



Timeout time : 7000 ms

# of Clients - X axis	# of Server Response - Y axis	# of Proxy Response - Y axis
200	106	94
175	108	67
150	96	54
125	77	48
100	71	29
75	62	13
50	50	0
25	25	0
10	10	0



5.4 The best time for timeout

In section 5.1, we could notice that as long as the proxy processes a client's request, the average time for a client to get a response is linearly higher. Even though there are other parameters, a "timeout" time and increasing numbers of clients' requests, play in the experiment in section 5.2, it also tells that the linearly increasing numbers of clients' requests of services and timeout time also linearly increases the average response time for a client to get an answer back. This is because some of clients' requests are processed by proxy, shown in section 5.3.

In the section 5.4 below, we would also like to study what the influence of another parameter, service time, is with the linearly increasing of numbers of clients on the timeout time on the extended SyD platform. Table 4, 5, and 6 below are the results of experiments.

Table 4. The timeout time on 500 ms of service time.

Service time 500MS

client 1

Timeout (ms)	AVG response time (ms)	# of server response	# of proxy response	% of server response
1000	927	1	0	1

client 3

Timeout (ms)	AVG response time (ms)	# of server response	# of proxy response	% of server response
1750	1433.00	2	1	0.666666667
2000	1450.67	3	0	1
2250	1482.33	3	0	1

client 5

Timeout (ms)	AVG response time (ms)	# of server response	# of proxy response	% of server response
2250	1977	3	2	0.6
2500	1591.8	5	0	1
2750	2106.2	4	1	0.8
3000	2060.2	4	1	0.8
3250	2164	5	0	1
3500	2156.2	5	0	1

client 7

Timeout (ms)	AVG response time (ms)	# of server response	# of proxy response	% of server response
3250	2579.14	5	2	0.714285714
3500	2689.71	5	2	0.714285714
3750	2779	5	2	0.714285714
4000	2755.42	6	1	0.857142857
4250	2862	6	1	0.857142857
4500	2166	7	0	1
4750	2517.43	7	0	1

client 10

Timeout (ms)	AVG response time (ms)	# of server response	# of proxy response	% of server response
4500	3940.1	6	4	0.6
4750	3542.8	7	3	0.7
5000	3712.9	8	2	0.8
5250	3850.5	8	2	0.8
5500	3689.3	10	0	1
5750	3850.3	10	0	1

Table 5. The timeout time on 750 ms of service time.

Service time 750MS

client 1

Timeout (ms)	AVG response time (ms)	# of server response	# of proxy response	% of server response
1250	1006	1	0	1

client 3

Timeout (ms)	AVG response time (ms)	# of server response	# of proxy response	% of server response
1750	2406.00	1	2	0.333333333
2000	1769.67	2	1	0.666666667
2250	1306	3	0	1
2500	2029.01	2	1	0.666666667
2750	1935.33	3	0	1
3000	1939	3	0	1

client 5

Timeout (ms)	AVG response time (ms)	# of server response	# of proxy response	% of server response
2750	1910.00	4	1	0.8
3000	2482.8	3	2	0.6
3250	1842.2	5	0	1
3500	2557.2	4	1	0.8
3750	2373.4	5	0	1
4000	2055.4	5	0	1

client 7

Timeout (ms)	AVG response time (ms)	# of server response	# of proxy response	% of server response
3750	3151.00	5	2	0.714285714
4000	3530.85	4	3	0.571428571
4250	3282.43	5	2	0.714285714
4500	3464.14	6	1	0.857142857
4750	3571.29	6	1	0.857142857
5000	2819.14	6	1	0.857142857
5250	3151.42	7	0	1
5500	3406.43	7	0	1

client 10

Timeout (ms)	AVG response time (ms)	# of server response	# of proxy response	% of server response
5250	4558.6	7	3	0.7
5500	4741.4	6	4	0.6
5750	4755.2	7	3	0.7
6000	4732	7	3	0.7
6250	4909.9	7	3	0.7
6500	4898.9	8	2	0.8
6750	4972	8	2	0.8
7000	5138	8	2	0.8
7250	5261.1	8	2	0.8
7500	5107	10	0	1
7750	5091.6	9	1	0.9
8000	5122.3	10	0	1

Table 6. The timeout time on 1000ms of service time.

Service time 1000MS

client 1

Timeout (ms)	AVG response time (ms)	# of server response	# of proxy response	% of server response
1500	1386	1	0	1

client 3

Timeout (ms)	AVG response time (ms)	# of server response	# of proxy response	% of server response
1750	2122	1	2	0.333333333
2000	2022	1	2	0.333333333
2250	2085.67	2	1	0.666666667
2500	2158.34	2	1	0.666666667
2750	2265.34	2	1	0.666666667
3000	2578	2	1	0.666666667
3250	2430	2	1	0.666666667
3500	2414.34	3	0	1

client 5

Timeout (ms)	AVG response time (ms)	# of server response	# of proxy response	% of server response
3500	3069.2	3	2	0.6
3750	3083.8	3	2	0.6
4000	3258.8	3	2	0.6
4250	3280.8	3	2	0.6
4500	3384	3	2	0.6
4750	3413.2	4	1	0.8
5000	3711.2	4	1	0.8
5250	3636.8	5	0	1
5500	3546.8	5	0	1

client 7

Timeout (ms)	AVG response time (ms)	# of server response	# of proxy response	% of server response
5250	4366.43	5	2	0.714285714
5500	4507.86	5	2	0.714285714
5750	4443.71	6	1	0.857142857
6000	4626	5	2	0.714285714
6250	4627.14	5	2	0.714285714
6500	4842.14	6	1	0.857142857
6750	4790.71	6	1	0.857142857
7000	4644.71	6	1	0.857142857
7250	4584.34	6	1	0.857142857
7500	4833.57	7	0	1

client 10

Timeout (ms)	AVG response time (ms)	# of server response	# of proxy response	% of server response
7500	5863.3	7	3	0.7
7750	5764	7	3	0.7
8000	5285.67	7	3	0.7
8250	6147.65	7	3	0.7
8500	6246	8	2	0.8
8750	6318	8	2	0.8
9000	6574.31	9	1	0.9
9250	6347.1	8	2	0.8
9500	6413.3	8	2	0.8
10500	6505.5	9	1	0.9
10750	6459.2	10	0	1
11000	5931.4	10	0	1

The service time is used as another additional parameter in the above experiment on determining the best timeout time because there is a process that needs to be done upon clients' requests on a server, such as beOnLookOut process used to look out a vehicle license plate numbers (in camera application described in chapter 6).

On the process of making decision, based on table 4, 5, and 6 above, to determine the timeout time on additional parameter, service time, with the increasing numbers of clients' requests, we maximize the best timeout time by taking the maximum timeout time for allowing all clients' requests to be processed and responded by server. Figure 26 below displays our result of experiment on determining the best timeout time for the extension of SyD platform.

Server Computation	500MS	750MS	1000MS
# Client	Timeout (MS)	Timeout (MS)	Timeout (MS)
1	1000	1250	1500
3	2000	2750	3500
5	3250	3750	5250
7	4500	5250	7500
10	5500	8000	10750

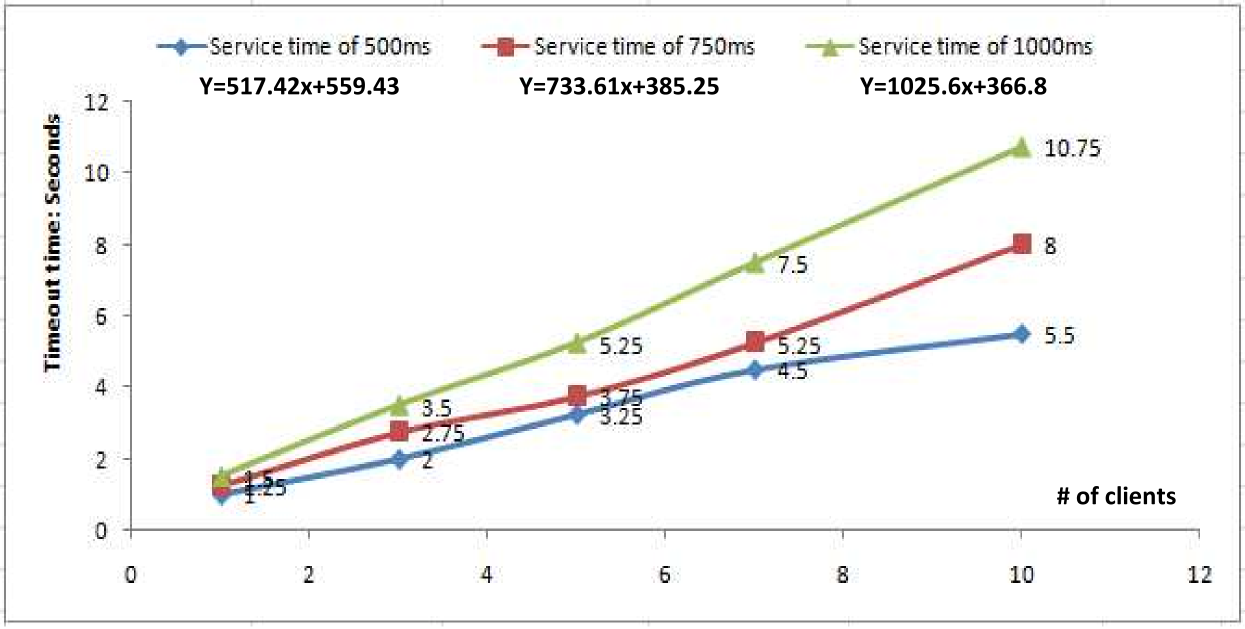


Figure 26. The increasing numbers of clients and service time on the server influence the timeout time linearly.

As a result shown in Figure 26 above, there is a linearly increasing of a time for timeout to be allowed for linearly increasing numbers of clients requesting services and the time for allowing services to be processed (service time). Based on our analysis of the experiment, we conclude the following best timeout time.

$$\boxed{\mathbf{T(x) = (ST * x) + K}}$$

T	➔ The best timeout time (ms).
ST	➔ The service time, such as beOnLookOut service computation (chapter 6) (ms).
x	➔ Numbers of clients request services.
K	➔ A constant number (ms). K = 600ms as a result of our experiment.

As an addition, we also conduct another experiment to improve the performance of the above experiment. In the above experiment, SyDDirectory module is used by each of the objects to get the data directly into the database. This is done by implementing direct JDBC-RMI remote connection into the SyDDirectory module. That situation, as we expected, creates much traffic congestion between the object hosting the database and the object requesting the data.

Instead, to boost the performance, when the object wants to request data into the database, we implement the basic RMI call to call the SyDDirectory module in the object hosting the database. Then, the data is fetched locally in that object. Figure 27 below pictures the comparison between the previous scenario and the new scenario.

The improvement of the experiment has been conducted as the following:

- Oracle 10g database server runs on Windows XP.
- The proxy is an object that runs in Windows Vista machine.
- The server application is an object that runs in Windows XP machine.
- The client runs in Windows Vista machine.

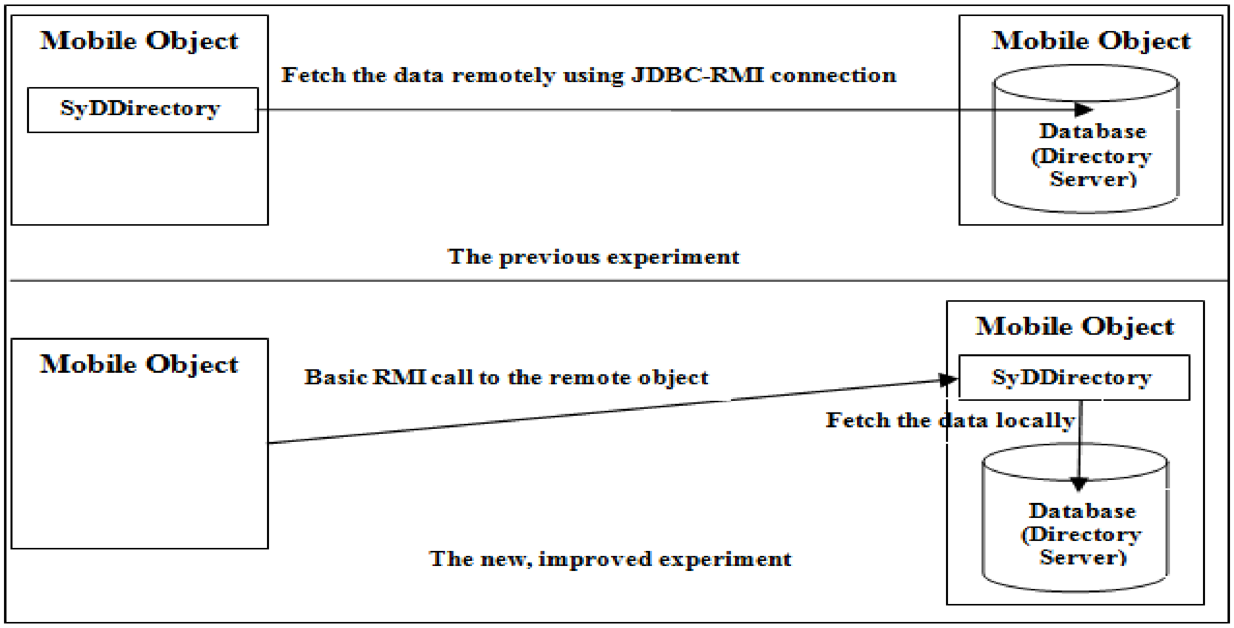
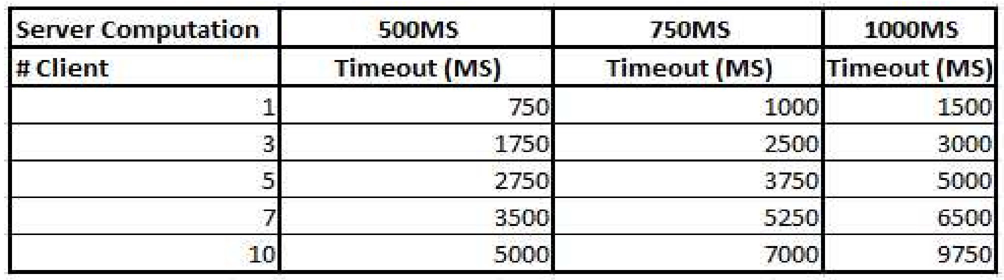

Mobile Object
SyDDirectory
Fetch the data remotely using JDBC-RMI connection
Mobile Object
Database (Directory Server)
The previous experiment
Mobile Object
Basic RMI call to the remote object
Mobile Object
SyDDirectory
Fetch the data locally
Database (Directory Server)
The new, improved experiment

Figure 27. The comparison of the new experiment scenario with the previous one.

Figure 28 below shows the summary of the experiment.



Server Computation	500MS	750MS	1000MS
# Client	Timeout (MS)	Timeout (MS)	Timeout (MS)
1	750	1000	1500
3	1750	2500	3000
5	2750	3750	5000
7	3500	5250	6500
10	5000	7000	9750

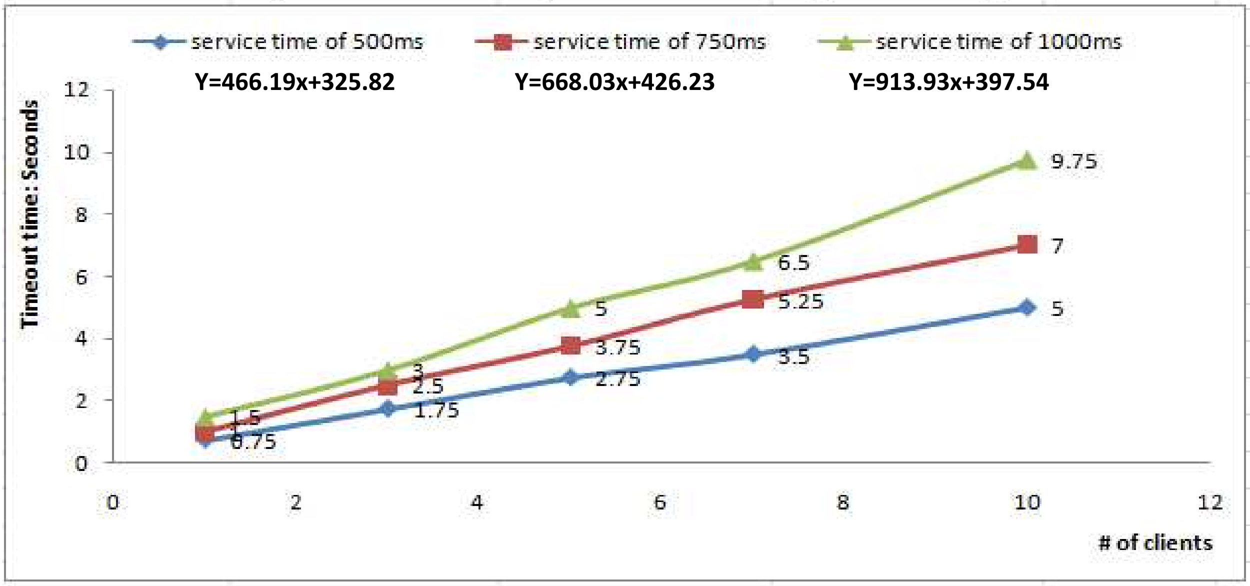


Figure 28. The improvement of the new experiment by (100 * # of clients) ms.

As a result shown in Figure 28 above, there is also a linearly increasing of a time for timeout to be allowed for linearly increasing numbers of clients requesting services and the time for allowing services to be processed (service time). It looks like that the performace is improved slightly by (100 * the numbers of clients) ms. Therefore, according to our analysis of the experiment, we conclude the following best timeout time.

$$\boxed{\mathbf{T(x) = ((ST - 100ms) * x) + K}}$$

T	➔ The best timeout time (ms).
ST	➔ The service time, such as beOnLookOut service computation (chapter 6) (ms).
x	➔ Numbers of clients request services.
K	➔ A constant number. K = 500ms as a result of our experiment.

6. CASE STUDY: A CAMERA APPLICATION TO MONITOR A STOLEN VEHICLE

The chapter presents the details of the camera application that has been developed to show the work progress of proxy module implementation on SyD and its SyDSync API implementation.

6.1 Application description

A camera application has been taken as an example of the working progress of proxy module in the extended SyD middleware platform. The example has been substantiated by taking into consideration of all the assumptions that are involved and its actual relevancies to the real world.

Nowadays, there are many cameras that have been installed on the roadways. The purpose of those cameras is to capture the vehicles' license tag numbers for all vehicles passing through the stop red light. Getting the license plate numbers, the police issue a ticket for the suspect and send a ticket to the suspect through mail based on the suspect's identity on the vehicle license tag numbers. The process might make the life easier, right? Also, in the long term, it will save the state much money by not hiring too many police for the purpose of monitoring any vehicles crossing the red light.

From that point of view, we make our decision to also use a camera application for our example application to demonstrate our SyD proxy module. The goal of using a camera in the example is the same, which is to capture a vehicle license tag numbers. But, our camera application is used for a different purpose. We use our camera application to look for stolen vehicles not to catch the suspect who breaks the traffic law.

Before we present with the actual details of how the camera application can be accomplished, we take the following assumptions on designing a camera application.

- Cameras have been installed on the roadways.
- Image processing software has been implemented in each of the cameras.
- Each of the cameras has a wireless card installed and could get its Internet connection automatically.
- There will not be any wireless network failures. But, we still consider the overloaded bandwidth on the network.
- The most important thing is that the cameras always work correctly on capturing images without any defects.
- Proxy of the camera is not a camera, but it is such kind of portable or non-portable devices (depending on the one who implements). It is assumed to be installed in the office of the police.
- Users' clients of the cameras are the police using mobile devices, such as cell phone, PDA, smart phone, etc.

With the assumption in mind, we are going to go in the details of the design and implementation of the camera for looking stolen vehicles using the extended SyD middleware platform with the proxy module.

6.2 Application design

Based on our idea of designing SyD, every user of the SyD applications is considered as an object. An object could act as a client, a server, or both a client and a server. An object will provide methods of services if it is a server. As a client, an object will normally request the registered services. An object will provide services and request remote services if it has both of the server and client's functionalities.

Design such camera application using the extended SyD middleware platform is not required tedious work. SyD platform already provided high-level application designs to develop such kind of application. API has been designed to ease the developers' job for implementing the camera application. Therefore, details like SyDDirectory, SyDEngine, SyDListener, SyDListenerDelegate, and its synchronization will be transparent enough on the developers. The section discusses the overall process design from the proxy to the camera and to the users' clients of camera.

6.2.1 Design of the camera proxy

As being mentioned above, the purpose of the proxy is to "substitute" the actual application. The design of the proxy should be standard enough and has the following functions:

- It should be an object and should be registered as an object server.
- Since it is a server, it needs to register its services to its local device and globally via Directory Service (Synchronization using downlink method indeed needs to be registered).
- Locate the place to store the state transactions (there is a default API to do this).
- It should be "smart." Not only it will "substitute" the job of the actual server application, but it will also store the missed of requested state transactions.

6.2.2 Design of the camera application

The purpose of having a camera in the application is used to look for the specified vehicle that has been stolen by capturing every vehicle license plate numbers on the roadways within period of time. The design of the camera itself should have both the server's and the client's capabilities. The following below are the things needed to be in a camera.

- It also should be an object, different from the proxy object.
- It consists of an object server and an object client.
- As a server, it needs to register its services (especially lookout services) into local device and globally via Directory Service.
- Similar to its proxy, it also needs to have a place to store the data for synchronization purposes.
- As a client, it needs to initiate the process of synchronization with its proxy within synchronization period of time.

6.2.3 Design of the camera client

The design of a client for requesting services to a camera, similar to a camera itself, also needs to act as an object that has a client's and a server's capabilities. As a client, its main purpose is to invoke the services provided by a camera server application. Lookout services provided in the design of a camera server application is the main services used to look out a stolen vehicle and is normally invoked by the client.

The client mainly needs to provide a vehicle license plate numbers and duration of time that the camera will look out the specified vehicle. With the given parameters, the client instantiates the invocation to a camera server application. The camera object's server will extract the given parameters and will start to look for the specified vehicle.

On the other hand, as a server, a client needs only to provide a service, such as a confirmation service. The camera object's client uses a confirmation service provided by the client's server to let a client knows that the suspect vehicle has been found or not.

6.2.4 Overall design of camera application

Figure 29 is the overall design of the actual camera application. Proxy works as a "substitution" of the camera application. Since a client and a camera need to act as object clients and object servers, both of them need to have SyDListener and SyDEngine to work as a server and as a client. On the other hand, a proxy needs only SyDListener component since it works as a server.

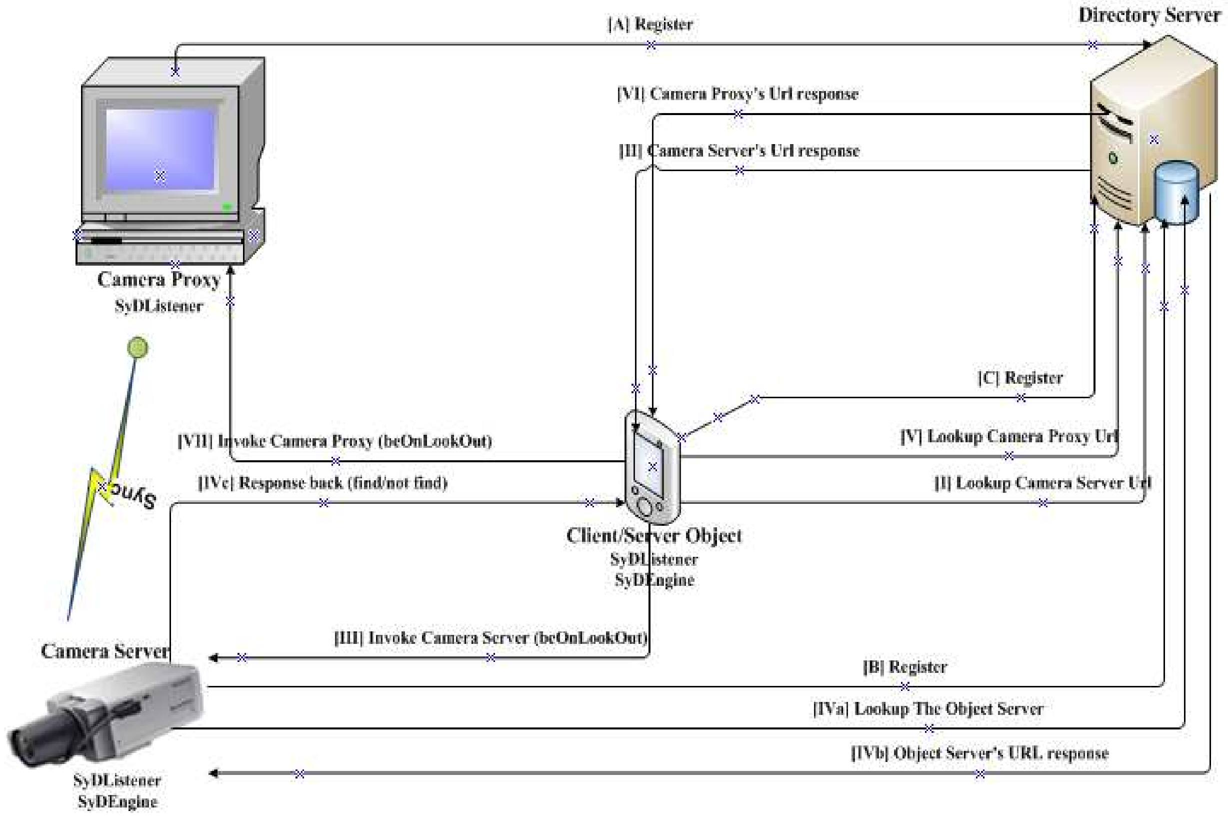

[A] Register
Directory Server
[VI] Camera Proxy's Url response
[II] Camera Server's Url response
Camera Proxy
SyDListener
[C] Register
[VII] Invoke Camera Proxy (beOnLookOut)
[V] Lookup Camera Proxy Url
[IVc] Response back (find/not find)
[I] Lookup Camera Server Url
Sync
Client/Server Object
SyDListener
SyDEngine
[III] Invoke Camera Server (beOnLookOut)
Camera Server
[B] Register
[IVa] Lookup The Object Server
[IVb] Object Server's URL response
SyDListener
SyDEngine

Figure 29. The overall design of camera application.

As shown in figure above, every request from a client to the servers (either a camera or a proxy) needs to go to the Directory Server first. The reason for this is because Directory Server holds all services and information of the registered servers. Therefore, prior to the invocation to the server application, client needs to know where the servers that need to be invoked is located. As being mention before, IP address is used in SyD for the purpose of locating the server object. A client, then, invokes a server upon receiving a server's address location. In the case if the server is busy or in off mode (timeout time goes on), the proxy takes care the responsibility of the server.

6.3 Application implementation

Implementation of the design comes later as long as the design is solid. The following below are the main files used to implement the camera application.

- **CameraModule.java**

Camera module contains only the interfaces of the remote method invocation of camera application.

- **CameraModuleImpl.java**

It contains the implementation of the remote method invocation of camera application. At the current implementation of the camera application, the followings are the API methods of services providing by the server applications once it is registered: setDownLink, runDownLink, beOnLookOut, readDeleteRequest, and sendConfirmation.

SetDownLink method is used to do one-time synchronization by the server with its proxy. But, runDownLink method is used to do continuous synchronization. ReadDeleteRequest method is used to read and delete the data. SetDownLink, runDownLink, and readDeleteRequest methods of services need to be registered in the proxy so that the client side in the camera could synchronize with the proxy and delete the data after synchronization. SendConfirmation method is used to send confirmation whenever the one, a stolen vehicle tag numbers, that being look for is found or is not found. Normally, the user's client's server needs to register the sendConfirmation method. So that whenever the camera application object either finds or does not find it, its client will invoke sendConfirmation on the remote client object's server to let it knows that the vehicle is found or not. beOnLookOut method, on the other hand, is used by the camera server to do lookout on specified vehicle. Camera server, indeed, needs to register the beOnLookOut module.

- **CameraProxyServer.java**

It contains the API modules of services for proxy server to register and to provide its services. It also contains registered services of other server applications. The services will be registered locally in the object device and globally via Directory Server. The main important methods that need to be registered here is the synchronization method (either setDownLink or runDownLink).

- **CameraServer.java**

It contains the actual server implementation and its registered services. It also contains the client implementation that invokes the synchronization method with its camera proxy.

- **CameraClientServer.java**

It contains the implementations and the services for server on the client site. The main service that needs to be provided is the method that allows the client to send confirmation message whether the vehicle has been found or not (sendConfirmation).

- **CameraMultiClients.java**

This is where the implementation for the client application to do remote method invocation toward the registered methods of services of the camera server and the camera proxy if the camera server fails.

6.4 Execution flow

In order for the camera application is up and running as described on figure above. It needs to follow the following execution flow.

- Directory Server needs to be up and running.
- All appropriate listener modules need to be setup and run.
- The proxy of the camera needs to be registered first.
- The camera's server is then registered with its services (especially beOnLookOut service) and binds it with its proxy by registering proxy object ID.
- Finally, the object client is used to do invocation by providing parameters, such as vehicle tag numbers, and period of time needed to do lookout.
- Figure 30 below tells all the sequence diagram of the camera application with its proxy and Directory Server.

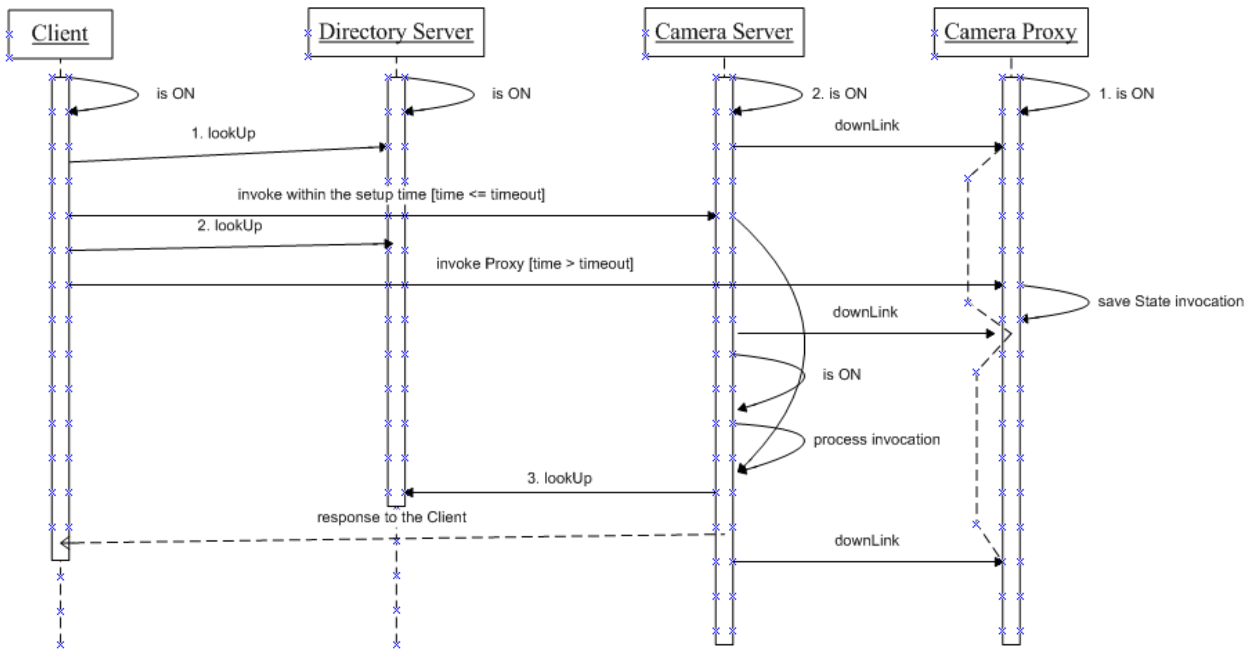


Figure 30. The sequence flow of the camera application.

6.5 Experimental results

After the design and the implementation of the camera application have been done, we run the experiment to study the influence of synchronization time between the proxy and the server against numbers of clients. Experiment is conducted on Windows Vista machine with 1.67 GHz Intel Centrino Duo processor and on Windows XP machine with Intel Pentium IV 1.7 GHz mobile. The camera and the Directory Server run on Windows XP machine. But, the proxy and the user’s client run on Windows Vista machine. There will be no failures on the network connection as being stated on the assumption above. We also use our standard estimation of our timeout time formula (described in chapter 5 above).

Running the camera application using only one client, which requests beOnLookOut service on the remote camera for about 1000ms, using timeout time of 1600ms (calculated based on chapter 5 above), as we have expected, it shows a steady linear average response time as shown in Figure 31 below.

Sync time (ms) - X axis	AVG response time (ms) - Y axis
10	1585
100	1605.4
500	1468.8
1000	1480.8
1500	1552.6

# of Client	**: 1**
beOnLookOut	**: 1000 ms**
timeout	**: 1600 ms**

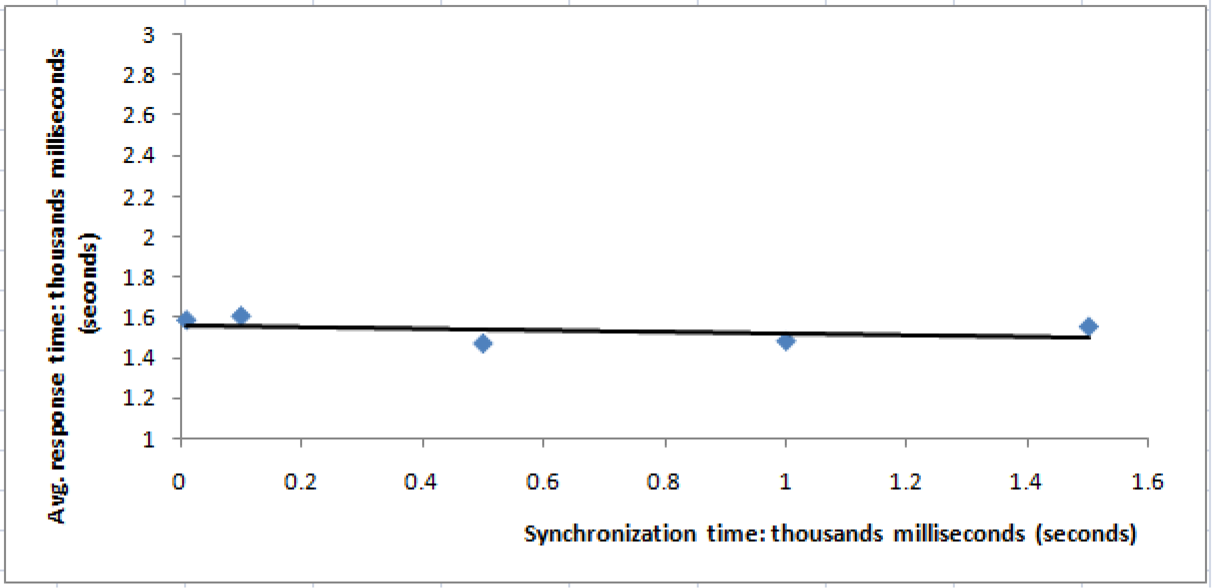


Figure 31. The sync time vs. avg. response time for one client invocation.

The reason behind a linear steady average response time is because our calculation timeout time (timeout(# of client) = (beOnLookOut Service time * # of client) + k, k = 600ms, explained in chapter 5) allows a good enough time for the user's client to do beOnLookOut invocation on the camera server application. A requested of beOnLookOut service is not going to be responded by the proxy (no state transaction being stored in the proxy). Because of that, there is no effect, which is carried by the synchronization of proxy and the camera server. We also expect a linear steady average response time higher than the above figure if we use 100 numbers of clients using our timeout calculation time of 100600ms with the 1000ms of beOnLookOut service.

We also want to know what happen if the proxy is involved on the requested beOnLookOut service. This case could be achieved if we let our timeout calculation time smaller than what it should be. Since our timeout calculation time depends on numbers of clients that try to request services from the camera server (chapter 5), we took 85% from the actual numbers of clients. For example, we would like to run with 100 clients requesting beOnLookOut service from camera server. But, for the timeout calculation, we use only 85 clients. Therefore, from what we will expect, approximately 15 clients will get responses from proxy.

Experiment is still taken in the same environment, where the proxy of the camera and the users' clients run on Windows Vista machine with Intel Centrino Duo processor and the camera and the Directory Server run on Windows XP machine. Figure 32 below shows the result of experiment for 100 clients with the beOnLookOut service time of 2000ms and the timeout time of 170600ms (only use 85 clients to calculate timeout time).

Analyzing the experiment shown in Figure 32, we saw that there is a minimum peak for synchronization time of 1000ms. We come to the conclusion that the minimum peak appears as the combination of two graphs, one graph is from the left (sync of 1ms to 1000ms) of 1000ms and the other graph is from the right of 1000ms (sync time of 1000ms to 10000ms).

The graph on the left of 1000ms shows negative gradient (decreasing slope). It is because that as the sync time is getting higher, it is lessening the average time for the users' client getting response from the camera server (synchronization time of 1000ms between the proxy and the camera happens not frequently as when the synchronization time of 1ms). On the other hand, the graph on the right of 1000ms shows positive gradient (increasing slope). The reason for the case is because there are some numbers of users' clients' requests that are kept in the proxy (camera server is too busy handling too many requests at the time). As we increasing the sync

Sync time (ms) - X axis	Avg. Response Time (ms) - Y axis
1	106547
10	106251.5
50	106248
100	106208
500	105955
1000	105702.5
5000	105907
10000	106274.5

# of Client	**: 100**
beOnLookOut	**: 2000 ms**
timeout	**: 170600 ms (for 85 clients)**

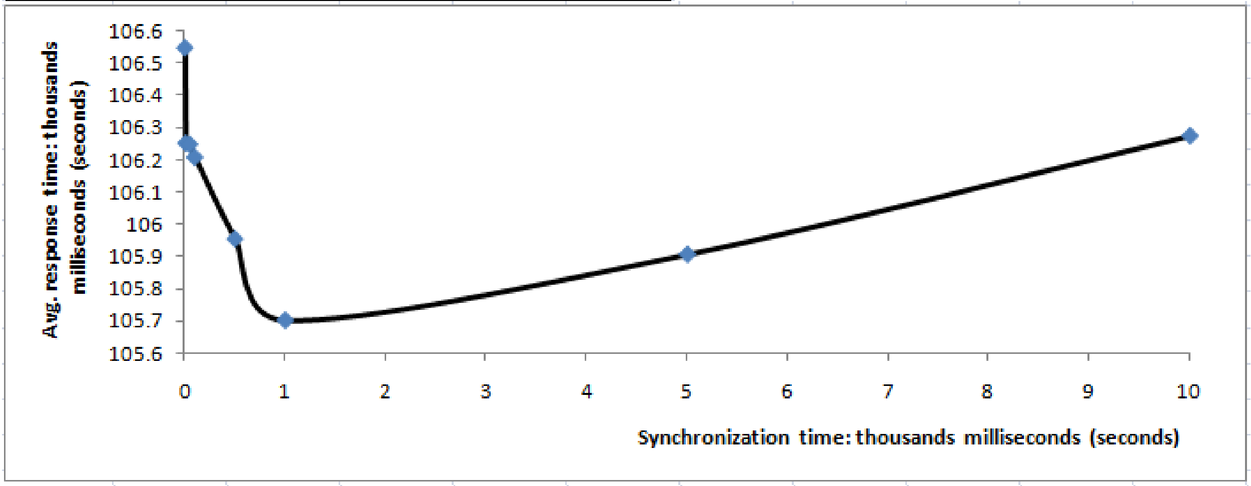


Figure 32. The sync time vs. avg. response time for one hundreds clients invocation with timeout time for eighty five clients.

time, the average response time for the users to get the response from the camera server is higher since the camera server will invoke the proxy's stored state transactions after the synchronization with its proxy happens.

In the other hand, we also analyze the performance using the improved technique by enabling remote RMI call into the remote object and getting the data locally (similar to the experiment shown in chapter 5). The similar environment is described below:

- The Directory Server is run on Windows XP.
- The camera proxy is an object that runs in Windows Vista machine.
- The camera server application is an object that runs in Windows XP machine.
- The client requesting services from the camera server runs in Windows Vista machine.

Different than the result above, the new scenario of the experiment shows only the increasing slope as shown in the figure 33 below. We believe that the trend is the result of the consistency of enabling the RMI call and the local data fetching. Because of it, synchronization time is not influenced by tremendous remote data access transactions to the Directory Server done by the clients. As long as the synchronization time is smaller, it is quick enough for the camera server to sync with its proxy and to make a quick response on the clients' requests.

Sync time (ms) - X axis	Avg. Response Time (ms) - Y axis
1	105140.5
10	105203.5
50	105297
100	105312.5
500	105333.7
1000	105369.7
5000	106210.5
10000	107055

# of Client	**: 100**
beOnLookOut	**: 2000 ms**
timeout	**: 162000 ms (for 85 clients)**

Timeout time has been improved by approximately 8500 ms (100ms * 85 clients).

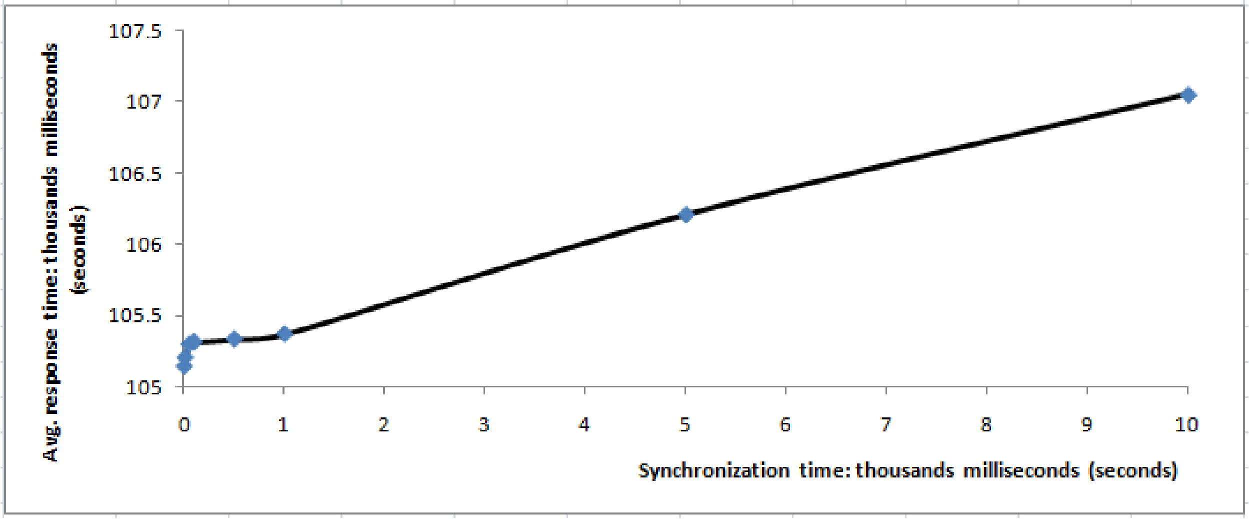


Figure 33. The improvement of sync time vs. avg. response time for one hundreds clients invocation with timeout time for eighty five clients.

6.6 Extending a camera application

The simple yet powerful scenario of developing camera application for a stolen vehicle could be extended for some other similar applications. It could be extended for military purposes or U.S. intelligent agent, such as look for Osama Bin Laden (the terrorist) as an example.

CONCLUSION

Proxy module has been successfully extended in the existing System on Mobile Devices (SyD) platform. There are two main reasons for extending the existing SyD platform. First, there is a need to provide disconnection tolerance of transactions upon failures in the existing SyD platform. Second, there is also a need to provide synchronization among the SyD object devices with their proxies in order to make the disconnection tolerance of transactions to happen.

The design, analysis, and implementation of the proxy module for SyD have been conducted and presented in the paper. The main SyD kernel (SyDDirectory, SyDListener, and SyDEngine) has been extended and synchronization (SyDSync) module has been provided to extend the SyD (SyDProxy). We have shown that linearly increasing numbers of clients request services and services time will also linearly increase a “timeout” time. Analysis to determine a best “timeout” time for allowing seamlessly invocation either to the object or to its proxy has been experimented. A simple yet powerful implementation of camera application has been implemented to show the work progress of the SyD proxy module.

For the future work, we would like to secure all of the SyD state transactions. An improvement of the SyD installation package will be provided. Also, we would like to have the SyD virtual machine to run the SyD byte code and to have the SyD Operating System.

REFERENCES

[1] M.Weiser, "Some Computer Science Issues in Ubiquitous Computing," Comm. ACM, Vol. 36, No. 7, July 1993, pp.75-84.

[2] David Chess, Colin Harrison, Aaron Kershenbaum, Mobile Agents: Are They a Good Idea?, pp. 25-47, Proceedings of the Second International Workshop on Mobile Object Systems, Jan Vitek, Christian Tschudin (Ed.), Lecture Notes in Computer Science, Springer-Verlag, Linz, Austria, Lecture Notes in Computer Science, Vol. 1222, July 1996.

[3] Wong, D., Paciorek, N., and Moore, D. 1999. Java-based mobile agents. *Commun. ACM* 42, 3 (Mar. 1999), 92-ff.

[4] Powell, M., and Miller, B. Process Migration in DEMOS/Mo. In Proceedings of the Ninth ACM Symposium on Operating Systems Principles (Bretton Woods, N.H., Oct. 11-13), ACM/SIGOPS, New York, 1983, pp. 110-119.

[5] Stamos, J., and Gifford, D. Remote evaluation. ACM Trans. Comput. Sys. 12, 4 (Oct. 1990), 537-565.

[6] Jul, E., Levy, H., Hutchinson, N., and Black, A. Fine-grained mobility in the Emerald system. ACM Trans. Comput. Sys. 6, 1. (Feb. 1998), 109-133.

[7] Gong-ping, Yang, Guang-zhou, Zeng. Mobile Agent Life State Mangement. IMACS Multiconference on Computational Engineering in Systems Applications. Vol. 1, 4-6 Oct. 2006. Page(s):448-451.

[8] Madiraju, Praveen, Prasad, Sushil K., Sunderraman, Rajshekhar, and Dogdu, Erdogan. An Agent Module for a System on Mobile Devices. In Procs. of the 3rd Intl. Workshop on Agents and Peer-to-Peer Computing (AP2PC) in conjunction with Third Intl. Joint Conf. on Autonomous Agents and Multi Agent Systems (AAMAS). LNCS, New York, July, 2004.

[9] Gian Pietro Picco. μCode: A lightweight and flexible mobile code toolkit. In Mobile Agents, Procs. Of the 2nd Intl. Workshop on Mobile Agents (MA), vol. 1477, Page(s): 160-171. Springer, LNCS, Stuggart, 1998.

[10] http://www.cs.utk.edu/~plank/ckp.html

[11] Alvisi, L., Marzullo, K. Message logging: pessimistic, optimistic, causal, and optimal. IEEE Transactions of Software Engineering. Vol. 24, Issue 2, Feb. 1998. Page(s):149-159.

[12] A. Borg, J. Baumbach, and S. Glazer. A Message Systems Supporting Fault Tolerance. Proc. Symp. Operating Systems Principles. Page(s):90-99, ACM SIG OPS, Oct. 1983.

[13] M.L. Powell and D.L. Presotto, "Publishing: A Reliable Broadcase Communication Mechanism," Proc. Ninth Symp. Operating SystemPrinciples, pp. 100-109. ACM SIGOPS, Oct. 1983.

[14] R.B. Strom and S. Yemeni, "Optimistic Recovery in Distributed Systems," ACM Trans. Computer Systems, vol. 3, no. 3, pp. 204-226, Apr. 1985.

[15] D.B. Johnson and W. Zwaenepoel, "Sender-Based Message Logging," Digest of Papers: 17th Ann. Int'l Symp. Fault-Tolerant Computing, pp. 14-19, IEEE Computer Society, June 1987.

[16] R.E. Strom, D.F. Bacon, and S.A. Yemini, "Volatile Logging in n-Fault-Tolerant Distributed Systems," Proc. 18th Ann. Int'l Symp. Fault-Tolerant Computing, pp. 44-49, 1988.

[17] A.P. Sistla and J.L. Welch, "Efficient Distributed Recovery Using Message Logging," Proc. 18th Symp. Principles of Distributed Computing, pp. 223-238, ACM SIGACT/SIGOPS, Aug. 1989.

[18] D.B. Johnson and W. Zwaenepoel, "Recovery in Distributed Systems Using Optimistic Message Logging and Checkpointing," J. Algorithms, vol. 11, pp. 462-491, 1990.

[19] S. Venkatesan and T.Y. Juang, "Efficient Algorithms for Optimistic Crash Recovery," Distributed Computing," vol. 8, no. 2, pp. 105-114, June 1994.

[20] E.N. Elnozahy and W. Zwaenepoel, "Manetho: Transparent Rollback-Recovery with Low Overhead, Limited Rollback and Fast Output Commit," IEEE Trans. Computers, vol. 41, no. 5, pp. 526-531, May 1992.

[21] Strom, R. AND Yemini, S. 1985. Optimistic recovery in distributed systems. *ACM Transactions on Computing Systems 3*, 3, 204-226.

[22] Alvisi, L. and Marzullo, K. 1998. Message logging: pessimistic, optimistic, causal and optimal. IEEE Transactions on Software Engineering 24, 2, 149-159.

[23] Chandy, M. and Lamport, L. 1985. Distributed snapshots: Determining global states of distributed systems. ACM Transactions on Computing Systems 31, 1, 63-75.

[24] Randell, B. 1975. System structure for software fault tolerance. In *Proceedings of the international Conference on Reliable Software* (Los Angeles, California, April 21 - 23, 1975). ACM, New York, NY, 437-449.

[25] Borg, A., Blau, W., Graetsch, W., Hermann, F. and Oberle, W. 1989. Fault tolerance under UNIX. ACM Transactions on Computing Systems 7, 1, 1-24.

[26] Inacio, chris, "Software Fault Tolerance." http://www.ece.cmu.edu/~koopman/des_s99/sw_fault_tolerance/

[27] Randell, B. 1975. System structure for software fault tolerance. In *Proceedings of the international Conference on Reliable Software* (Los Angeles, California, April 21 - 23, 1975). ACM, New York, NY, 437-449.

[28] Ioannidis, J., Duchamp, D., and Maguire, G. IP-based protocols for mobile internetworking. In Proceedings of ACM SIGCOMM Symposium on communication Architecture and Protocols (1991), pp. 235-245.

[29] Schlichting, R. D., and Schneider, F. B. Fail-stop processors: an approach to designing fault-tolerant distributed computing systems. ACM Trans. Comput. Syst. 1, 3(1985), 222-238.

[30] Rangarajan, S., Ratnam, K., Dahbura, A.T. A fault-tolerant protocol for location directory maintenance in mobile networks. Twenty-fifth International symposium of Fault-Tolerant Computing, 1995. 27-30 June 1995. Pages(s):164-173.

[31] Gifford, D. K. Weighted voting for replicated data. In Proceedings of the Seventh ACM Symposium on Operating Systems Principles (Pacific Grove, Calif., Dec. 1979)

[32] Herlihy, M. 1987. Concurrency versus availability: atomicity mechanisms for replicated data. *ACM Trans. Comput. Syst.* 5, 3 (Aug. 1987), 249-274.

[33] Paris, J-F. Voting with witnesses: A consistency scheme for replicated files. In Proceedings of the sixth IEEE International Conference on Distributed Computing Systems (Cambridge 1986).

[34] El Abbadi, A. and Toueg, S. Maintaining availability in partitioned replicated databases. ACM Trans. Database Syst. 14, 2 (June 1989).

[35] Long, D. E. Analysis of replication control protocols. In Proceedings of the IEEE Workshop on Management of Replicated Data, Houston, Tex, Nov. 1990.

[36] Satyanarayanan, M. 2002. The evolution of Coda. *ACM Trans. Comput. Syst.* 20, 2 (May. 2002), 85-124.

[37] Hara, T. 2003. Replica allocation methods in ad hoc networks with data update. *Mob. Netw. Appl.* 8, 4 (Aug. 2003), 343-354.

[38] S. Alagar, R. Rajagopalan, S. Venkatesan, “Tolerating Mobile Support Station Failures,” Computer Science Technical Report, Univ. of Texas at Dallas, November, 1993.

[39] Sushil K. Prasad, Vijay Madisetti, Shamkant B. Navathe, Raj Sunderraman, Erdogan Dogdu, Anu Bourgeois, Michael Weeks, Bing Liu, Janaka Balasooriya, Arthi Hariharan, Wanxia Xie, Praveen Madiraju, Srilaxmi Malladi, Raghupathy Sivakumar, Alex Zelikovsky, Yanqing Zhang, Yi Pan, and Saied Belkasim. SyD: A Middleware Testbed for Collaborative Applications over Small Heterogeneous Devices and Data Stores,, In *Proceedings of ACM/IFIP/USENIX, 5th International Middleware Conference*, Toronto , Ontario , Canada , October 18th - 22nd, 2004.

APPENDIX A SyDSync API

The following appendix describes the methods in the SyDSync module. The SyDSync API is provided for easing the developers' jobs to synchronize between the objects (normally between the mobile object server and its proxy). The uplink and downlink methods are the main methods in the SyDSync module. The uplink method is normally used for the purpose of setting up the data in the object. In this case, the developer of the SyD server could use the uplink method to setup the server's proxy. The downlink method is used for the synchronization purpose. Upon the synchronization with its proxy, the server invokes the proxy to get the stored states of transactions. Then, the transactions are parsed and are processed by the server so that the server could give the responses on the client's requests.

CONSTRUCTOR DETAIL
SyDSync
public SyDSync()
Defines a SyD synchronization object with the initial synchronization time of 1000ms.

METHOD DETAIL

Synchronization Data Manipulation
createAFile
public String createAFile(String _fileName)
Create a data file with the specified file name if a file has not existed yet.
Parameters:
_fileName – a name of a data file.
Returns:
The physical location of a created file (a path of a file).

readData
public StringBuffer readData(String _loc)
Read a data based on the specified location.
Parameters:
_loc – a location of a data file.
Returns:
A data file.

writeData

public void writeData(StringBuffer _data, String _loc)

Write a data on the specified location. Data normally written in XML format.

Parameters:

_data – a data to be written on the specified location.

_loc – a location of a data file.

formatData

public StringBuffer formatData(StringBuffer _data)

Format a data.

This function is normally used to pass the XML data remotely into remote objects. It eliminates '<', '>', '\', and '/' character.

Parameters:

_data – a XML data needed to be formatted.

Returns:

A new format data.

cleanUpServerResData

public StringBuffer cleanUpServerResData(StringBuffer _data)

Cleanup a data responded back by remote object.

After processing an object client's request, an object server sends a response back. A response from a server is wrapped with XML SyD SOAP envelope format. The function un-wraps the envelope format and gets a data response.

Parameters:

_data – a wrapped SyD likely SOAP data response message.

Returns:

An un-wrapped data response message.

Synchronization Data Requests Manipulation

writeEmptyRequest

public void writeEmptyRequest(String _loc)

Write an empty request bundled in SyD likely XML SOAP message.

Parameters:

_loc – a location to write a request.

createRequest

```
public StringBuffer createRequest(String _objectID,
                                  String _methodName,
                                  Vector _parameterTypeList,
                                  Vector _parameterValueList,
                                  Vector _objectTypeList,
                                  Vector _objectValueList,
                                  Vector _objectNameList)
```

Create a request bundled in SyD likely XML SOAP message.
Parameters:
_objectID – an object id of a remote object, which should process a request (an object server).
_methodName – a remote method invocation. A service requested by an object client.
_parameterTypeList – a list contains of all parameters types of a requested method of service.
_parameterValueList – a list contains of all parameters values of a requested method of service.
_objectTypeList – a list contains of all live objects types.
_objectValueList – a list contains of all live objects values.
_objectNameList – a list contains of all live objects names.
Returns:
A XML request data bundled in XML SyD likely SOAP message.

updateRequest
public void updateRequest(StringBuffer _data, String _loc)
Update a request bundled in SyD likely XML SOAP message.
Parameters:
_data – an additional request data.
_loc – a location to update a request.

readDeleteRequest
public StringBuffer readDeleteRequest(String _loc)
Read stored requested data and delete it in the specified location path.
Parameters:
_loc – a location of request data.
Returns:
Stored, requested data.

Synchronization Main Methods – The main methods in SyDSync
uplink
public void uplink(StringBuffer _data, String _loc)
Setup live objects parameters in a remote object.
The function is normally used to setup live parameter objects in proxy through server by remote invocation.
Parameters:
_data – a XML data contains live object parameters.
_loc – a location to keep object parameters setup.

downlink
```
public void downLink(String _recipientServerPort,
                     String _recipientServerAppName,
                     String _recipientObjectID,
```

String _storeRecipientDataToloc,
String _reqServerAppName,
String _reqServerParamValue,
String _reqServerMethodName)

Get stored requested data in remote objects, keep it in local object, and process stored requests. Responses are sent after the process.
The function is normally used for synchronization purposes among remote objects (object servers with their proxies). Responses are sent to objects that request methods of services.
Parameters:
_recipientServerPort – a listener port of a remote object that requests services.
_recipientServerAppName – a remote object name that requests services.
_recipientObjectID – a remote object id that request services.
_storeRecipientDataToLoc – a remote object location to store requested services.
_reqServerAppName – a remote object name that provides requested services.
_reqServerParamValue – parameter values of requested services.
_reqServerMethodName – a service name that was requested.

contDownLink
private class contDownLink implements Runnable
A continuous synchronization thread object.

runDownLink
public void runDownLink(Long _elapsedTime,
String _recipientServerPort,
String _recipientServerAppName,
String _recipientObjectID,
String _storeRecipientDataToloc,
String _reqServerAppName,
String _reqServerParamValue,
String _reqServerMethodName)

Run continuous synchronization thread object for a given period of time.
Parameters:
_elapsedTime – a time specified to run synchronization process.
_recipientServerPort – a listener port of a remote object that requests services.
_recipientServerAppName – a remote object name that requests services.
_recipientObjectID – a remote object id that request services.
_storeRecipientDataToLoc – a remote object location to store requested services.
_reqServerAppName – a remote object name that provides requested services.
_reqServerParamValue – parameter values of requested services.
_reqServerMethodName – a service name that was requested.

<u>Synchronization Get Data Methods</u>

getCurrentDirectory
private String getCurrentDirectory()

Get current directory.
Returns:
A directory location path.

getFormatObjectId
public String getFormatObjectId(StringBuffer _data)
Get Object ID from the formatted XML data.
Parameters:
_data – a formatted XML data.
Returns:
An object ID of an object that provides a service.

getFormatMethodName
public String getFormatMethodName(StringBuffer _data)
Get method of service name from the formatted XML data.
Parameters:
_data – a formatted XML data.
Returns:
An object ID of an object that provides a service.

getFormatParamT
public Vector getFormatParamT(StringBuffer _data)
Get parameters types from the formatted XML data.
Parameters:
_data – a formatted XML data.
Returns:
A container contains parameter types.

getFormatParamV
public Vector getFormatParamV(StringBuffer _data)
Get parameters values from the formatted XML data.
Parameters:
_data – a formatted XML data.
Returns:
A container contains parameter values.

getFormatObjectT
public Vector getFormatObjectT(StringBuffer _data)
Get live object parameters types from the formatted XML data.
Parameters:
_data – a formatted XML data.
Returns:
A container contains live object parameter types.

getFormatObjectV

public Vector getFormatObjectV(StringBuffer _data)
Get live object parameters values from the formatted XML data.
Parameters:
_data – a formatted XML data.
Returns:
A container contains live object parameter values.

getFormatObjectName
public Vector getFormatObjectName(StringBuffer _data)
Get live object parameters names from the formatted XML data.
Parameters:
_data – a formatted XML data.
Returns:
A container contains live object parameter names.

getNumOfSavedRequest
public int getNumOfSavedRequest(String _loc)
Get numbers of stored requests.
Parameters:
_loc – a location of stored requests.
Returns:
A numbers of stored requests.

getObjectId
public Vector getObjectId(StringBuffer _data, int _numOfSavedRequest)
Get object IDs from the stored XML requests data.
Parameters:
_data – a stored XML requests data.
_numOfSavedRequest – numbers of stored requests.
Returns:
A container contains object IDs.

getMethodName
public Vector getMethodName(StringBuffer _data, int _numOfSavedRequest)
Get method names from the stored XML requests data.
Parameters:
_data – a stored XML requests data.
_numOfSavedRequest – numbers of stored requests.
Returns:
A container contains method names.

getParamT
public Vector getParamT(StringBuffer _data, int _numOfSavedRequest)
Get parameters types from the stored XML requests data.
Parameters:

_data – a stored XML requests data.
_numOfSavedRequest – numbers of stored requests.
Returns:
A container contains parameters types.

getParamV
public Vector getParamV(StringBuffer _data, int _numOfSavedRequest)
Get parameters values from the stored XML requests data.
Parameters:
_data – a stored XML requests data.
_numOfSavedRequest – numbers of stored requests.
Returns:
A container contains parameters values.

getObjectT
public Vector getObjectT(StringBuffer _data, int _numOfSavedRequest)
Get live object parameters types from the stored XML requests data.
Parameters:
_data – a stored XML requests data.
_numOfSavedRequest – numbers of stored requests.
Returns:
A container contains live object parameters types.

getObjectV
public Vector getObjectV(StringBuffer _data, int _numOfSavedRequest)
Get live object parameters values from the stored XML requests data.
Parameters:
_data – a stored XML requests data.
_numOfSavedRequest – numbers of stored requests.
Returns:
A container contains live object parameters values.

getObjectName
public Vector getObjectName(StringBuffer _data, int _numOfSavedRequest)
Get live object parameters names from the stored XML requests data.
Parameters:
_data – a stored XML requests data.
_numOfSavedRequest – numbers of stored requests.
Returns:
A container contains live object parameters names.

fillData
private void fillData(String _dataType, String _dataValue, Vector _vData)
Fill the specified data based on its data type into a container.
Parameters:

_dataType – a data type.
_dataValue – a data value.
_vData – a container holds all data.

APPENDIX B APIs for SyD

This appendix lists the rest of all the APIs in the SyD middleware platform.

SyDObject module

The module is used as a wrapper for each of the objects applications to implement SyD, such as proxy, server, or client. ObjProxy is used for the object to act as a proxy, ObjAppo is used for the object to act as a server, and ObjClient is used for the object to act as a client.

ObjClient

ObjClient

public ObjClient()

Client object constructor.

ObjClient

public ObjClient(Vector _ServerAppName, Vector _paramValue, String _methodName)

Another client object constructor.

Parameters:

_ ServerAppName – the name of the server app that is going to be invoked remotely.
_paramValue – the parameter values of the requested method or service.
_methodName – the requested method or service name.

run

public void run()

To run the object client to invoke the remote object.

run

public String run(boolean isRevoceryServer)

To also run the object client to invoke the remote object.

Parameters:

isRevoceryServer – the boolean value to state that the object will always invoke the live object.

Returns:

The result of the remote invocation.

getDirectory

public String getDirectory()

To get the location of the directory server.

Returns:

The location of the directory server.

getListenerPort

public String getListenerPort(String serverAppName)
    To look up for the listener port based on the given server name.
    Parameters:
        serverAppName – the name of the server.
    Returns:
        The listener port for the specified server name.

getDirecUrl

public String getDirecUrl(String serverAppName)
    To look up for the directory url based on the given server name.
    Parameters:
        serverAppName – the name of the server.
    Returns:
        The directory url for the specified server name.

getMethodName

public String getMethodName(String serverAppName)
    To look up for the method or service name based on the given server name.
    Parameters:
        serverAppName – the name of the server.
    Returns:
        The method or service name for the specified server name.

getParamType

public String getParamType(String methodName)
    To look up for the parameter types based on the method name.
    Parameters:
        methodName – the name of the service or method.
    Returns:
        The parameter types for the specified method name.

lookupAndInvoke

public String lookupAndInvoke(int listenerPort, String dirUrl, Vector serverAppName, Vector paramType, Vector paramValue, String methodName)
    To look up and invoke the remote object.
    Parameters:
        listenerPort – the remote object listener port.
        dirUrl – the directory url of the remote object.
        serverAppName – the name of the remote server.
        paramType – the parameter types of the remote method or service.
        paramValue – the parameter values of the remote method or service.
        methodName – the name of the method or service.
    Returns:
        The result of the remote invocation.

<u>ObjAppo / ObjProxy</u>

ObjAppo / ObjProxy

public ObjAppo()

Server object constructor.

getAppoPort / getProxyPort

public String getAppoPort()

To get the RMI port of the server object.

Returns:

The RMI port of the server object.

getDirectoryServerPort

public String getDirectoryServerPort()

To get the port of the directory server.

Returns:

The port of the directory server.

getListenerPort

public String getListenerPort()

To get the listener port of the server object.

Returns:

The listener port of the object server.

getAppoUrl / getProxyUrl

public String getAppoUrl()

To get the location of the object server.

Returns:

The location of the object server.

getDirectoryUrl

public String getDirectoryUrl()

To get the location of the directory server.

Returns:

The location of the directory server.

getProxyID

public String getProxyID(String proxyObjectName)

To get the proxy ID of the given object server ID.

Parameters:

proxyObjectName – the ID of the object server.

Returns:

The location of the object server.

setAppo

public void setAppo(String attribute, String userName)
To set the livebit of the object (normally object server).
Parameters:
attribute – 1=> turn off and 2 => turn on.
userName – the name of the object server.

publishAppoDoc / publishProxyDoc

public String publishAppoDoc(String userName, String userPassword, String appoUrl, String appoID, String appName, Vector methodNames, Vector returnTypes, Vector paramTypes, int portNum, int listenerPort)
To create SyD likely SOAP doc for the object publication.
Parameters:
userName – the name of the object server.
userPassword – the password of the object server.
appoUrl – the location of the object server.
appoID – the proxy ID of the object server. "null" is given if the object registered as the object proxy.
appName – the application name.
methodNames – the methods or services that are going to be registered.
returnTypes – the return types of the methods or services.
paramTypes – the parameter types of each of the methods or services.
portNum – the RMI port of the object server.
listenerPort – the listener port of the object server.
Returns:
The SyD likely SOAP document.

registerAppo / registerProxy

public void registerAppo(String appServerName, int appoPort, String directoryServerName, int directoryServerPort, Object appInstance, String publishAppoDoc)
To register the object server.
Parameters:
appServerName – the name of the object server.
appoPort – the RMI port of the object server.
directoryServerName – the name of the directory server.
directoryServerPort – the port of the directory server.
appInstance – the remote object application.
publishAppoDoc – the SyD likely SOAP doc.

SyDUtil module

The module is used as the tool utilities on processing SyD.

<u>Publisher</u>

Publisher

public Publisher()

The constructor of the publisher method.

Publisher

public Publisher(String newdoc)

The constructor of the publisher method.

Parameters:

newdoc – a SyD likely SOAP message.

getString

public String getString()

To get the string document of the SyD likely SOAP message.

Returns:

The string of the SyD likely SOAP message.

getMethodNames

public Vector getMethodNames()

To get the method names.

getReturnTypes

public Vector getReturnTypes()

To get the return types.

getParamTypes

public Vector getParamTypes()

To get the parameter types.

createPublishUserMethodsRequest

public void createPublishUserMethodsRequest(String userID,String userPasswd,String userURL, String proxyID,String appName, Vector methodNames, Vector returnTypes, Vector paramTypes, String serverPort, String listenerPort)

To create the SyD likely SOAP document for the user publishied methods registration.

<u>SyDPropertyFile</u>

getValue

public String getValue(String propName,String name) throws MissingResourceException

To get the property value in the file based on the property name file.

Parameters:

propName – the name of the file holds the properties.

name – the name of the property.

Returns:

The property value.

<u>**SyDDoc**</u>

SyDDoc

public SyDDoc()

Constructor of SyD document.

SyDDoc

public SyDDoc(StringBuffer xmlString)

Another constructor for SyD document.

Parameters:

xmlString – the SyD likely SOAP message document.

getString

public StringBuffer getString()

To get the string of SyD likely SOAP document.

createRequest

public void createRequest(String objectID, String methodName, Vector parameterTypeList, Vector parameterValueList)

To create requests of the services.

getObjectID

public String getObjectID()

To get the object ID.

getMethodName

public String getMethodName()

To get the method name.

getParameterValue

public Vector getParameterValue()

To get the parameter value.

getParameterType

public Vector getParameterType()

To get the parameter type.

createResponse

public void createResponse(Object ob)

To create response.

SyDListener module

There are three main functions of the module: to register object (SyDRegistrar), to listen and execute the local services (SyDListener), and to communicate with the remote object

(SyDListenerDelegate)

SyDListenerDelegate

invoke

public String invoke(String inputString) throws IOException

To communicate with the remote object server.

Parameters:

inputString – the SyD likely SOAP doc containing the methods or services to be invoked

Returns:

SyD likely SOAP doc response.

SyDListener

work

private void work(Socket clientSocket)

To communicate with the remote object client.

Parameters:

clientSocket – the TCPIP socket used to listen on the requests of services.

invoke

private String invoke (String message)

To parse the message and get the response upon local method invocation.

Parameters:

message – the SyD likely SOAP documents containing the request of service.

Returns:

The response of the invocation.

public String invoke (String objectName, String methodName, Vector parameterTypes, Vector parameterValues)

To invoke the local registered methods or services.

Parameters:

objectName – the object ID of the server.

methodName – the method of service name.

parameterTypes – the parameter types of the method.

parameterValues – the parameter values of the method.

Returns:

The response of the invocation.

SyDRegistrar

register

public void register (Object appInstance, String publishDoc, int liveBit)

To register the object with the services (register globally and locally afterward).

Parameters:
appInstance – the application method of service name to be registered.
publishDoc – the published document.
liveBit – the status of the registered object.

registerToRMIRegistry
public void registerToRMIRegistry (Object appInstance, String objectName)
To register the object locally.
Parameters:
appInstance – the application method of service name to be registered.
objectName – the object ID of the registered object.

registerToDirectoryService
public void registerToDirectoryService (String publishDoc, int liveBit)
To register the object globally.
Parameters:
publishDoc – the published document of the object in the form of SyD likely SOAP.
livebit – the status of the object.

SyDEngine module
The module is used for remote objects invocation. Normally, the object client to the object server.

<u>SyDDispatcher</u>

invoke
public Vector invoke(Vector userlist, String methodname, Vector paramtype, Vector paramvalue) throws IOException
To invoke the remote object (normally done by the object client).
Parameters:
userlist – the list of the object servers that is going to be invoked.
methodname – the requested method or service name.
paramtype – the parameter types of the requested method or service.
Paramvalue – the parameter values of the requested method or service.
Returns:
The result of the object invocation.

SyDDirectory module
The module contains all of the connection to the Directory Server.

<u>MemberShip</u>

MemberShip

public MemberShip() throws RemoteException
The membership constructor.

getConnection
public static Connection getConnection() throws SQLException
The SQL connection.
Returns:
The connection to the Directory Server.

lookUp
public String lookUp(String objecttype,String returnattritutename, String attributename, String attributevalue) throws RemoteException
To lookup the value.

setUp
public void setUp(String objecttype, String attributename, String attributename2, String attributevalue, String attributevalue2) throws RemoteException
To update the value.

publish
public String publish(String info, int liveBit) throws RemoteException
To register the object.

lookUpObject
public Vector lookUpObject(String objecttype,String returnattritutename, String attributename, String attributevalue) throws RemoteException
To look up for attributes, such as url, specified on an object or a proxy of an object
Returns:
A list contains: isRecoveryServer(FALSE/TRUE), object url, object id, listener Port.

advanceLookUp
public Vector advanceLookUp(String objecttype,String returnattritutename, String attributename, String attributevalue) throws RemoteException
Advanced look up for fetching more attributes

turnOff
public Vector turnOff(String objecttype, String attributename, String attributename2, String attributevalue, String attributevalue2) throws RemoteException
To turnoff the status of an object and to get the proxy information of the object.
Returns:
A list contains: object ID, url, and listener Port.

addMember
public void addMember(String groupName, String name) throws RemoteException
To add a member in the group.

deleteMember
public void deleteMember(String groupName, String name) throws RemoteException
To delete a member in the group.

listMember
public String listMember(String groupName) throws RemoteException
To list all the members in the group.

findMember
public String findMember(String groupName,String name) throws RemoteException
To find a member in the group.

findGroup
public String findGroup(String name) throws RemoteException
To find a group for a specified member.

unpublish
public void unpublish(String info) throws RemoteException
To un-publish.

listGroup
public String listGroup( ) throws RemoteException
To list all the groups.

APPENDIX C Camera application

Camera Module and Implementation

CameraModule.java

```
package syd.sydapp.Apps.camera;

import java.rmi.*;
import java.util.*;

public interface CameraModule extends Remote
{
        //String readData(String _loc) throws RemoteException;
        void setIdentities(Boolean _setCamera, String _objID, String _appName) throws
RemoteException;
        //void setDownLink(String _storeRecipientDataToloc, String _reqServerAppName,
String _reqServerParamValue, String _reqServerMethodName) throws RemoteException;
        void runDownLink(Long _elapsedTime, String _storeRecipientDataToloc, String
_reqServerAppName, String _reqServerParamValue, String _reqServerMethodName) throws
RemoteException;
        Boolean beOnLookOut(String _characters, Long _requestedTime, String _requestTo,
String _responseTo) throws RemoteException;
        String readDeleteRequest(String _loc) throws RemoteException;
        void sendConfirmation(Boolean _isFound) throws RemoteException;
}
```

CameraModuleImpl.java

```
package syd.sydapp.Apps.camera;

import java.lang.*;
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.math.*;
import syd.sydutil.*;
import syd.syddirectory.*;
import syd.sydobject.*;

public class CameraModuleImpl extends UnicastRemoteObject implements CameraModule
```

```
{
        private SyDSync serverSync;
        private boolean isCameraOn = true;
        private String objectID = "";
        private String appName = "";
        //private Vector responseTo = new Vector();
        private String responseTo = "";
        private Registry r;
        private MemberShipI member = null;
        private String genObjectID = "";
        private String genUserID = "";
        private long totResponseTime;

        public CameraModuleImpl() throws RemoteException
        {
                try
                {
                        String host =
InetAddress.getLocalHost().getHostAddress();//getHostName();
                        String url = "rmi://" + host + "/CameraModule"; // Original
                        System.out.println("url = " + url);
                        Naming.rebind(url,this);
                        System.out.println("Server bound to: " + url);

                        r = LocateRegistry.getRegistry(host);
                        member = (MemberShipI)r.lookup("DirectoryService");

                        serverSync = new SyDSync();
                }
                catch(java.net.UnknownHostException ex)
                {
                        System.err.println("Couldn't get local host");
                        System.exit(1);
                }
                catch(RemoteException ex)
                {
                        System.err.println("Couldn't connect to rmiregistry");
                        System.exit(1);
                }
                catch(MalformedURLException ex)
                {
                        System.exit(1);
                }
                catch(Exception ex)
                {
```

```
				System.exit(1);
			}
		}

		public void setIdentities(Boolean _setCamera, String _objID, String _appName)
		{
			isCameraOn = _setCamera.booleanValue();
			objectID = _objID;
			appName = _appName;
		}

		//public String readData(String _loc)
		//{
		//	StringBuffer result = serverSync.readData(_loc);
		//	return result.toString();
		//}

		private String getRecepientServerPort()
		{
			SyDPropertyFile prop = new SyDPropertyFile();
			String serverPort = prop.getValue("appo", "appserverport");

			return serverPort;
		}

		public void runDownLink(Long _elapsedTime, String _storeRecipientDataToloc, String
_reqServerAppName, String _reqServerParamValue, String _reqServerMethodName)
		{
			serverSync.runDownLink(_elapsedTime, getRecepientServerPort(), appName,
objectID, _storeRecipientDataToloc, _reqServerAppName, _reqServerParamValue,
_reqServerMethodName);
		}

		//public void setDownLink(String _storeRecipientDataToloc, String
_reqServerAppName, String _reqServerParamValue, String _reqServerMethodName)
		//{
		//	serverSync.downLink(getRecepientServerPort(), appName, objectID,
_storeRecipientDataToloc, _reqServerAppName, _reqServerParamValue,
_reqServerMethodName);
		//}

		public String readDeleteRequest(String _loc)
		{
			StringBuffer result = serverSync.readDeleteRequest(_loc);
			return result.toString();
```

```
	}

	public void sendConfirmation(Boolean _isFound)
	{
		if (_isFound.booleanValue())
			System.out.println( "THE STRING THAT HAS BEEN LOOKED FOR IS 
FOUNDED..." );
		else
			System.out.println( "THE STRING THAT HAS BEEN LOOKED FOR IS 
NOT FOUNDED..." );


		String filePath = serverSync.createAFile("data3.txt");

		StringBuffer readData = serverSync.readData(filePath);
		String readD = readData.toString();
		int i = readD.indexOf(":");
		readD = readD.substring(i+1, readD.length()).trim();
		System.out.println("Start Time: " + readD);
		long startTime = Long.parseLong(readD);

		long endTime = System.currentTimeMillis();
		endTime -= startTime;
		System.out.println("End Time: " + endTime);

		totResponseTime += endTime;
		System.out.println("Total Response Time: " + totResponseTime);

		filePath = serverSync.createAFile("data4.txt");
		StringBuffer data = new StringBuffer();
		data.append("Total Response Time: ").append(totResponseTime);
		serverSync.writeData(data, filePath);
	}

	public Boolean beOnLookOut(String _characters, Long _requestedTime, String 
_requestTo, String _responseTo)
	{
		boolean isRequestAccepted = true;
		//responseTo.addElement(_responseTo);
		responseTo = _responseTo;
		String storeReqLoc = "";

		if (isCameraOn)
		{
			Thread t = new Thread(new lookOut(_characters, _requestedTime));
```

```
                    t.start();
                }
                else
                {
                    Vector paramTList = new Vector();
                    paramTList.addElement("java.lang.String");
                    paramTList.addElement("java.lang.Long");
                    paramTList.addElement("java.lang.String");
                    paramTList.addElement("java.lang.String");
                    //paramTList.addElement("java.lang.String");
                    Vector paramVList = new Vector();
                    paramVList.addElement(_characters);
                    paramVList.addElement(_requestedTime);
                    paramVList.addElement(_requestTo);
                    paramVList.addElement(_responseTo);
                    storeReqLoc = serverSync.createAFile("CameraProxyDoc.xml");
                    //paramVList.addElement(storeReqLoc);
                    Vector objTList = new Vector();
                    Vector objVList = new Vector();
                    Vector objNList = new Vector();

                    try
                    {
                        // Get ObjectID of the requestTo Server
                        genUserID =
member.lookUp("SYD_USER","userID","userName", _requestTo.toString());
                        genObjectID =
member.lookUp("USER_APPO_MAPPING","objectID","userID",genUserID.toString());
                    }
                    catch(Exception e){}

                    StringBuffer dataReq = serverSync.createRequest(genObjectID,
"beOnLookOut", paramTList, paramVList, objTList, objVList, objNList);
                    serverSync.updateRequest(dataReq, storeReqLoc);
                }

                try
                {
                    Thread.sleep(_requestedTime.longValue());
                }
                catch(InterruptedException e) {}

                return Boolean.valueOf(isRequestAccepted);
            }
```

```
	private class lookOut implements Runnable
	{
		private long responsetime=0, starttime=0, endtime=0;
		private Integer length;
		private String characters;
		private Boolean isCharMatch = Boolean.valueOf(false);
		private Long requestedTime;

		public lookOut(String _characters, Long _requestedTime)
		{
			characters = _characters;
			requestedTime = _requestedTime;
			length = new Integer(characters.toString().length());;
		}

		public void run()
		{
			starttime = System.currentTimeMillis();

			while ((endtime - starttime) <= requestedTime.longValue())
			{
				//System.out.println("RESPONSE TIME: " + (endtime - 
starttime));
				//System.out.println("REQUESTED TIME: " + requestedTime);
				String randomChar = generateCharacters(length);
				System.out.println("GENERATED RANDOM CHAR: " + 
randomChar);
				randomChar = "123";
				if (randomChar.equals(characters.toString()))
				{
					isCharMatch = Boolean.valueOf(true);
					break;
				}
				endtime = System.currentTimeMillis();
			}

			//for (long i=0; i < requestedTime.longValue(); ++i)
			//{
				//System.out.println("HELLLLLLLLLLLLLLLLLLLLLLLLLO");
			//}

			//if (!isCharMatch)
			//	System.out.println("CHARACTERS IS NOT FOUND....");
			//else
```

```
                    //        System.out.println("CHARACTERS IS FOUND...." +
responseTo.elementAt(0));

                    //for (int i=0; i<responseTo.size(); ++i)
                    //{
                              Vector serverAppName = new Vector();
                              //serverAppName.addElement(responseTo.elementAt(i));
                              serverAppName.addElement(responseTo);
                              Vector paramValue = new Vector();
                              paramValue.addElement(isCharMatch.toString());
                              ObjClient objClient = new ObjClient(serverAppName,
paramValue, "sendConfirmation");
                              objClient.run();
                    //}

                    //responseTo.clear();
          }
     }

     private String generateCharacters(Integer _charLength)
     {
          String randomChars = "";
          Vector vLetters = getVLetters();

          for (int i = 0; i < _charLength.intValue(); i++)
          {
                    Random generator = new Random();
                    int index = generator.nextInt( vLetters.size() );

                    randomChars += String.valueOf(vLetters.elementAt(index).toString());
          }

          return randomChars;
     }

     private Vector getVLetters()
     {
          Vector vLetters = new Vector();
          vLetters.addElement("A");
          vLetters.addElement("B");
          vLetters.addElement("C");
          vLetters.addElement("D");
          vLetters.addElement("E");
          vLetters.addElement("F");
          vLetters.addElement("G");
          vLetters.addElement("H");
          vLetters.addElement("I");
          vLetters.addElement("J");
          vLetters.addElement("K");
          vLetters.addElement("L");
          vLetters.addElement("M");
          vLetters.addElement("N");
          vLetters.addElement("O");
          vLetters.addElement("P");
          vLetters.addElement("Q");
          vLetters.addElement("R");
          vLetters.addElement("S");
          vLetters.addElement("T");
          vLetters.addElement("U");
          vLetters.addElement("V");
          vLetters.addElement("W");
          vLetters.addElement("X");
          vLetters.addElement("Y");
          vLetters.addElement("Z");
          vLetters.addElement(" ");
          vLetters.addElement("a");
          vLetters.addElement("b");
          vLetters.addElement("c");
          vLetters.addElement("d");
          vLetters.addElement("e");
          vLetters.addElement("f");
          vLetters.addElement("g");
          vLetters.addElement("h");
          vLetters.addElement("i");
          vLetters.addElement("j");
          vLetters.addElement("k");
          vLetters.addElement("l");
          vLetters.addElement("m");
          vLetters.addElement("n");
          vLetters.addElement("o");
          vLetters.addElement("p");
          vLetters.addElement("q");
          vLetters.addElement("r");
          vLetters.addElement("s");
          vLetters.addElement("t");
          vLetters.addElement("u");
          vLetters.addElement("v");
          vLetters.addElement("w");
          vLetters.addElement("x");
          vLetters.addElement("y");
```

```
            vLetters.addElement("z");
            vLetters.addElement("1");
            vLetters.addElement("2");
            vLetters.addElement("3");
            vLetters.addElement("4");
            vLetters.addElement("5");
            vLetters.addElement("6");
            vLetters.addElement("7");
            vLetters.addElement("8");
            vLetters.addElement("9");
            vLetters.addElement("0");
            return vLetters;
        }

        public static void main(String[] args)
        {
            try
            {
                CameraModuleImpl server = new CameraModuleImpl();
            }
            catch(RemoteException ex)
            {
                System.err.println("Trouble creating server: "+ex.getMessage());
                ex.printStackTrace();
            }
        }
    }
```

Object Camera Proxy

CameraProxyServer.java

```
package syd.sydapp.ProxyApp;

import java.util.*;
import java.net.*;
import java.lang.reflect.*;
import java.rmi.registry.*;
import java.io.*;

import syd.sydlistener.*;
import syd.sydutil.*;
import syd.sydobject.*;
import syd.sydapp.Apps.camera.*;
import syd.syddirectory.*;
```

```
public class CameraProxyServer
{
	private static String userName = "hello";
	private static String userPassword = "hello1";
	private static String proxyURL = "";
	private static int proxyPort = 0;
	private static String publishProxyDoc = "";
	private static Vector methodParams;
	private static Vector params;
	private static Vector methods;
	private static Vector returnTypes;
	private static int directoryServerPort = 0;
	private static int listenerPort = 0;

	private static BufferedReader stdin = new BufferedReader(new
InputStreamReader(System.in));

	public static void main(String args[]) //throws IOException
	{
		ObjProxy myProxy = new ObjProxy();

		proxyPort = Integer.parseInt(myProxy.getProxyPort());
		proxyURL = myProxy.getProxyUrl();
		directoryServerPort = Integer.parseInt(myProxy.getDirectoryServerPort());
		listenerPort = Integer.parseInt(myProxy.getListenerPort());

		try
		{
			System.out.print("Username: ");
			userName = stdin.readLine();
			System.out.print("Password: ");
			userPassword = stdin.readLine();
		}
		catch(Exception e){}

		try
		{
			// Setting up method object
			methods = new Vector();
			returnTypes = new Vector();
			methodParams = new Vector();
			params = new Vector();

			// method 1
```

```
methods.addElement(new String("setIdentities"));
returnTypes.addElement(new String("java.lang.Void"));
methodParams.addElement(new String("java.lang.Boolean"));
methodParams.addElement(new String("java.lang.String"));
methodParams.addElement(new String("java.lang.String"));
params.addElement(methodParams);

// method 3
//methods.addElement(new String("setDownLink"));
//returnTypes.addElement(new String("java.lang.Void"));
//methodParams = new Vector();
//methodParams.addElement(new String("java.lang.String"));
//methodParams.addElement(new String("java.lang.String"));
//methodParams.addElement(new String("java.lang.String"));
//methodParams.addElement(new String("java.lang.String"));
//params.addElement(methodParams);

// method 2
methods.addElement(new String("beOnLookOut"));
returnTypes.addElement(new String("java.lang.Boolean"));
methodParams = new Vector();
methodParams.addElement(new String("java.lang.String"));
methodParams.addElement(new String("java.lang.Long"));
methodParams.addElement(new String("java.lang.String"));
methodParams.addElement(new String("java.lang.String"));
params.addElement(methodParams);

// method 3
methods.addElement(new String("readDeleteRequest"));
returnTypes.addElement(new String("java.lang.String"));
methodParams = new Vector();
methodParams.addElement(new String("java.lang.String"));
params.addElement(methodParams);

// method 4
methods.addElement(new String("runDownLink"));
returnTypes.addElement(new String("java.lang.Void"));
methodParams = new Vector();
methodParams.addElement(new String("java.lang.Long"));
methodParams.addElement(new String("java.lang.String"));
methodParams.addElement(new String("java.lang.String"));
methodParams.addElement(new String("java.lang.String"));
methodParams.addElement(new String("java.lang.String"));
params.addElement(methodParams);
```

```
                    // method 5
                    methods.addElement(new String("sendConfirmation"));
                    returnTypes.addElement(new String("java.lang.Void"));
                    methodParams = new Vector();
                    methodParams.addElement(new String("java.lang.Boolean"));
                    params.addElement(methodParams);


                    // Setting up Server Application
                    CameraModuleImpl helloSydObject = new CameraModuleImpl();

                    // Publish proxy doc
                    publishProxyDoc = myProxy.publishProxyDoc(userName, userPassword,
proxyURL, "null", "CameraModule", methods, returnTypes, params, proxyPort, listenerPort);

                    // Register myProxy
                    myProxy.registerProxy(proxyURL, proxyPort, proxyURL,
directoryServerPort, helloSydObject, publishProxyDoc);
            }
            catch (Exception e)
            {
            }

            // Setting up the data
            SyDSync newSync = new SyDSync();
            String filePath = newSync.createAFile( "CameraProxyDoc.xml" );
            newSync.writeEmptyRequest( filePath );

            // Client of the Camera Proxy Server
            // -----------------------------------------------------------------------------------------------
------------ START setIdentities
            ObjClient objClient = new ObjClient();
            String dirurl = objClient.getDirectory();

            Registry r;
            MemberShipI member = null;
            String userid = "";
            String objectID = "";

            try
            {
                    r = LocateRegistry.getRegistry(dirurl);
                    member = (MemberShipI)r.lookup("DirectoryService");
                    userid = member.lookUp("SYD_USER","userID","userName",
userName.toString());
```

```
                    objectID = 
member.lookUp("USER_APPO_MAPPING","objectID","userID",userid);
                }
                catch(Exception e) {
                    System.out.println("Error " + e);
                }

                Vector serverAppName = new Vector();
                serverAppName.addElement(userName);
                Vector paramValue = new Vector();
                paramValue.addElement("false");
                paramValue.addElement(objectID);
                paramValue.addElement(userName);

                String methodName = "setIdentities";
                objClient = new ObjClient(serverAppName, paramValue, methodName);
                objClient.run(true);
                // -------------------------------------------------------------------------------------------
------------ END setIdentities

                return;
        }
}
```

Object Camera Server

<u>CameraServer.java</u>

```
package syd.sydapp.Apps.camera;

import java.util.*;
import java.net.*;
import java.lang.reflect.*;
import java.rmi.registry.*;
import java.io.*;

import syd.sydlistener.*;
import syd.sydutil.*;
import syd.sydobject.*;
import syd.syddirectory.*;

public class CameraServer
{
        private static String userName = "";
        private static String userPassword = "hello1";
```

```
	private static String proxyObjectName = "";
	private static String appServerName = "";
	private static int portNum = 0;
	private static String objUrl = "";
	private static String publishAppoDoc = "";
	private static Vector methodParams;
	private static Vector params;
	private static Vector methods;
	private static Vector returnTypes;
	private static String directoryServerName = "";
	private static int directoryServerPort = 0;
	private static int listenerPort = 0;

	private static BufferedReader stdin = new BufferedReader(new
InputStreamReader(System.in));

	public static void main(String args[])
	{
		/*
		if (args.length != 6)
		{
			System.out.println("Usage: java TestSyDRegistrar rmiServerPort flag
directoryServerName directoryServerPort userName");
			System.exit(1);
		}
		*/

		ObjAppo myAppo = new ObjAppo();

		portNum = Integer.parseInt(myAppo.getAppoPort());
		directoryServerPort = Integer.parseInt(myAppo.getDirectoryServerPort());
		listenerPort = Integer.parseInt(myAppo.getListenerPort());
		objUrl = myAppo.getAppoUrl();

		try
		{
			System.out.print("Username: ");
			userName = stdin.readLine();
			System.out.print("Password: ");
			userPassword = stdin.readLine();
			System.out.print("Proxy Object Name: ");
			proxyObjectName = stdin.readLine();
		}
		catch(Exception e){}
```

```
// Setting Up Methods
                methods = new Vector();
                returnTypes = new Vector();
                methodParams = new Vector();
                params = new Vector();

                // method 1
                methods.addElement(new String("setIdentities"));
                returnTypes.addElement(new String("java.lang.Void"));
                methodParams.addElement(new String("java.lang.Boolean"));
                methodParams.addElement(new String("java.lang.String"));
                methodParams.addElement(new String("java.lang.String"));
                params.addElement(methodParams);

                // method 3     void setDownLink(String _storeRecipientDataToloc, String
_reqServerAppName, String _reqServerParamValue, String _reqServerMethodName) throws
RemoteException;
                //methods.addElement(new String("setDownLink"));
                //returnTypes.addElement(new String("java.lang.Void"));
                //methodParams = new Vector();
                //methodParams.addElement(new String("java.lang.String"));
                //methodParams.addElement(new String("java.lang.String"));
                //methodParams.addElement(new String("java.lang.String"));
                //methodParams.addElement(new String("java.lang.String"));
                //params.addElement(methodParams);

                // method 2     Boolean beOnLookOut(String _characters, Long _requestedTime,
String _recipientAppName, String _recipientObjectID, String _storeReqLoc) throws
RemoteException;
                methods.addElement(new String("beOnLookOut"));
                returnTypes.addElement(new String("java.lang.Boolean"));
                methodParams = new Vector();
                methodParams.addElement(new String("java.lang.String"));
                methodParams.addElement(new String("java.lang.Long"));
                methodParams.addElement(new String("java.lang.String"));
                methodParams.addElement(new String("java.lang.String"));
                params.addElement(methodParams);

                // method 3
                methods.addElement(new String("readDeleteRequest"));
                returnTypes.addElement(new String("java.lang.String"));
                methodParams = new Vector();
                methodParams.addElement(new String("java.lang.String"));
                params.addElement(methodParams);
```

```
                    // method 4
                    //runDownLink(Long _elapsedTime, String _storeRecipientDataToloc, String
_reqServerAppName, String _reqServerParamValue, String _reqServerMethodName)
                    methods.addElement(new String("runDownLink"));
                    returnTypes.addElement(new String("java.lang.Void"));
                    methodParams = new Vector();
                    methodParams.addElement(new String("java.lang.Long"));
                    methodParams.addElement(new String("java.lang.String"));
                    methodParams.addElement(new String("java.lang.String"));
                    methodParams.addElement(new String("java.lang.String"));
                    methodParams.addElement(new String("java.lang.String"));
                    params.addElement(methodParams);


// End Setting Up Methods

                    try
                    {
                              publishAppoDoc = myAppo.publishAppoDoc(userName, userPassword,
objUrl, myAppo.getProxyID(proxyObjectName), "CameraModule", methods, returnTypes,
params, portNum, listenerPort);
                              CameraModuleImpl helloSydObject = new CameraModuleImpl();
                              myAppo.registerAppo(objUrl, portNum, objUrl, directoryServerPort,
helloSydObject, publishAppoDoc);

                    }
                    catch (Exception e)
                    {
                              System.out.println("Error in setting up the Application Server: " +
e.toString());
                    }

                    // Setting up the data
                    SyDSync newSync = new SyDSync();
                    String filePath = newSync.createAFile( "CameraAppoDoc.xml" );
                    newSync.writeEmptyRequest( filePath );

                    // Client of the Camera Server
                    // ------------------------------------------------------------------------------------------
------------ START setIdentities

          //paramValue.addElement("//export//home//students//jgunawan//syddemo//syd//sydapp//P
roxyApp//CameraAppoDoc.xml");
                    //paramValue.addElement(
"C:\\Users\\Joseph\\Desktop\\syddemo\\syd\\sydapp\\ProxyApp\\CameraAppoDoc.xml" );
```

```
		ObjClient objClient = new ObjClient();
		String dirurl = objClient.getDirectory();

		Registry r;
		MemberShipI member = null;
		String userid = "";
		String objectID = "";

		try
		{
			r = LocateRegistry.getRegistry(dirurl);
			member = (MemberShipI)r.lookup("DirectoryService");
			userid = member.lookUp("SYD_USER","userID","userName",
userName.toString());
			objectID =
member.lookUp("USER_APPO_MAPPING","objectID","userID",userid);
		}
		catch(Exception e) {
			System.out.println("Error " + e);
		}

		Vector serverAppName = new Vector();
		serverAppName.addElement(userName);
		Vector paramValue = new Vector();
		paramValue.addElement("true");
		paramValue.addElement(objectID);
		paramValue.addElement(userName);
		String methodName = "setIdentities";

		objClient = new ObjClient(serverAppName, paramValue, methodName);
		objClient.run(true);
		// ---------------------------------------------------------------------------------------------
------------ END setIdentities

		// ---------------------------------------------------------------------------------------------
------------ START runDownLink
		// void runDownLink(Long requestedTime, String _storeRecipientDataToloc,
String _reqServerAppName, String _reqServerParamValue, String _reqServerMethodName)
throws RemoteException;

		serverAppName.clear();
		serverAppName.addElement(userName);
		paramValue = new Vector();
		paramValue.addElement( "10000" );
```

```
                    paramValue.addElement( filePath );
                    //paramValue.addElement(
"C:\\Users\\Joseph\\Desktop\\syddemo\\CameraAppoDoc.xml" );
                    paramValue.addElement( "CameraProxy" );
                    //paramValue.addElement(
"//export//home//students//jgunawan//syddemo//CameraAppoDoc.xml" );
                    paramValue.addElement(
"C:\\Users\\Joseph\\Desktop\\syddemo\\CameraProxyDoc.xml" );
                    paramValue.addElement( "readDeleteRequest" );
                    methodName = "runDownLink";

                    objClient = new ObjClient(serverAppName, paramValue, methodName);
                    objClient.run(true);
                    // ------------------------------------------------------------------------------------------------
----------- END runDownLink


                    return;
          }
}
```

Object Camera Client

Server of Camera Client

CameraClientServer.java

```
import java.util.*;
import java.net.*;
import java.lang.reflect.*;
import java.rmi.registry.*;
import java.io.*;

import syd.sydlistener.*;
import syd.sydutil.*;
import syd.sydobject.*;
import syd.sydapp.Apps.camera.*;
import syd.syddirectory.*;

public class CameraClientServer
{
          private static String userName = "hello";
          private static String userPassword = "hello1";
          private static String proxyURL = "";
          //private static String proxyServerName = "";
```

```
	private static int proxyPort = 0;
	private static String publishProxyDoc = "";
	private static Vector methodParams;
	private static Vector params;
	private static Vector methods;
	private static Vector returnTypes;
	//private static String directoryServerName = "";
	private static int directoryServerPort = 0;
	private static int listenerPort = 0;

	private static BufferedReader stdin = new BufferedReader(new
InputStreamReader(System.in));

	public static void main(String args[]) //throws IOException
	{
		ObjClientServer myProxy = new ObjClientServer();
		//proxyServerName = myProxy.getProxyServerName();
		proxyPort = Integer.parseInt(myProxy.getProxyPort());
		proxyURL = myProxy.getProxyUrl();
		//directoryServerName = myProxy.getDirectoryServerName();
		directoryServerPort = Integer.parseInt(myProxy.getDirectoryServerPort());
		listenerPort = Integer.parseInt(myProxy.getListenerPort());

		try
		{
			System.out.print("Username: ");
			userName = stdin.readLine();
			System.out.print("Password: ");
			userPassword = stdin.readLine();
		}
		catch(Exception e){}

/*
		System.out.println("Username: " + userName);
		System.out.println("UserPassword: " + userPassword);
		System.out.println("proxyPort: " + proxyPort);
		System.out.println("proxyUrl: " + proxyURL);
		System.out.println("directoryServerName: " + directoryServerName);
		System.out.println("directoryServerPort: " + directoryServerPort);
*/
		try
		{
			// Setting up method object
			methods = new Vector();
			returnTypes = new Vector();
```

```
                        methodParams = new Vector();
                        params = new Vector();

                        // method 1
                        //void sendConfirmation(Boolean _isFound) throws RemoteException;
                        methods.addElement(new String("sendConfirmation"));
                        returnTypes.addElement(new String("java.lang.Void"));
                        methodParams.addElement(new String("java.lang.Boolean"));
                        params.addElement(methodParams);

                        // Setting up Server Application
                        CameraModuleImpl helloSydObject = new CameraModuleImpl();

                        // Publish proxy doc
                        publishProxyDoc = myProxy.publishProxyDoc(userName, userPassword,
proxyURL, "null", "CameraModule", methods, returnTypes, params, proxyPort, listenerPort);

                        // Register myProxy
                        myProxy.registerProxy(proxyURL, proxyPort, proxyURL,
directoryServerPort, helloSydObject, publishProxyDoc);
                }
                catch (Exception e)
                {
                }
                return;
        }
}
```

Client of Camera Client

<u>ClientThread.java</u>

```
import syd.sydengine.*;
import syd.sydutil.*;
import syd.sydlistener.*;
import syd.syddirectory.*;
import syd.sydobject.*;

import java.lang.*;
import java.util.*;
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
import java.net.*;
```

```
public class ClientThread implements Runnable
{
	private Vector ServerAppName;
	private Vector paramValue;
	private int pos;
	private String methodName;

	public void setupClient(Vector _serverAppName, Vector _paramValue, int _pos, String
_methodName)
	{
		ServerAppName = _serverAppName;
		paramValue = _paramValue;
		pos = _pos;
		methodName = _methodName;
	}

	public void run()
	{
		System.out.println("\n\nStart to run the client # " + pos + " ...");
		ObjClient myClient = new ObjClient(ServerAppName, paramValue,
methodName);
		myClient.run();
		System.out.println("Finish running the client # " + pos + " ...");
	}
}
```

CameraMultiClients.java

```
import syd.sydengine.*;
import syd.sydutil.*;
import syd.sydlistener.*;
import syd.syddirectory.*;

import java.lang.*;
import java.util.*;
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
import java.net.*;

public class CameraMultiClients
{
	private static BufferedReader stdin = new BufferedReader(new
InputStreamReader(System.in));
```

```
	public static void main(String[] args) throws IOException
	{
		SyDSync newSync = new SyDSync();

		// Holding multiple clients & values
		Vector multiClients = new Vector();
		Vector multiValues = new Vector();

		System.out.println("Running Trial Application V.1.1 (Camera Application) ...");

		System.out.println("How many client(s) do you want to run (X for eXit)?");
		String userAnswer = stdin.readLine();
		while ( (userAnswer.equals("X")) || (Integer.parseInt(userAnswer) == 0) )
		{

			if (userAnswer.equals("X"))
			{
				System.out.println("Exit the application!");
				System.exit(0);
			}

			System.out.println("There must be at least one client need to be run.");
			System.out.println("Please enter number of client(s) do you want to run (X
for eXit)?");
			userAnswer = stdin.readLine();
		}

		int numOfClients = Integer.parseInt(userAnswer);

		String serverName = "";
		for (int i = 0; i < numOfClients; i++)
		{
			Vector ServerAppName = new Vector();
			Vector paramValue = new Vector();

			if (i == 0)
			{
				System.out.print("Enter the camera application name: ");
				serverName = "CameraServer";
				//serverName = stdin.readLine();
				ServerAppName.addElement( serverName );
			}
			else
				ServerAppName.addElement( serverName );
			System.out.print("Enter the vehicle license plate to be looked for: ");
```

```
				paramValue.addElement( "jh43iotre" );
				//paramValue.addElement( stdin.readLine() );
				System.out.print("Enter the duration of the time for looking (in 
miliseconds): ");
				String lookOutTime = "2000";
				paramValue.addElement( lookOutTime );
				//paramValue.addElement( stdin.readLine() );
				paramValue.addElement( serverName ); // serverName suppose to be a 
server that the client would like to invoke, but the server is not there (off).
				System.out.print("Where do you want to get a response from the camera 
after look out for " + lookOutTime + " miliseconds: ");
				paramValue.addElement( "CameraClient" );
				//paramValue.addElement( stdin.readLine() ); // response to what server 
client
				// location to store the unanswered request
				//paramValue.addElement( 
"//export//home//students//jgunawan//syddemo//CameraAppoDoc.xml" );

				multiClients.addElement( ServerAppName );
				multiValues.addElement( paramValue );
			}

			Vector clientTemp = new Vector();
			for (int i = 0; i < numOfClients; i++)
			{
				Random generator = new Random();
				int clientNo = generator.nextInt( numOfClients );

				if (clientTemp.size() == 0)
					clientTemp.addElement( clientNo );
				else
				{
					while( clientTemp.contains( clientNo ) )
						clientNo = generator.nextInt( numOfClients );
					clientTemp.addElement( clientNo );
				}
			}

			String filePath = newSync.createAFile("data3.txt");
			long startTime = System.currentTimeMillis();
			StringBuffer data = new StringBuffer();
			data.append("Start Time: ").append(startTime);
			newSync.writeData(data, filePath);

			for (int i = 0; i < clientTemp.size(); i++)
			{
				int pos = (Integer)clientTemp.elementAt(i);
				ClientThread expectedClient = new ClientThread();
				expectedClient.setupClient( ((Vector)multiClients.elementAt( pos )),
((Vector)multiValues.elementAt( pos )), pos, "beOnLookOut" );
				Thread threadClient = new Thread( expectedClient );
				threadClient.start();
			}
		}
}
```