CONCORDIA UNIVERSITY ENGINEERING

AND COMPUTER SCIENCE

COURSE: DISTRIBUTED SYSTEMS

DESIGN (COMP6231)

SUMMER – 2015

CORBA & Web Services Implementation for GIPSY

BY: TEAM 4

Nehal Bhagat
Parikshit Pandya
Kumaran Ayyappan
Nandhini Devaraj

PROFESSOR

NAME: Serguei A. Mokhov
EMAIL: mokhov@cs.concordia.ca

DATE OF SUBMISSION

13 SEPTEMBER, 2015

ABSTRACT

GIPSY (General Intensional Programming System) is a research project which extends GIPSY's multi-tier architecture implementation [1] to RMI, CORBA, and WS for future scalability studies. The tiers used in GIPSY's multi-tier architecture are GMT (management), DGT (generators of demands), DWT (workers), and DST (store, caching of worker's results. RMI implementation is the process of implementing Hashtable-based DST by extending DSTWrapper and process demands accordingly. CORBA implementation is similar to RMI module, the goal here, is the same to process a demand by storing it in hash table, processing it and giving back when it's processed. The conversion from GIPSY to CORBA and vice versa, is done through Adapter/Delegate classes which takes CORBA demand which is further converted into GIPSY type, processed and converted back into the CORBA format. The IDL used in CORBA reflects the interfaces developed in RMI implementation. The same concept is used for Web Services (SOAP) where WSDL is created from annotated web services classes in eclipse which is again linked through adapter/delegate classes. Thus, Scalability studies are made by evaluating the performance measures for RMI, CORBA and Web Service implementations.

TABLE OF CONTENT

1. Introduction………………………………...……………………………..4
2. Description……………....…………………………………………………..5
3. GIPSY Architecture……...…………………………………………………..7
4. Deployment Architecture……………………………………………………9
5. Implementation………………………………………………………………10
    5.1 RMI…………………………………………………………………...
    5.2 CORBA………………………………………...……………11
    5.3 Web Services………………………………...………………12
    5.4 Performance Evaluation……………………………………………..13
6. Logger
    6.1 Overview……………………………………………………………...19
    6.2 Design and implementation……………………………………………….. 19
7. Utilities
    7.1 IDE……………………………………………………………………..20
    7.2 Repository……………………………………………………………...20
    7.3 Language………………………………………………………………20
8. How to run?..23
9. Limitations…………………………………………………………………26
10.Acknowledgement…………………………………………………….……..27
11. References…………………………………………………………..…….28

TABLE OF FIGURES

Figure 2.1 GIPSY flow diagram…………………………………………………6
Figure 3.1 GIPSY architecture…………………………………………………..8
Figure 4.1 Adapted GIPSY architecture…………………………………………....8
Figure 4.2 Design Layout of Logger mechanism
Figure 4.1.1 Class diagram for RMI……………………………………………10
Figure 4.2.1 Class diagram for CORBA………………………………………..11
Figure 4.2.1 Class diagram for WS……………………………………………..14
Figure 8.3.1 WSDL………………………………………………………………..24
Figure 8.3.2 getHashTableValue…………………………………………………..24
Figure 8.3.3 setHashTable……………………………………………………25

1. Introduction

GIPSY1 is a research project that extends GIPSY (General Intentional Programming System). GIPSY's multi-tier run-time currently uses either JMS or Jini as middleware. The tiers are GMT (management), DGT (generators of demands), DWT (workers), and DST (store, caching of worker's results) using Jini or JMS Transport Agents (TAs). Needed to have a concrete JAX-WS and SOAP based middleware for its TAs to expose GIPSY's tiers as services.

The project uses RMI, CORBA, Web Services (SOAP) as middleware for the existing GIPSY's multi-tier architecture and test cases are made to compare the performance of each of the middleware for scalability studies. In the description section, we have introduced the general concept of GIPSY, what's the basic idea of the entire project and how it works, the flow and hierarchy of different interfaces and classes used. Next, in implementation, description for each module is described as what have been done in RMI, CORBA and Web Service as well. After implementing all these modules, comparison of these modules have been done using different test cases for sending 10,100 and 1000 demands in four different machines with identical environment 10 times.

2. Description

Here we briefly describe the background and previous work of GIPSY since the project is its extension. The GIPSY basically consists of a flexible compiler and a scalable runtime system for the compilation and execution of intentional programs, where the compiler translates any flavor of intentional programs into source-language independent runtime resources, and the runtime system uses the runtime resources to execute the program in a demand-driven and distributed manner, i.e. computation requirements are wrapped into demands and are distributed among networked computers, so that the required computations can be executed distributive and concurrently to shorten their overall computation time. [1]

The GIPSY framework consists of three main subsystems: a flexible compiler called the General Intentional Programming Compiler (GIPC), a language-independent runtime system called the General Education Engine (GEE) and a component called the Runtime Interactive Programming Environment (RIPE) that provides visual user interfaces to allow user interaction with the runtime system. Our main focus is on GIPSY's runtime system, GEE. The early architectural design of GEE, i.e. the runtime system, specified that the GEE generally had three parts: the Intentional Demand Propagator (IDP) that generates demands, the Intentional ValueWarehouse (IVW) that caches the values of computed demands, and the Ripe Function Executor (RFE), i.e. the worker that does functional computation.

Later, a generic Demand Migration Framework (DMF) for migrating demands in heterogeneous and distributed environment was proposed for the GIPSY runtime system, and a Demand Migration System (DMS) implementing the DMF using the Jini technology and the JMS technology was provided. In DMS, Demand Generators (DGs) and Demand Workers (DWs) can use the Transport Agents (TAs) to connect to Dispatcher Proxies (DPs) to send and receive demands via the DS across heterogeneous and distributed environment. The dispatching is achieved by labeling demands with three different states: pending, in-process, and computed. Each demand also has a signature called demand signature serving as the unique identifier of this demand within the DS. The demand migration and dispatching process involves: when a demand is generated by the Demand Generator and is put into the Demand Space, its state is pending; when the demand is picked up by a Demand Worker, its state transits to in-process; after the demand is computed by the Demand Worker, its state transits to computed and the computed demand with the same signature is put back into the Demand Space so that it can be picked up by the Demand Generator. Illustration of which is shown below

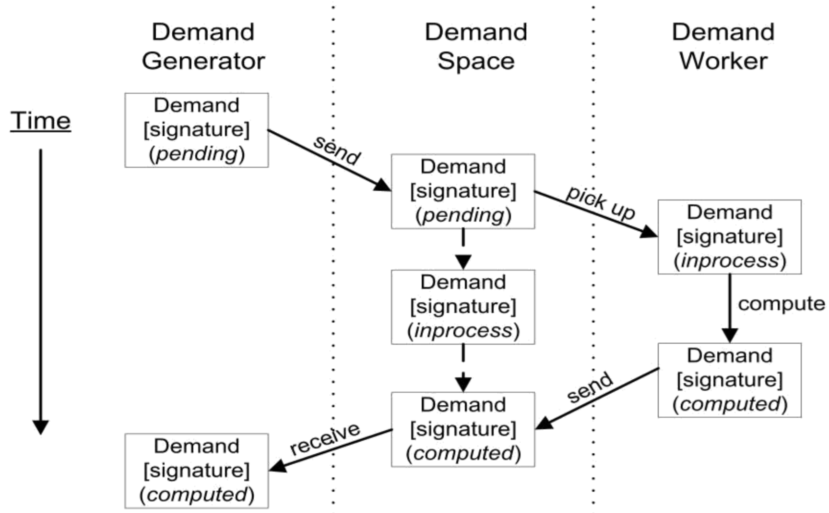


Figure 2.1 GIPSY flow diagram [1]

3. GIPSY Architecture

GIPSYs multi-tier architecture is composed of 4 main tiers : (a) A Demand Store Tier (DST) that acts as a middleware between tiers in order to migrate demands, provides persistent storage of demands and their resulting values, and exposes Transport Agents (TAs) used by other tiers to connect to the DST; (b) A Demand Generator Tier (DGT) that generates demands according to the declarations and resources generated for the program being evaluated. (c) A Demand Worker Tier (DWT) which processes demands by executing method defined in such a dictionary. The DWT connects to the DST, retrieves pending demands and returns back the computed demands to the DST; (d) A General Manager Tier locally and remotely controls and monitors other tiers (DGT, DWT and DST) by exchanging system demands. Also, the GMT can register new nodes, move tier instances from one node to another, or allocate/de-allocate tier instance from/on a registered node. [2]

The GIPSY runtime system is a distributed multi-tier and demand-driven framework composed of 4 main tiers: (a) A Demand Store Tier (DST) that acts as a middleware between tiers in order to migrate demands; (b) A Demand Generator Tier (DGT) that generates demands; (c) A Demand Worker Tier (DWT) which processes demands; (d) A General Manager Tier which locally and remotely controls and monitors other tiers (DGT, DWT and DST). Also, the GMT can register new nodes, move tier instances from one node to another, or allocate/de-allocate tier instance from/on a registered node.

The architecture is displayed in the figure below:

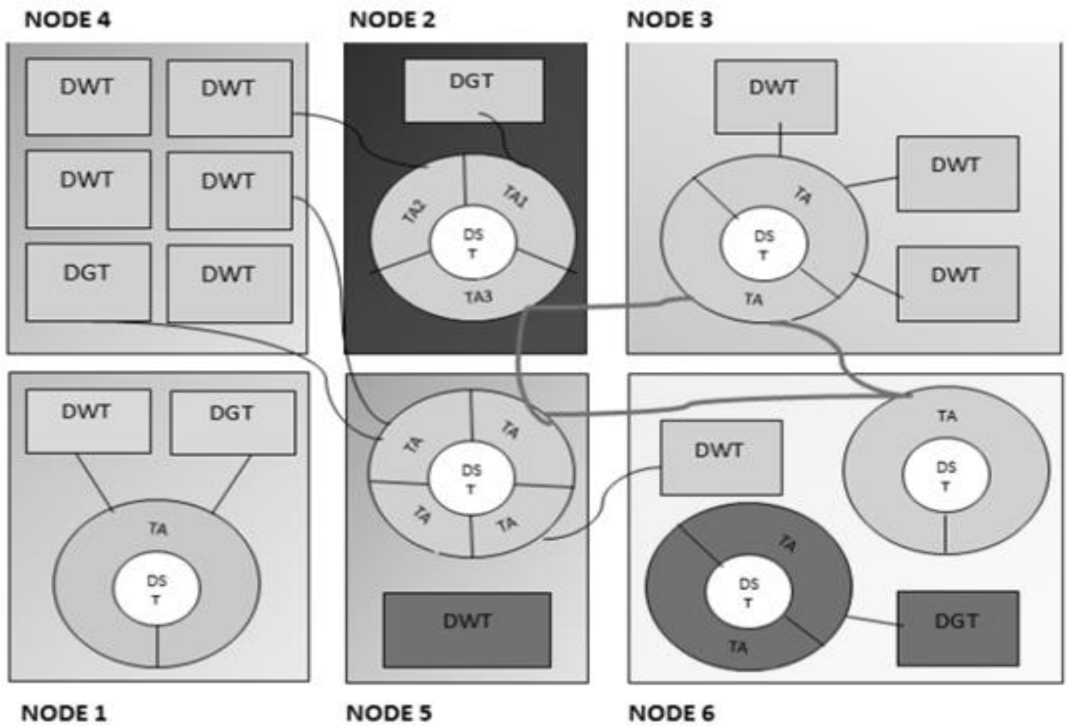


Figure 3.1 GIPSY Architecture [4]

4. Deployment Architecture

The Project uses single instance of each tier such as DWT (Demand Worker Tier), DGT (Demand Generator Tier), TA (Transport Agent) for each of the implementations such as RMI, CORBA and WS (Web Service). The required number of demand is generated in DGT which is carried away by sending request through Transport Agent (TA) and finally it is stored in the Demand Store Tier (DST).

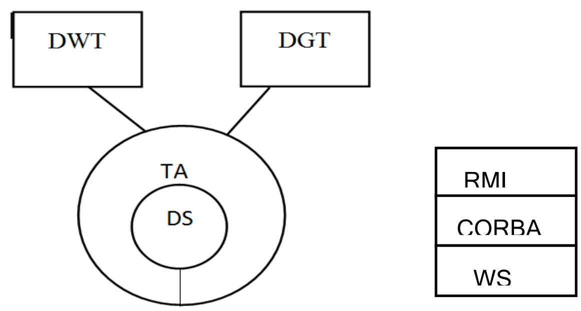


Figure 4.1 Adapted GIPSY Architecture

All coding is deployed using JDK 1.7 version of JAVA consistently throughout the project in Eclipse-Luna Environment.

5. Implementation

The extension of multi-tier architecture of GIPSY's runtime system with RMI, CORBA and Web Service technology as middleware. The goal of each implementation is same but they differ in their implementation.

5.1 RMI

- RMIDSTWrapper: A class that extends the DSTWrapper class which starts the RMI services. And binds objects to RMI registry.
- IRMIDSTWrapper class: It is Remote interface for RMIDST that extends Remote.
- RMIDST: A class that implements the remote interface of RMI. Contains a method to store and fetch values from Hashtable by taking unique string as key and the object of IDemand class as value.
- RMITransportAgent class: Works as middle man class between DWT and DST. Delegates demands to store and fetches pending demands and gives to worker.
- RMITransportAgentProxy: a class that extends RMITransportAgent. It is the proxy that uses the backend protocol. This proxy implements the IRMITransportAgent hence allowing the clients to use it transparently.
- IRMITransportAgent: An interface for RMITrasportAgent that extends Remote.
- RMIDemandDispatcher: A class that extends DemandDispatcher and implements IRMIDemandDispatcher.
- IRMIDemandDispatcher: a class that extends both Remote and IDemandDispatcher.
- Ilogger: Provides interface for logger class.
- logger: Implements Ilogger and logger class has methods such as info(),debug(), error() to make log entries which is being send to client class.
- client: It take log message that comes from logger class and generates clientLogger file and send log message over UDP to server.
- server: UDP server listens client on port 1099, it keep listening until you shut down server. It receives log message from client and makes log entry to serverLogger file.
- RMIDSTWrapperTest: Test case to check all classes have proper implementation.

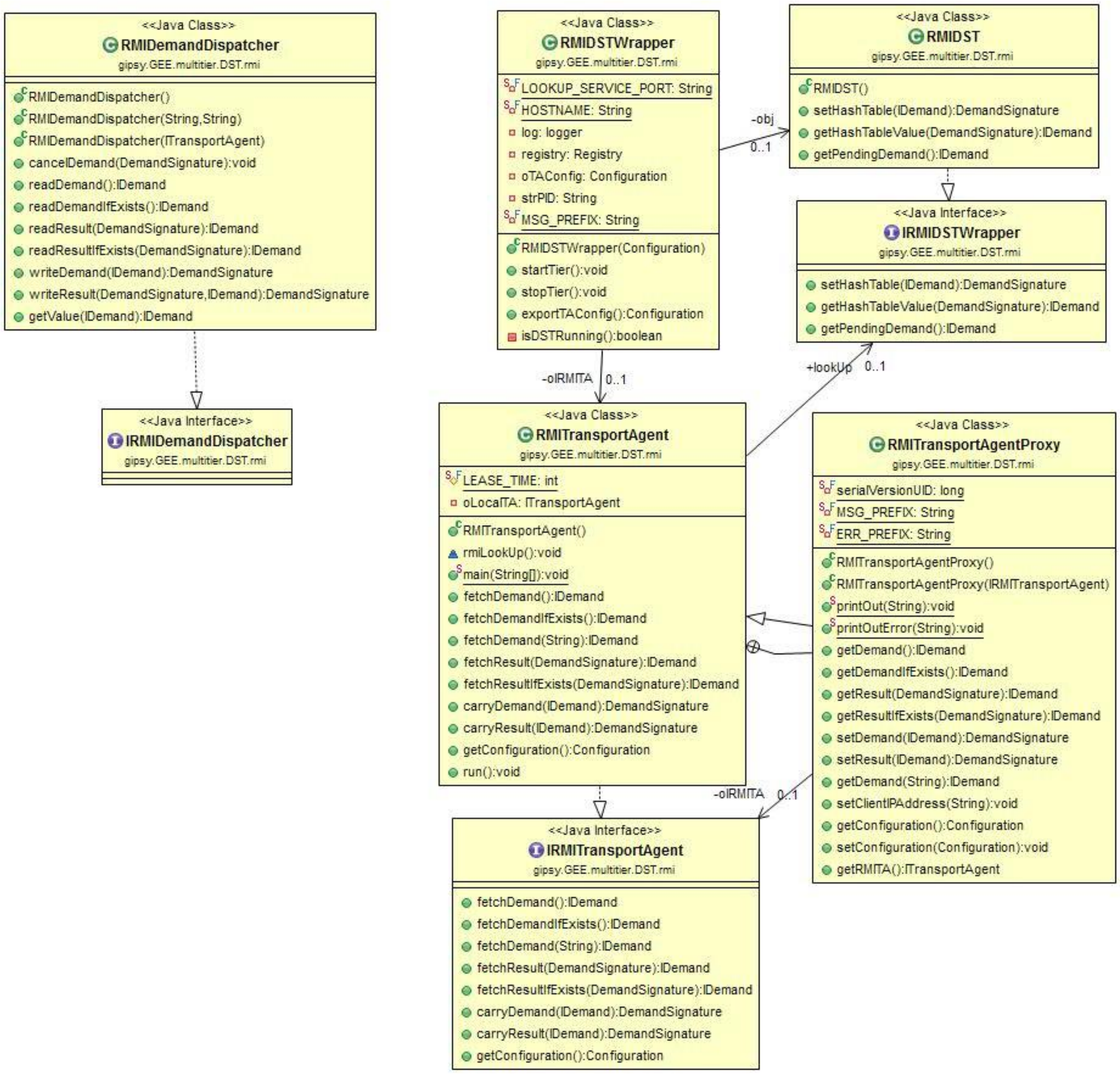


Figure 4.1.1 Class Diagram for RMI

5.2 CORBA:

- CORBA.idl **:** a file for defining interfaces i.e. ICORBADSTWrapper and ICORBATransportAgent which is further compiled to generate CORBA related classes like helper, holder, POA.
- CORBAAdapter : It converts between CORBADemandSignature and DemandSignature and CORBADemand and ProceduralDemand.
- Delegate/Wrapper : It implements the standard GIPSY interfaces and link to the CORBA servant and the Adapter when translating data structures back and forth.
- ICORBAServer: It contains the methods which set the root portable object adapter for the server.
- ICORBADST: It's an interface for declaring methods for storing and retrieving values to and from the hash table.
- ICORBATA: It declares all the methods to fetch, carry demands and results which further will be implemented by CORBATransportAgent.

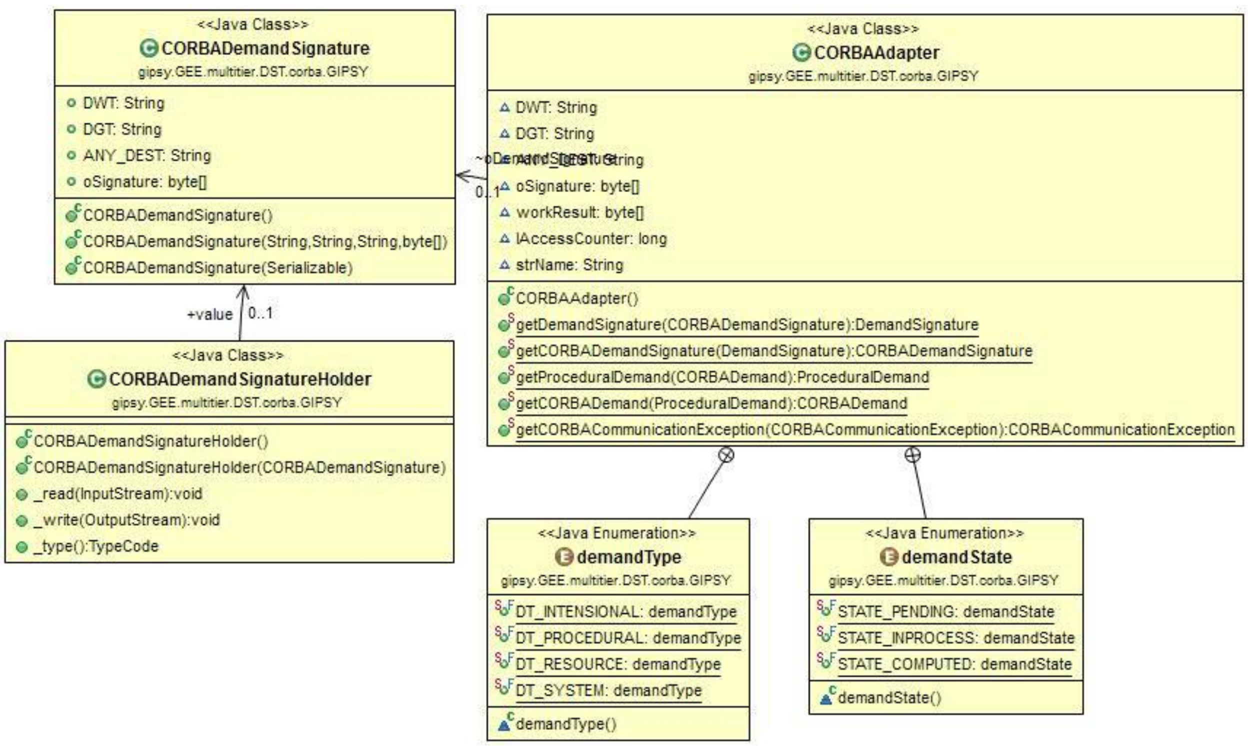


Figure 4.2.1 Class diagram for CORBA

5.3 Web Service:

- WSDSTWrapper: The class is the implementation of web methods, contains method to store and get Hash Table Value and also to get pending demand.
- WSDemand: The class auto-generated from WSDL that implements Serializable object and provide helper functions for serialization and de-serialization.
- WSDSTWrapper: The class auto-generated from WSDL that extends remote interface and throws Remote Exception, provide methods such as setValue and print value.
- WSDSTWrapperProxy: The class that acts as proxy stubs for setting and getting property values, set and get end point values.
- WSDSTWrapperService: The class auto-generated from WSDL that extends rpc service of xml
- WSDSTWrapperServiceLocator: The class auto-generated from WSDL that extends client service to get location of WS service and its name.
- WSDSTWrapperSoapBindingStub: The class auto-generated from WSDL that acts as a helper function to create call, SOAP bind and set value.
- WSValueHouse: The class that contains method like getpendingvalue, setvalue, getvalue.
- Logger: The class that has methods such as info, debug, error to make log entries.
- LoggerClient: The client class that request for log entry to server through SOCKET and port.
- LoggerServer: The server class that responds to client request and stores logger details.

Class diagram for web services in next page.

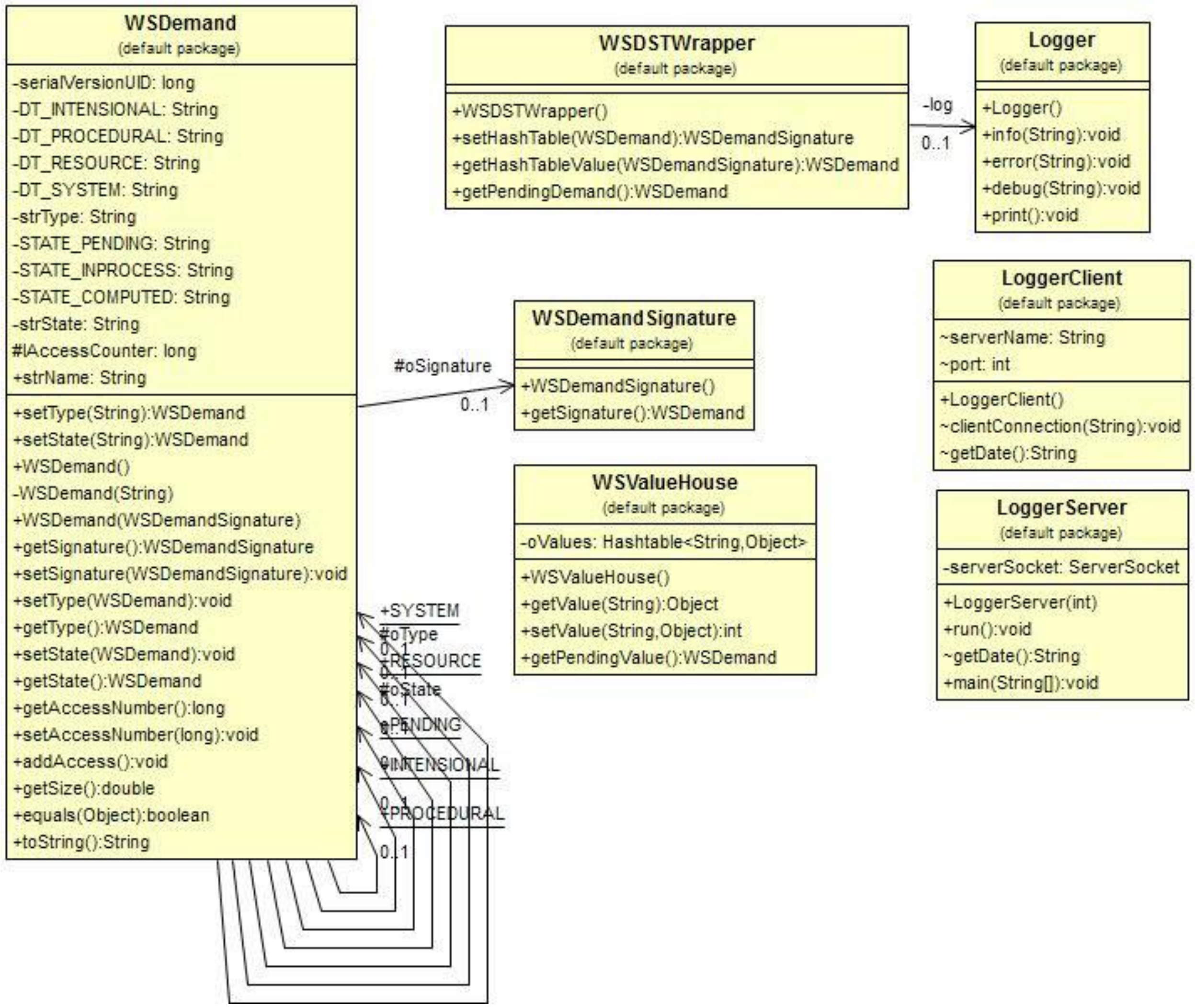


Figure 4.3.1 Class diagram for WS

5.4 Performance Evaluation

Performance evaluation is measured by running *RMIDSTWrapperTest.java* which is under the package *gipsy.tests.junit.GEE.multitier.DST.rmi* for 10 times with a sequence of 10, 100 and 1000 demands in a single machine and also in four different machines with identical configuration. The report thus summarizes the outcome of the RMI performance evaluation survey for the GIPSY1 project by analyzing the test result through Graph.

RMI Performance Test

5.4.1 Single Machine

Performance evaluation for a single machine is measured by running *RMIDSTWrapperTest.java* which is under the package *gipsy.tests.junit.GEE.multitier.DST.rmi* for 10 times with a sequence of 10, 100 and 1000 demands. The memory measured through the performance evaluation is the JVM memory usage. The output obtained while running in the Java version 8 and update 31 is shown as below:

System specification:

OS Name	Microsoft Windows 10 Home
Version	10.0.10240 Build 10240
OS Manufacturer	Microsoft Corporation
System Type	x64-based PC
Processor	IntelI CoreI i3-3227U CPU @ 1.90GHz, 1901 Mhz, 2 Core(s), 4 Logical Processor(s)
BIOS Version/Date	Dell Inc. A05, 1/3/2013
SMBIOS Version	2.7
Platform Role	Mobile
Installed Physical Memory (RAM)	4.00 GB
Total Physical Memory	3.87 GB
Available Physical Memory	620 MB
Total Virtual Memory	5.25 GB
Available Virtual Memory	1.57 GB
Page File Space	1.38 GB
Page File	C:\pagefile.sys

Figure 4.4.1.1: Single Machine-System Specification

No	Turnaround Time in Milliseconds			Used Memory in MB			Free Memory in MB			Total Memory in MB			Max Memory in MB		
	Demand			Demand			Demand			Demand			Demand		
	10	100	1000	10	100	1000	10	100	1000	10	100	1000	10	100	1000
1	4014	8714	35629	10	14	16	49	44	41	59	59	58	882	882	882
2	5372	7966	36355	10	14	16	49	44	41	59	59	58	882	882	882
3	3809	7857	36404	10	14	16	49	44	41	59	59	58	882	882	882
4	3461	7798	37827	10	14	12	49	44	47	59	59	60	882	882	882
5	4447	7787	35730	10	14	16	49	44	40	59	59	58	882	882	882
6	3594	8499	35730	10	14	17	49	44	40	59	59	58	882	882	882
7	3640	7798	36393	10	14	16	49	44	41	59	59	58	882	882	882
8	3539	8511	30195	10	14	16	49	44	40	59	59	57	882	882	882
9	3644	8052	37099	10	14	16	49	44	41	59	59	57	882	882	882
10	3602	8603	36861	10	14	17	49	44	40	59	59	58	882	882	882

Figure 4.4.1.2: Single Machine- Performance evaluation

5.4.2 Different Machine with identical Configuration

The Evaluation result of *RMIDSTWrapperTest.java* which is under the package *gipsy.tests.junit.GEE.multitier.DST.rmi* of four identically configured machines for 10 times with a sequence of 10, 100 and 1000 demands. The memory measured through the performance evaluation is the JVM memory usage. The output obtained while running in the Java version 8 and update 45 is shown as below:

System specification:

OS Name	Microsoft Windows 7 Enterprise
Version	6.1.7601 Service Pack 1 Build 7601
OS Manufacturer	Microsoft Corporation
System Type	x64-based PC
Processor	IntelI CoreI i7-4790 CPU @ 3.60GHz, 3591 Mhz, 4 Core(s), 4 Logical Processor(s)
BIOS Version/Date	Dell Inc. A15, 2/2/2015

SMBIOS Version	2.7
Installed Physical Memory (RAM)	8.00 GB
Total Physical Memory	7.94 GB
Available Physical Memory	4.66 GB
Total Virtual Memory	15.9 GB
Available Virtual Memory	12.0 GB
Page File Space	7.94 GB

Figure 4.4.2.1: Different Machine with Similar Configuration- System Specification

First Machine:

No	**Turnaround Time in Milliseconds**			**Used Memory in MB**			**Free Memory in MB**			**Total Memory in MB**			**Max Memory in MB**		
	Demands			**Demands**			**Demands**			**Demands**			**Demands**		
	10	**100**	**1000**	**10**	**100**	**1000**	**10**	**100**	**1000**	**10**	**100**	**1000**	**10**	**100**	**1000**
1	3267	3844	11995	18	15	17	104	107	127	123	123	145	1808	1808	1808
2	3301	3837	8353	18	15	12	104	107	124	123	123	137	1808	1808	1808
3	3254	3792	8390	18	15	9	104	107	127	123	123	137	1808	1808	1808
4	3264	3766	9251	18	15	7	104	107	135	123	123	143	1808	1808	1808
5	3327	3898	9409	18	15	7	104	107	129	123	123	137	1808	1808	1808
6	3327	3883	8914	18	15	9	104	107	155	123	123	164	1808	1808	1808
7	3273	3770	8761	18	15	14	104	107	128	123	123	143	1808	1808	1808
8	3267	3801	8812	18	15	14	104	107	141	123	123	155	1808	1808	1808
9	3278	3765	8384	18	15	8	104	107	136	123	123	144	1808	1808	1808
10	3259	3774	9328	18	15	9	104	107	126	123	123	137	1808	1808	1808

Figure 4.4.2.2: Different Machine with Similar Configuration- Performance evaluation of First Machine

Second Machine:

No	**Turnaround Time in Milliseconds**			**Used Memory in MB**			**Free Memory in MB**			**Total Memory in MB**			**Max Memory in MB**		
	Demands			**Demands**			**Demands**			**Demands**			**Demands**		
	10	**100**	**1000**	**10**	**100**	**1000**	**10**	**100**	**1000**	**10**	**100**	**1000**	**10**	**100**	**1000**
1	3382	3796	9295	10	8	7	112	114	129	123	123	137	1808	1808	1808
2	3258	4055	8401	10	8	22	112	114	143	123	123	166	1808	1808	1808

3	3260	3809	9526	10	8	24	112	114	128	123	123	153	1808	1808	1808
4	3255	3777	8372	10	8	21	112	114	131	123	123	152	1808	1808	1808
5	3274	3542	8026	10	8	4	112	114	143	123	123	148	1808	1808	1808
6	3262	3609	8575	10	8	28	112	114	136	123	123	164	1808	1808	1808
7	3258	3549	8285	10	8	14	112	114	129	123	123	144	1808	1808	1808
8	3263	3498	8543	10	8	8	112	114	136	123	123	144	1808	1808	1808
9	3254	3642	8496	10	8	16	112	114	128	123	123	145	1808	1808	1808
10	3268	3828	8308	10	8	20	112	114	140	123	123	144	1808	1808	1808

Figure 4.4.2.3: Different Machine with Similar Configuration- Performance evaluation of Second Machine

Third Machine:

No	**Turnaround Time in Milliseconds**			**Used Memory in MB**			**Free Memory in MB**			**Total Memory in MB**			**Max Memory in MB**		
	Demands			**Demands**			**Demands**			**Demands**			**Demands**		
	10	**100**	**1000**	**10**	**100**	**1000**	**10**	**100**	**1000**	**10**	**100**	**1000**	**10**	**100**	**1000**
1	3573	3921	8887	18	16	23	104	106	128	123	123	152	1808	1808	1808
2	3283	3794	8652	18	17	23	104	105	157	123	123	181	1808	1808	1808
3	3358	3846	8572	18	17	23	105	106	128	123	123	152	1808	1808	1808
4	3694	3952	8295	18	17	22	105	106	128	123	123	151	1808	1808	1808
5	3592	3744	8399	18	17	5	105	106	138	123	123	145	1808	1808	1808
6	3278	3894	9340	18	17	13	105	106	141	123	123	155	1808	1808	1808
7	3346	3964	8729	18	17	23	105	106	128	123	123	152	1808	1808	1808
8	3244	3872	8213	18	17	22	105	106	128	123	123	152	1808	1808	1808
9	3564	3859	8437	18	17	19	105	106	135	123	123	155	1808	1808	1808
10	3398	3941	8637	18	17	10	105	106	126	123	123	137	1808	1808	1808

Figure 4.4.2.4: Different Machine with Similar Configuration- Performance evaluation of Third Machine

Fourth Machine:

No	Turnaround Time in Milliseconds			Used Memory in MB			Free Memory in MB			Total Memory in MB			Max Memory in MB		
	Demands			Demands			Demands			Demands			Demands		
	10	**100**	**1000**	**10**	**100**	**1000**	**10**	**100**	**1000**	**10**	**100**	**1000**	**10**	**100**	**1000**
1	3470	3851	9048	10	8	4	112	114	135	123	123	140	1808	1808	1808
2	3358	3765	9496	10	8	26	112	114	126	123	123	153	1808	1808	1808
3	3496	3986	9024	10	8	15	112	114	127	123	123	143	1808	1808	1808
4	3523	3832	8949	10	8	4	112	114	144	123	123	148	1808	1808	1808
5	3422	3950	9495	10	8	5	112	114	139	123	123	145	1808	1808	1808
6	3555	3789	8433	10	8	15	112	114	123	123	123	138	1808	1808	1808
7	3649	3895	9356	10	8	15	112	114	127	123	123	143	1808	1808	1808
8	3455	3955	9011	10	8	20	112	114	124	123	123	145	1808	1808	1808
9	3254	3864	8137	10	8	8	112	114	128	123	123	137	1808	1808	1808
10	3647	3923	8421	10	8	28	112	114	136	123	123	164	1808	1808	1808

Figure 4.4.2.5: Different Machine with Similar Configuration- Performance evaluation of Fourth Machine

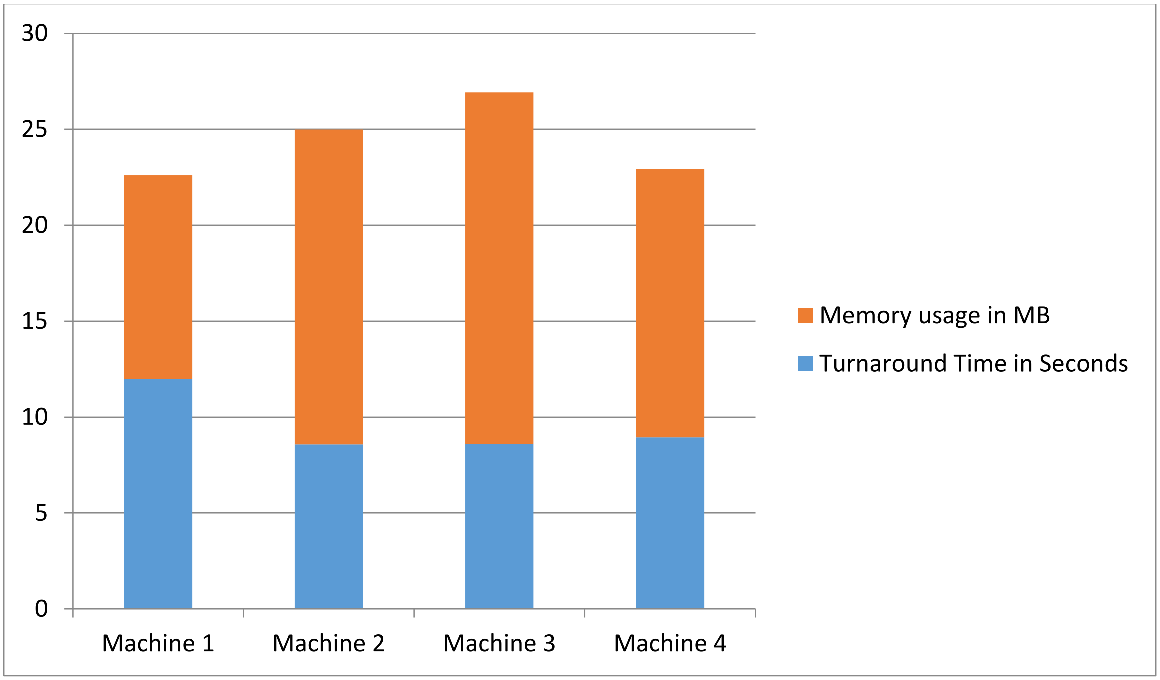


Figure 4.4.2.6: Comparison chart of Time and Memory used in four machines for 1000 demands

The Graph analyses the Average Turnaround time (in seconds) and Average Memory usage (in Mega Byte) of JAVA Virtual Machine calculated for four different machines with identical configuration.

6. Logger

6.1 Overview

Logging is the process of writing log messages during the execution of a program to a central place. Java contains the Java Logging API. This logging API allows you to configure which message types are written. Individual classes can use this logger to write messages to the configured log files. The java.util.logging package provides the logging capabilities via the Logger class. This feature can be achieved by including jar file from http://logging.apache.org/log4j/2.x/.

Thus logging allows you to report and persist error and warning messages as well as info messages (e.g., runtime statistics) so that the messages can later be retrieved and analyzed. The object which performs the logging in applications is typically just called Logger. [3]

6.2 Design and Implementation

The various Logger levels used in our project are 1. Debug – Fine grained informational events that are most useful to debug an application. 2. Info – informational messages that highlight the progress of the application at coarse-grained level. 3. Warn – potentially harmful situations. 4. Error – error events that might still allow the application to continue running. 5. Fatal – very severe error events that will presumably lead the application to abort. [7]

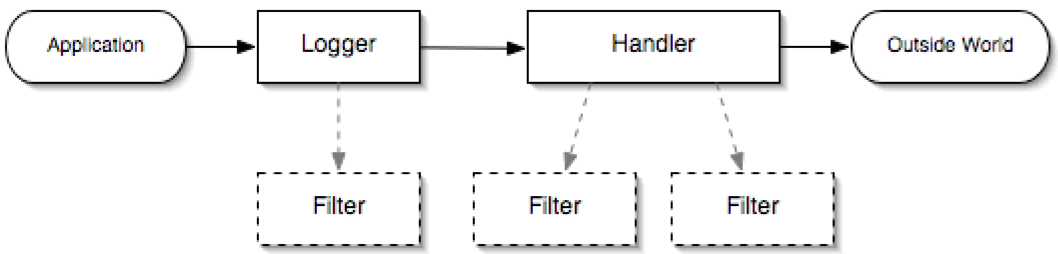


Figure 5.2.1 Design Layout of Logger Mechanism

- Logger: The class that has methods such as info, debug, error to make log entries.
- Logger Client: The client class that request for log entry to server through SOCKET and port.
- Logger Server: The server class that responds to client request and stores logger details.

7. UTILITIES

7.1 IDE

We used Luna Version of Eclipse IDE (Integrated Development Environment) which is an open source community, it provides cross platform software tool that enabled us to extend additional plug-ins and provided all developing needs.

7.2 Repository

We used Bit Bucket repository which is a web-based for hosting project online that allowed us to work together in a team and avoid conflicts of multiple editing. It also made us to work on the project consistently as well as concurrently.
GIT client: We used Source Tree GIT client that allows us to store the project locally on our machine.

7.3 Language

The Project is implemented in JAVA Language which is concurrent, class-object and Object Oriented. We used (JAVA Development Kit) JDK 1.8 version of Java.
http://www.vogella.com/tutorials/Logging/article.html

8. How to run?

Here are few steps to run RMI/CORBA/WS.

8.1 RMI

Step-1 Reach to source folder on console. For example in our case G:/workspace/gipsy/src

Run -> gispyrmi.bat

This command compiles all classes.

Step-2 Now, run -> rmic_gipsy.bat

This creates stub for RMI classes.

Step-3 finally, run -> gipsyrmitest.bat

This runs the junit test case which is been created to test RMI implementation.

8.2 CORBA

Step-1 Reach to source folder on console. For example in our case G:/workspace/gipsy/src

From command prompt, run -> idlj –fall gipsy / GEE / multitier / DST / corba /GIPSY.idl

You have set of .java classes for future

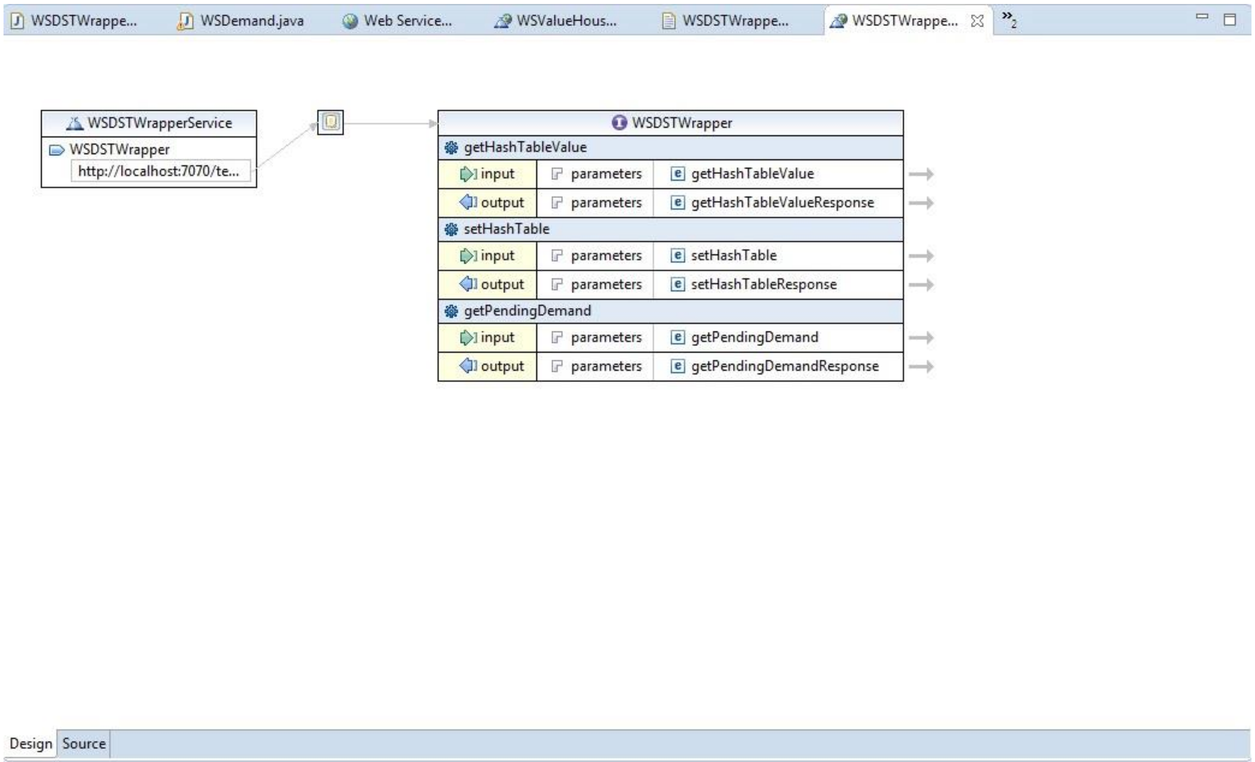
Step-2 Run CORBATest.java in eclipse to check CORBA implementation.

8.3 WS

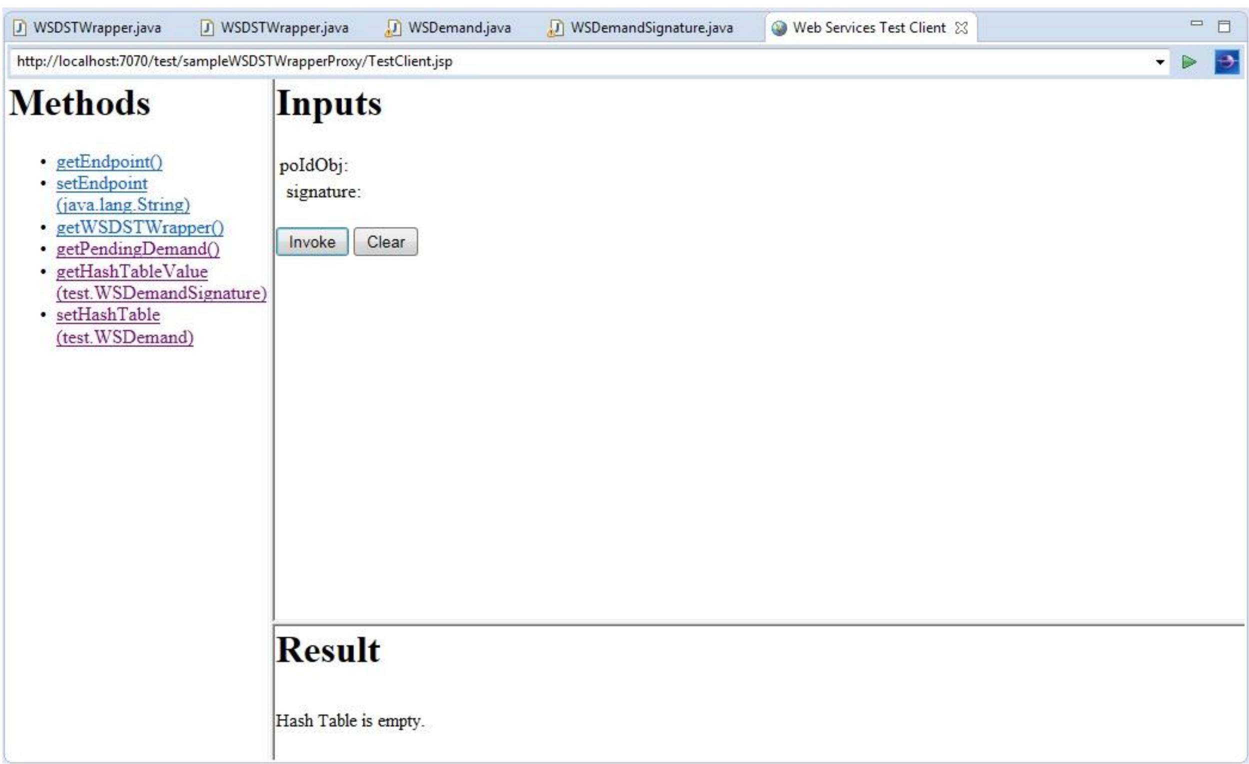
Step-1 Start Tomcate v7 server

Step-2 Run -> TestClient.jsp on server

Below is the screen shots from web services.



8.3.1 WSDL



8.3.2 getHashTableValue

8.3.3 setHashTable

9. Limitations

- There is only one registry in RMI that is in RMIDSTWrapper class which starts when DST tier starts.
- Tested only on single machine.
- CORBA is less scalable compared to Web Services.
- For web services, the conversion to and from XML during the communication process takes too much time.
- Web service does not maintain the connection throughout the application life cycle.
- Performance evaluation can be extended for other two tiers namely CORBA and WS for GIPSY1 with multiple workers and their results can be compared and analyzed using graphs, as future work.
- Local call on Multi-threaded components in the multi-tier framework can be made as a future work.
- No GIPC service provided.
- Web service is limited to SOAP , it can be applied to REST services as a future work.

10. Acknowledgement

We wish to express our profound gratitude to Prof. Serguei A. Mokhov, the instructor of the course COMP6231 for his excellent guidance and providing necessary information regarding different reports and useful papers to make the project more precise and accurate.

We appreciate his efforts for giving us such attention and valuable time to solve our doubts and queries and leading us to the right path in completing our project of analyzing different technologies for GIPSY architecture.

We appreciate the lab facilities provided by the Concordia University which really helped in completing the project providing necessary environment.

11. REFERENCES

[1] Yi Ji. “ Scalability evaluation of the GIPSY runtime system. Master’s thesis”, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, March 2011. http://spectrum.library.concordia.ca/7152/.

[2] Agrawal," N. *Towards refactoring of DMARF and GIPSY Case Studies".*

[3]"*Java Logging API - Tutorial*. "(2015, 05 15). Retrieved from www.vogella.com: http://www.vogella.com/tutorials/Logging/article.html

[4] Sleiman Rabah, Serguei A. Mokhov, and Joey Paquet. "An interactive graph-based automation assistant: A case study to manage the GIPSY’s distributed multi-tier run-time system". In Ching Y. Suen, Amir Aghdam, Minyi Guo, Jiman Hong, and Esmaeil Nadimi, editors, Proceedings of the ACM Research in Adaptive and Convergent Systems (RACS 2013), pages 387–394, New York, NY, USA, October 2011–2013. ACM. ISBN 978-1-4503-2348-2. doi: 10.1145/2513228.2513286. Pre-print: http://arxiv.org/abs/1212.4123.

[5]Yi Ji, Serguei A. Mokhov, and Joey Paquet. "Unifying and refactoring DMF to support concurrent Jini and JMS DMS in GIPSY". In Bipin C. Desai, Sudhir P. Mudur, and Emil I. Vassev, editors, Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering (C3S2E’12), pages 36–44, New York, NY, USA, June 2010–2013. ACM. ISBN 978-1-4503-1084-0. doi: 10.1145/2347583.2347588. Online eprint http://arxiv.org/abs/1012.2860.

[6]Serguei A. Mokhov. "On design and implementation of a heterogeneous web services, CORBA, RMI, and TCP/IP-based distributed stock broker system". Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, August 2006.

[7] "Logging in Java applications using Log4j", http://www.cosenonjaviste.it/logging-in-java-applications-using-log4j/

[8] “Remote Method Invocation tutorial”, https://www.youtube.com/watch?v=mdqR7oy8N-s

[9] “Remote Method Invocation API”, http://docs.oracle.com/javase/7/docs/api/java/rmi/package-summary.html

[10] CORBA Technology and the Java™ Platform Standard Edition, “https://docs.oracle.com/javase/7/docs/technotes/guides/idl/corba.html”

[11] SOAP Web Service Tutorial, “http://www.java2blog.com/2013/03/soap-web-service-tutorial.html”.

[12]” SOAP Web Services 10 - Understanding the WSDL”, https://www.youtube.com/watch?v=E76xW1JTVXY