Building Reliable Web Services: Methodology, Composition, Modeling and Experiment

Pat. P. W. Chan

Supervised by Michael R. Lyu

Department of Computer Science and Engineering
The Chinese University of Hong Kong

29 November 2007

Outline

- Introduction
  - Contribution
  - Related Work
  - Web Services
- Problem Statement
- Methodologies for Web Service Reliability
- New Reliable Web Service Paradigm
  - Optimal Parameters
  - Experimental Results
- Web Service Composition Algorithm
  - Experimental Results
  - Discussion
- Modeling of the Paradigm
- Conclusion and Future Work

Introduction

- Service-oriented computing is becoming a reality.
- Web Service is a promoting technique in the internet.
- The benefit of interoperability, reusability, and adaptability.
- The problems of service dependability, security and timeliness are becoming critical.
- Reliability is an important issue.
- Existing web service model needs to be extended to assure reliability.
- We propose experimental settings and offer a roadmap to dependable Web services.

Contribution

- Surveyed on reliability methodologies
- Surveyed on Web services reliability and Web service composition techniques
- Proposed an architecture for dependable Web services
- Proposed an algorithm for Web services composition
- Developed reliability models for the proposed scheme
- Performed experiments for evaluating the reliability of the system and the correctness of the algorithm

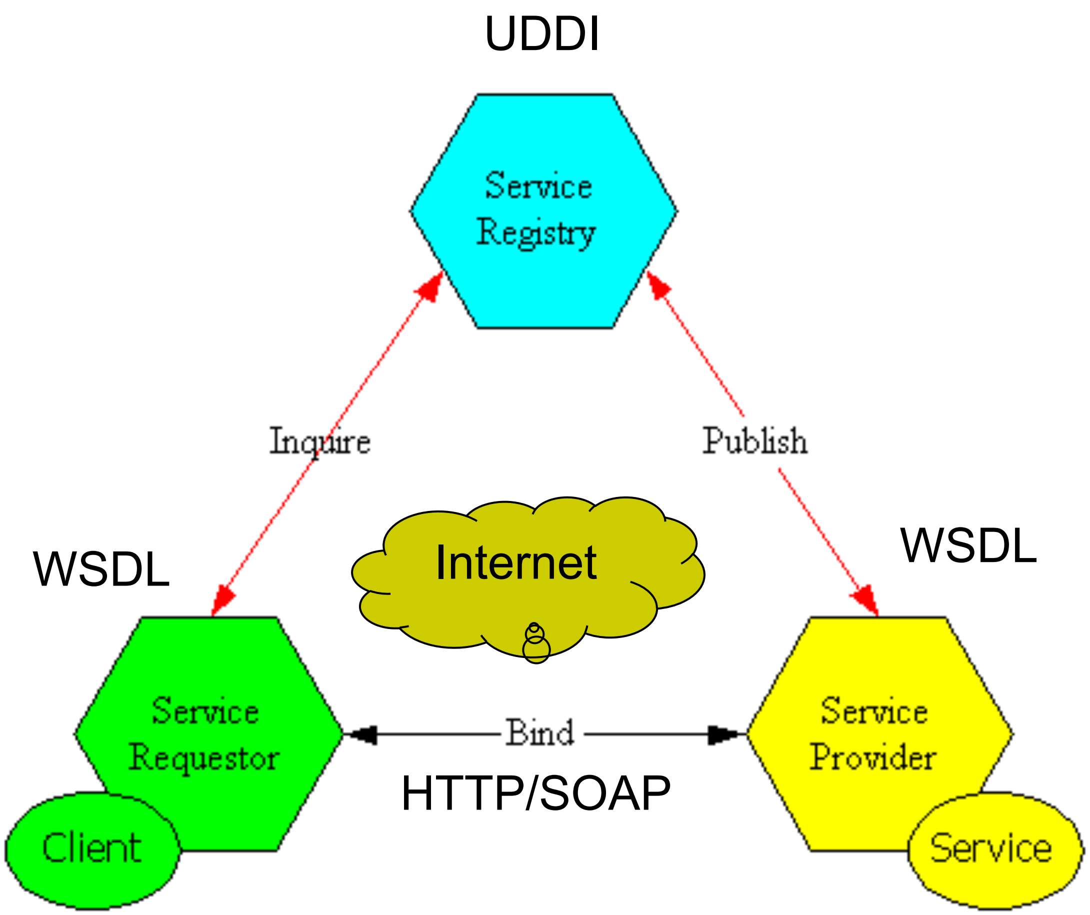
Reliability

- "a measure of the success with which the system conforms to some authoritative specification"
  - Guaranteed delivery
  - Duplicate elimination
  - Ordering
  - Crash tolerance
  - State synchronization

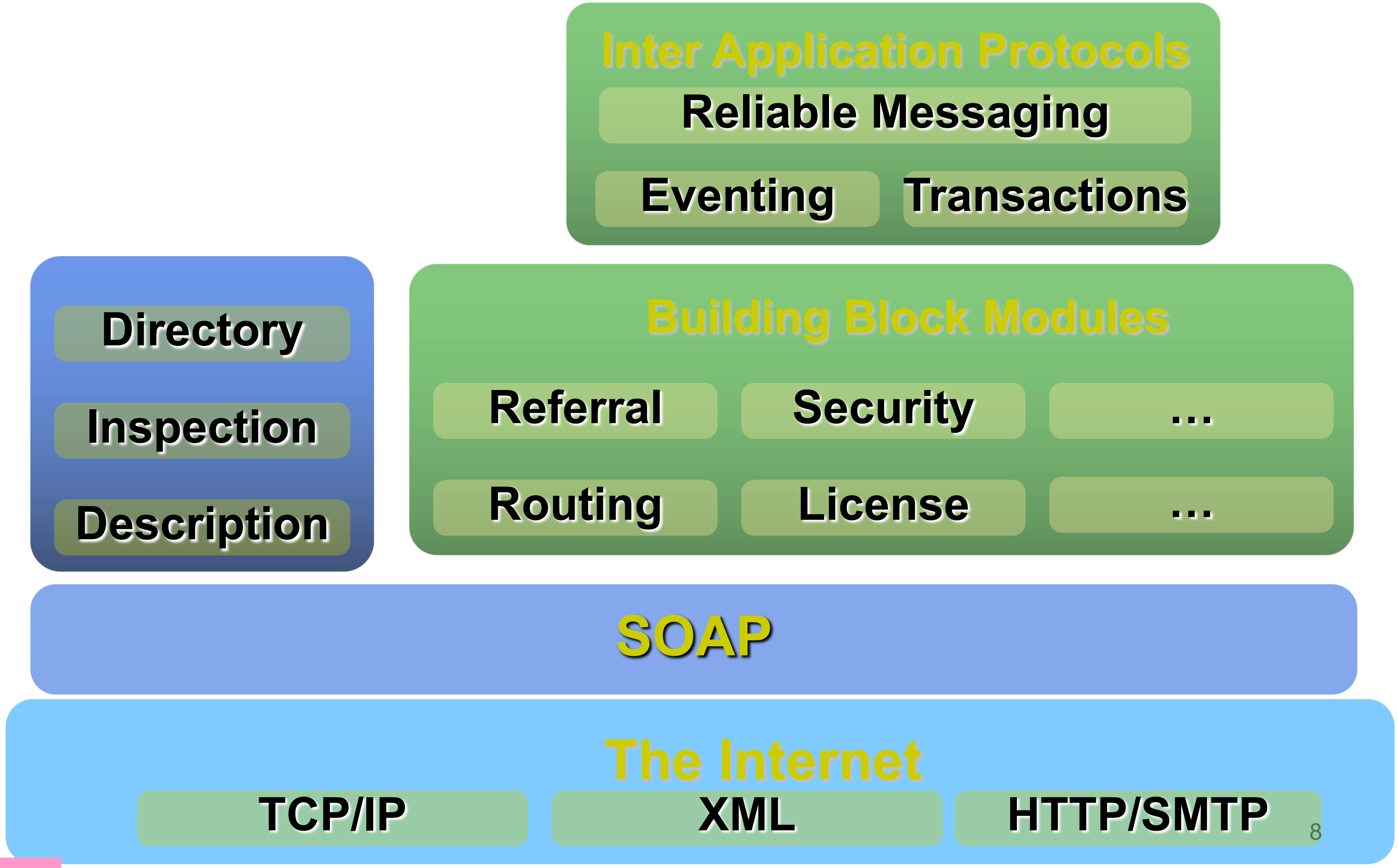
What are Web Services ?

- Self-contained, modular applications built on deployed network infrastructure including XML and HTTP
- Use open standards for description (WSDL), discovery (UDDI) and invocation (SOAP)

Web Services



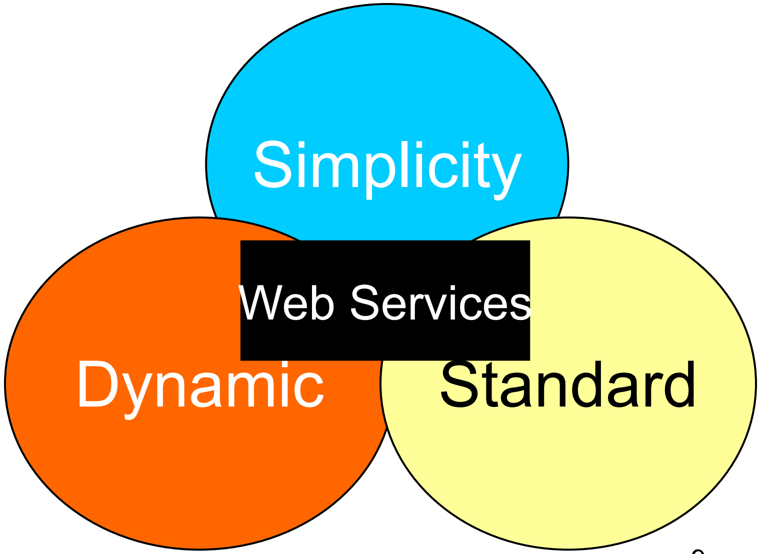
Web Services Architecture



Web Services

- Benefits of WS
  - Service-oriented
  - Highly accessible
  - Open specification
  - Easy integration
- Number of system using Web service including: shopping, e-banking…

Build common infrastructure reducing the barriers of business integration with lower costs and faster speed.



Problems of Web Services

- Transaction
  - Atomicity is not provided
- Security
  - Insecure Internet transportation
- **Reliability**
  - The internet is inherently unreliable
  - No single underlying “transport protocols” address all the reliability issues.

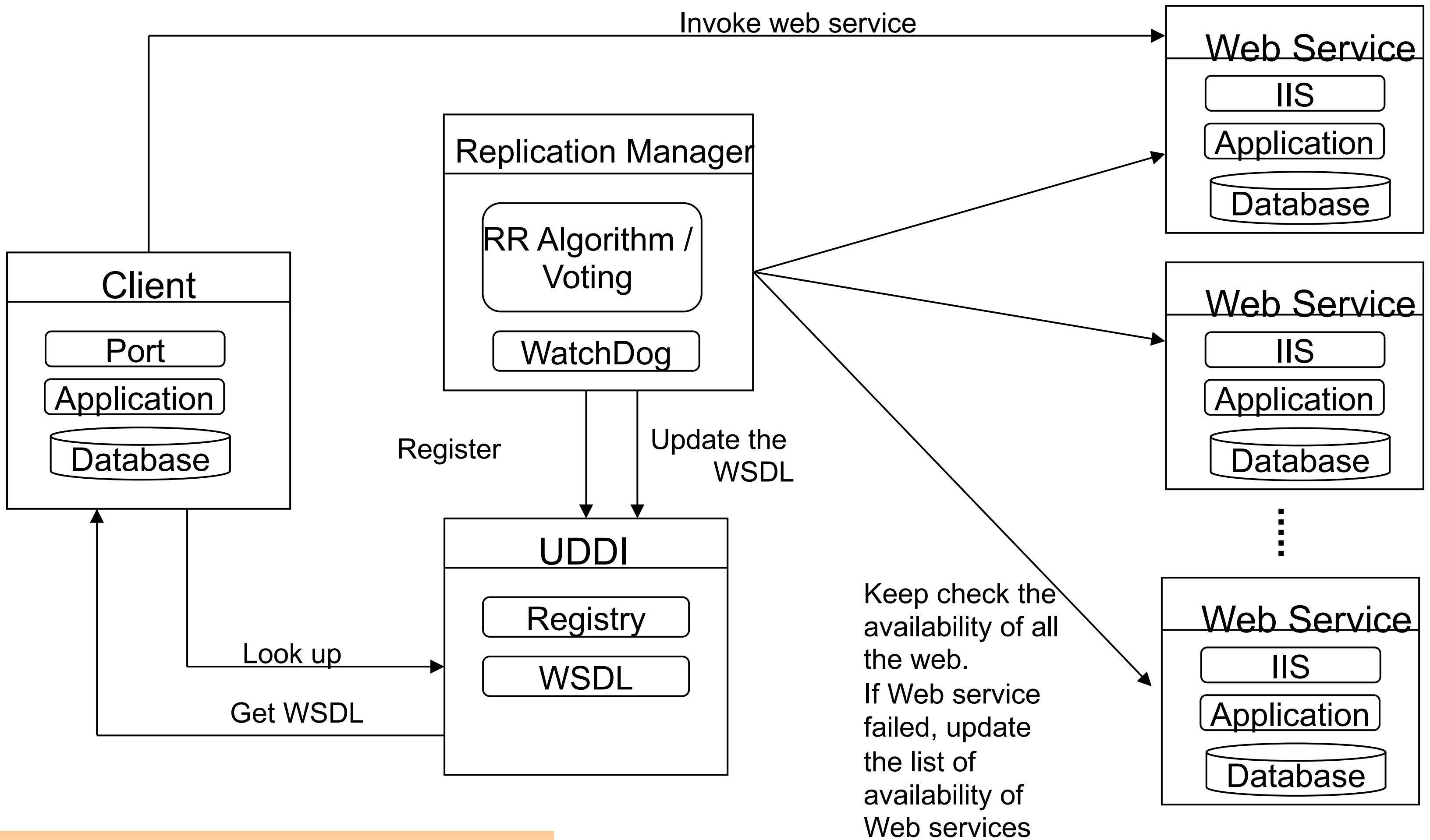
Problem Statement

- Fault-tolerant techniques
  - Replication
  - Diversity
- Replication is one of the efficient ways for providing reliable systems by time or space redundancy.
  - Increasing the availability of distributed systems
  - Key components are re-executed or replicated
  - Protect against hardware malfunctions or transient system faults
- Another efficient technique is design diversity
  - Employ independently designed software systems or services with different programming teams,
  - Defend against permanent software design faults.
- We focus on the analysis of the replication techniques when applied to Web services.
- A generic Web service system with spatial as well as temporal replication is proposed and investigated.

Road Map for Research

- Redundancy in time
  - Retry
  - Reboot
- Redundancy in space
  - Sequentially
  - Parallel
  - Majority voting using N modular redundancy
  - Diversified version of different services

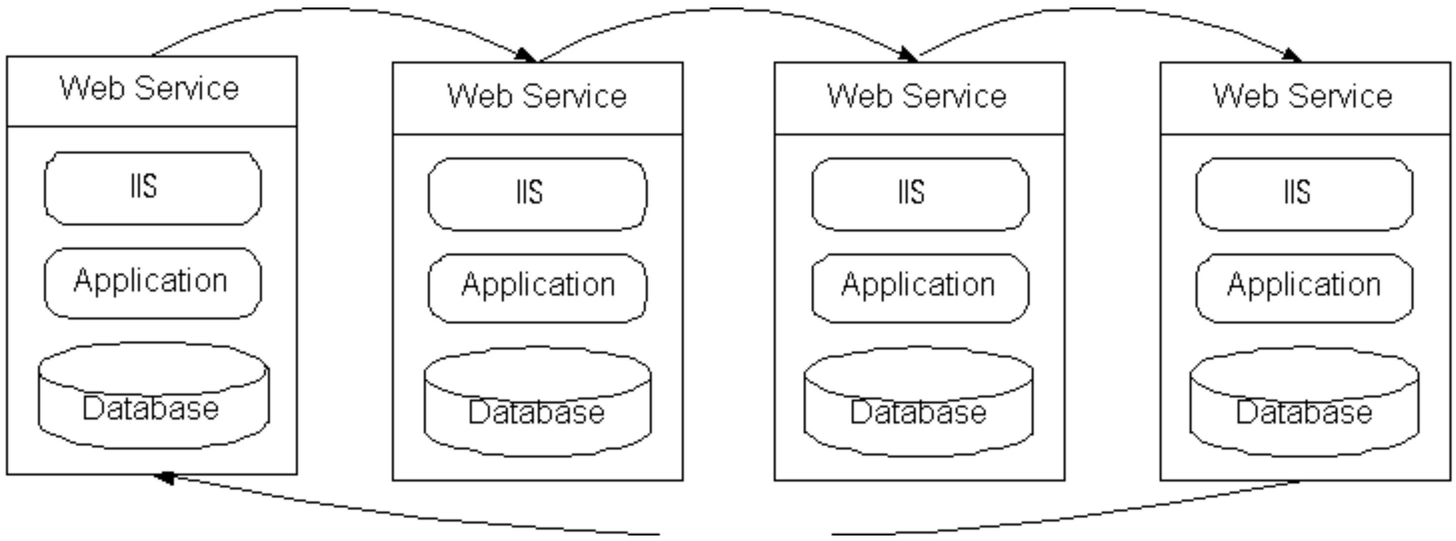
Proposed Paradigm



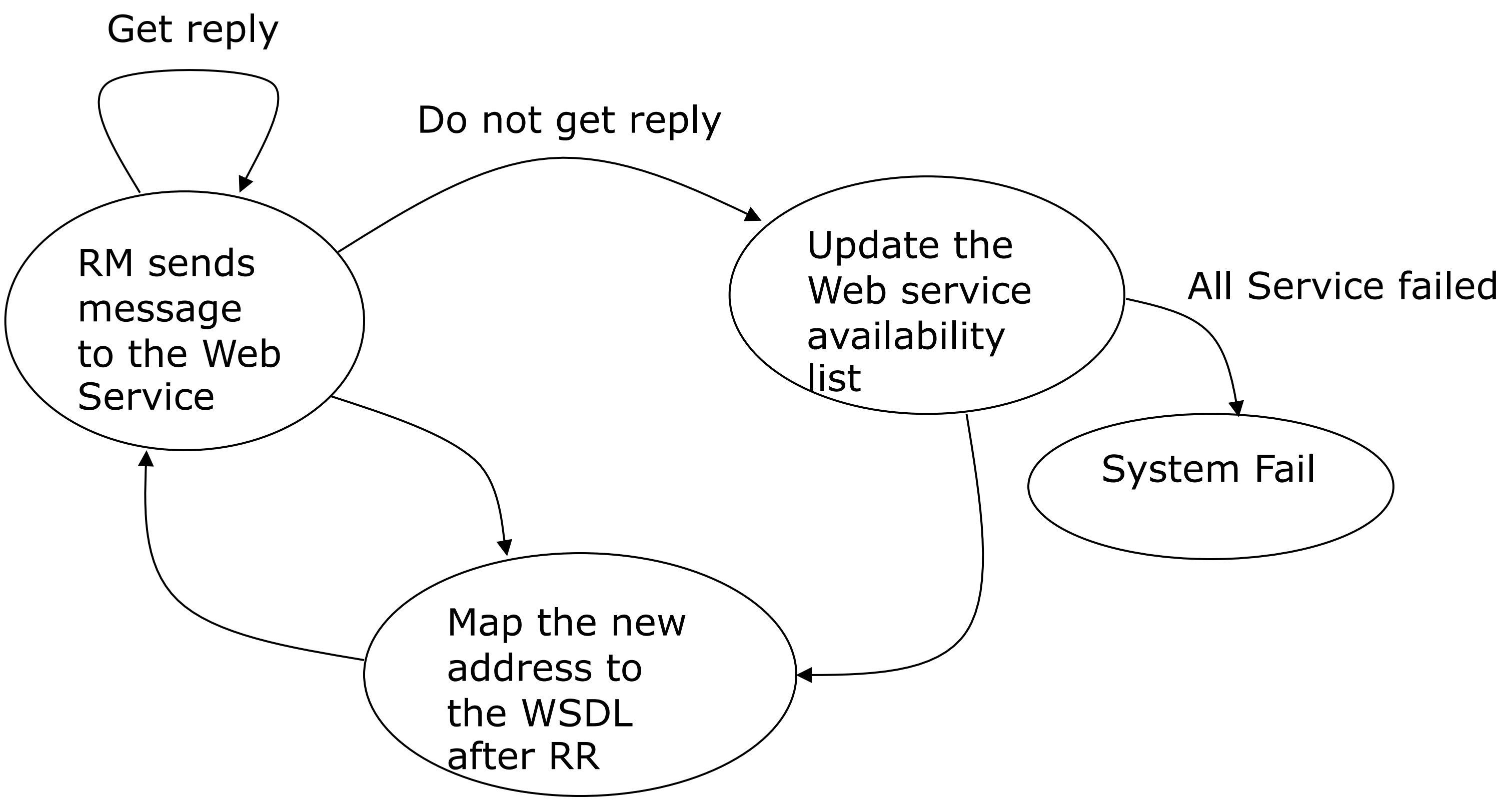
Different Approaches

- Replication
  - Round-robin scheduling algorithm
- Design Diversity
  - N-version programming
  - Recovery block

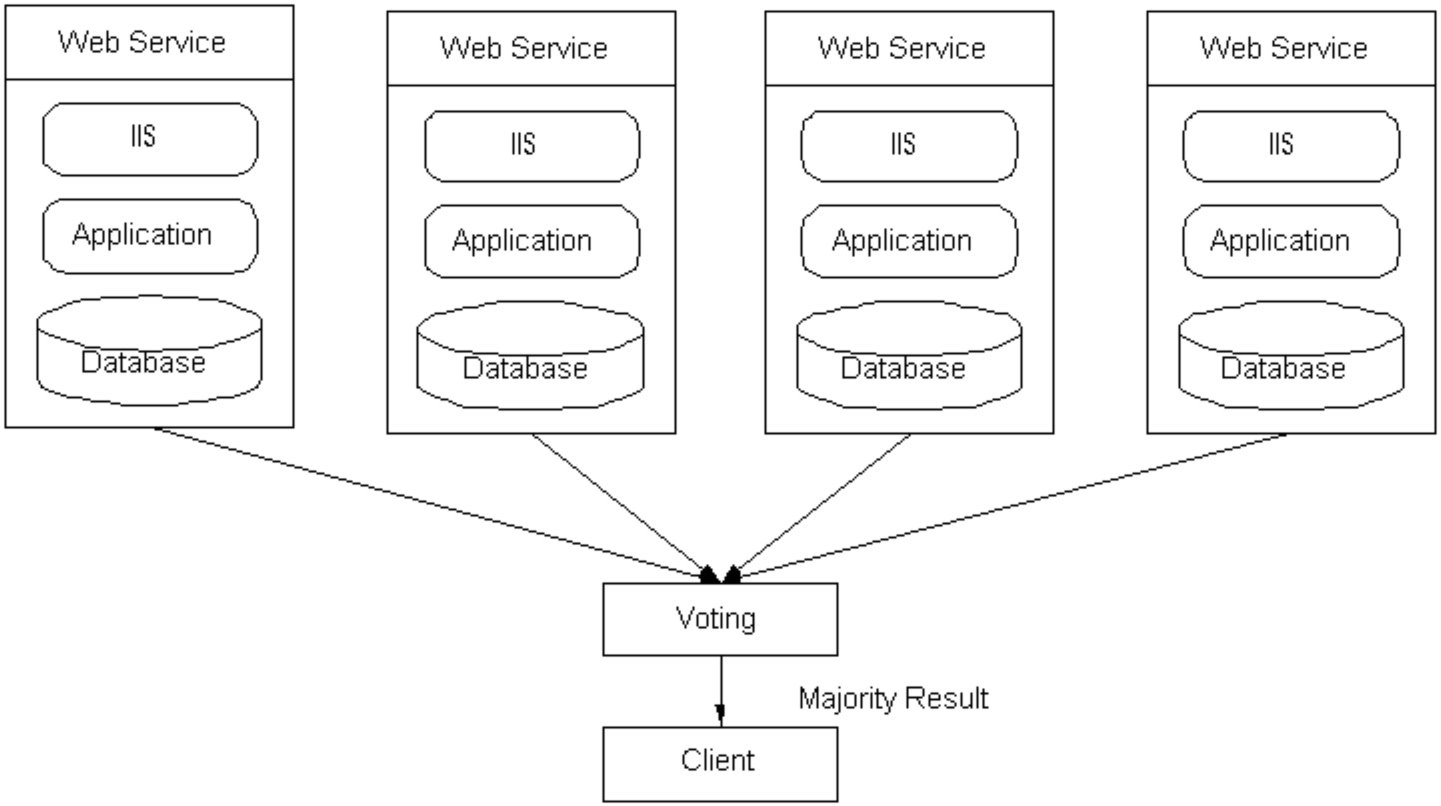
Replication: Round-robin



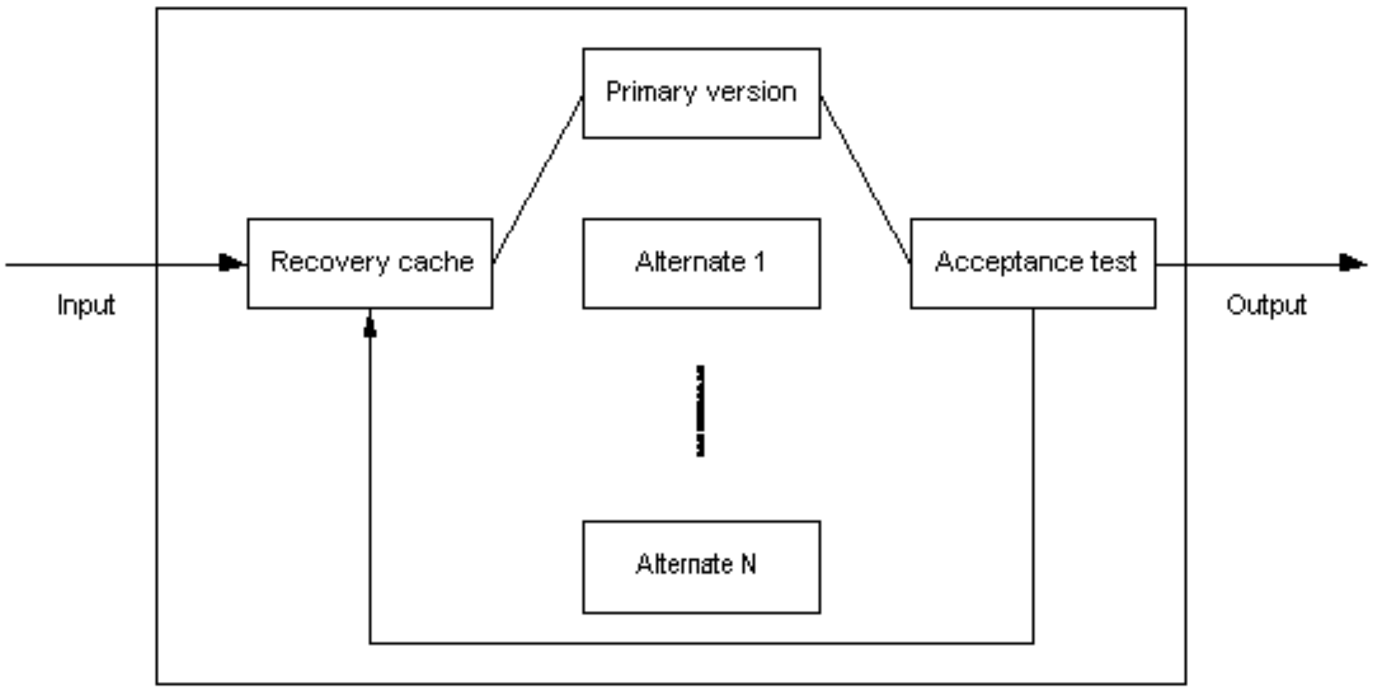
Work Flow of the Replication Manager



Design Diversity: Parallel N-Version Programming



Design Diversity: Recovery Block



Experiments Variations

- A series of experiments are designed and performed for evaluating the reliability of the Web service.

	1	2	3	4	5	6	7	8
Spatial replication	0	0	0	0	1	1	1	1
Reboot	0	0	1	1	0	0	1	1
Retry	0	1	0	1	0	1	0	1

Varying the parameters

- Number of tries
- Timeout period for retry in single server
- Timeout period for retry in our paradigm
- Polling frequency
- Number of replicas
- Load of server

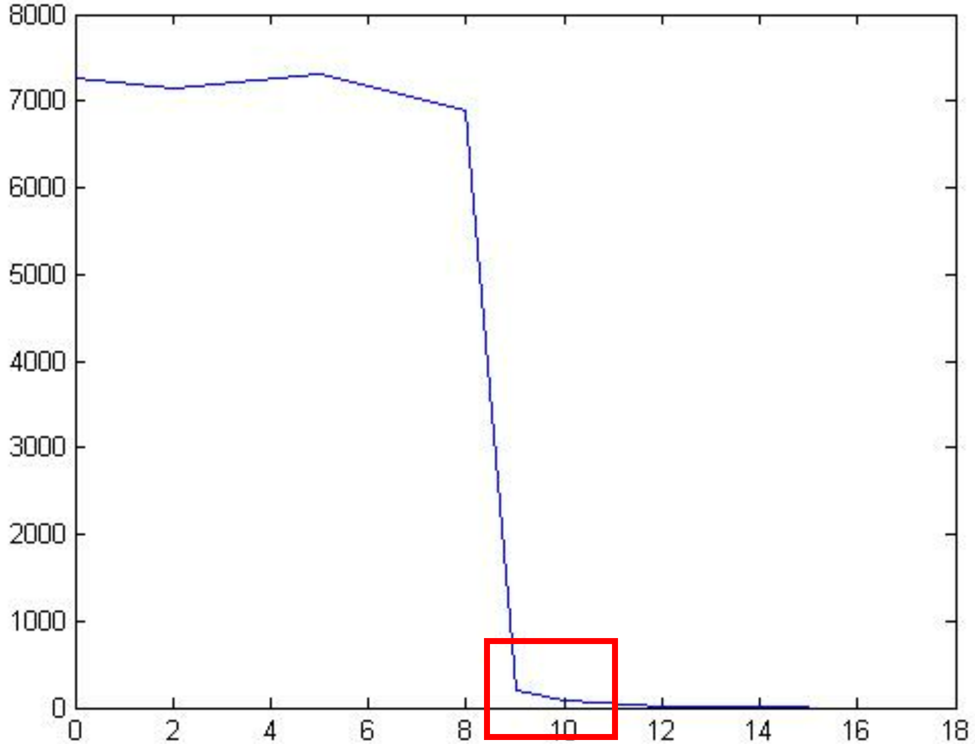
Number of tries

Number of tries	Number of failures in Temp	Number of failures in Perm
0	95	76
1	2	2
2	0	0
3	0	0
4	0	0
5	0	0

Timeout period for retry in single server

Timeout period for retry (s)	Number of failures in Temp	Number of failures in Perm
0	95	7265
2	2	7156
5	0	7314
6	0	6890
7	0	189
8	0	82
9	0	11
10	0	2
12	0	0
14	0	0
16	0	0
[illegible]	0	0

Timeout period for retry in single server



# of failure

Timeout period

Timeout period for retry in our paradigm

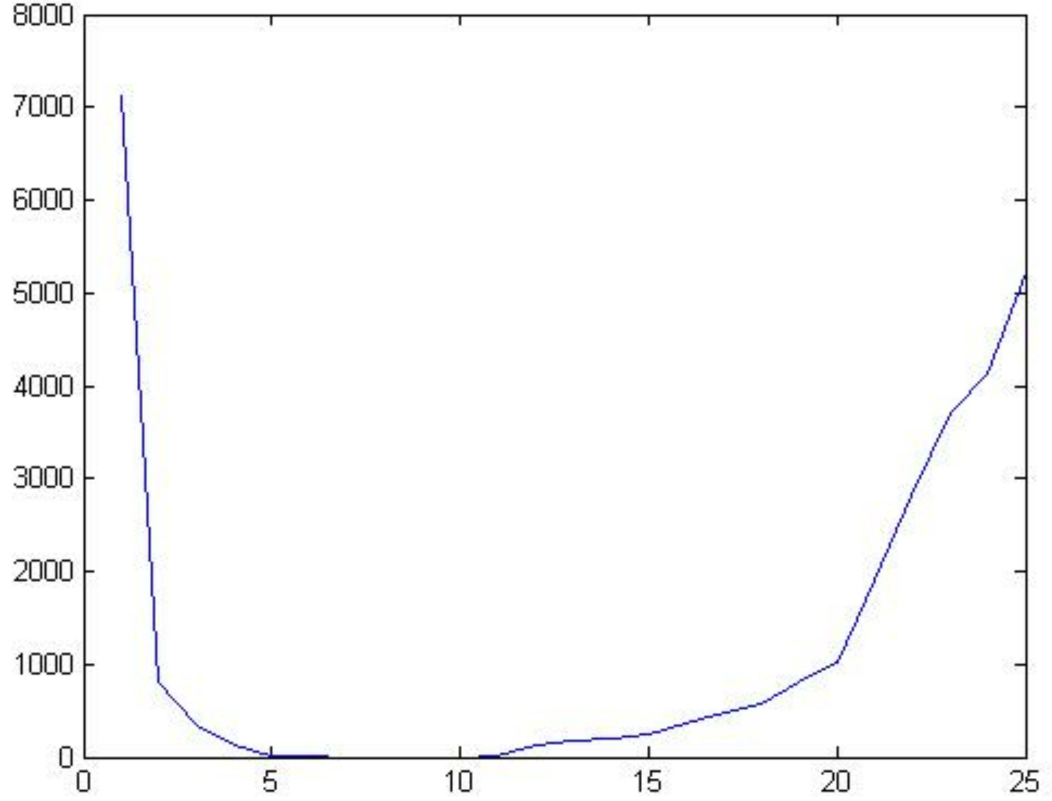
Timeout period for retry (s)	Number of failures in Temp	Number of failures in Perm
0	2	81
2	0	2
5	0	0
10	0	0
20	0	0

Polling frequency

Polling frequency (number of requests per min)	Number of failures in Temp	Number of failures in Perm
0	0	7124
1	0	811
2	0	30
5	0	12
10	0	1
15	213	254
20	1124	1023

Polling frequency

of failure



Polling frequency

Number of Replicas

Number of replicas	Number of failures in Temp	Number of failures in Perm
No replica	91	8152
2	2	356
3	0	0
4	0	0

Load of Web Server

Load of the web server (%)	Number of failures in Temp	Number of failures in Perm
70	0	0
75	0	0
80	2	3
85	10	14
90	512	528
95	3214	3125
98	8792	8845
99	8997	8994

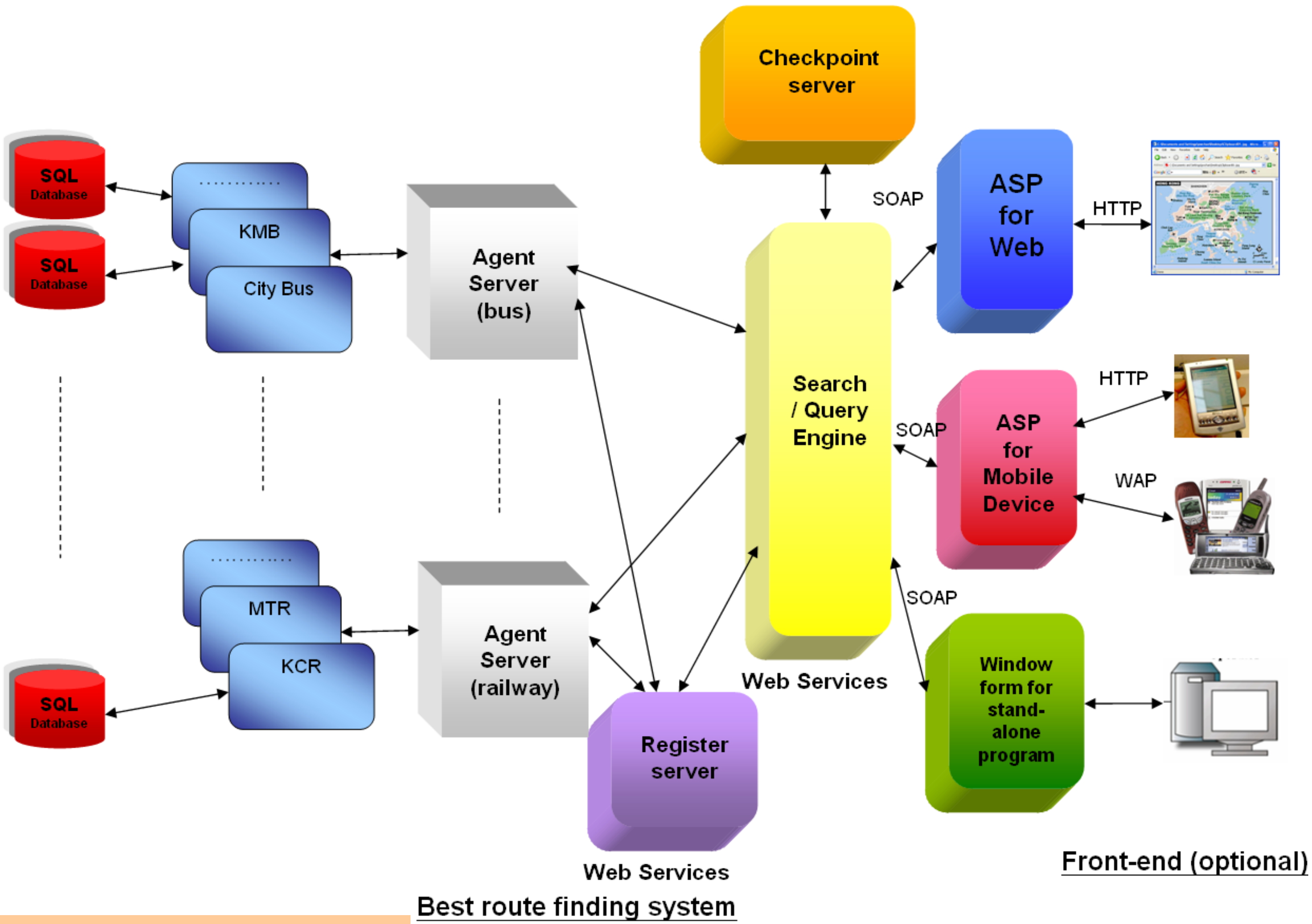
Summary of Parameters

- Number of tries = 2
- Timeout period for retry in single server = 10s
- Timeout period for retry in our paradigm = 5s
- Polling frequency = 10 request per min
- Number of replicas = 3
- Load of server < 75%

Testing system

- Best Route Finding.
- Provide traveling suggestions for users.
- Starting point and destination.
- The system needs to provide the best route and the price for the users.

System Architecture



Experimental Setup

- Examine the computation to communication ratio
- Examine the request frequency to limit the load of the server to 75%
- Fix the following parameters
  - Computation to communication ratio (e.g 10:1)
  - Request frequency

Experimental Setup

Communication time: Computation time	143:14 (10:1)
Request frequency	1 request per min
Load	78.5%

Timeout period of retry	1 min
Timeout for Web service in RM	1s (web service specific)
Polling frequency	10 requests per min
Number of replicas	5
Max number of retries	5
Round-robin rate	1 s

Experiment Parameters

- Fault mode
  - Temporary (fault probability: 0.01)
  - Permanent (fault probability: 0.001)
- Experiment time 5 days (7200 requests)
- Measure:
  - Number of failures
  - Average response time (ms)
- Failure definition:
  - 5 retries are allowed. If there is still no correct result from the Web service after 5 retries, it is considered as a failure.

Experimental Result with Round-robin

(failures / response time in ms)

	1	2	3	4	5	6	7	8
Experiments	Single server	Single server with retry	Single server with reboot (continues no response for 3 requests)	Single server with retry and reboot	Spatial Replication RR	Hybrid approach RR+Retry	Hybrid approach RR+ Reboot	All round approach RR spatial + Retry (5 times) + Reboots
Normal case	0 / 183	0 / 193	0 / 190	0 / 187	0 / 188	0 / 195	0 / 193	0 / 190
Temp	705 / 190	0 / 223	723 / 231	0 / 238	711 / 187	0 / 233	726 / 188	0 / 231
Perm	6144 / --	6337 / --	1064 / --	5 / 2578	5637 / --	5532 / --	152 / 187	0 / 191

Experimental Result with N-Version

(failures / response time in ms)

	1	2	3	4	5	6	7	8
Experiments	Single server	Single server with retry	Single server with reboot (continues no response for 3 requests)	Single server with retry and reboot	Spatial Replication Voting	Hybrid approach Voting+ Retry	Hybrid approach Voting + Reboot	All round approach Voting spatial + Retry (5 times) + reboots
Normal case	0 / 183	0 / 193	0 / 190	0 / 187	0 / 189	0 / 190	0 / 188	0 / 188
Temp	705 / 190	0 / 223	723 / 231	0 / 238	0 / 190	0 / 190	0 / 189	0 / 187
Perm	6144 / --	6337 / --	1064 / --	5 / 2578	3125 / 191	3418 / 192	40 / 189	0 / 188

Experimental Result with Recovery Block (failures / response time in ms)

	1	2	3	4	5	6	7	8
Experiments	Single server	Single server with retry	Single server with reboot (continues no response for 3 requests)	Single server with retry and reboot	Spatial Replication Voting	Hybrid approach Voting+ Retry	Hybrid approach rollback + Reboot	All round approach rollback spatial + Retry (5 times) + reboots
Normal case	0 / 183	0 / 193	0 / 190	0 / 187	0 / 191	0 / 189	0 / 193	0 / 188
Temp	705 / 190	0 / 223	723 / 231	0 / 238	0 / 205	0 / 203	0 / 204	0 / 201
Perm	6144 / --	6337 / --	1064 / --	5 / 2578	3478 / 215	3245 / 208	201 / 211	0 / 201

Summary of the proposed paradigm

- Temporal replication improves the reliability.
- Spatial replication further improves the reliability of Web services.
- N-version programming approach is the most reliable and efficient.

Web Service Composition Algorithm

- N-version programming
  - Reliable
  - Efficient
- Composition
  - WSDL – Web Services Description Language
  - WSCI – Web Services Choreography Interface
- Verification
  - BPEL – Business Process Execution Language
  - Petri-Net

WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
…
<portType name="BRF">
  <operation name="shortestpath">
    <input message="tns:startpointDestination"/>
    <output message="tns:pathArray"/>
  </operation>

  <operation name="addCheckpoint">
    <input message="tns:pathArray"/>
    <output message="tns:addAcknowledgement"/>
  </operation>
…
</operation>
</portType>
```

WSCI

```
<correlation name=“pathCorrelation”
      property=“tns:pathID”></correlation>
<interface name=“busAgent”>
  <process instantiation="message">
    <sequence>
      <action name="ReceiveStartpointDest“ role="tns:busAgent“
        operation="tns:BRF/shortestpath">
       </action>
       <action name="Receivecheckpoint“ role=" tns:busAgent“
        operation="tns:BRF/addCheckpoint">
        <correlate correlation=“tns: pathCorrelation”/>
        <call process=“tns:SearchPath”/>
       </action>
</sequence>
</process>
```

Algorithm 1 Algorithm for Web service composition

Require: $I[n]$: required input, $O[n]$: required output

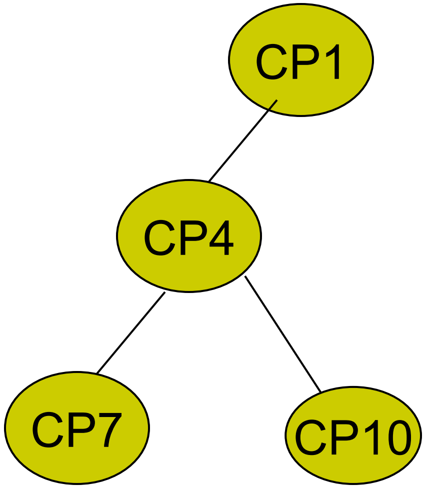
1: CP_n: the n^{th} Web services component
2: **for all** $O[i]$ **do**
3:     Search the WSDL of the Web services, and find the CP_n 's operation output = $O[i]$. Then, insert CP_n into the tree.
4:     **if** the input of the operation = $I[j]$ **then**
5:         Insert the input to the tree as the child of CP_n.
6:     **else**
7:         Search the WSCI of CP_n, WSCI.process.action = operation.
8:         Find the previous action needing to be invoked.
9:         Search the operation in WSDL equal to the action.
10:         **if** input of the operation = $I[i]$ **then**
11:             Insert input to the tree as the child of CP_n
12:         **else**
13:             go to step (8)
14:         **end if**
15:     **end if**
16:     until reaching the root of WSCI and not finding the correct input, search other WSDL with output = $I[j]$, insert CP_m as the child of CP_n and go to step (7) to do the searching in WSCI of CP_m.
17: **end for**

Web service composition

1. Output
2. ***Operation*** in **WSDL**
3. Find the ***output*** information in *CP1* (Web service component)
4. If ***Input*** of the operation == required input

5. Else
   search in the **WSCI** of CP1 to find ***action*** == ***operation***
6. Get the pervious ***action*** involved
7. Search in **WSDL** to find ***operation*** == ***action***
8. If ***Input*** of the operation == required input

Else, till the root of **WSCI**



Web Service Composed Tree

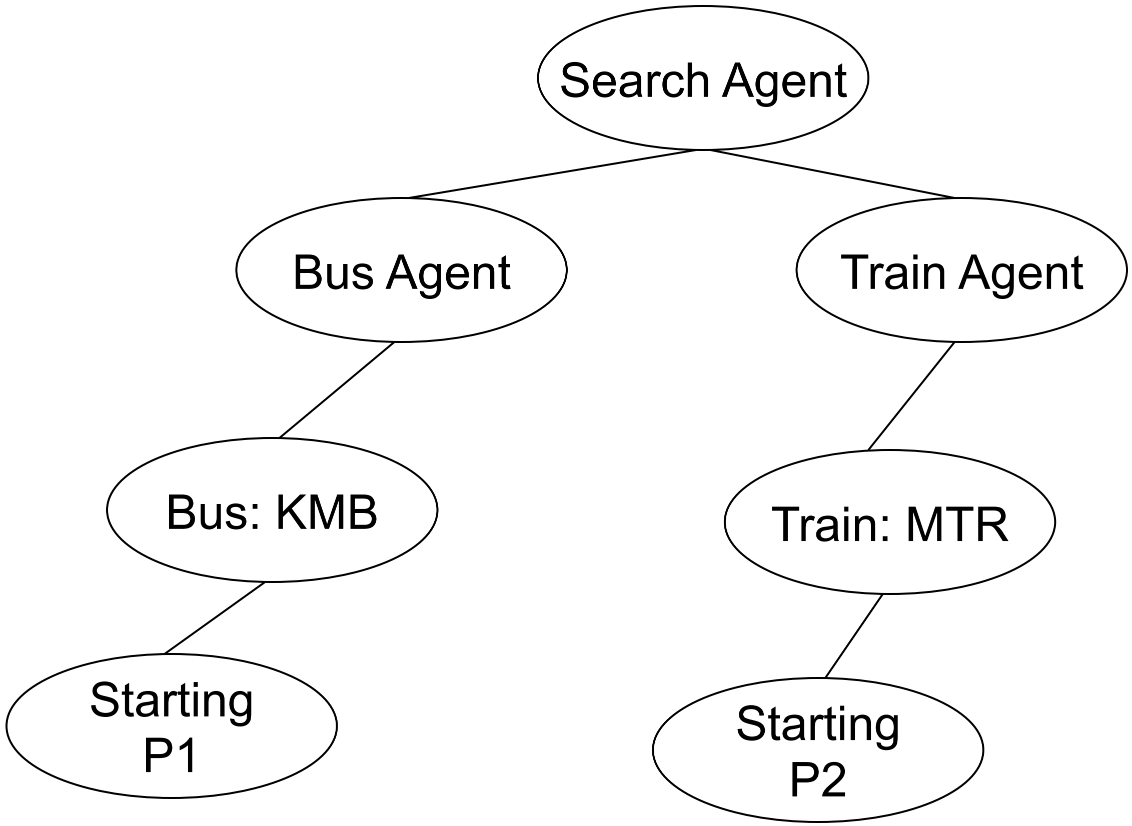


Table 4.1: Petri-Net building blocks of basic activities

Building Block type	Description
Invoke	The Invoke activity directs a Web service to perform an operation.
Reply	The Reply activity matches a Receive activity. It has the same partner link, port type, and operation as its matching Receive. Use a Reply to send a synchronous response to a Receive.
Empty	The Empty activity is a no operation instruction in the business process.
Assign	The Assign activity updates the content of variables.
Terminate	The Terminate activity stops a business process.
Throw	The Throw activity provides one way to handle errors in a BPEL process.
Wait	The Wait activity tells the business process to wait for a given time period or until a certain time has passed.

Table 4.2: Petri-Net building blocks of structure activities

Building Block type	Description
While	Repeat the same sequence of activities as long as some condition is satisfied.
Switch	Use "case-statement" to produce branches.
Sequence	Definition of a series of steps for the orderly sequence.
Link	Link different activities work together.
Flow	A series of steps should be specified in parallel implementation.

Petri-Net– Basic Activities

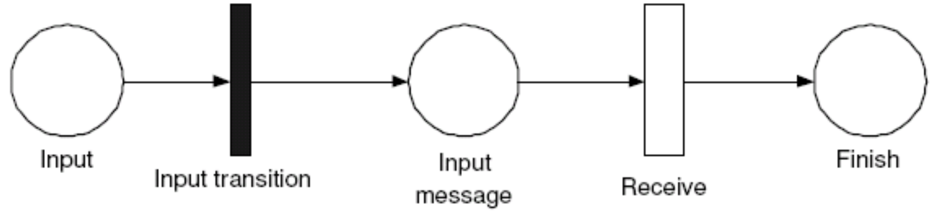


Figure 4.3: Basic Petri-Net building block – Receive.

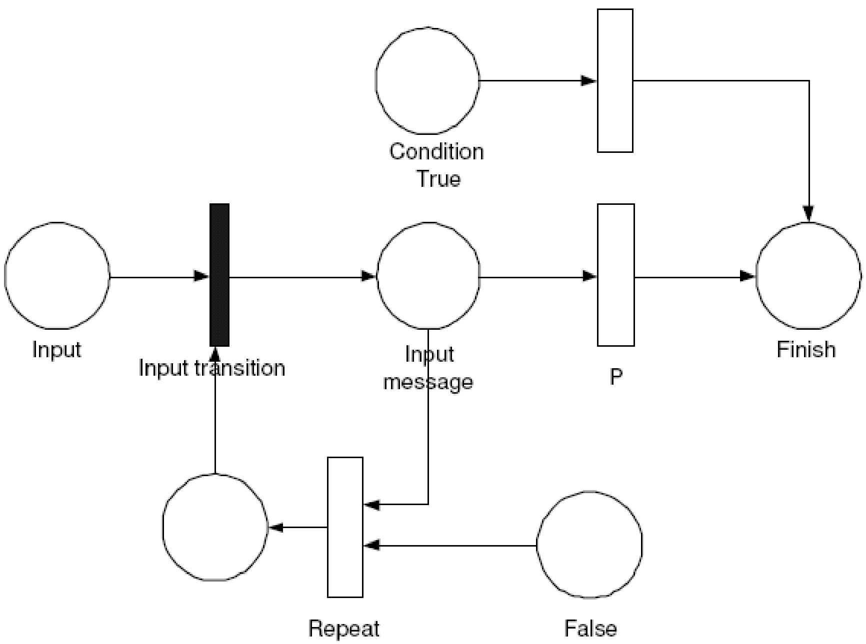


Figure 4.5: Basic Petri-Net building block – Wait.

Petri-Net– Structure Activities

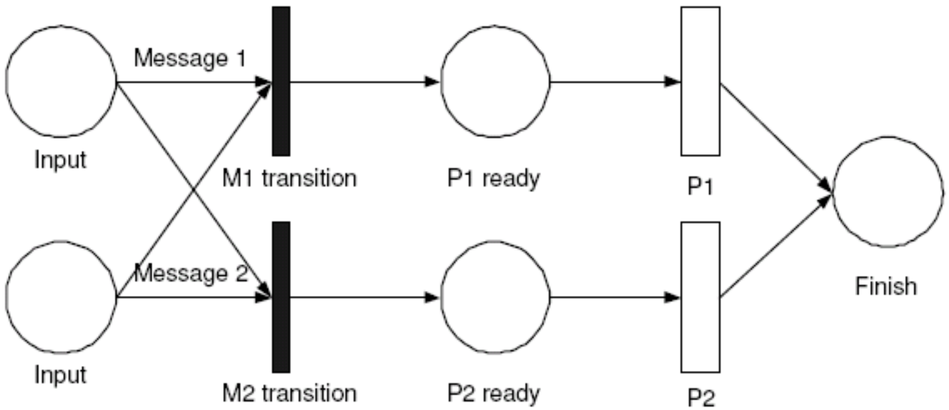


Figure 4.10: Structure Petri-Net building block – Pick.

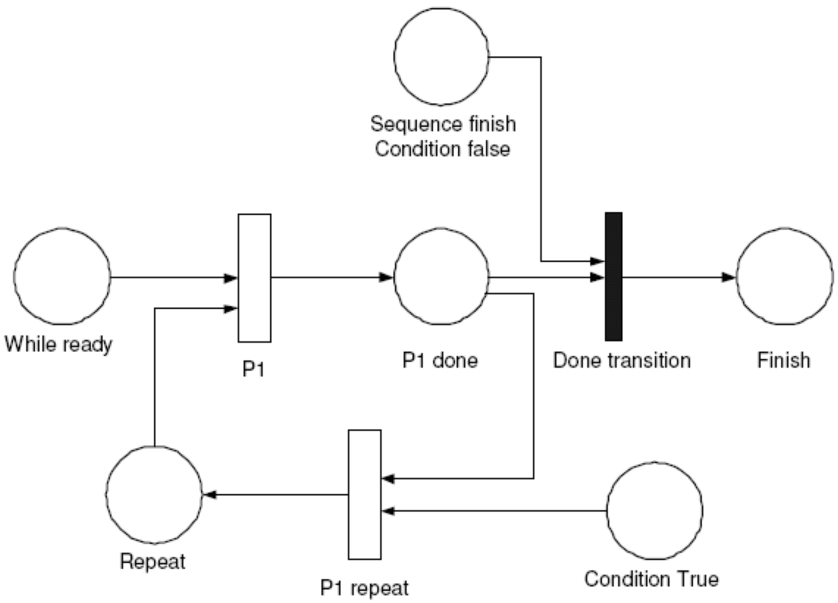


Figure 4.11: Structure Petri-Net building block – While.

Composed Petri-Net

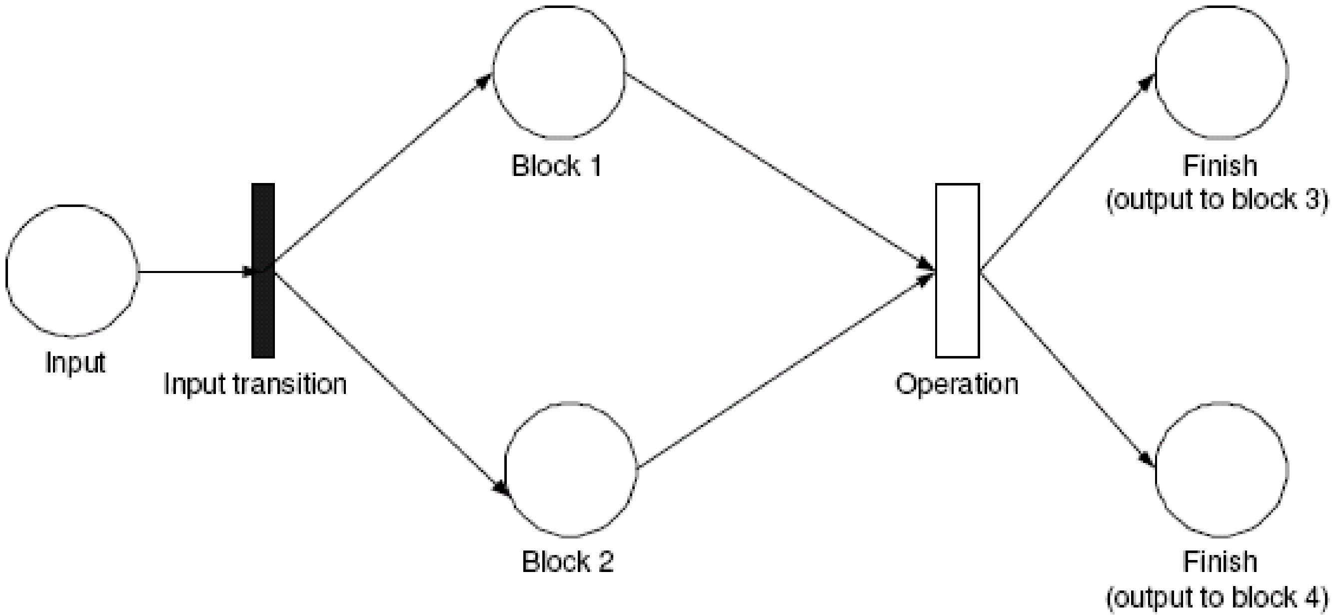


Figure 4.15: Composed Petri-Net building block graph.

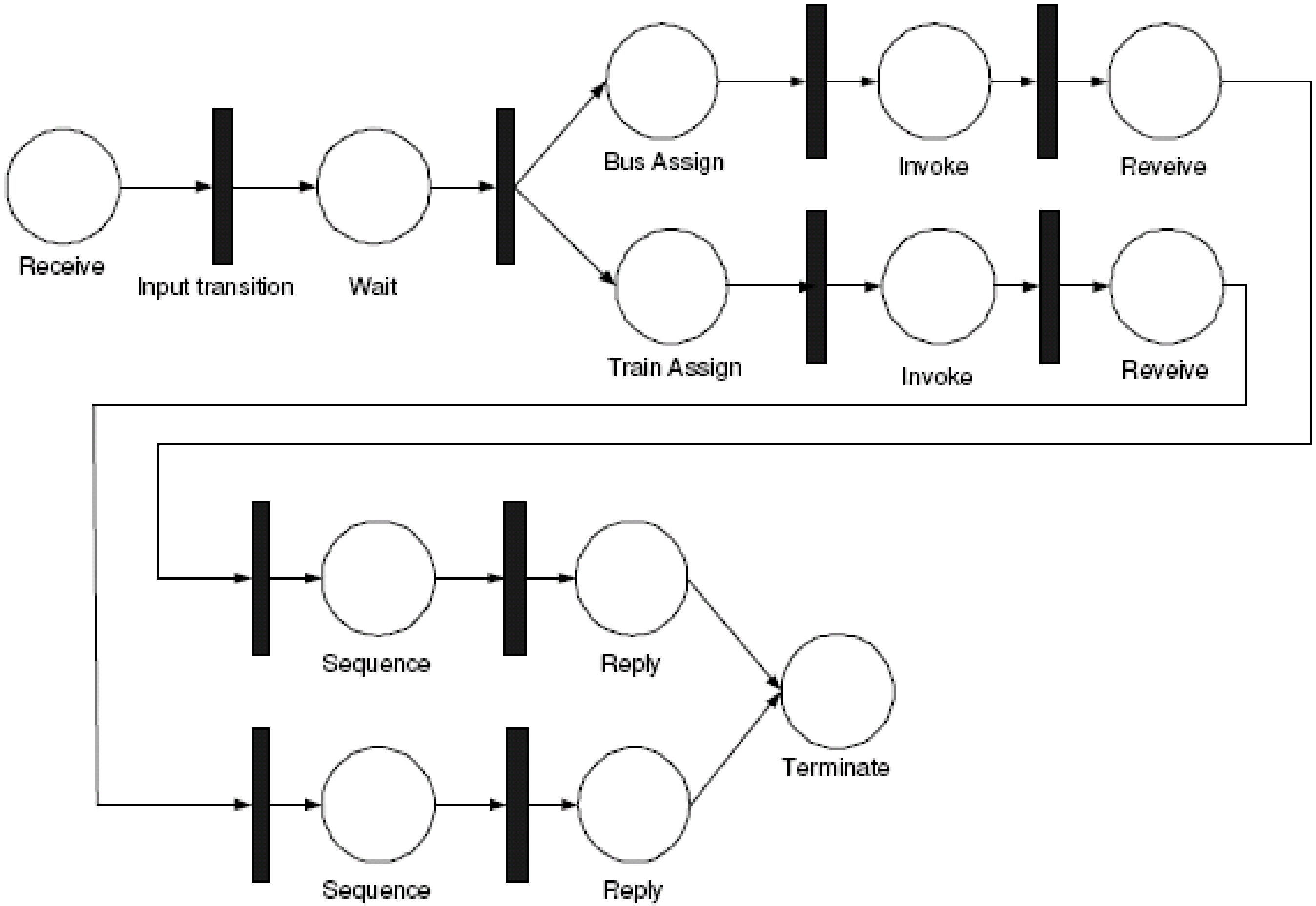


Figure 4.16: The Petri-Net of a BRF.

Table 6.16: Program metrics of the 15 versions

ID	Lines	Line without comment	Number of function	Complexity	time for composition (s)	Deadlock free	Acceptance test
01	3452	3052	59	64	-	yes	pass
02	2372	1982	47	87	-	yes	pass
03	2582	2033	26	45	-	yes	pass
04	3223	3029	78	124	-	yes	pass
05	2358	2017	34	89	-	yes	pass
06	4478	3978	56	107	-	yes	pass
07	1452	1320	38	46	-	yes	pass
08	5874	5275	80	124	-	yes	pass
09	3581	3214	45	74	-	yes	pass
10	4578	4187	47	113	-	yes	pass
11	2364	2015	36	76	-	yes	pass
12	2987	2336	65	147	1.48	yes	pass
13	4512	3948	75	155	1.74	yes	pass
14	3698	3247	60	192	1.58	yes	pass
15	4185	3856	34	88	1.62	yes	pass

Summary of the Web Service Composition Algorithm

- The composition algorithm is proposed with the use of WSDL and WSCL
- The BPEL of the composed Web services are generated
- Petri-Net is employed to avoid deadlock
- Acceptance tests are set for checking the correctness
- Experiments are performed
  - Efficient
  - Accurate
  - Deadlock-free

Experimental Setup

- Same as the previous setting
- Employ the composed Web services (BRF)
- Fault Injection
  - Temporary
  - Permanent
  - Byzantine failure
  - Network failure

Experimental Result (1)

Table 6.18: Experimental results without spatial redundancy

Experiments (number of failure / response time(s))	1	2	3	4
Normal case	5/186	3/192	2/190	3/187
Temporary	1025/190	4/223	1106/231	4/238
Permanent	8945/3000	8847/3000	1064/3000	5/1978
Byzantine failure	315/188	322/208	314/186	326/205
Network failure	223/187	2/227	239/193	3/231
Average	2102/730	1835/770	541/220	68/568

Experimental Result (2)

Table 6.19: Experimental results with Round-robin

Experiments (number of failure / response time(s))	5	6	7	8
Normal case	5/216	3/225	3/224	1/220
Temporary	1114/215	2/281	1072/218	3/284
Permanent	5682/3000	5362/3000	222/217	3/224
Byzantine failure	142/219	6/259	177/222	2/224
Network failure	229/223	2/253	211/227	2/222
Average	1434/775	1075/804	328/222	2/235

Experimental Result (3)

Table 6.20: Experimental results with N-version programming

Experiments (number of failure / response time(s))	5	6	7	8
Normal case	0/219	0/220	0/216	0/217
Temporary	0/221	0/222	0/219	0/216
Permanent	3136/221	3427/223	189/229	0/221
Byzantine failure	0/221	0/219	0/220	0/218
Network failure	0/220	0/222	0/218	0/217
Average	627/220	685/221	38/218	0/217

Experimental Result (4)

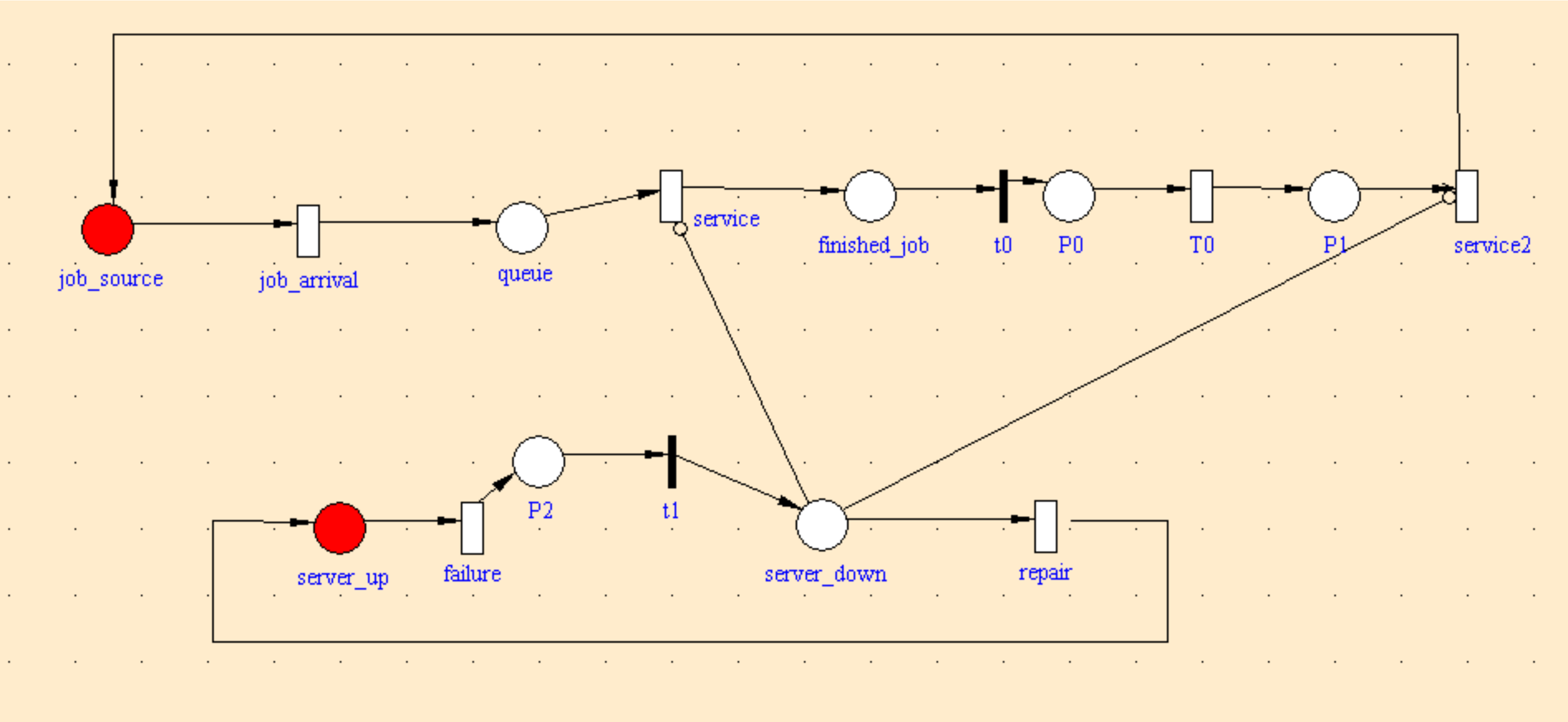
Table 6.21: Experimental results with recovery block

Experiments (number of failure / response time(s))	5	6	7	8
Normal case	0/221	0/219	0/224	0/219
Temporary	0/235	0/231	0/237	0/231
Permanent	3473/241	3250/238	201/242	0/231
Byzantine failure	0/224	0/230	0/225	0/224
Network failure	0/225	0/226	0/228	0/224
Average	695/231	650/229	40/231	0/228

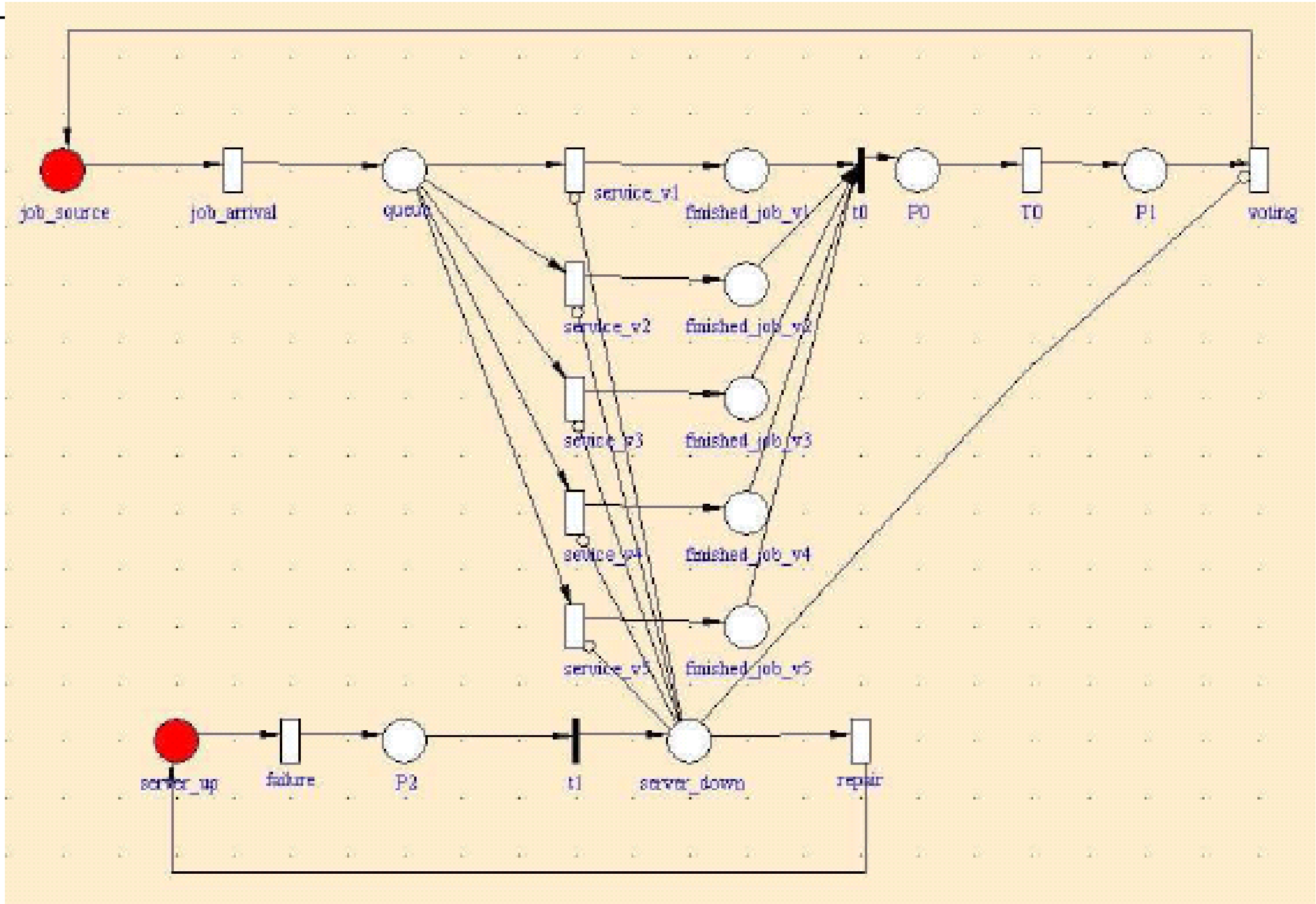
Modeling

- Modeling can check the reliability, correctness, deadlock-free and performance of the system
- We employed
  - Petri-Net
  - Markov chain model

Petri-Net (Four identical replicas)

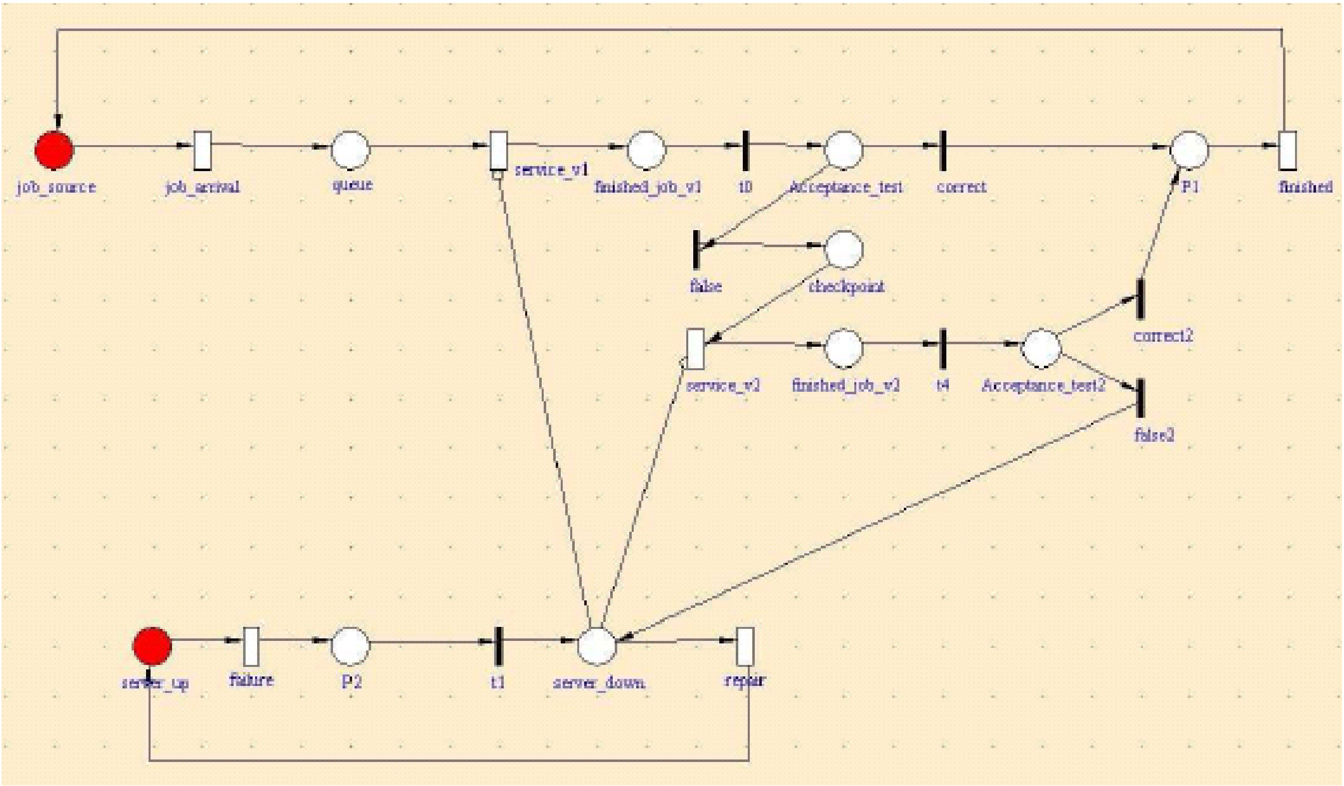


Petri-Net (N-version Web service with voting)

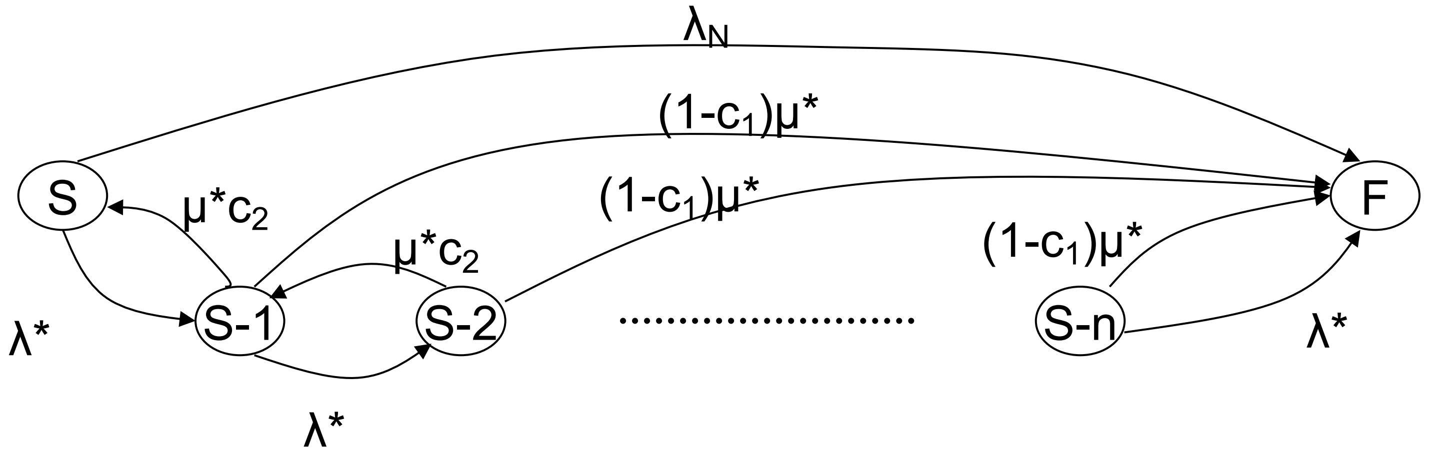


Modeling

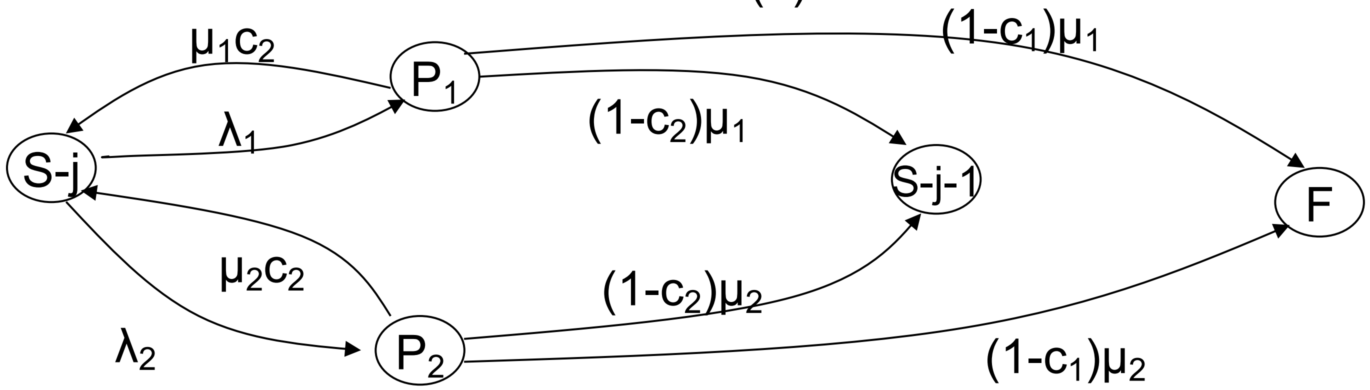
Petri-Net (Recovery Block)



Reliability Model



(a)



(b)

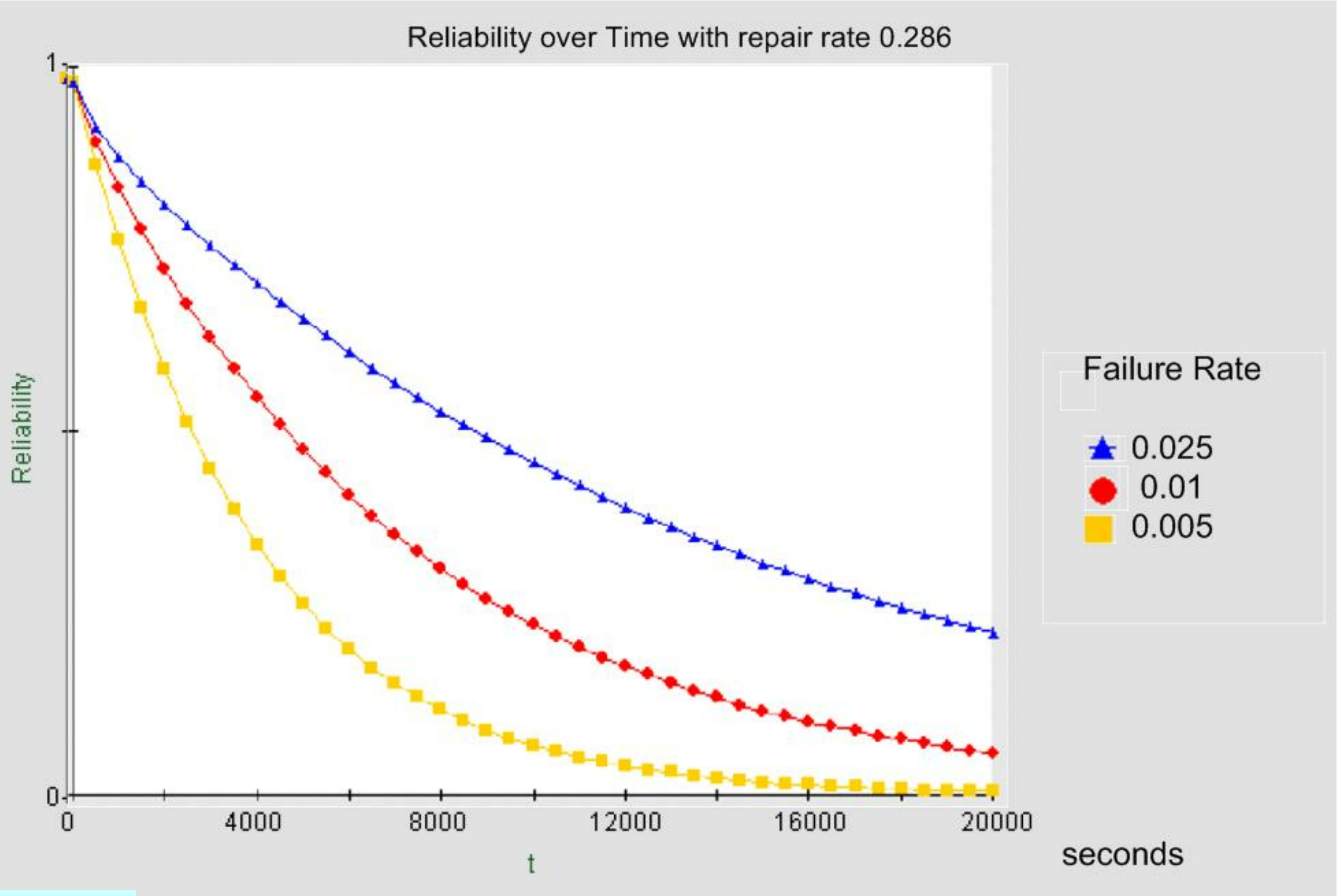
Reliability Model

$$\mu^* = \lambda_1 \times \mu_1 + \lambda_2 \mu_2$$

$$\lambda^* = \lambda_1 \times (1 - C_2)\mu_1 + \lambda_2 \times (1 - C_2)\mu_2$$

ID	Description	Value
λ_n	Network failure rate	0.02
λ^*	Web service failure rate	0.025
λ_1	Resource problem rate	0.142
λ_2	Entry point failure rate	0.150
μ^*	Web service repair rate	0.286
μ_1	Resource problem repair rate	0.979
μ_2	Entry point failure repair rate	0.979
C_1	Probability that the RM response on time	0.9
C_2	Probability that the server reboot successfully	0.9

Outcome (SHARPE)



Conclusion

- Surveyed replication and design diversity techniques for reliable services and the state-of-the-art Web service composition algorithm.
- Proposed a hybrid approach to improving the reliability of Web services.
- Optimal parameters are obtained.
- Proposed a Web service composition algorithm and verified by Petri-Net.
- Carried out a series of experiments to evaluate the availability and reliability of the proposed Web service system.
- Employ Petri-Net and Markov chain to model the system to analysis the reliability and performance.

Future Work

- Improve the current fault-tolerant techniques
  - Current approach can deal with hardware and software failures.
  - How about software fault detectors?
- N-version programming
  - Different providers provide different solutions.
  - There is a problem in failover or switch between the Web Services.
- Application
  - Different requirements
  - Realize in the Internet.

Q&A