A Novel Approach for Business Logic Evaluation Model in Web Service Environment

Divya.A1 and V. Renuka2

1 Department of Computer Science and Engineering, IFET college of Engineering
Villupuram, India

2 HCL technologies,
Bangalore, India

Abstract
Many business services need the dynamic adhoc solution for incorporating the emergency changes in business logic frequently in order to meet the customer’s demand in the ever-changing business market. This requires an urgent need of a dynamic model to facilitate the business analyst to do the business critical emergency changes reliably and sophisticatedly by themselves without depend on the IT staff. In this paper we propose an agent based approach for incorporating the changes on the business logic by evaluating the dependency relation between the business rules, functions and parameters with respect to the business policy constraints. Further a business logic evaluation model, which provides the grounding semantics to support the automation of change management through agents with a set of change operators that allows specifying a change in a precise and formal manner. It also evaluates the consistency of changes made at service business logic with respect to the change requirements, policy enforcement and the service performance. For analyzing the dependency we use Finite State Machine (FSM) which acts as a knowledge base and provide intelligence for the agents. Hence the business logic evaluation model automates the dynamic changes through the agents that consistently monitor and control the changes. Thus this work caters a new dimension in business logic management especially in service computing environment by significantly enhancing the efficiency of business logic change through the agents.

Keywords: Business Logic Management, Change Management, Agent Coordination, multi-agent systems, Computational Complexity, Business Logic Evaluation Model, Finite State Machine, Business Logic Automation, Context analysis.

1. Introduction

A web service provides a platform-independent protocols and standards for exchanging data between applications by establishing a communication interface between service providers and consumers through which programmable application logic can be accessible via standard web protocols such as SOAP and XML. Consumers of a web service do not need to know anything about the underlying service platform, business object model or programming logic used to implement the service; they only need to understand how to call the service and how to customize it for their requirements. An agent is an intelligent process powerful enough to manage the computational resource which can act as a proxy for those entities (human, organization) that own the service. However services might be changed or modified and may undergo changes during its life cycle. Hence there is need for effective change management framework in ever-changing business areas to satisfy the requirements as and when needed. The process of change reaction could be very complicated, which makes it impractical to manage changes in a manual way. Therefore, a systematic framework with the approach of agents is required to provide a disciplined support for change management in web services and managing these changes without affecting the business functionality. Agents can be used to deal with these changes propagated during design time or even during run time. There are different approaches for managing changes in long term composed services at different levels. The relation between agent and service is not static but dynamic and will be established during runtime. Changes can be propagated either at service developer’s side or at service provider’s side. Change management plays a critical role in business process as the necessity for change increases with the market demand and technology. But there are some changes which are critical i.e., these changes have to be satisfied immediately as per user requirements and within the specified constraint. Though there are many existing change management approaches to satisfy the normal changes, they fail to support the dynamic changes within the business constraint or with respect to business policy. Hence there arises a need for supporting these dynamic changes in a flexible manner. In order to sustain the dynamic changes at run time, a framework is needed that supports runtime changes along with the evaluation of impact made by those changes. Agents are capable of exhibiting flexible problem-solving behavior in pursuit of their design objectives. The necessary measures often depend on the change that has been carried out on the service provider side where changes can be mainly classified into two ways – Internal changes and External changes. Internal changes to a web service are propagated by the service developer where external changes are the changes made by the third party service providers based on service level agreement. And also the order of execution, time/space complexity analysis and execution
functions, parameters and their relations as business logic decomposition of the business logic into business rules, platform. The core functionality behind this model is business analyst as a service provider under source control efficiently managing the web service business logic by the source manager also acts as a repository for the change analyzer and agent by providing sufficient knowledge about changes. The corresponding logic is decomposed into rules, functions & parameters using business logic analyzer. The software agent implementing the web service mainly depends on four constraints--agent id, behavior, process and business logic set. However agents differ from web services in many ways as far as the development mode is concerned. Web services have traditionally been transient and stateless processes that exist only during service execution. An agent, however, is often persistent and resource bound, providing only a single service to its peers at any given time. Web services implemented with the help of agents have distinguished advantages as agents are problem solvers, proactive, goal-oriented context-aware, and autonomous. They also can be used to identify problems and solve them with clear workflows and well-formed logics. Agents are proactive i.e., they are capable of solving problems both reactively and proactively where reactives indicates that the particular problem is solved within time constraints and proactive problems are long term goal oriented approach. And the context aware agents are situated or embedded in a particular environment over which they have partial control and observability—they receive inputs related to the state of their environment through the initiators and act on the environment through decision making and knowledgebase system.

Initially, business logic model was developed for efficiently managing the web service business logic by the business analyst as a service provider under source control management system. Further the model was extended to share the business logic across the enterprises through ESB (Enterprise Service Bus) in B2B service integration platform. The core functionality behind this model is decomposition of the business logic into business rules, functions, parameters and their relations as business logic set which can be processed by the finite state machine for dynamic management of the business logic in service computing environment.

2. Literature Survey

In [1] author defines the concept of the SOA system with agent by introducing the problem oriented approach to manage the recovery action and applied for management of the integrated rescue system. Qianhui Liang presented the XML Markup Language web ontology Language-Business process customization (OWL-BPC). It describes a framework to deal with inconsistencies and misalignment [2]. M. Brian Blake developed the several natural language processing approaches such as TSM-L,TSM-P,TSM-LP by analysis the existing web services. By using TSM-L to provide the largest percentage of valid predictions [3]. On Lutz Lowis presented ATLIST a new vulnerability analysis method with improved transferability for secure business process and services. By enabling the identification of vulnerabilities, ATLIST supports the identification and assessment of risks [4]. In [5], [6], define the Business Process Execution Language (BPEL). But In [5] author defines self-supervising BPEL processes that assess their behavior and react to monitoring and recovery. To introduced WSCoL and WSRel, provide a complete and coherent solution able to address many diverse business domains and user needs. Finding useful services is very complex in several applications. In [6] develop the BPEL ranking platform to find service repository and describe the structure based discovery can be reformulated by service discovery. It can be reduced the problem of service behavioral matching to graph matching problem. Wenli Dong develops and implements the multi-agent test environment for BPEL-based Web Service composition. To define and analyze the ontology based on XML which provides on test case generation and test evaluation. [14]. To overcomes the fast shifting e-commerce trends Tae-Hyung Kim introduces a design methodology, Service-Oriented Design with Aspects (SODA). Using aspects and services is represented as basic Petri Nets leads to a reusable, customizable, maintainable, and traceable design [7]. In [8] the standard WS techniques, fail to realize dynamic WSDi, as they rely on nonfunctional service attributes. The key to automation of WSDi is the incorporation of semantics into a richer WSDM. It should encompass functional and nonfunctional aspects like QoS. Jocelyn Simmonds described the framework for runtime monitoring of Web service and allow UML 2.0 SDs ensures that the framework is usable by practitioners to specify safety and liveness properties [9]. In [10] presented a framework based on a combination of multi-agent and case based reasoning (MACE-SCM) to
address the different uncertainty situation and facilitating collaboration between firms within the supply chain. Mohammed Ketel proposed an integration framework for web services and mobile agent in wireless networks. Combining both technologies provides reliability and ease of use for any user [11]. Sandeep Kumar Garg proposed a multi agent based semantic web services composition to address negotiation conditions, input validation and proper selection model in the service composition process. To present a selection models to perform rating of the agents based on their QoS parameters [12]. In [13] presented an agent based approach for the coordinated negotiation architecture and renegotiation of QoS constraints for service compositions to guarantee end-to-end QoS. Yongsheng Ding represent the web services by introduce the bio-entity. By constructing web services emergent agent to obtain the desirable characteristics and compositing of Web services and also build its simulation platform to analyze the flexibility of WSEA [14]. On Weiming Shen introduced a new concepts called as ishop floor for internet enabled intelligent shop floor, web and agent technology to changing shop floor environment and customer demand [15]. The author [8] defines to enable functional and non-functional attributes of web services. In [16] show how the concept of utility function policies can be used to define over functional and non-functional properties In [17] defines web services intelligent agent technology presented for Family Wealth Management (FWM) problems such that complex, dynamic and distributed process which requires the system has a high degree of cooperative problem-solving capability. On T. Madhusudan proposed the simulation-based approach for scheduling web service execution and presented experimental studies to highlight the utility and to cope with the dynamics in a web services environment [18].

3. Business Logic Evaluation Model

Business Logic Evaluation Model demonstrated in fig1 provides an abstract framework which guides to manage the existing business logics and to shape up in different forms for persuading various complex demands. The major task of the model is to cope up the existing logics, adapt meaningful changes into it and amend into different form for swaying various requisites. It accomplishes these tasks through six chief components such as Logic Analyzer, Dependency Analyzer, FSM simulator, Agent Coordinator, Context Analyzer and Schema Generator.

Business Logic Analyzer is mainly to analyze the given request and to ascertain necessary rules and functions to process the requisite. Context analyzer here discovers the environment which the given request correlated to. It grasps the request, slices it into numerous parts, scrutinizes each part as a separate keyword and ascertains the domain which the detained request related to. In the ascertained domain it traces required business logics to process the request. If it is the first time for logic analyzer to access the predicted business logic, it dissects the logic into business rule, function and domain variable set. Then it builds up Business Logic schema through schema generator. BL schema depicts the logic flow obviously. Then it uncovers required rules and function set in the business logic. Next time while accessing the logic, Logic Analyzer recognizes the required business rules and functions of the isolated logic directly from rule/function repository. The repository also contains service name with its ID for each sustained rule. Through which source manager locates business rule in the business logic.

![Business Logic Evaluation Model](image)

Dependency Analyser discovers dependencies with the ascertained rules and functions in the business logic through the identified change criteria. The presence of this FSM simulator is for the automation of the business logic set for its abstraction and completeness. For
constructing conceptual model for the business logic, Turing Machine(TM) computational model has been incorporated and the dynamic behavior of the logic change is observed with the help of state transition table (STT) constructed from the transition function with respect to the change scenario. It basically acts a knowledge repository for the agents.

Change Analyzer is to let authorized one to do meaningful changes, monitor and manage the changes. It initially decides whether the change can be handled at run time. If that indeed is the case, the change has to be planned, for example through the help of BL Analyzer. The proposed model guides business analyst and other management people to do modification at BL schema level by considering all the run time issues and also business policy constraints. The actual execution of the change leads to the creation and/or alteration of code and when this change is propagated it probably causes other code fragments to change as well. Change Analyzer scrutinizes the validity of modified rules, functions and parameters through TM simulator. Finally, the source manager accepts the valid changes and automatically builds and deploys the service within the reaction time and closes the change request entry in the change log. In association with the dependency analyzer it assesses the business logic through methods such as Order of execution, Time complexity & execution plan. In that, the order of execution ensures the work flow of the business logic along with the consistent updating of the logic sequence with respect to the changes. The Time complexity measure is used to predict the approximate complexity nature of the changes. Execution plan contains the set of estimated execution path for rules and functions.

The Agent Coordinator is one of the major component which is responsible for analyzing the change request and triggering the specified agent for processing. It is also responsible for handling the change request that involves more than one dependency set. Thus it handles and coordinates all the three agents and act as a multi-agent system. Each Agent A has its own set of parameters as goals, behavior, knowledge & context. The context analyzer provides the environment under which the agent processes the request. Finally the Run time manager is responsible for building and deploying the services after fruitful changes and also it helps to debug the exceptions in the service logic sophisticatedly.

4. FSM Based Change Measure with Agents

The rules and functions under logic form a dependency group which is analyzed and verified for every change initiated externally. Those dependency groups assist in performing changes and results in an effective change. Those changes that are made based on the dependency analysis are measurable, complete and finite. Any change that is not based on the dependency analysis is said to violate the outlined properties and may result in a failure. Dependency analysis based on each factor is performed by a finite state machine (FSM). The FSM simulates the pattern based on the behavior of the rules & functions, which can be easily verified and measured at each stage.

Algorithm Change_Evaluation (Rule, Function, Parameter)

```
//Input : Change_Request
//Storage: RuleSet, FunctionSet, ParameterSet, DependencySet
//Output: Change_Report

//Methodology: Analyze dependency and process changes
//Agent: A{B,K,G,C} \{A_0,A_T,A_E\}

Begin
//CR be the ChangeRequest that contains set of rules to be changed
   CR \rightarrow Fetch the rule to be modified
   //map the actual rule in the Ruleset with the CR
   Start \rightarrow start_state
do
   state \rightarrow get current_tapecontent()//move tape
   next \rightarrow next_tape
   state \rightarrow next
   While (state!=CR or state!=EndState)
   if (CR \in RuleSet) then
      D \rightarrow FindTransition(CR ,state, TransitionTable)
      Fetch the related functions & parameters for the CR
      //Analyze the dependency relation based on the change factor(s)
      For each rule in dependency set
         Begin
            trigger agent A_j
            Update the parameters B,K,G,C
            if (CR \in Dependencyset) then//map the CR with the Dependencyset
               //Analyze dependency with associated rules in the set (policy constraint)
               Modify B,K,G,C
            //Discard changes
            Else
            Verify context
        End
   //Measure change
   Current_state := prev_state
   \Delta C := abs( New_state – prev_state)
   Current_state := new_state
   Output(\Delta C) //submit report
End
```

The implementation of FSM for analysis makes the changes measurable, complete and finite. The Complete Logic is decomposed into rules, functions & parameters and they are grouped under their respective tapes for easy retrieval and mapping. Each one of them has a finite state control that controls the movement of the pointer through
the tapes and helps in traversing the tapes all along the way in the Turing machine. For example if there is a request for a particular rule (say \(R_3\)), then that request is send to the rule set that has the finite control over the rule tapes. The requested rule is fetched from the tape by moving the pointer one by one over the tape. This process is similar for functions & parameters.

In case of a service request for a rule including all, the request is send to each control and retrieval of service including the rule, function & parameter are done easily. Each rule & function based on the three factors (execution order, Time complexity, and execution path) are grouped in the dependency set. This dependency set is further decomposed into dependent rules, functions & parameters. They play a vital role in optimizing the changes through agents which are classified as \(A_O\), \(A_T\) & \(A_E\) that are concerned with the order of execution, time complexity & execution path. The extent of changes made and their impact over the logic are analyzed by those agents, which perform an analysis over the changes by verifying the old state and new state of the logic. They are used for submitting reports over the changes and they provide a satisfactory acknowledgement to the user based on the changes made.

An Agent \(A \{B,K,G,C\}\) is encompassed with the set of behavior \(B\), knowledge \(K\), goals \(G\) & context \(C\) that are responsible for autonomous processing of the change request within the constraint. The parameter \(K\) is a subset of \(A (K \subseteq A)\) and the dependency set \(D (K \subseteq D)\) of the logic \(L\). The agent using the parameter \(K\) process the dependency existing between the rules, functions and parameters in the logic. The parameter \(B\) determines the behavior of the agent in which the reaction of the agent for a particular set of input request is analyzed. The parameter \(G\) sets a predefined group of goals or tasks to be performed based on the request it analyzes or the initial action it encounters.

The agent receives the input it has to process and analyze it using the knowledge set \(K\) and works autonomously to achieve the required goal \(G\). The context set \(C\) determines the context of the service request and process it according to that value. The context plays a vital role mainly in the runtime and long term service handling. Thus in general the integration of all the defined parameters \((B,K,G,C)\) ensures the functionality of an Agent. The Algorithm Change Evaluation provides an effective procedure for change evaluation and management in a FSM using agents.

5. Dependency Analysis Methods

5.1 Dependency Analysis by Time Complexity

In a business logic \(L\) encompassing set of rules \(R\), functions \(F\), parameters \(P\) & dependency \(D\), the dependency \(D\) can be determined based on the time complexity of a business logic \(L\). In a normal structural programming language, the time complexity of a logic depends on the time complexity of its corresponding rules and functions. This factor is extended further and used in the business logic for analyzing the dependency. A logic is said to satisfy the run-time management, if the logic executes in polynomial time.

![Fig 2. Evaluation Model of Time and Space Complexity](image)

The major components of the above time/space complexity model are change analyzer, Agent\((A_T)\) context analyzer, time analyzer & computational engine. Whenever there comes a change request, the source manager searches for the logic where the change is to be made. Then the request processed to the analyzer which contains four relational sets namely Rule set, Function set, Parameter set and Dependency set. The analyzer evaluates whether the change request exists in the dependency set and whether that particular rule / function / parameter exists in their corresponding set where the change is to be made. Then the request is transferred to the FSM simulator that generates the FSM for the corresponding change request using the STT(state transition table) and passes the generated FSM to the run-time manager that has change analyzer to determine whether the change request can be implemented in run-time. The context analyzer is involved.
in analyzing the environment under which the Agent is processing and if the request is given in runtime then it moves the control to the runtime manager. After the evaluation of the change request by the run-time manager the control is transferred to the computational engine. The computational engine tells whether the change to be made is with respect to time or storage space. Then the change request is passed to the criterion function that evaluates whether the change request exceeds the maximum constraint, because beyond the limit the change can’t be made. After the changes takes place the control moves to the runtime manager. After the change is made the change made is measured and places the change in the relational set.

5.2 Time Complexity Analysis by FSM

If for every input business logic of ‘n’ business rules, M makes at most T(n) moves before halting, then M is said to be a T(n) time bounded Turing machine, or of time complexity T(n).

Let \( L = (\gamma_1, \gamma_2, \gamma_3, \gamma_4, \ldots, \gamma_n) \) be the set of business logics, \( R = (a_1, a_2, a_3, a_4, \ldots, a_n) \) be the set of business rules, \( F = (\beta_1, \beta_2, \beta_3, \beta_4, \ldots, \beta_n) \) be the set of business function associated with the rules. Then the time complexity of the function \( f(\beta_1) \) is denoted by \( t(\beta_1) \).

![Time Complexity of Business Logic](image)

Let us assume that \( \beta \in \alpha \) and \( \alpha \in \gamma \). Assume that ‘\( \gamma \)’ has some ‘\( \alpha \)’ rules and contains some ‘\( \beta \)’ functions, then the time complexity of the rule \( 1(\alpha_1) \) is given by,

\[
T(\alpha_1) = t(\alpha_1 \beta_1) + t(\alpha_1 \beta_2) + \ldots + t(\alpha_1 \beta_n)
\]

Upto ‘n’ rules, \( T(\alpha_i) = \sum_{k=1}^{n} t(\alpha_i \beta_k) \)

Since a business logic can contain ‘n’ number of rules, time complexity of business logic is, \( T(\gamma_1) = t(\alpha_1) + t(\alpha_2) + \ldots + t(\alpha_n) \)

Since there are ‘n’ business logic, the total time complexity is given by,

\[
T(\gamma_n) = \max[\sum_{k=1}^{n} t(\gamma_k \alpha_1) + \sum_{k=1}^{n} t(\gamma_k \alpha_2) + \ldots + \sum_{k=1}^{n} t(\gamma_k \alpha_n)]
\]

Table 1. Computational Result of Time Complexity

<table>
<thead>
<tr>
<th>Logic</th>
<th>Rule</th>
<th>Function</th>
<th>Time complexity of each Function</th>
<th>Time complexity of each Rule</th>
<th>Time complexity of each Business Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \gamma_1 )</td>
<td>( \alpha_1 )</td>
<td>( \beta_1, \beta_2, \ldots \beta_f )</td>
<td>( t(\beta_1), \ldots, t(\beta_f) )</td>
<td>( t(\alpha_1) = \sum_{k=1}^{f} t(\alpha_1 \beta_k) )</td>
<td>( T(\gamma_1) = t(\alpha_1) + t(\alpha_2) + \ldots + t(\alpha_n) )</td>
</tr>
<tr>
<td>( \gamma_2 )</td>
<td>( \alpha_2 )</td>
<td>( \beta_1, \beta_2, \ldots \beta_f )</td>
<td>( t(\beta_1), \ldots, t(\beta_f) )</td>
<td>( t(\alpha_2) = \sum_{k=1}^{f} t(\alpha_2 \beta_k) )</td>
<td>( T(\gamma_2) = t(\alpha_1) + t(\alpha_2) + \ldots + t(\alpha_n) )</td>
</tr>
<tr>
<td>( \gamma_r )</td>
<td>( \alpha_r )</td>
<td>( \beta_1, \beta_2, \ldots \beta_f )</td>
<td>( t(\beta_1), \ldots, t(\beta_f) )</td>
<td>( t(\alpha_r) = \sum_{k=1}^{f} t(\alpha_r \beta_k) )</td>
<td>( T(\gamma_r) = t(\alpha_1) + t(\alpha_2) + \ldots + t(\alpha_n) )</td>
</tr>
</tbody>
</table>

5.3 Time Complexity Processing through Agents

An Agent \( A_T \), along with the FSM as the processing component does the time complexity analysis for the rules & functions under the given logic. Any change request that involves in the alteration of the time complexity of the rules would be triggered to the agent \( A_T \), through the agent coordinator \( A_C \). Now the agent \( A_T \) takes in charge for processing the request. It starts processing the request using the FSM through series of moves in the tapes. The set of dependent rules & functions in terms of time complexity act as a knowledge repository for further processing of the request and decides the actions to be taken for the change request given in the logic. Also the context analyzer helps in supplying the sufficient details about the environment in which the agent is working.
Decisions on the change request are handled efficiently through the information such as environment and nature of the request such as runtime changes and long term composed services.

Here are the methods through which the dependency among the rules based on the time complexity is analyzed and added as a knowledge to the agent A_T involved in processing the request dependent on time complexity.

5.4 Dependency based on Time Complexity

The dependency between rules in a logic L can be determined by analyzing the time complexity between them. If the time complexity T is unique for a set of rules or functions, then they are said to be dependent. Every rule of similar time complexity and processing time are grouped under one category and they are analyzed by the agent A_T during the change request processing involved with time.

**Definition:**

If there exists a common polynomial order time complexity or similar pattern of computation between two or more rules, then they are said to be dependent and form a subset in the dependency set. (i.e.) If \( \{ T(R_1) = T(R_2) = \ldots = T(R_n) \} \), then the value dependent on functions in the rule set with \( c \) as a constant then \( \{ R_1, R_2, R_3, \ldots, R_n \} \) are said to be dependent and \( \{ R_1, R_2, R_3, \ldots, R_n \} \subseteq D_T \), where \( D_T \) is the set containing the rules of similar time complexity.

**Proof:** Let us consider ‘n’ rules \( R_1, R_2, R_3, \ldots, R_n \), of same time complexity \( T(R) = k \), where \( k \) is a constant. If there are \( x \) functions in each rule, then (\( \forall \) rules under the logic L) where \( n \) is the value dependent on functions in the rule set with \( c \) as a constant then \( \{ R_1, R_2, R_3, \ldots, R_n \} \) are said to be dependent and \( \{ R_1, R_2, R_3, \ldots, R_n \} \subseteq D_T \), where \( D_T \) is the set containing the rules of similar time complexity.

Every rule has at least one function \( F \) such that \( T(F) \) is the maximum. So there exists at least one function in every rule which have equal time complexity say \( k \). Those functions are composed of parameters & inputs and they are having common time complexity. So there exists dependency among them. Hence the rules which are composed of those dependent functions are said to be dependent and they are included in the set \( D_T \) for analysis by agents.

5.5 Upper bound and Lower bound Analysis

Apart from the normal analysis and storage of rules of similar time complexity the concept of optimality, upper bound and lower bound analysis makes the retrieval of dependent rules more specific and reduces the complexity in analyzing the given change request. Those parameters are discussed in the following section as follows.

Every rule need not execute all its functions for every input & parameter. For example in a login process, if the entered username is not correct, then the system need not verify password. Hence the function verify password remains idle for invalid username and remains in the same state. In this case, the time & space complexity of login process, encompassing the functions username & password verification varies. Also in some cases, there may be increase in the complexity due to the addition of new function to the existing rule. Hence the time complexity of a rule can be classified based on the number of functions it contains as lower bound and upper bound. Lower bound of a rule is defined as the minimum number of functions that are needed to execute the rule. It is denoted as \( T_L(R) \). Upper bound of a rule is defined as the maximum number of functions that are required to execute that rule. It is denoted as \( T_U(R) \). The rules are classified based on this criteria in the dependency set \( D_T \). If \( T_L(R_1) = T_L(R_2) \) then the rules \( R_1 \) & \( R_2 \) are said to have same lower bound and similarly for \( T_U(R) \). Thus \( T_L(R), T_U(R) \subseteq D_T \).

5.6 Optimality

If the upper bound & lower bound of a rule are equal, then that rule is said to be optimal. (i.e.) \( T_L(R) = T_U(R) \). Thus optimal rules under a given logic \( L \) form a dependency. Rules that are dependent based on the optimality have the capability of executing any number of functions & there is no variation of complexity for these optimal rules.

**Definition:** If a rule is composed of set of functions of equal time complexity, then that rule is said to be optimal. (i.e.) if \( T(R_1) = T(F_{1b}) = T(F_{2b}) = \ldots = T(F_{nb}) \). Then \( T_L(R_1) = T_U(R_1) \). Where \( F_{1b}, F_{2b}, \ldots, F_{nb} \) are set of functions under \( R_1 \).

**Proof:** Let \( R_1 \) be the rule and \( F_{1b}, F_{2b}, \ldots, F_{nb} \) be the functions under it. To prove that \( R_1 \) is optimal, it is enough to show that \( T_L(R_1) = T_U(R_1) \) for all functions in \( R_1 \). This can be proved by the induction method.

**Basis step** Since a function is the basis for any rule & a rule cannot be empty, let us consider a single function \( F_{1b} \) under \( R_1 \).

Then, \( T_L(R_1) = T(F_{1b}) \) \( T(U(R_1) = T(F_{1b}) \) since there is only one function, \( T_L(R_1) = T_U(R_1) \) holds for one function.
Now consider a rule $R_\lambda$ with ‘n’ functions. Then,

$$T_L(R_\lambda) = T_U(R_\lambda) = T(F_{1\lambda}) = T(F_{2\lambda}) = \cdots = T(F_{n\lambda}) = k \text{ (by induction)}$$

Now to prove the above condition is true for any set of functions, consider $R_\lambda$ with $n+1$ functions, consider

$$T_L(R_\lambda) = T(F_{1\lambda}) = T(F_{2\lambda}) = \cdots = T(F_{n\lambda}) = T(F_{(n+1)\lambda})$$

Hence, $\min(k + T(F_{(n+1)\lambda})) = \max(k + T(F_{(n+1)\lambda}))$

This is possible only if $T(F_{(n+1)\lambda}) = k$. Thus the rule $R_\lambda$ is said to be optimal if the time complexity of all its function are same. In the above concepts and methods specified, the agents are more specific and they help in easy retrieval of the dependent rules that are associated with the given change request for processing.

5.7 Bounded Changes

In general for any rule $R$ of size $n$, the time complexity is given as $T(n) = n^a + D$. $T$ is a polynomial function dependent on $n$. This function is said to attain a maximum or minimum value for a particular value $n$. In order to obtain the maximum & minimum value of $T(n)$ for a particular rule, the function has to satisfy the conditions based on the Extreme value theorem.

Definition If a function $f$ is continuous on a closed interval $[a,b]$ then $f$ attains an absolute maximum value $f(c)$ and an absolute minimum value $f(d)$ at some numbers $c$ and $d$ in $[a,b]$. By applying the above condition on the function $T(n)$, we can calculate the maximum and minimum value for $T$.

Before change,

$$T(n) = n^b + D \text{ (where } n^b \text{ is the order of complexity before change)}$$

Since $T$ is a polynomial function on $n$, it is continuous on $[1,m]$. By condition $1$, $n = 0$, so $T(0) = D$. On $n = 0$, $T = D$ (i.e. the time complexity of a rule is a constant one if there is no input applied to it). At extreme points $1$ & $m$, $T(1) = D + 1$ and $T(m) = m^b + D$.

Thus by overall analysis, $T(n) = n^b + D$ (where $n^b$ is the order of complexity after change)

$$T(n) = n^b + D \text{ (where } k \text{ is a constant depending on the changes)}$$

Since $T$ is again a polynomial function on $n$, it is continuous on $[1,m]$ and differentiable

$$\frac{dT(n)}{dn} = \frac{kb(n^{b-1})}{dn}$$

On applying $\frac{dT(n)}{dn} = 0$, $n = 0$ and $T(0) = D$, $T(1) = D + 1$,

$$T(m) = (m^b + D)$$

By analysis $T(n) = \begin{cases} 
D, & \text{if } n = 0, \text{(min)} \\
D + 1, & \text{if } n = 1 \\
m^b + D, & \text{if } n = m, \text{(max)} 
\end{cases}$ After change)

From the result, we can understand that the complexity of the function remains the same even after changes for the minimum input of the function.

6. Dependency Analysis by Order of Execution

In a business logic $L$ encompassing set of rules $R$, functions $F$, parameters $P$ & dependency $D$, the dependency set $D$ can be determined based on the order of execution of rules & functions. In a normal structural programming language, the order of execution depends on the control, branch & functions. In an object oriented programming, the message sequence determines the order of execution. This concept is extended further and used in the business logic for analyzing the dependency.

A logic is said to be executed successfully, if each and every rule and function under it executes in order. Any change in the order of execution of rules is mapped in the

![Evaluation Model for Order of Execution](image-url)
dependency set of that rules. The order of execution is one of the dependency evaluation methodologies in change management that assists in the enabling changes through agents. In response to the changes from the analyst, the source manager sorts out the required logic. The source manager also acts as a repository for the change analyzer and agent by providing sufficient knowledge about changes. The corresponding logic is decomposed into rules, functions & parameters using BL analyzer. Then the requested change is fetched from the corresponding ruleset or function set. The fetched rule or function is analyzed with the dependency set for consistency. The transition function for that change is analyzed and an equivalent FSM is generated by the FSM simulator. The next state of the particular rule or function is predefined using the FSM state transition table. The Agent(AO), with the set of parameters A{B,K,G,C} is the major component of the model that utilizes this STT and provides the appropriate control flow in the logic. The resultant model is analyzed for its complexity such as space & time analysis and verifies the pre-existing constraints through the context analyzer. As a result, the change request is computed and the impact made through the changes are measured in the computational engine. Finally the changes are added to the change analyzer and dependency set for further changes in the future for the agents. After changes in the order of execution, the STT is modified according to the change request. Also those changes are registered and stored in the dependency set for evaluation and monitoring.

6.1 Example for finding Order of Execution

For example consider an airline ticket reservation system. The various rules under this system are login, book_ticket & billing. The login process provides the appropriate authentication by various functions such as username & password verification. The book_ticket provides three functionality as flight details, availability_check & schedule_verification. The billing provides the functionality of card_validation & amt_calculation. In this system, the book_ticket is executed only after the completion of authentication in the login process. Thus the rulebook_ticket is dependent on the login process. Thus only after the execution of functions in login such as username verification & password verification, the rule book_ticket can continue. Similarly the billing process can be done only if the required seats are allocated in the book_ticket. Hence billing is dependent on the functionality of book_ticket.

**Schema for airline ticket reservation**

```xml
<login>
  <username>
    <verify_username type="function", id="F11">
      <Username type="parameter", id="P111"/>
    </verify_username>
    <verify_password type="function", id="F12">
      <password type="parameter", id="P121"/>
    </verify_password>
  </username>
</login>
</AIRLINE_TICKET_RESERVATION>
```

6.2 Dependency Analysis Approach by Finite Automata

The existence of dependency through order of execution can be analyzed through a finite state machine that contains set of rules & functions as its finite state and transitions between the rules denotes the dependency through the order of execution. In the logic level, the set of rules form the finite states with functions as transitions. A rule is said to be dependent in this model, if there is a transition from one rule to another on accepting a function. Level of dependency can be even extended to the functions where the parameters act as the transition from one function to another. If M is a finite state machine then it defined formally as M(ω, α, ω0, σ, E) Where, ω is finite set of states (i.e.) rules, α as finite domain set, ω0 as initial state or rule, σ transition function and Hasan end state. In the context of the business logic, this end state is
considered as an exception state. There exists a transition from any state to the exception state E, if the rule or function encounters an exceptional input or parameter denoted as e.

![Finite State representation for login process](image)

**Fig 4.** Finite State representation for login process

![FA representation for rules in the given schema](image)

**Fig 5(a).** FA representation for rules in the given schema

**Table 2.** Transition table for FA representing rules

<table>
<thead>
<tr>
<th>Transition</th>
<th>R_1</th>
<th>R_1</th>
<th>R_2</th>
<th>R_2</th>
<th>R_2</th>
<th>R_3</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ω</td>
<td>σ(ω, F_11)</td>
<td>σ(ω, F_12)</td>
<td>σ(ω, F_13)</td>
<td>σ(ω, F_21)</td>
<td>σ(ω, F_22)</td>
<td>σ(ω, F_23)</td>
<td>σ(ω, F_31)</td>
<td>σ(ω, F_32)</td>
<td>σ(ω, F_33)</td>
<td>σ(ω, e)</td>
</tr>
</tbody>
</table>

6.3 Change Measurability in Order of Execution

The dependency analysis based on the order of execution of rules under logic helps in the rate of change in the rule & logic. Any significant change in the rule results in the following changes as Change in time complexity, Change in the dependency and Change in the functionality. Based on the analysis of above parameters, any change in the order of execution of the rule is defined as,

\[ d(R) \propto \omega \] (where \( \omega \) is a transition function)

\[ d(R) \propto F \] (where \( F \) is function corresponding to the rule)

\[ d(R) \propto 1/T \] (where \( T \) is the time complexity of the rule before change)

Combining all the three, we get

\[ d(R) = \frac{k \omega F}{n^c} \] (1)

(Where \( T \) is considered as polynomial \( n^c \) with \( c \) as constant).

Now rate of change of rule in order of execution \( \frac{\partial R}{\partial \omega} \) with respect to \( \omega \), \( F \), \( T \) is given as

\[ \frac{\partial R}{\partial \omega} = \frac{k F}{n^c} \]

Since the \( \omega \) is a transition function mapping two rules, \( F \) is a function with its own construct & expressions & \( T \) is also a polynomial function, they are said to be differentiable. Hence \( \frac{\partial R}{\partial \omega} = \frac{\partial R}{\partial F} \) & \( \frac{\partial R}{\partial T} \) said to be exist and finite.

\[ \frac{\partial R}{\partial F} = \frac{k}{n^c} \]

\[ \frac{\partial R}{\partial T} = - \frac{k}{n^c+1} \]

So, \( \frac{\partial R}{\partial \omega} = \frac{k F}{n^c} \left[ \frac{1}{\omega} + \frac{1}{F - c/n} \right] \) from (1)

The \( \omega \), \( F \), \( n \) are finite functions and hence \( \left[ \frac{1}{\omega} + \frac{1}{F - c/n} \right] \) is a constant value for various values in \( \omega \), \( F \), \( n \). Thus \( \frac{\partial R}{\partial \omega} \propto d(R) \), which indicates that the rate of change of rule in execution order with respect to \( \omega \), \( F \), \( T \) is dependent on the rule that changes.

7. Dependency on execution plan

A set of rules are said to be dependent, if the execution plan yields a similar execution path for them. The Agent \( A_E \) is responsible for analyzing those changes in an appropriate manner through the dependent set \( D_E \) containing the set of rules that follows similar execution path.
The execution path is one of the dependency evaluation methodology for processing the changes that are involved in addition of new rules or function to the existing logic. The evaluation model for the execution path is shown in the fig 6. The source manager also acts as a repository for the change analyzer and agent by providing sufficient knowledge about changes. The fetched rule or function is analyzed with the existing dependency set for consistency. The Agent (AE), with the set of parameters A{B,K,G,C} is the major component of the model that utilizes the dependency set as a knowledge and provides the logic flow. The schema parser is concerned with decomposition of the given request into rules and functions. The rule plan cache contains the set of dependent rules and functions of similar execution path. The plan cost estimator adds the cost of the rule based on the function it encloses. In case of new rule apart from the existing in plan cache, the newly added service is added to the dependency for future analysis for the agents.

7.1 Phases of Execution

The execution of a rule takes place in two phases as
1) Rule optimization
2) Plan verification

Rule optimization
In this phase, the rule optimizer will calculate the execution path for the rule represented in the schema based on already created execution plan. It will provide some number to each and every step given in plan which is called as the estimated cost for prepared execution plan.

The Agent AE utilizes the estimated cost for every rule and function and process the request.

Plan verification
In this phase, the availability of the execution plan in cache is tested. If the plan is already available in the cache called as 'rule plan cache', then consider that plan and execute the given rule. If there is no plan in the cache, then add the estimated plan to the cache for further reference. Thus the similarity in the process of execution of rules by estimated plan & actual plan can be analyzed for the dependency.

7.2 Dependency analysis

The dependency of rules based on execution plan can be analyzed by the plan dependency analyzer. It is concerned with fetching the estimated execution path of a particular rule from the optimizer and compare with the execution path of other rules under the logic L. The rules of similar path are stored in the same category and are categorized as plan dependent rules. These plan dependent rules are processed by the corresponding agent AE and executed without any recompilation, which saves the time & space for the given logic.

8. Agent Characteristics:

The agent possesses knowledge to process and sustain changes. The relation between agent and service is not static but dynamic and will be established during runtime. The following are the properties of the agents used in the logic.

Agent Autonomy: The agent is partially autonomous since it needs to be triggered by a request from another agent or user.

Agent Reactivity: The agent reacts on initial triggering by another agent or user and possible execution errors during service processing.

Agent Pro-activity: Specifying an agent to have an understanding of how to interact with the change request and exceptions it is programmed to access.

Agent Behavior: Depends on the complexity of the agent under the process. Based on the nature of analysis done and the context in which the agent operates.

Agent Actions: They are set of actions or tasks to be performed by the agent for the given change request.

Agent Preconditions: They are the set of conditions or rules that need to be true during the request processing by the agents. For example, for a change request with respect to execution order, the request should not disturb the basic policy requirements of the logic.

Agent Effect: It is the result or operation performed by the agent after processing the request without altering the policy and requirements of the service.

Table 3: Agent characteristics with dependency parameters
9. Change Analysis through Multi-Agents

Agent is usually considered as one kind of computation entities which have abilities to adapt to the environment as well as deal neatly and autonomously in order to reach a series of goals under the specific environment with those characteristics such as the autonomy, the sociality, the reactivity, initiative and so on. The multi-Agent system means the system in which many agents accomplish tasks together by mutual communication and coordination. In the multi-Agent system, agents cooperate mutually, and solve the large-scale complex problem together. It compensates for single agent insufficiency, meanwhile coordinateately deals with each member’s contradictions and conflicts between goals and behaviors through the consultations means. If there arises a change request involving multiple changes such as alteration in the time complexity, execution order & policy variation of the logic, then the agent corresponding to all the above dependent factors process together and act as a multi-agent system. This unified approach and coordination results in successful processing of the required request. In case of any exception or failure in any one of the processing agents, the whole request is discarded and the entire logic remains unaltered. Thus the multi-Agent systems provide an atomicity property for the change request in the given logic. The given change request is decomposed into the corresponding rules, functions & params. Based on the type of change made, the agent coordinator sends the request to the respective Agent for processing using the FSM simulator.

In case of a request including multiple changes, the agent coordinator does the job of information sharing and handling workflow between the various agents. This coordination is done through the FSM which specifies the information flow between the agents. The generated bit pattern based on the request and the parameter it depends is stored in a multi-tape Turing machine with each individual tape storing the bit pattern generated. Based on the bit pattern and the tape id, the agent coordinator maintains the workflow through the tape movement and comparisons between the tapes. Each tape is utilized by the corresponding agent for processing the request.

<table>
<thead>
<tr>
<th>Order of Execution</th>
<th>Time complexity</th>
<th>Execution path</th>
<th>Input request Bit Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>010</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>011</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>110</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>111</td>
</tr>
</tbody>
</table>

Fig 7 Change request representation in Tapes

10. Performance Measure

The change request and its impact over the logic should be performed within the defined time limit. Hence there arises the need for performance measure in handling the changes. The agents with its predefined knowledge and autonomous behavior made changes reliable and complete within the specified time limit. Let $T_{cr}$ denotes the total time elapsed...
for accomplishing effective changes through change criteria in the whole change management process. In general, $T_{cr}$ is the summation of time taken to identify required business rules to process the request, time taken to explore dependency at business rule, function and parameter. However the complexity may vary depending on the method involved in satisfying the change. Let us assume $T_r$ be time taken to identify list of rules, functions & parameters to process the request, $T_d$ be the time taken to locate the dependency set for corresponding rule, function or parameter and $T_m$ is the time to measure changes.

**Change Measure (Normal)**

In general $T_{cr} = T_r + T_d + T_m$

$T_d = (T_{do} + T_{dt} + T_{de})$

where $T_{do}$, $T_{dt}$, $T_{de}$ refers the time taken to observe dependency in order of execution, time complexity and execution path.

$T_{do} = \sum T_{sst} + \sum T_{sst}$ ($T_{sst}$ denotes the time taken to compute the dependency order through the STT)

$T_{dt} = \sum T_{tm} + \sum T_{tm}$ ($T_{tm}$ denotes the time taken to compute the complexity dependency by TM)

$T_{de} = \sum T_{tp} + \sum T_{tp}$ ($T_p$ represents the time taken to analyze the plan cache)

Total time taken for the whole process in satisfying the change request is

$T_{cr} = T_r + (T_{do} + T_{dt} + T_{de}) + T_m$.

**Change Measure (With Agents intervention)**

The Agent $A\{B,K,G,C\}$ encompassed with the set of behavior, knowledge, goals and context utilizes the dependency set and reduces the change processing time $T_{cr}$.

Consider $T_{cr}^a$ as the total time taken for processing the request using agents.

$T_{cr}^a = (T_r + T_K^a + T_m)$ ($T_K^a$ is the total time taken for the agents to analyze the knowledge set based on the dependency $D$)

Now $T_K^a = (T_{ko}^a + T_{kt}^a + T_{ke}^a)$

where $T_{ko}^a$, $T_{kt}^a$, $T_{ke}^a$ refers the time taken to analyze the knowledge set $K$ using the dependency in order of execution, time complexity and execution path.

Total time taken in satisfying the change request through agents is

$T_{cr}^a = T_r + (T_{ko}^a + T_{kt}^a + T_{ke}^a) + T_m$

Since $(K \subseteq D)$, the analysis of $K$ by the agent is very small when compared to the dependency set where the entire set is considered for the processing. However in the case of an agent, the agent coordinator takes care of the specified task and determines the control which takes less time than the dependency analysis. Hence $T_{cr}^a << T_{cr}$.

Based on the above measurements the following graphs are drawn comparing the processing time of the given change request in the agent oriented and normal business logic with respect to order execution, time complexity and execution path.

**Fig.8(a)**

**Fig.8(b)**

**Fig 8: Dependency ratio vs Processing time (Order of Execution)**

<table>
<thead>
<tr>
<th>Dependency Ratio ($r$)</th>
<th>Processing time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal ($r+\alpha n$)</td>
<td>Agents ($r$)</td>
</tr>
<tr>
<td>$\alpha$ is number of moves</td>
<td>$\alpha=0$</td>
</tr>
<tr>
<td>0.095</td>
<td>0.312</td>
</tr>
<tr>
<td>0.224</td>
<td>0.554</td>
</tr>
<tr>
<td>0.312</td>
<td>0.776</td>
</tr>
<tr>
<td>0.508</td>
<td>1.121</td>
</tr>
<tr>
<td>0.612</td>
<td>1.754</td>
</tr>
<tr>
<td>1.222</td>
<td>2.189</td>
</tr>
<tr>
<td>1.351</td>
<td>2.743</td>
</tr>
<tr>
<td>1.516</td>
<td>3.121</td>
</tr>
<tr>
<td>1.808</td>
<td>3.754</td>
</tr>
</tbody>
</table>

**Fig.9(a)**

**Time Complexity**

- **Agent**
- **Normal**
11. Conclusion

The proposed model provides a powerful platform to enable changes in the service business logic more sophisticatedly through agents in such a way interoperability between the services are managed with flexibility and ease. This paper identifies various dependency properties needed for integrating and managing changes in the business logic for agents within the specified constraint. The concept of multi-agent system and their coordination methods are discussed with the logic of FSM in service. The model is also indulged in policy mapping and change measure in such a way that the whole logic remains undisrupted. We have also analyzed the QoS through the performance measure made on agents in handling changes. This provides a clear outline for the service providers to integrate the business logic and manage the changes. Thus a flexible service manageability is supported within the logic for changes.

References

[3]. M. Brian Blake, Senior Member, IEEE, and Michael F. Nowlan,” Knowledge Discovery in Services (KDS): Aggregating Software Services to Discover Enterprise Mashups”, IEEE Transactions on Knowledge and Data Engineering, vol. 23, no. 6, pp. 889-901, June 2011
[8]. KyriakosKritikos and DimitrisPlexousakis,“ Requirements for QoS-Based Web Service Description and Discovery”, IEEE transactions on services computing, vol. 2, no. 4, pp. 320-337, October-December 2009
[9]. Jocelyn Simmonds, Yuan Gan, Marsha Chechik, Shiva Nejati, Bill O’Farrell, Elena Litani, and Julie Waterhouse"


[12]. Mohammed Ketel,” Mobile Agents Based Infrastructure for Web Services Composition”, IEEE, 2008


