FRAMEWORK FOR DYNAMIC SERVICE INTEGRATION USING AGENTS
Divya.A
IFET College of Engineering, Villupuram, Tamilnadu, India

Abstract- Service Integration is a platform for building complex service logics by integrating disparate service logics from diverse enterprises. Integrating service logic from distinct enterprises to persuade the demand exactly is not a painless task which entails to discern required pieces of service logics, accessibility level to consume the logic must be established, discovered endorsed service logics must be extracted evocatively with its dependent parts and must determine proper mode to integrate the service logics. The present demand is to automate all these progresses and develop complex logics robotically. Software agents are now increasingly used in commercial applications to solve these kinds of issues. In this paper, we present a novel integration mechanism based on multi agent system to integrate the service logics mechanically. The proposed integration system accomplishes the process through disparate agents such as Discoverer agent, SLA Negotiation Agent, DA (Dependency Analyzer Agent), PE (Property Evaluation) Agent and SI (Service Integration) agent. Each agent in the system plays a vital role to craft the service integration, an automated one. Authors presented various evaluation metrics and results for the proposed work. Experimental results show the effectiveness of model for various requisites.

Keywords: Dynamic Service Integration, Software Agent, Dependency analyzer, Finite State Machine, Business to Business Collaboration

1. Introduction

Service integration is designed to enable the organizations to attain integration maturity for integration between enterprise applications among partners. But the current market demand does not satisfied with integrating applications or web services as a whole. It steeps to haul out essential piece of logic from the entire service and to build up a complete service by integrating these unearthed logics. Integration in this level has many challenges. Required logic to resolve the requisite must be located among the desperate service of various partners. Located service logic must be extracted with it dependent parts in the logic and must be integrated in proper mode. Moreover accessibility level to utilize the discovered logic must be verified before going in to integration process. Though demand for dynamic service integration grows high and also has many advantages such as reusability, business agility, etc. these issues makes it trivial.

To address these issues, we proposed a multi agent based Service integration system which facilitates to integrate the service logics robotically. Agent is an autonomous entity which observes and acts upon an environment. It may learn or use knowledge to achieve their goals. Intelligence of agent includes methodical, functional, procedural or algorithmic search, find and processing, approaches. A multi-agent system is a system composed of multiple interacting agents. It is used to solve problems that are difficult or impossible for an individual agent or a monolithic system to solve. Each agent in the proposed multi agent system plays a crucial and unique role in automating this process. Service Discovery agent, a primary agent in the system dynamically looks up the required service logic through its advanced search component, logic discoverer. SLA Negotiation Agent is a significant agent used for security testing which substantiates the authorization level to exploit the logic. DA Agent, a chief agent in the system plays an imperative role in the integration process whose main role is to analyze the dependency among the service logics. It carries out this through Dependency Analysis approach which employs Turing Machine (TM) to depict the flow of service logic. Through TM it analyzes the dependency and extracts intellectually. As a final step, extracted logic must be integrated in efficient way as it satisfies the requisite exactly. SI (Service Integration) Agent, a key agent analyzes the dependency between the extracted service logics and ascertains better way for integration. Through these agents, system automates service integration. During the integration process, there is possibility for bugs and access violations. To look up this, there is a separate agent, PE Agent which looks up the process and fixes the issues through various functional and non-functional assessments such as computability, completeness, accessibility and configurability. This assures enterprises to share and integrate the service logics dynamically with cent percent confident without developers intervention at any stage of this process. Evaluation metrics to evaluate the work is explained briefly and experimental results are presented graphically.

The paper proceeds as follows: Section 2 discusses related works from research world and industry. Section 3 describes the working of proposed framework. Section 4 gives detailed view of SLA Negotiation Agent. Section 5 gives detailed account of Dependency analysis approach. Section 5 provides brief outline of service integration implementation methodology. Section 6 presents evaluation metrics and results of the proposed
work. Finally, section 7 presents our conclusion and future work.

2. Related works

There are plenty of publications in the area of using Web Services to support service integration, B2B and integration since this area has attracted researches from various research institutes. XuHuiyang proposed a novel modern service industry service integration system based on SOA to integrate and compose common services. The system establishes an agent layer between service users and platform identifies actual service components required for the request, orchestrates and composes the identified service components in right way [1]. The system can only integrate the services in component level but in many case it requires to integrate at various level. Dong Hui-fang presented an approach to SOA based service integration which makes the service granularity flexible to change and makes the service components reusable at different grade of granularities. As a result, a great number of systems can maintain the existing good framework and mutually provide all necessary and easy-to-use services when they need to share information and interact with each other through integration [2]. Hui Zhang proposed an agent-based Web services integration model which integrates the service ensuring the QOS of integration system. To realize the QOS, he constructed QoS-based integration path selection algorithm which improves service integration efficiency and reduces the integration cost through this selection mechanism [3]. WenOuyang proposed Web service integration algorithm to integrate the existing Web services in order to satisfy a pool of tasks and, at the same time, to minimize the Web service hop count. This problem plays important roles when the response time is an important factor in evaluating the performance of the application [4]. KaCheuk WU presented an integrated and personalized tour planning portal based on Semantic Web service technologies for knowledge management which employs ontology to classify and manage service [5]. Above works pave way to semi-automatically integrates the services but there is no standard approach to dynamically integrate the services.

Zhaoen Jiang proposed an innovative model, ‘Multi-layer Structure for Dynamic Service Integration’ (MSFDSI) in SOA which adds authorized institution and a service integration & analysis adapter to achieve the service authorization, service analysis and dynamic service integration. [7]. In [8], the authors discussed TaMeX, a software framework for developing intelligent multi agent applications for integrating web-based application services. This architecture is based on an extensible integration specification language and a run-time environment consisting of reflective intelligent agents able to interpret and execute integration specifications, defined in the language. Ye Zhong et al [9] presented a Mobile Agent based three tier architecture of Web Service integration, and designed the agent interaction layer which is the key part of the architecture. Hui Zhang proposed the agent-based Web services integration model AWSIM. Comparing to the traditional Web services integration method, AWSIM is more open and intelligent, and more suitable to the large complicated system [10]. To an enterprise, the information integration is very important and difficult and, a web service based information integration framework was proposed in [11], and the intrinsic problems that traditional information integration technologies often meet have been solved. In [12], the authors have given the idea of pragmatic web service framework and the functionalities of different types of agents. The agents will firstly catch and analyze user’s requirements and generate the abstract web service workflow and pragmatic frame which is a frame to store the purpose, context, communication and negotiation. In [13], the authors provided a solution to the problems of both industrial maintenance and enterprise integration by integrating industrial systems and field devices. W.J.Yan [14] proposed B2B integration approach for SME by leveraging the characteristics of Web Services which utilizes pull and push mechanisms for effective information exchange and sharing between trading partners. This approach has been incorporated in a B2Bi Gateway which enables SMEs to participate in business-to-business collaboration by making use of Web Services [15]. Timon C. Du and Hsien-Ling Chen propose an active collaboration and negotiation framework (ACNF), which is a negotiation support system that uses active documents with embedded business logics or business rules that can adapt to different collaborative strategies in a business-to-business (B2B) environment [16]. Sangseung Kang [17], Business rules are business statements that define some aspect of a business. They describe, constrain and control the structure, operations and strategy of the business. In his paper, he analyzes business rules and business rule systems, and present requirements and considerations for the business rule expression and system.

3. Framework for Dynamic Service Integration

As service integration is the process of aggregating service logics of diverse partners and builds up a new service. It entails a complex interface to locate the required logic in the whole service and integrate them in proper way. Moreover it requires to validate authentication and authorization to view and utilize the partner’s logic. System accomplishes these tasks with a group of agents such as discovery agent, SLA Negotiation Agent, DA Agent, Builder Agent, PE Agent, SI Agent. Fig.1 exemplifies briefly the working of proposed system for dynamic service integration.

a. Discovery Agent

Discovery Agent discovers required business logics to process the requisite. Through logic slicer, it slices the acquired request into numerous parts and transforms it into standard format. For each part of the formatted request, it discovers correlated service from directory to process the request. The directory contains all registered services. Each registered service associated with metadata which contains authentication, authorization particulars and description of the service logic such as sub logic facts, etc.
Directory receives the request, passes it as a query to registered service and finds required services and service logics. If exact service logics are located, system continues its process with located logic. If slight alteration is needed at any part of logic to fulfill the request, it executes and goes off. If associated service or service logic found for the request but the requestor is not authenticated to consume, it suggests extending the partnership. If service not found for any part, it modifies the request slightly and displays the request in different form. If user clicks at any of the modified request, it develops logic for that request.

b. SLA Negotiation Agent
As service integration is integrating service logics of diverse partners, security is critical issue. Authorization level should be verified each time after locating the logic. Initially, it formulates the Contract between them in XML format. It holds all necessary information such as contract initiated & expire date, service name, logic name, accessibility level for each logic, etc. At each time after locating the logic, it verifies contract formulated in XML. It is explained in section 4.

c. Dependency Analyzer Agent
Logic extraction is a crucial part in the integration process which needs to extract the logic discovered by discoverer with its dependent pieces and develop it as a complete service. DA Agent analyzes the dependent rules, functions and parameters of each located rules and function through Rule Bound and Function Bound Analysis approach. It discovers dependencies with the ascertained rules and functions in the business logic through rule/function bound analysis which is discussed briefly in the next section. Rule bound analysis method analyzes the dependency by simulating Turing Machine (TM) through TM simulator as its transition transits the logic flow. Then it extracts the dependent parts sited through TM and develops a new service. TM simulator is a behavior model composed of a finite number of states, transitions between those states, and actions, similar to a flow graph in which one can inspect the way logic runs when certain conditions are met. The presence of this simulator is for the automation of parsing through the business logic for its abstraction. It identifies the primitive business functions for business scenarios from the function sets and checks if those are computable functions. Once a new service is built, property evaluator appraises the Business Logic with the

Fig 1. Agent based Service Integration Framework
interoperable goals such as Computability, Traceability, and Completeness and other goals such as Configurability and Accessibility. The determined rules, functions and parameters are extracted separately and new service is developed.

3.3 Service Integration Agent

After developing complete services, SI analyzes proper pattern to aggregate the developed services. Normally services can be integrated in four ways such as union, composition, substitution and Reducibility. Initially, it scrutinizes the relationship between sliced formatted requests from the obtained request and through which it attempts to uncover the integration pattern. For instance, requisite is to develop a service for “search by file type and content type”. Sliced and formatted request is “search by file type’ & ‘Search by content type’”. Relation between the two is ‘&’, accordingly it uses ‘Union’ pattern to integrate the services. Same way, it analyzes the relationship between the sliced-formatted request and ascertains pattern to integrate the service. After detecting the pattern with initial process, Integration Adapter (IA) scrutinizes discovered pattern is adaptable to integrate the services and accomplish the requisite exactly through TM which is discussed briefly in section 5 with necessary theorems and examples. If the developed services cannot be combined straightforwardly with the pattern ascertained by IA due to data type mismatch, etc., it robotically creates wrapper class in between them and aggregates the services. After aggregating the service, it verifies the similarity measure between the given requisite and developed service, accordingly develops graph with dynamic graph generation tool OBIEE (Oracle Business Intelligence Enterprise Edition). Finally Runtime builder/deployer builds and deploys the integrated service. Runtime Engine estimates execution time of integration process and generates graph with OBIEE.

3.4 Property Evaluator (PE) Agent

PE Agent investigates the defects in the logic and validates the quality of business logic by automating various functional and non-functional assessments to ensure that developed or integrated logic persuades the requisite absolutely. Functional assessments assess the business logic through properties such as computability, completeness and configurability to ensure efficiency and performance of the logic. It is a collaborative agent works collaboratively with all.

4. SLA Negotiation Agent

Authorization and authentication are critical issues as this is the process of sharing service logics among the partners. So accessibility is a critical issue as it is used to describe the degree to which the requestor can access service logic. Before going into the integration process, contract made between them should be verified, accordingly system must allow them to use. In some cases, service logic can be accessible by the requestor but not fully, partially a rule or function in the logic. So to meet those cases, accessibility level of each and every rule, function and parameters are verified by SLA Negotiation Agent. For this, it generates FOL(First Order Logic) for each and every line of code as in the contract. FOL scrutinizes each and every parameter in it’s is accessible, if so it returns 1. This FOL is used as states in TM, if particular FOL gives result as 1, it will move on to next state otherwise it will reside in it itself. IF TM halts with in a time limit, it means requestor authorized to access particular part of logic. Same way, it checks for each and every part of logic and goes for integration. Table 1 gives illustration for FOL conversion and algorithm 1 describes the algorithm for Accessibility [19].

5. Dependency Analysis Approach

Dependency analysis mainly to analyze the dependency between the business rule to be isolated with other business rule, business function and parameter in the business logic. DA Agent does this task intellectually by collaborating with other agents in the system. As a first step, it categorizes business logic into rule set, function set and parameter set. Next, it develops Business Logic schema as its tag categorizes business rule, functions and parameters and it depicts the flow from one business rule to other business rule, function, parameter and vice-versa. BL schema for business logic ‘credit card validation’ presented in listing 1 is depicted in listing 2. At the third step Turing machine is constructed whose tapes represent business rule, functions and parameters. Turing machine represents transition and dependency between each business rule, function and parameter and vice-versa. This section explains this whole process with business logic ‘Credit card Validation’ shown below.
RULE CODE FIRST ORDER LOGIC

<table>
<thead>
<tr>
<th>RULE</th>
<th>CODE</th>
<th>FIRST ORDER LOGIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>get(String account) Select * from withraw where account no like {$Keyword}</td>
<td>$\exists a[account(a)] \rightarrow w[withdraw(w)] \land a[account_no(a)]$</td>
</tr>
<tr>
<td>R2</td>
<td>get (user id,pin no) if(user.equals(user id) &amp;&amp; pass.equals(pino)) { out.println(&quot;Login Successful&quot;);} else out.println(&quot;Login Failed,Please try Again&quot;) } }</td>
<td>$\exists y[login(y)] \land z[password(z)] \rightarrow login\ success(y,z). Or \exists y[login(y)] \land \neg z[password(z)] \rightarrow login\ success(y,z).$</td>
</tr>
</tbody>
</table>

Table 1. First Order Logic Conversion

Algorithm Accessibility(BusinessLogic, AccessPoint, AccessControl)

// Assume that the input BusinessLogic is decomposed into set of Rules(R), set of Functions(F), set of Parameters(P) and set of Dependency Relation(R) i.e BL consist of 4-tuples (R,F,P,D)
//Each Rule contains an Access Point which is the entry point of the business rule code segment
And it ensures the accessibility of the modifiable business logic based on the Access Control Type
Begin
For each predefinedAccessPoint
For i = 1 to n // let 'n' be the no of access point
Slice[i]=DoSegment(BusinessLogic,AccessPoint)
For each slice in Slice[n] do
 // convert each code segment(slice) into first order logic (FOL)
FOL[i]=Convert_to_FOL(slice)
//AccessControlType can be of three categories such as subject, resource and environment
//User,Owner,Admin,etc
UserObject[]=ExtractSubject(FOL[i])
//Database, FileSystem, CommerceServer
ResourceObject[]=ExtractResource(FOL[i])
// Datacenter, Organization
EnvironObject[]=ExtractEnvironment(FOL[i])
AccessControlType=[UserObject,ResourceObject,EnvironObject]
//Construct Finite State Machine for the complete business logic
Construct_FSM(FOL[i])
Begin
Let 'M' be the Finite State Machine which consists of 5-tuples $M = (Q, \Sigma, \delta, q_0, F)$
where
- $Q$ is a finite set of states, each state stores equivalent First Order Logic (FOL) of fragmented code slice
- $\Sigma$ is a finite set of symbols, the input alphabet, which is nothing but the access control specification.
- $\delta: Q \times \Sigma \rightarrow Q$ is a transition function, makes transition from one state to other state if the required Access control specification satisfies.
- $q_0 \in Q$ is the initial state, entry point of FOL[i], where i = 1
- $F \subseteq Q$ is a set of final states. Exit point of FOL[i], where i = n.

1947
Note: The fact that $\hat{\varepsilon}$ is a function implies that every vertex has an outgoing transition for each member of $\Sigma$.

We can also define an extended transition function $\hat{\varepsilon}: Q \times \Sigma^* \times \{0,1\} \rightarrow Q$ and transition function such as $\hat{\varepsilon}_i: Q \rightarrow Q$, $\varepsilon_i: \{0,1\} \rightarrow Q$, and transition function such as $\varepsilon_i: Q \rightarrow Q$, representing negation of the input access specification $r_i$.

Create State for each FOL[i] with respect to the slice.

Construct State Transition Table (STT) for the identified states and map transition between the states.

End

Do

Signal $\Rightarrow$ DoTraversal(FSM, FOL[i], AccessControl, AccessControlType, STT)

While (Not FinalState(FSM) && FOL[i]!=NULL)

If Signal returns 1 then Accessibility is TRUE for all set of Access Point

Else Accessibility is denied for the Current Access Point ‘x’ with reason.

End

Listing 1. Business Logic for Credit card validation

```java
import java.sql.*;
import java.util.*;
public class creditcardvalidation implements creditcardvalidIF{
    public String validation(String cardno, Date expdate){
        String status="";
        //Validate card length
        if (cardno.length() < 13 || cardno.length() > 16)
            status+="Invalid Cardno: Out of range";
        // Verify Expiry date
        Calendar curdate = Calendar.getInstance();
        Date todate=curdate.getDate();
        if(todate>expdate)
            status+="Sorry! Credit card expired";
        //*** Validate cardno
        int val=0,val1=0;
        for(int i=1;i<=cardno.length();i++)
            if(i%2==0)
                val+=Integer.parseInt(cardno.charAt(i));
            else
                val1+=Integer.parseInt(cardno.charAt(i));
        if(((val+val1)%10)!=0)
            status+="Invalid creditcard";
        if(status="")
            status="valid credit card";
        return status;
    }
```

BL schema for above logic generated by DA Agent is shown in listing 2 which is a complete translation of logic in listing 1 and describes the flow of above logic absolutely. The tags of BL schema classify the logic into business rules, functions and parameters. It also narrates the scope of each business rule, function and parameter and dependency with other. It is given as an input to the TM Simulator which simulates the TM to analyze the transition from one element to other and generates BL pattern accordingly. The BL pattern portrays the dependency between each construct conceptually. TM for BL schema in listing 2 is imparted in Fig 3 and corresponding BL pattern is offered in table 1. This Turing machine is consumed by various components in the model at various levels to trace out bugs, scrutinize efficiency of the logic, etc. At this stage, TM is simulated to analyze the dependency and examine the correctness of the logic. If it transits till end state within a time limit; it implies that the logic can run efficiently with minimum space and time complexity, it adds essential information to rule repository as shown in table 2.
Listing 2. Business Logic Schema

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<BL name="validation" include="cardno,expdate" includeType="xs:String,xc:Date" excludeType="xs:String" id="M1"/>
<BF name="String" name="status" minOccurs="null" id="BP1"/>
<BR name="Validate length" id="BR11"/>
<BF id="BF11" type="if" test="cardno\t13 and cardno\t16"/>
<BR name="status" value="Invalid cardno: Out of range" id="BP11"/>
<BR/>
<BF name="Verify Expired date" id="BR12"/>
<BF type="func" accessmodule="getInstance()" uses="calendar" id="BF121"/>
<BR name="curdate" type="Calendar" id="BP121"/>
<BR type="func" accessmodule="getDate()" uses="curdate" id="BF122"/>
<BR name="todate" type="Date" id="BP122"/>
<BR type="if" test="todate\texpdate" id="BF123"/>
<BR name="status" value="Sorry! Credit card expired" id="BP123"/>
<BR/>
<BR name="Validate Entered digits" id="BR13"/>
<BF type="int" name="val\tmimoccurs\t0" id="BP31"/>
<BF type="int" name="val\tmimoccurs\t0" id="BP32"/>
<BF type="for\" minoccurs\t1\" steps\t1\" test="le\ t\tcardno length\" names="BF131"/>
<BF type="if\" test="1 mod 2\" id="BF1311"/>
<BF type="func" name="In\t" accessmodule="accessmodule1\" characterAt\"(" accessmodule="parseInt()" uses="cardno" id="BF13111"/>
<BF type="func" name="In\t" accessmodule="accessmodule1\" characterAt\"(" accessmodule="parseInt()" uses="cardno" id="BF13121"/>
<BF type="func" name="In\t" accessmodule="accessmodule1\" characterAt\"(" accessmodule="parseInt()" uses="cardno" id="BF131121"/>
<BF type="if\" test="val\t1+val\tmod\t10\" id="BF132"/>
<BR name="status" value="Invalid cardno" id="BP1321"/>
<BR type="if\" test="status\tnull" id="BF111"/>
<BR name="status" value="Card valid" id="BP11"/>
</BF>
</BL>
```

Fig 2. Turing Machine for Business Logic ‘Credit card Validation
Through the transition of Turing machine, dependency analyzer analyzes the dependency between the rules, functions and parameters. Figure 3 illustrates storage representation of BL pattern in memory. The data structure employed here for storage is stack joint with linked list. Business logic, rules, functions and parameters are stored separately in each stack as shown in fig 3. Once the required rule is located in rule repository it refers corresponding pattern in storage and traces dependent fragments. Accordingly situates tags in schema for each ID in pattern. Fig 3 depicts schema mapping for each bit of the pattern.
Listing 3. BL Schema for validating credit card no

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Implementation>
  <packagename>sql.*</packagename>
  <packagename>util.*</packagename>
  <class name="creditcardvalidation" implements="creditcardvalidIF">
    <validation includes="cardno,expdate" includetype="xs:String,xs:Date" excludestype="xs:String">
      <BP type="for" minoccurs="1" step="+1" test="i le cardno.length">
        <BP type="if" test="i mod 2">
          <BF type="func" name="In" accessmodule1="charAt()" accessmodule="parseInt()" use="cardno">
            <BPop add="val" par1="In" name="val"/>
          </BF>
        </BP>
        <BP type="else">
          <BF type="func" name="In" accessmodule1="charAt()" accessmodule="parseInt()" use="cardno">
            <BPop add="val1" par2="In" name="val1"/>
          </BF>
        </BP>
      </BF>
      <BF type="if" test="val+val1 mod 10">
        <BP type="func" name="In" accessmodule1="charAt()" accessmodule="parseInt()" use="cardno">
          <BPop add="val+val1" par1="In" name="val"/>
        </BP>
      </BF>
      <BP type="else"/>
      <BP type="func" name="status" accessmodule1="charAt()" accessmodule="parseInt()" use="cardno">
        <BPop add="status" par2="In" name="valid"/>
      </BP>
    </validation>
  </class>
</Implementation>
```

Listing 4. Business Logic for validating Credit card no.

```java
import java.sql.*;
import java.util.*;
public class creditcardvalidation implements creditcardvalidIF{
  public String validation(String cardno, Date expdate){
    String status="";
    int val=0, val1=0;
    for(int i=1;i<cardno.length;i++)
      if(i%2==0)
        val+=Integer.parseInt(cardno.charAt(i));
      else
        val1+=Integer.parseInt(cardno.charAt(i));
    if(((val+val1)%10)!=0)
      status+="Invalid creditcard";
    if(status="")
      status="valid credit card";
    return str;
  }
}
```

Fig 4. New Turing Machine for extracted logic
Now for instance the requisite is to 'develop a service for validating credit card no'. The Discovery Agent splits up the requisite into copious keywords and pursuits related service for each of its keyword. With the help of SLA Negotiation agent, it verifies the authorization level of each ascertained service. From the list of identified authorized services, Discovery Agent reveals precise one to process the request. Accordingly spots list of business rules to process the request. Here the requested service can be built from the above service through the Business logic associated with Business Rule BR\(_1\). DA Agent analyzes dependency between the required Business Rule BR\(_1\) with other business rule, business function and parameter and ascertains through BL pattern stored in memory. First DA Agent spots BL pattern associated with exposed business rule to process the request. Accordingly it finds out the dependent fragments and tracks matching schema for each symbol of the pattern. It generates separate BL schema with the tracked tags in the existing schema. BL schema generated for our case is shown in listing 3. As the schema and logic are tightly coupled, it ascertains business logic correlated with the extracted schema and develops a new logic. Extracted logic is publicized in listing 4. Now with BL pattern it performs a reverse process to construct Turing machine for this new logic as it is necessary to trace the bugs and scrutinize efficiency of the logic. With the revealed BL patterns in rule, function and parameter level, it uncovers the segment associated with them and new TM is simulated with the detected segment. Locating required division for the spotted pattern and newly constructed Turing machine for our requisite is shown in fig 4.

6. Pattern Recognizer

After ascertaining required logic with its dependent fragments, pattern to integrate the services should be discovered. Here SI Agent accomplishes this task through request formatted by Discovery agent and TM constructed by DA Agent. Each pattern is discussed in detail here with illustrations.

a. Union

Theorem: Suppose \( M_1=(Q, \Sigma, q_0, \delta_1,T_1,H_1,B) \) and \( M_2=(Q, \Sigma, q_0, \delta_2,T_2,H_2,B) \) accept business logics \( L_1 \) and \( L_2 \) respectively. Let \( M \) be an Turing Machine defined by \( M=(Q, \Sigma, q_0, \delta, T, H,B) \) where \( Q \) be the set of States of \( M \), \( \Sigma \) be the input alphabet, \( q_0 \) be the start state, \( \delta \) transition function=\( \Sigma \times (T \cup \{B\}) \rightarrow (Q \cup \{H\}) \times (T \cup \{B\}) \times (R,L,S) \) is a partial function (that is, possibly undefined at certain points), \( T \) be the tape symbol, \( H \) be the halt sate, \( B \) be the blank symbol, \( Q=q_0 \times Q_2, q_0=(q_0,q_0) \), and the transition function \( \delta \) is defined by the formula \( \delta((p,q),a)=((\delta_1(p,a), \delta_2(q,a)) \) (for any \( p \in Q_1, q \in Q_2 \) and \( a \in \Sigma \)). Then If \( H=\{(p,q)|p \in H_1 \) or \( q \in H_2 \} \), M accepts the logics \( L_1 U L_2 \).

Proof: Since acceptance by \( M_1 \) and \( M_2 \) is defined in terms of the functions \( \delta_1 \) and \( \delta_2 \) respectively and acceptance by \( M \) in terms of \( \delta \), we need the formula \( \delta((p,q),r)=((\delta_1(p,r), \delta_2(q,r)) \)

which holds for any \( x \in \Sigma^* \) and any \( (p,q) \in Q \), and can be verified easily by using mathematical induction. A rule \( r \) is accepted by \( M \) if and only if \( \delta((q_1,q_2),r) \in A \). If the set \( A \) is defined as \( H=\{(p,q)|p \in H_B \) or \( q \in H_B \} \), \( M \) accepts the logics \( L_1 U L_2 \) this is the same as saying that \( \delta_1(q_1,r) \in A_1 \) or \( \delta_2(q_2,r) \in A_2 \), or in other words, that \( r \in L_1 U L_2 \).

Example Consider the requisite is to develop a service for search by file type and content type. Vista search is a book search service which contains business rules to search by file type, date of publication and language. Same way, Florasearch is product search service which contains business rules to search by content type, price and brand name. So the required service can be built with the two rules existing in the vista and flora services. Initially, business logic for the required rules are extracted and developed as separate services. WSDLs of the developed services are shown in fig 5.a and 5.b. Subsequently, FSM is constructed to annotate the logic flow. Here the services are integrated using union construct and final result is shown in fig 5.c. Similarity measure between request and final result is shown in fig 5.d.

5.2 Composition

Let us consider the composition of functions associated with the business rules of a particular domain and co domain with an example. Let \( f, g \) and \( h \) be three functions and the logical relation of first two functions are described as \( f:A \rightarrow B \) and \( g:B \rightarrow C \). This shows the relation between a domain and co domain. Then for any \( x \in A \), \( f(x) \) is in the domain of \( g \), and it is therefore possible to derive as \( g(f(x)) \). More generally, if \( f:A \rightarrow B, g: B \rightarrow C \) and the range of \( f \) is a subset of \( B \), then \( g(f(x)) \) is called composition of \( g \) and \( f \) and is written as \( h=g \circ f \). For example, the function \( h \) is defined by \( h(x)=\sin(x) \) is \( g \circ f \), where \( g(x)=\sin(x) \) and \( f(x)=x^2 \). The function \( f \circ g \) on the other hand, is given by the formula \( \sin(x)^2 \). When we compute \( g(f(x)) \), we will take the formula for \( g \) and replace every occurrence of \( x \) by the formula for \( f(x) \).
It can also be verified by just tracing the definitions that if \( f: A \rightarrow B \), \( g: B \rightarrow C \) and \( h: C \rightarrow D \), then the functions \( h \circ (f \circ g) \) and \( (h \circ f) \circ g \) from \( A \) to \( D \) are equal, and they are computed as follows. For \( x \in A \), first take \( f(x) \); then apply \( g \) to that element of \( B \) to obtain \( g(f(x)) \); then apply \( h \) to that element of \( C \) to obtain \( h(g(f(x))) \). We summarize this property of composition by saying composition is associative.

Consider the business rules
\[
\text{Rule 1: } r_1 = \{ f_1, f_2, \ldots, f_n \} \quad \text{and} \quad\text{Rule 2: } r_2 = \{ g_1, g_2, \ldots, g_n \}
\]
then \( g(f(x)) \) is called composition of rule \( r_1 \) and rule \( r_2 \) with their associated functions \( g \) and \( f \) and is written
\[
h = g \circ f,
\]
in general, \( h = g \circ f \). The function \( h \) is the composition function which is defined as \( h(x) = g(f(x)) \).

Let us show that if \( f: A \rightarrow B \) and \( g: B \rightarrow C \) are both one-to-one, then so is the composition \( g \circ f \). To say that \( g \circ f \) is one-to-one means that whenever \( g \circ f(x_1) = g \circ f(x_2) \), then \( x_1 = x_2 \). But if \( g(f(x_1)) = g(f(x_2)) \), then since \( g \) is one-to-one, \( f(x_1) = f(x_2) \). Therefore, since \( f \) is also one-to-one, \( x_1 = x_2 \).

Similarly, if \( f: A \rightarrow B \) and \( g: B \rightarrow C \) are both onto, \( g \circ f \) is also onto. For any \( z \in C \), there is an element \( y \in B \) with \( g(y) = z \), since \( g \) is onto and there is an element \( x \)
\( f(x) = y \) since \( f \) is onto. Therefore, for any \( z \in C \), there is an \( x \in A \) with \( g(f(x)) = g \circ f(x) = z \).

**Example** - Let the requisite is to develop tour reservation service to reserve for accommodation and travels automatically through this. This can be developed simply by composing reservation module of hotel and travels reservation services. WSDL of the composed service is shown in fig 6.a and similarity measure is shown in fig 6.b.

---

5.3 **Substitution**

**Theorem** : Business Logsics are closed under substitution.

**Proof** Let \( L \) be a BL, \( BL \subseteq \Sigma^* \) (where \( \Sigma^* \) be the business domain set which includes Rules, Functions, Business variables and Dependency Relation), and for each Rule \( r \in \Sigma \) let \( BL_{Rulei} \) be a complete BL. Let \( BL = L(G) \), \( G = (R, F, P, D) \), and for each Rule \( r_i \) in \( \Sigma \) let \( BL_{Rulei} = BL(G_{Rulei}) \), where \( G_{Rulei} = (R_{Rulei}, F_{Rulei}, P_{Rulei}, D_{Rulei}) \). Assume that \( R \cap R_{Rulei} = \emptyset \), for all \( \text{Rules} i \in \Sigma \). And \( R_{Rulei} \cup R_{Rulej} = \emptyset \), for all \( \text{Rules} i \neq j \in \Sigma \). Construct \( G' = (R', F', P', D') \), where \( R' = R_{Rulei} \cup R \), \( F' = F \cup \text{functions} \in F_{Rulei} \).
5.3 Substitution

**Theorem:** Business Logics are closed under substitution.

**Proof** Let \( L \) be a BL, \( BL \subseteq \Sigma^* \) (where \( \Sigma^* \) be the business domain set which includes Rules, Functions, Business variables and Dependency Relation), and for each Rule \( r_1 \) in \( \Sigma \) let \( BL_{r_1} \) be a complete BL.

Let \( BL = L(G) \), \( G = (R, F, P, D) \), and for each Rule \( r_1 \) in \( \Sigma \) let \( BL_{r_1} = BL(G_{r_1}) \), where \( G_{r_1} = (R_{r_1}, F_{r_1}, P_{r_1}, D_{r_1}) \). Assume that \( R \cap R_{r_1} = \emptyset \), for all rules in \( \Sigma \) and \( R_{r_1} \cap R_{r_2} = \emptyset \), for all \( r_1 \neq r_2 \) in \( \Sigma \).

Construct \( G' = (R', F', P', D') \), where
\[
P' = \bigcup_{param1 \in \sum} P_{param1} \cup \{ Rule_1 \rightarrow function_{1..x}(Param_{1..y}) | Rule_1 \rightarrow Function_{1..x} \in D, Rule_1(Function_{1..x}) = D_{rule1} \text{ for each Rule } Rule_1 \text{ in } \Sigma \text{ and Function}_{1}(Param_{1..y}) = D_{function1} \text{ for each Function } Function_1 \text{ in } \Sigma \}
\]

Example - Consider the requisite is to extend an authentication service by adding encryption part into it. Let the security service containing the required encryption part. Now this can be added to our service by extracting the required logic as explained in union method. The extracted logic can be substituted into the required place in our service. Now this extended service can be built and deployed. WSDL of deployed service is shown in fig 7.a and similarity measure is shown in fig 7.b.

5.4 Reducibility

If we can establish that one decision logic, \( L_1 \), can be reduced to another, \( L_2 \), or that having a general solution to \( L_2 \) would guarantee a general solution to \( L_1 \), then it is reasonable to say informally that \( L_1 \) is no harder than \( L_2 \). It should then follow that if \( L_2 \) is solvable (or equivalently, if \( L_1 \) is unsolvable).

Reducing one decision logic to another - If \( L_1 \) and \( L_2 \) are decision logics, we say \( L_1 \) is reducible to \( L_2 \) (written \( L_1 \leq L_2 \)) if there is an algorithm procedure that allows us, given an arbitrary instance I of \( L_1 \), to find an instance \( F(I) \) of \( L_2 \) so that for every I, the answer for the two instances I and \( F(I) \) are the same.

Example - Consider the requisite is to extend an authentication service by adding encryption part into it. Let the security service containing the required encryption part. Now this can be added to our service by extracting the required logic as explained in union method. The extracted logic can be substituted into the required place in our service. Now this extended service can be built and deployed. WSDL of deployed service is shown in fig 7.a and similarity measure is shown in fig 7.b.

Fig 8.a. WSDL of authentication service holding 6 parameters

Fig 8.b. WSDL of reduced service holding 4 parameters

Fig 8.c Similarity Measure between Requisite & Service Built
a. Computability

Though the system integrates the service logics dynamically, it is essential to verify that developed complex logic computable with in a time limit. Desirable characteristics of business logic or business function to be a computable one must have exact instructions which are clearly defined (i.e., core business logic), finite in length. Thus every computable function finite program (logic) that completely describes how the function is to be computed. If the business function is given a k-tuple \( x \) in the domain of \( f \), then after a finite number of discrete steps the function must terminate and produce \( f(x) \). Intuitively, the business function proceeds step by step, with a specific rule to cover what to do at each step of the calculation. Only finitely many steps can be carried out before the value of the function is returned. If the function is given a k-tuple \( x \) which is not in the domain of \( f \), then the function might go on forever, never halting, then it is hold by some sensitive exception. Or it might get stuck at some point with some basic types of exception, but it must not pretend to produce a value for \( f \) at \( x \). Thus if a value for \( f(x) \) is ever found, it must be the correct value and the business function is totally computable. Although the function may use only a finite amount of storage space during a successful computation, there is no bound on the amount of space it used. It is assumed that additional storage space can be given to the function whenever the function asks.

b. Completeness

The problem is to prove the business rules are complete and also to show the rules are semantically valid. When the correspondence between the syntax and semantics tighter, we would say the logic is complete. Generally, the business logic consists of four major tuples such as Rules, Functions, Parameters, and dependency relation which are related as:

\[
\exists \text{rules} \land \exists \text{functions} \land \exists \text{parameters} \land \exists \text{dependency relation}
\]

Therefore any semantically valid argument can be captured by formal proof. The choices of rules are to be made for its completeness. It can be easily done when the system is in static phase. When it is done in runtime the conditional variable and iterative variable are also changes. When these variables are modified it brings out bugs in the logic. So these variables must be declared with certain range. The extra complications come about because of these restrictions, needed to guarantee soundness of the rules, on the status of the variables. An algorithm has been proposed by considering the above standards [18].
Algorithm Compute_Primitive_Business_Function(Business LogicSet)
Input : BusinessLogicSet [L]
Output : Computational factor with total or partial computability evaluation
Methodology : Primitive Recursive Function (PRF) : to prove the given logic is computable or not
// ßr [] : Array for Business Rules (ßr [] = [ßf[1..n]]
// ßf[] : Array for Business Function (ßf[] = Ißf[] ∪ Cßf[])
// Ißf[] : Array for Initial Business Function
// Cßf[] : Array of Composite Business Function
// Pßf[] : Array of Primitive Business Function (Pßf[] = Pßf[Ißf[],Cßf[]])
Begin
(i) Input Business Logic Set(L)
// Business Logic Set consist of 4-tuples such as Set of Business Rules, Set of Functions associated with the Business Rules, Set of Domain variables called as Business Parameters and Set of Dependency Relations exist between the first 3-tuples. L = { R,F,P,D} 
L = { { ßr 1, ßr 2 ,… n }, { ßf 1, ßf 2,… n }, {P 1, P 2,… n } {D 1, D 2,…n } }
(ii) Phase 1: Evaluate Initial Business Function Ißf

For each BL[i] (where i = 1) in BL:
   If ßf[i] = Compute Initial_function(BL[i])
   Then ßf[i] = BL[i] where in BL[i] is computable within the polynomial time.
// Initial Functions includes { Zßf (x) = 0 | Ußf (x) = 1 | Ißf(x) = x | Prßf (x₁, x₂, x₃,… xₙ) = xᵢ | Sßf (x) = x+1 | Pßf (x) = x-1 | Cßf (x) = x ', etc}
End if
End loop
(iii) Phase 2: Evaluate Composite Business Function Cßf

For each ßf[i] (where i = 1) in ßf[i]:
   Cßf[i] = Apply composition over Initial Business function ßf[i]
// Cßf[i] = [ßf₁, ßf₂, ßf₃] and Cßf[i] = computed Initial Business Function from the Phase 1
Where ßf₁, ßf₂, ßf₃ are generated Initial Business Function from the Phase 1
Cßf[i] = [ßf₁, ßf₂, ßf₃] where in Cßf[i] is computable within the polynomial time.
End if
End loop
(iv) Phase 3: Evaluate Primitive Business Function Pßf

For each ßf[i] (where i = 1) in ßf[i]:
   Pßf[i] = Apply recursion over Initial Business function ßf[i]
// Pßf[i] = [ßf₁, ßf₂, ßf₃] and Pßf[i] = computed Initial Business Function from the Phase 1
From defined initial business functions ßf₁, ßf₂, ßf₃ are generated Initial Business Function from the Phase 1
Rßf[i] = [ßf₁, ßf₂, ßf₃] where in Rßf[i] is computable within the polynomial time.
End if
End loop
(v) (Computability factor
   If (Input BusinessLogic(BL) ) is identifiable in all the three phases then
   BL is totally computable
Otherwise BL is partially computable
End if
End.

Completeness  L={R,F,P,D}
Let ‘L’ be the Business Logic, which consists of 4-tuples such as
R – Set of Rules
F - Set of Functions
P - Set of Parameters
D- Set of dependency relation
Theorem: The Business Logic ‘L’ is complete iff the associated Rules, Functions, Parameters and the dependency relations are complete.
Proof: Parameters are complete.
Input Parameters <I1,I2,I3> are bound with discrete values then return
Boolean value (True)
c. Configurability

Configurability focuses on establishing and maintaining consistency of performance over the life cycle. Before extracting and developing service logic from remote service, we must verify resources required to build and execute the service logic is available in the system. If particular resource is not available it can go and search for alternative way and do changes accordingly. System adopts this modification only when it is trustworthy.

System Logic is formed of a set of code segment 'cs' consist of set of system information for configuration and maintenance. Configuration logic is part of system logic points out set of logic related to configurability. It holds set of methods such as connectors, drivers, resource types, credentials, access methods, etc. If any modification done in the system logic, it should be verified that it is not affected the associated business logic and updation is trustworthy. To perusal the modification, initially system logic is expressed in first order logic form. Whenever modification is done, modified logic will be exposed in to FOL and compared with the original FOL. Configurability is meaningful only when original FOL and new FOL are same. Let CL and CL' be configuration logic before and after modification, it verifies FOL(CL) = FOL(CL'). Also it verifies business logic output is same before and after modification. Let BL and BL' be business logics before and after modification, it verifies BL ∩ BL' = φ.

For each modification it verifies code segment and examines the updation is fruitful. Modification is reliable only when resource modified before and after are of same type. i.e., Database connection type is allowed to modify only when it is replaced with some other database connection type not with network connection type or some other. Let f(x) be original resource defined in the CL, g(x) be the modified one, it verifies f(x)f(γx)=g(x)=c, some
constant, i.e., the modification is allowed in CL only when resource types are same. Table 2 illustrates FOL conversion for configurability and Algorithm 4 describes the working of configurability in integration process.

8 Evaluation Results

After integrating service logics and developing a complex logic, SI Agent evaluates the performance of the integration process, compares with its past integrations and plots graphs graphically. Here performance is measured by service integration time.

(a) Service Integration Time

Service Integration time is defined as the time elapsed to accomplish the whole integration process. In other word, it is the total time required for each piece of framework to execute. Service integration time is computed by summing up logic discover time, service alignment time, service compilation time, schema generation time and service deployment time.

(b) Logic Discovery Time

Logic Discovery time is defined as the time spent by the Discovery Agent to process the request, locate the required business logics and extract the located business logics.

T_{ld}=\text{Time taken to process the request (T}_{p}\text{)+Time taken to locate the rule(T}_r\text{).}

Let n be number of located logics and T be the time taken to identify the rule, then T_{ld}=n*T_{ld}.

(c) SLA Negotiation Time

SLA Negotiation Time (T_{sla}) is time taken by the SLA Negotiation Agent to verify the policy in the contract for each and every service logic located by Discoverer Agent. It is the summation of collaboration time T_{coll} and time taken to do verification for each logic T_{agg}.

T_{sla}=\sum_{i=1}^{n}(T_{coll})+T_{agg}(2)

d. Dependency Analysis Time

T_{da} denotes total time elapsed for DA Agent to accomplish dependency analysis process in the whole service management progress. T_{da} is summation of time taken to identify required business rules to process the request, time taken to explore dependency at business rule, function and parameter. Let assume T_{be} be time taken to identify list of rules to process the request and T_{be} be time taken to locate context free grammar in memory for corresponding rule, function or parameter. T_{be} be total time taken to ascertain dependency at rule, function and parameter level for each located rule and n be number of logics discovered by Discovery agent.

\[ T_{da}=(T_{da} + T_{df} + T_{f})+T_{da} = \frac{dT_{da} +dT_{df} + dT_{f}}{dt} \quad (3) \]

Here, T_{da}, T_{df} and T_{f} refers time taken to observe dependency at rule, function and parameter level respectively.

\[ T_{da}=T_{df} + (r*T_{df}) + (f*T_{df}) + T_{f} \]

\[ T_{df}=T_{df} + (f*T_{df}) + (p*T_{df}) \]

T_{df} implies time taken to inspect dependency at rule, function and parameter level to manage the changes at runtime respectively.

\[ T_{f}=T_{f} + (p*T_{f}) \]

Total time taken for the whole process of RBA is

\[ T_{da}=T_{da} + T_{df} + T_{f} + dt \quad (4) \]

e. Service Integration Time

Service integration time is time taken by the SI agent to integrate the service logics extracted by DA Agent. It is the summation of time taken for the agent to identify the pattern to integrate the services logics(T_{p}), time taken to collaborate with DA Agent to get the extracted service (T_{da}) and time taken to deploy the service T_{df}.

\[ T_{si}=T_{si} + T_{da} + T_{df} + T_{f} \quad (5) \]

f. Property Evaluation Time

Property Evaluation Time (T_{pe}) is time spent by PE Agent to accomplish the property evaluation in the service integration process.

g. Total Integration Time

\[ T_{t}=T_{t} + T_{da} + T_{df} + T_{f} + T_{si} \quad (7) \]

SI Agent evaluated the effectiveness of work with various service logics in various constructs to produce various outcomes. Results are depicted in table 3 and pictured graphically in Fig 9.a-f.
Algorithm Configurability(Business logic, System logic, Configuration Request)
Input: Business logic [BL], code segment for System logic [SL].
Output: Configuration logic, Audit log
Methodology: Mapping the configuration logic into first order logic for semantic level configuration Verification. Further change and configuration manager component performs the complete evaluation of the configuration logic before and after the changes are done

Begin
(i) Get the configuration request Creq.
(ii) Analyze and extract the system logic w.r.t to the configuration request.
(iii) Classify Connectors, Drivers, Controllers, Credentials , Access Details, etc of extracted system logic.
(iv) Let 'Lc' be the configuration logic (i.e. identified system logic for configuration change) Lc=[{Drivers}, {Connectors}, {Controllers}, {Resource Types}, {Security logic}, etc] If 'Lc' is the code segment for drivers and 'Lc' is considered as a configuration logic then
   Convert_to(fol) \rightarrow x(Driver DriverObject (x)) =
   Else If 'Lc' is the code segment for Connectors then
   Convert_to(fol) \rightarrow x(Connector ConnectorObject (x)) =
   Else If 'Lc' is the code segment for Controllers then
   Convert_to(fol) \rightarrow x(Controller ControllerObject (x)) =
   Else If 'Lc' is the code segment for security logic then
   Convert_to(fol) \rightarrow x(Credentials SecurityObject (x)) =
   (v) Change Manager: Equivalence Verification of source logic with the modified logic
   (i.e. Equivalence of input configuration logic associated with the business logic (L₀₁) with the modified configuration logic associated with the business logic (L₀₂))
   (L₀₁ ∩ L₀₂) U (L₀₂ ∩ L₀₁) = \alpha
   (v) Configuration Manager: Let 'x' be the output generated by the configuration logic 'Lc' before configuration and 'y' be the output generated by the configuration logic after configuration w.r.t to the configuration request then the output function g(x) and g(y) can be represented as follows:
   GenerativePower[g(x)] is equivalent to GenerativePower[g(y)] : such that g(x) \cap g(y) = \alpha
   Where '\alpha' be the constant which can be \alpha (i.e. null)
   (vii) Generate 'Lc' and update the configuration log.

End
Table 3. Evaluation results of service integration

<table>
<thead>
<tr>
<th>Service (S1)</th>
<th>Service (S2)</th>
<th>Integrated Service</th>
<th>Td (ms)</th>
<th>Tsu (ms)</th>
<th>Ta (ms)</th>
<th>Tx (ms)</th>
<th>Tiu (ms)</th>
<th>TSi (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick Search</td>
<td>Quick Search</td>
<td>Semantic Search</td>
<td>0.954</td>
<td>1.132</td>
<td>0.567</td>
<td>0.674</td>
<td>0.395</td>
<td>5.722</td>
</tr>
<tr>
<td>Security service</td>
<td>Security service</td>
<td>Alert Service</td>
<td>0.876</td>
<td>1.354</td>
<td>0.678</td>
<td>0.8</td>
<td>0.563</td>
<td>4.271</td>
</tr>
<tr>
<td>Online Shopping</td>
<td>Online Shopping</td>
<td>Mail Service</td>
<td>0.979</td>
<td>0.996</td>
<td>0.623</td>
<td>0.710</td>
<td>0.482</td>
<td>3.79</td>
</tr>
<tr>
<td>Registration</td>
<td>Registration Service</td>
<td>Security Service</td>
<td>0.785</td>
<td>1.046</td>
<td>0.637</td>
<td>0.659</td>
<td>0.475</td>
<td>3.904</td>
</tr>
<tr>
<td>Billing service</td>
<td>Billing service</td>
<td>Banking service</td>
<td>0.851</td>
<td>1.267</td>
<td>0.598</td>
<td>0.692</td>
<td>0.495</td>
<td>3.903</td>
</tr>
<tr>
<td>Online Shopping</td>
<td>Online Shopping</td>
<td>Security service</td>
<td>0.912</td>
<td>1.160</td>
<td>0.620</td>
<td>0.786</td>
<td>0.477</td>
<td>3.955</td>
</tr>
<tr>
<td>Login service</td>
<td>Login service</td>
<td>E-Medical</td>
<td>0.865</td>
<td>0.899</td>
<td>0.712</td>
<td>0.683</td>
<td>0.495</td>
<td>3.605</td>
</tr>
<tr>
<td>Medical Service</td>
<td>Medical Service</td>
<td>Best Doctor service</td>
<td>0.785</td>
<td>0.963</td>
<td>0.617</td>
<td>0.672</td>
<td>0.483</td>
<td>3.522</td>
</tr>
<tr>
<td>Translator</td>
<td>Translator</td>
<td>Search service</td>
<td>0.956</td>
<td>1.175</td>
<td>0.683</td>
<td>0.625</td>
<td>0.492</td>
<td>3.931</td>
</tr>
</tbody>
</table>

Fig 9.a Logic Discovery time

Fig 9.b SLA Negotiation Time
In this paper we presented a powerful framework based on Multi Agent System for dynamic service integration which facilitates enterprises to share their service logic more sophisticatedly and securely with their business network partners. The proposed framework integrates the required service logics robotically without developer’s intervention at any stage. Agents proposed in this paper effectively identify dependency between the service logics and helps to integrate potentially. Also the model evaluates various properties such as computability, completeness, and configurability to precede the integration process efficiently. Mathematical methodology and algorithm for property evaluation are discussed in detail. Also the agent automatically evaluates works and pictures it graphically. This proves that this would be a standard platform for service providers to share their resources sophisticatedly and securely.

REFERENCES


