Application of Adaptive Decision Tables to Enterprise Service Bus Service Selection

Fabiana S. Santana (fabiana.santana@usp.br)*#, Claudio Barberato (cbarberato@fei.edu.br)#, Renata L. Stange (rlstange@usp.br)*, João J. Neto (joao.jose@poli.usp.br)*, Antonio M. Saraiva (amsaraiv@usp.br)*

** Departamento de Computação e Sistemas Digitais – Escola Politécnica da Universidade de São Paulo – São Paulo – Brazil*
 # Departamento de Ciência da Computação – Centro Universitário da FEI – São Paulo – Brazil

***Abstract* — An Enterprise Service Bus, ESB, is an essential part of a Service-Oriented Architecture, SOA. Among other attributions, an ESB needs to manage service ranking and selection during the application running time. This is a highly complex computational problem. This paper presents a solution for this problem based on adaptive decision tables. One of the main features of the presented solution is to allow the integration of many other algorithms as adaptive functions. This is desirable because many algorithms were already proposed but with different requirements, so it is necessary to propose a solution able to integrate different contexts. Besides, a general solution must be able to support the integration requirements that are necessary to ESB and SOA. In order to illustrate the viability and relevance of the proposal, examples are also presented, along as the future researches that may be developed considering this work as a starting point.**

***Keywords* — Adaptive systems, Adaptive decision tables, Enterprise service bus, Service-oriented architecture, Adaptive algorithms.**

I. NOMENCLATURE

BPEL – Business Process Execution Language
CORBA – Common Object Requesting Broker Architecture
DCOM – Distributed Component Object Model
ERP – Enterprise Resourcing Planning
ESB – Enterprise Service Bus
FTP – File Transfer Protocol
HTTP – HyperText Transfer Protocol
IT – Information Technology
JBI – Java Business Integration
JDBC – Java DataBase Connectivity
JEE – Java platform, Enterprise Edition
POP3 – Post Office Protocol version 3
QoS – Quality of Service
RMI – Remote Method Invocation
SMTP – Simple Mail Transfer Protocol
SOA – Service Oriented Architecture
SOAP – Simple Object Access Protocol
SOC – Service Oriented Computing
UDDI – Universal Description, Discovery and Integration
WSDL – Web Services Description Language
WS-* – Family of W3C standards
XML – Extensible Markup Language
XMPP – Extensible Messaging and Presence Protocol

II. INTRODUCTION

A SYSTEM architecture may be defined as the fundamental organization of the components of a system and their relationships, among themselves and with the external environment [1]. Usually, basic principles, named as *architectural patterns*, conduct the architectural design, development and system evolution over time [5]. Architectural patterns are a main concern to IT professionals, which everyday must face the challenge to improve and maximize the usage of available resources while are steady pressured for reducing development and maintenance costs, in addition to improve the quality skills of each solution. Among the main and most expensive resources are the software programs which have already been developed.

Fig. 1 presents the architectural evolution over time, from Monolithic based to Services based solutions, as discussed in ENDREI et al. [5]. Monolithic is considered the simplest pattern while Services is considered the most sophisticated one. Monolithic applications are self-contained, single-tiered and contain the main program, the user interface and the data access code. In order to attend main concerns of IT professionals, such as reuse and maintenance, monolithic applications are usually not recommended. For instance, if a problem is found, the whole application must be revised, not only the damage part. In this sense, architectural patterns presented in Fig. 1 show evolutions to software engineering by adding layers and establishing communication among them. Distributed systems are required for the last three.

The alternatives presented in Fig. 1 usually are seen as evolutions in architectural paradigms. This is due to each solution claims to be better than previous considering the following criteria [5]: 1) Management of corporative systems; 2) Improvements in system scalability; 3) Cost reduction; 4) Collaboration and reuse of solution; and 5) Integration and interoperable skills. However, requirements of each software packages direct the architectural pattern choice.

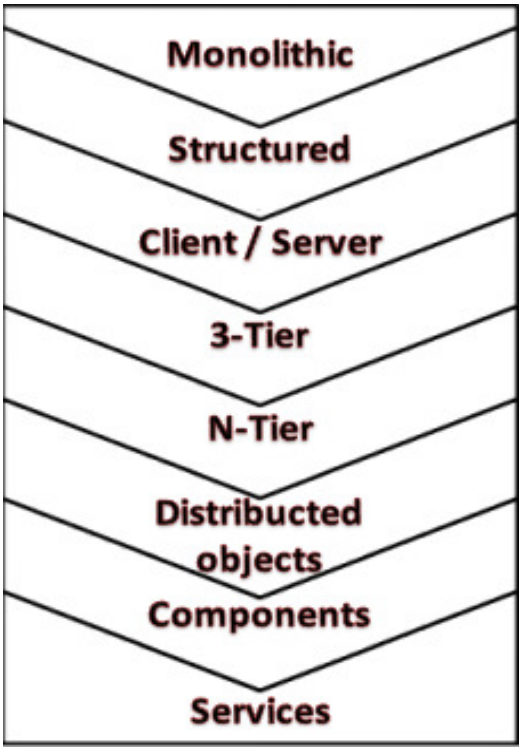

Fig1. Architectural evolution over time.

SOC is a new paradigm to develop software using services as basic units. It has evolved from component-based software frameworks, such as JEE [http://java.sun.com/javaee/], CORBA [http://www.corba.org/], .NET [http://www.microsoft.com/NET/] and DCOM [http://msdn.microsoft.com/en-us/library/ms809340.aspx]. IBM [http://www.ibm.com/], Microsoft [http://www.microsoft.com/], SUN [http://www.sun.com/], Hewlett-Packard [http://www.hp.com/], Oracle [http://www.oracle.com/], SAP [http://www.sap.com/] and other major companies in this area consider Web services as an adequate approach to SOC adoption [9].

Open Internet protocols XML and HTTP are the basis for Web services standards (e.g.: SOAP, WSDL, UDDI, and BPEL), which makes SOC adoption cheaper and easier [20] [9]. Services may be offered by applying many different technologies, but nowadays Web services are probably the most adopted one [13]. As main ideas presented in this work do not depend on any specific service technology and are applied to Web services as well as are applied to many other service technologies, this work will not discuss all of them in details unless strictly necessary, focusing on only this one to discuss the concepts presented, whenever it were necessary.

SOC, Service Oriented Computing, enables the execution of transactions across multiple platforms, providing advanced software interoperability. The construction of a system in SOC demands the integration of services in providers distributed worldwide. Thus, each application may be designed based on a business process, representing the steps to solve the computational problem. A business process may integrate, during its execution, different IT systems, available in different service providers. Process' steps may be provided by internal or external (outsourcing) service providers. Cost and performance are among the main reasons to define how a service will be provided. Once these definitions are made, one of the main challenges in SOC is related to service creation, composition and selection, over the Internet, especially in the case of outsourcing [9], where the conditions are not under the control of applications owners.

SOA is the architectural paradigm related to SOC. Applications which present requirements such as collaboration and reuse of interoperable solutions, high integration needs (e.g.: ERP systems), interoperability skills (e.g.: distributed systems) or reusable components, usually should consider SOA as architectural pattern and SOC as the correspondent development paradigm [11]. This is the case of many problems which concerns to IT professionals.

Initial investments for SOA adoption usually require a financial and computational effort which may be considerable, but, after that, the development cost of each application tends to be reduced over time, as reuse may be applied in large scale. It is also possible to implement SOA for reusing legacy systems. This strongly reduces software programming efforts [5] [11]. However, as wisely said by BROOKS [3], there are "no silver bullets" when the subject is software development and maintenance, and SOA-based systems also have their intrinsic problems, which may not be disregarded.

Most common problems are related to QoS, Quality of Service. QoS represents a wide range of issues that must be treated in order to offer a good solution to the final user. QoS issues in service technologies must consider many different properties [10] [13]. For instance, related to Web services, it is possible to enumerate: 1) Availability: time that a service keep operating, in percentage values; 2) Security: include authentication mechanisms to offer and use services, data integrity and reliability, and data confidentiality, among others; 3) Response time: time taken by a service to respond to requests which have been done; and 4) Throughput: related to the rate a service can process requests, usually as a function of time.

In addition to QoS, others issues must be considered. For instance, consider the case of non-deterministic applications, such as ecological niche modelling [19]. They are typical cases where the correctness of the results is usually more relevant than the response time, nevertheless the response time must be considered anyway. However, to describe each issue related to service selection is out of the bounds of this work and,

whenever necessary.

In an architectural perspective able to disregard the complex details of dealing with services technology, the implementation of a SOC application may be reduced mainly to choose and integrate services. This represents a simple alternative to implement applications in a very fast and simple way, as desired by many IT professionals. But there are problems intrinsic to the services nature which must be addressed. For instance, when a service is requested, it may present some problems related to QoS, such as availability; it is usually an unpredictable problem.

The service orchestration concept allows the design and implementation of software solutions by detailing the requirements (e.g.: interface, inputs, outputs) of the services necessary to the application [9]. If this was performed without to tie each service to specific providers, actions may be taken to prevent and correct application problems due to problems in the services, such as to replace a problematic service by other, in running time. Therefore, a main challenge of SOA-based solutions is service ranking and selection, so as to develop applications able to choose and replace services in running time.

In order to provide an adequate services' management, SOA-based systems use ESBs [9]. Nevertheless, a very much relevant aspect of an ESB is the service ranking and selection itself [4]. This might not be a concern of some researchers yet because services technologies are still incipient in many companies, but certainly will became a problem when many service providers start to compete with each other. When that happens, the main question will be: "How to select a service, if many different providers furnish analogous solutions?". Service selection may be provided by ranking the available services according to specific criteria and applying some selection technique among the many computational techniques.

Adaptive techniques [7] [16] may enable the construction of algorithms and a framework for service ranking and selection. They can be applied to choose the best or a subset of good services among a service list. This list may be indexed and re-indexed according to the service behaviour, considering results of their previous executions. At this point, it is important to note that QoS and other service attributes may not remain fixed over time, and a good service today may loose its quality tomorrow, so the re-indexation, or re-ranking, is a demanding task. Considering this specific aspect of the problem, adaptive technologies seem to be the most appropriate solution. Among many adaptive techniques, adaptive decision tables seem to be more adequate, since its structure is related to UDDIs, which store services information.

This work proposes a solution to construct ESBs using adaptive decision tables, which will be able to manage changes in services issues based on QoS and, considering these changes, to re-ranking the services by changing ranking and selection rules over time. Other services constraints also are considered. The paper starts by presenting some IT backgrounds, where SOA, ESB and adaptive decision tables are discussed. Then, the proposal of adaptive decision table for service ranking and selection is presented. After that, the architectural solution is presented, showing where adaptive decision tables would be included, this time in an architectural context. For the sake of comprehension, this will be explained using a factual system reference architecture. Examples to illustrate the application of the overall proposal of adaptive decision tables for service selection are also presented, and discussion and future works lead to the conclusion of this work.

III. IT BACKGROUND

A. *Service-oriented architectures*

A reference model [5] is a standard to decompose a system in parts, which must be combined to collaboratively solve the original problem. An architectural pattern describes the elements of a software architecture, the relationships among them and the restrictions to specify the fundamental structure of an application so as to obtain a complete architectural solution. A software architecture (or concrete architecture) presents the structure and organization furnished by the system, subsystems and components, and the interactions among them, in order to build systems able to fulfil identified and analyzed properties [5].

SOA is a paradigm to organize distributed competences, controlled by different providers or not [5], where reuse may be more possible than imagined before. Competences, in the system development context, are usually developed to solve problems when they appear. However, in a distributed scenario, a part may have needs compatible with competences already developed by others. Since the competences may already be available, SOA proposes services as the technology to provide that, and the required details to use them are known, a part may use the competences furnished by the other [5]. So, it is possible to define a service as the competence to execute jobs for others.

More specifically, SOA assumes that applications provide functionalities as reusable services [14]. A service is a self-contained component which may be accessible through a standardized and pre-defined interface. Any application may implement and offer services which may be used by other applications. Therefore, it is possible to implement complex business process by combining services from different sources, which is named as service orchestration.

To offer a service, a service provider must register it on an UDDI (or equivalent), which is a central naming

service. Then, when a service was requested, consumer applications have sources to look for available services, retrieve information about connection to the respective service providers, and get service description necessary to define the usage skills of a service. Any service-based technology (e.g.: Web services, SOAP) may be used to implement SOA.

A SOA-conformity solution must [5]: have entities identified as services, according to the reference model definitions; define how visibility is established among services consumers and providers; identify how to mediate interaction; be able to understand how effects of services will be understand; associate description with services; be able to identify the execution context required to support interaction; and identify policies and contracts.

B. Enterprise service bus

The construction of applications based on service orchestration is possible in a distributed scenario even when the service providers do not use the same operating system, programming languages or data models. This flexibility is fundamental for integration, since it enables the link among very different systems and environments.

Therefore, a powerful solution to integrate services based on open standards and able to support SOA is required, and this solution is usually named as Enterprise Service Bus [14]. Thus, an ESB needs to deal with open standards and infrastructures to integrated distributed systems, which requires: 1) Service routing; 2) Service invocation and mediation; 3) Abilities to integrate distributed applications and services with reliability; and 4) Security skills.

ESBs main components are routers, transformers, adapters and bridges, so as to integrate and interoperate applications over different middlewares (software to connect other software components), providing communication skills. [14].

In terms of resources, an ESB should offer support to SOAP, WSDL – Web Services Description Language, UDDI – Universal Description, Discovery and Integration, and WS-*, the family of W3C standards. In addition, communication mechanisms as JBI, RMI, JDBC, SMTP, POP3, FTP or XMPP are also required. For message routing and transportation of sources decoupled from the destinations (allowing a sender to send messages without specifying exact destinations), transformations or translations resources (e.g.: transport protocol, message format and content) are demanding, as much as the usage of a common data format (XSL, for instance, is a powerful tool to use XML messages). Besides, adapters are recommended to connect APIs and data structures, in addition to facilities to administrate the infrastructure, among others identified requirements.

The proposal in this work is that ESBs become able to provide the facilities to accomplish all these tasks, and the solution proposed in this work should be aggregated to ESBs implementation.

C. Adaptive Decision Tables

A conventional decision table [17] is a device composed of conditions rows and actions rows, where the columns represent rules associated to conditions and actions. The basic decision table operation is: the conditions defined by the rules are verified and, if one of them is satisfied, this rule is considered as a valid one and all action associated to that rule are performed. Table 1 illustrates the concept by presenting a conventional decision table where c_1, c_2, …, c_n are the conditions rows,

Table 1 – A conventional decision table.

		0	1	2	3	4	5	6	7	8	9
Condition rows	c_1										
	c_2										
	…										
	c_n										
Actions rows	a_1										
	a_2										
	…										
	a_m										

and a_1, a_2, …, a_m are the actions rows.

Adaptive decision tables [6] [8] are adaptive devices which may be obtained by extending conventional (non-adaptive) decision tables, by adding rows to encode the adaptive actions to be performed. These actions must be able to alter the rules defined in the original decision table, so as to change the behaviour of respective conventional decision table. Adaptive rules are usually checked before and/or after conventional rules. By modifying the rules defined in the original decision table, it may be possible to remove rules, to add rules and to change the behaviour of an established rule [15].

Table 2, adapted from NETO [15] and BRAVO ET AL. [2], presents an adaptive decision table with adaptive function rows, b_{a1}, b_{a2}, …, b_{af}. c_1, c_2, …, c_n are the conditions rows and a_1, a_2, …, a_m are the actions rows, as in the conventional decision table, but they may be changed according to adaptive functions rows.

IV. Adaptive Decision Table for Service Ranking and Selection

A conventional decision table to service ranking and selection may be constructed simply by using the following: 1) A defined number of lines to describe the conditions; 2) A defined number of lines to describe the actions; and 3) Each service will be describe in a column. 1) and 2) are from the original definition, and 3)

introduces the services in the decision table.

The condition rows c_1, c_2, …, c_n, are the n conditions described by QoS of Web services, each attribute represented by a c_i, $1 \leq i \leq n$. Besides availability, security, response time and throughput, the c_i, $1 \leq i \leq n$, conditions also may include other functional and non-functional conditions, such as: 1) Correctness of the answer; 2) Overload of a service in a specific moment; and 3) User preference (when user interaction is available).

The list of criteria may be changed according to the application, then it is desirable that an Adaptive ESB have mechanisms to configure (and adapt) also this list.

There are real conditions which are related to a numerical interval (e.g.: 0 ≤ response time ≤ MAX_TIME, where MAX_TIME is a constant value). In these cases, in fact, the condition may be split in two related conditions c_i, $1 \leq i \leq n$, presenting the minimum and maximum values for each service attribute, so as to offer numeric values to allow the decision table implementation (e.g.: c_1: 0 ≤ response time and c_2: response time ≤ MAX_TIME). The format itself is not that relevant, but the aim of defining a numerical interval must be achieved. Table 3 presents the structure, similar to the structure presented in Adaptive Decision Tables defined to ADAPTGARP algorithm [2]. Each condition c_i, $1 \leq i \leq n$, is split into two other conditions: c_{iA} and c_{iB}, $1 \leq i \leq n$, respectively for the lower and upper bounds of the interval, i.e., c_{iA} means $v_i \geq i_{I1}$ and c_{iA} means $v_i \leq i_{I2}$, where v_i is the variable which is being analyzed (e.g.: response time), and $[i_{I1}, i_{I2}]$ is the acceptable interval (e.g.: [0,10] seconds for response time). The same feature also might be provided by adding extra columns instead of lines. If the interval was not necessary (e.g.: service availability), the previous condition definition may be applied.

Defined the structure of the adaptive decision table, the practical question related to service attributes/issues is: "how to evaluate each of them in order to be represented by numerical values?" The answer is not unique and is not simple, since each attribute must be studied separately. Common QoS attributes will probably have values defined by companies or systems, related to what may or may not be accepted, but other attributes may be more difficult to evaluate. Consider, for instance, the correctness of an answer of a non-deterministic experiment, implemented as a Web service. This is highly dependant of the problem. However, even in this case, some metrics must be adopted by the research community involved with the specific problem, since it is supposed that every model must be evaluated. So, correctness must be studied in the respective community, and the acceptable interval (or decision, in case of Boolean experiments) for this issue may be defined to be inserted in the decision table. Ecological niche modelling, for instance, includes a statistical evaluation of each generated model [19], and there are studies about these statistics. The results of both may drive the interval definition. Other criteria, such as user evaluations, may be more simplistic to define, since it is just necessary to ask the user to attribute a number in the range of a to b, and, then, assume [a,b] as the valid interval (e.g.: marks from 1 to 10). Nevertheless, the main point is that, since a value or an interval was defined, the adaptive decision table for service ranking and selection is ready to be used.

It is necessary to define, then, how to use the adaptive decision table. Since a condition c_i, $1 \leq i \leq n$ (eventually defined by c_{iA} and c_{iB}, $1 \leq i \leq n$), is satisfied, a respective action a_j, $1 \leq j \leq m$, may be associated to that. In this case, as the aim is to ranking and select services, a score may be associated with each number belonging to the numerical interval associated to the condition c_i, $1 \leq i \leq n$ (c_{iA} and c_{iB}, $1 \leq i \leq n$). The score may be a number or a function, depending on the complexity of the criteria.

A scoring algorithm must be defined by the usage of an extra structure to store services information, which would be integrated to the adaptive decision table to include the results obtained after execution of each action a_i, $1 \leq i \leq n$. Several proposals were also presented for achieving

Table 2 – An adaptive decision table.

		0	1	2	3	4	5	6	7	8	9
Condition rows	c_1										
	c_2										
	…										
	c_n										
Actions rows	a_1										
	a_2										
	…										
	a_m										
Adaptive functions rows	ba_1										
	ba_2										
	…										
	ba_f										

that and each situation must consider which of them is more adequate, if there are any. To analyze and discuss all presented proposals for service ranking is out of the bounds of this work but it is a job to be done. If any of the proposals fit, the problems must be a new proposal must be discussed.

Only in the moment when a service was invocated, all services in the table able to match the requirements of that service must be evaluated and their score must be evaluated. After calculating the scores of the fitting services, ranking and selection are simple tasks, which can be reduced, in algorithm theory, to a vector sorting problem. So, the solution described so far, based on conventional decision tables only, would be enough to solve the problem if the attributes of the services would be fixed and reliable, over time.

However, to assume this fact as an absolute true is not possible. There are several issues involved in service technologies which may lead to a wide set of problems [12]. For instance, service providers may present internal errors and they may affect the quality of all services furnished by them, communication problems may occur by many different internal and external factors, a service provider (and thus the required service) may be overloaded, among many others regular problems that Internet heavy users are familiar to deal from time to time.

So, as time goes by, adaptive mechanisms must be considered (and included in the adaptive part of the decision table) to manage adaptive decision table lines, including, removing or altering the lines according to adaptive functions described in b_{a1}, b_{a2}, …, b_{af} lines. It is necessary to evaluate the values of each interval attributed to each c_i, $1 \leq i \leq n$ (eventually defined by c_{iA} and c_{iB}, $1 \leq i \leq n$) condition, and the actions rules defined to each a_j, $1 \leq j \leq m$, in order to including, removing and changing their relevance according to system evolutions. This will permit to change the scoring behaviour so as to perform services ranking and selection.

There are general criteria, once a list of c_i, $1 \leq i \leq n$ (eventually defined by c_{iA} and c_{iB}, $1 \leq i \leq n$) conditions were defined. For instance, the columns related to the available services should be dynamically increased. Suppose availability quality attribute of service; it may receive an initial value, which may be improved or reduced by the adaptive functions each time a service was invoked.

Main problems, again, rely on the adaptive lines which are directly related to each problem, since different problems have different behaviours, and therefore the changes in their behaviours also are dependant of the scope of the system studied. For instance, an Internet Banking must have a response time upper bounded by 15 or 20 seconds, while a modelling system that executes very complicated algorithms may take from hours to days to reach a solution, so a response time of a week would be easily acceptable. Other example, it may be difficult to decide if a faster service is better than a more precise one (e.g.: routing services), or if a faster service is more often available than other [12].

Table 3 – An adaptive decision table with intervals.

		0	1	2	3	4	5	6	7	8	9
Condition rows	c_{1A}: $v_1 \geq i_{11}$										
	c_{1B}: $v_1 \leq i_{12}$										
	c_{2A}: $v_2 \geq i_{21}$										
	c_{2B}: $v_2 \leq i_{22}$										
	…										
	c_{nA}: $v_n \geq i_{n1}$										
	c_{nB}: $v_n \leq i_{n2}$										
Actions rows	a_1										
	a_2										
	…										
	a_m										
Adaptive functions rows	ba_1										
	ba_2										
	…										
	ba_f										

Even though, related problems are a matter of adaptive decision table intervals adjustment and user decisions, and for each problem domain all the criteria may be studied and defined. In any way, this does not reduce the solution credit, since its application still remain valid after an initial configuration. Therefore, the proposal provides a framework to re-evaluate services criteria using adaptive techniques, as desired.

V. ARCHITECTURAL SOLUTION – HOW TO INTEGRATE ADAPTIVE DECISION TABLES IN THE ESB CONTEXT

As an ESB requires dealing with many different issues and the application of each adaptive decision table may be dependant of the domain of each application, a single and fixed adaptive decision table will hardly be the solution for all ranking and selection service problems for SOA application, specially considering a more general architectural context, such as a Web portal. Therefore, the construction of an ESB able to incorporate more than one adaptive decision table is the main challenge of the architectural solution.

From the architectural viewpoint, it must be designed an environment to control this feature in an adequate way, since a company may have a single ESB to integrate many different applications, from many different domains (e.g.: financial companies).

Consider that to implement the adaptive proposal previously presented in this work, any domain must have its respective adaptive decision tables, configured and adequate to the specific conditions of each problem to be solved. In the financial company, for instance, there are systems to both human resource control and financial investments.

At first sight, the implementation of so many adaptive decision tables may seem a huge challenge and a very difficult task, but it is important to remind this is the SOA world, so it only will be necessary to create a service able to generate generic adaptive decision tables integrated to the ESB, able to configure such tables according to each domain. So, the implementation of adaptive decision tables is not an architectural problem by itself (more difficult than to implement that will be to define and configure the criteria, but this problem in not related to the system architecture).

The real architectural problem, which requires an architectural study in order to be solved, is how to incorporate a *Solution Manager* to existing ESBs patterns, in order to enable the adequate usage of adaptive decision tables. This Solution Manager must be able to associate each application to the right table without much effort, so as to avoid performance reduction. In addition, the Solution Manager must be able to treat different sources of services, with different patterns.

However, despite of the challenge to make the Solution Manager real in the ESB pattern, since this component were defined, the selection service and ranking problem will be solved, and many different approaches to solve the selection problem itself will may be integrated to an ESB, improving its skills to perform dynamic service ranking and selection in an intelligent and adaptive environment. Therefore, the effort to solve such a problem is more than valid. In this work, an example of a reference architecture including an service bus will serve as the basis to present the proposal.

In SANTANA ET AL. [18], a reference architecture for ecological niche modelling were proposed [19]. This example was chosen because of its complexity and because its architecture, among others features, details the main characteristics of any ESB, marked in the figure by the lightgrey color, avoiding to tie the solution to any specific platform, such as Java.net OpenESB [https://open-esb.dev.java.net/]. Therefore, it is effective to take a closer look to this architecture, focusing mainly on the ESB features.

At first, however, it is interesting to provide a briefly explanation about ecological niche modelling systems, so as to make the example understandable. Ecological niche [19] is an important concept used as a foundation for determining geographic species (e.g: plants and animals) distribution. It is related to the conditions that allow a species survival, disregarding external factors, such as human influence, biotic interactions and geographic barriers, which might prevent from a species to grow within the ecological niche area. The main hypothesis is that if a species can be found in certain conditions, then it should be able to survive and reproduce in any place with the same conditions. The modelling technique based on this concept aims to obtain areas similar to those where the species is known to occur, resulting in an ecological niche model.

Many different computational algorithms may be applied to obtain a niche model (including adaptive algorithms [2]). These algorithms mainly consider species occurrence and absence data, represented as coordinates of points in the geo-referrenced space of the study area, and data about the environmental conditions that are relevant to the species survival at those same points. Therefore, a model may be obtained in many different forms, and by using services spread worldwide.

Modelling algorithms produce models to represents the probability of finding species under the time–space conditions described by the input data. The model may be projected onto a map of the study region or is applied to obtain projections onto different regions or periods in time (past, present, and future), enabling the studies of the research community in ecological and environmental areas.

Fig. 2 presents the results of six experiments applying ecological niche modelling technique for mapping *Ouratea spectabilis*, a tree that can be found in Open Cerrado, Cerrado sensu stricto and Cerradão, a vegetation

of the Brazilian Savannah. Models are projected onto a map of the State of São Paulo, Brazil, and are related to 2006. Algorithms applied are: 1) Minimum distance; 2) Climate space model; 3) Bioclim; 4) Garp best subsets; 5) Distance to average; and 6) Environmental distance. Blue points are the original species occurrence points and the color scale represents the probability to find the species, where black means the probability is equal to zero. This figure was extracted from [19] so as to illustrate the problem to be solved by the presented architecture.

This technique has already been successfully used to propose scenarios for sustainable use of the environment, to evaluate the potential of invasive species, to evaluate climatic changes impacts on biodiversity, to delineate potential routes of infections and diseases and to indicate potential priority areas for conservation, among others.

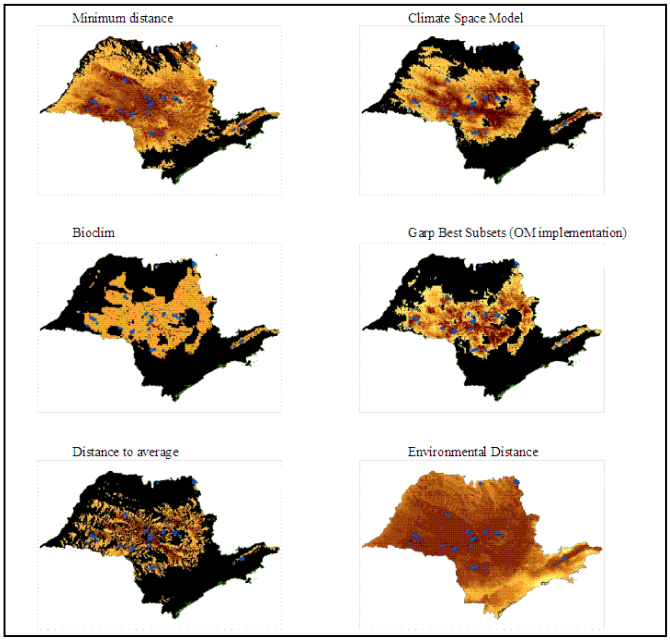

Fig 2. Results of six ecological niche modelling experiments with Ouratea spectabilis *species, using same input data and six different modelling algorithms. Models are projected onto the State Of São Paulo – Brazil.*

Ecological niche model generation is a complex computational process, involving a variety of data, techniques and software packages. It is data-intensive, since environmental variables may be stored in very large data files, of the order of many gigabytes. Data sources are frequently distributed and accessed via Internet, which may require adequate connectivity and bandwidth. Data quality and format are other relevant issues, demanding data pre-processing, cleaning and formatting, so as to provide adequate input data for software packages. A wide variety of methods and spatiotemporal data-analyses can be applied, and specific data conversion may be required to run the same experiment and to compare results obtained by different algorithms. The connection with all the sort of components presented in Fig. 3 may be required.

The reference architecture, presented in Fig. 3 [18], considers mainly the service-oriented architectural style. Service Bus (the equivalent to an ESB within this architectural proposal) receives the requests from applications and calls the appropriate services; after a service processing is finished, answers are stored in the results Repository and the client is notified, receiving a reference to retrieve desired results from the Repository (this may be put available in the Web portal or to be sent to the client, for instance, by e-mail, sms or other communication technology).

Clients may access the applications using the portal so as to offer easy and standard interfaces to the users. Applications must interact with the Service Bus, which invokes necessary services, even if they were distributed over different service providers, using some specific service protocol, such as Web services. The Service Bus must guarantee the requests delivery and, when necessary, must transform the data before communicating with the services. The Repository must store results that require huge processing effort and thus are invoked in an asynchronous way. Other services usually are Web services hosted in application servers on the Internet.

In order to increase the reusability and modifiability of the system, because each layer just knows the neighbouring ones, the architecture is organized into layers. Layers are: *client* – access devices, e.g. *web browsers, wap-phones* or *pagers*, for user input data for processing and receiving answers; *presentation* – offers services using a *web portal*, creating a single entry point for the system; *integration* – integrates applications with services, required for a complete business process, including a *service bus*, for routing and conversion services, and a *repository*, that offers mechanisms for temporary storing results; *business* – services required for the complete execution of a business process, including services based on OGC, *Web Feature Service* (WFS) and *Web Map Service* (WMS) (OGC, 2006); *resources* – data base applications, legacy systems and other devices required for precision agriculture.

Observe that this solution [18] already predicted the inclusion of a Service Engine, even before the adaptive technique was considered. In this proposal, the Service Engine has the attribution to search UDDIs so as to find services based in their description (usually, using BPEL or related W3C patterns).

However, to solve the architectural problem presented in this paper, the Service Engine must be extended to a major component which will encapsulate the current functions in other component, in this work named as *Service Search*. The *Solution Manager* would be other component of the Service Engine, with the function to specifically ranking and select services based on adaptive decision tables, but implementing

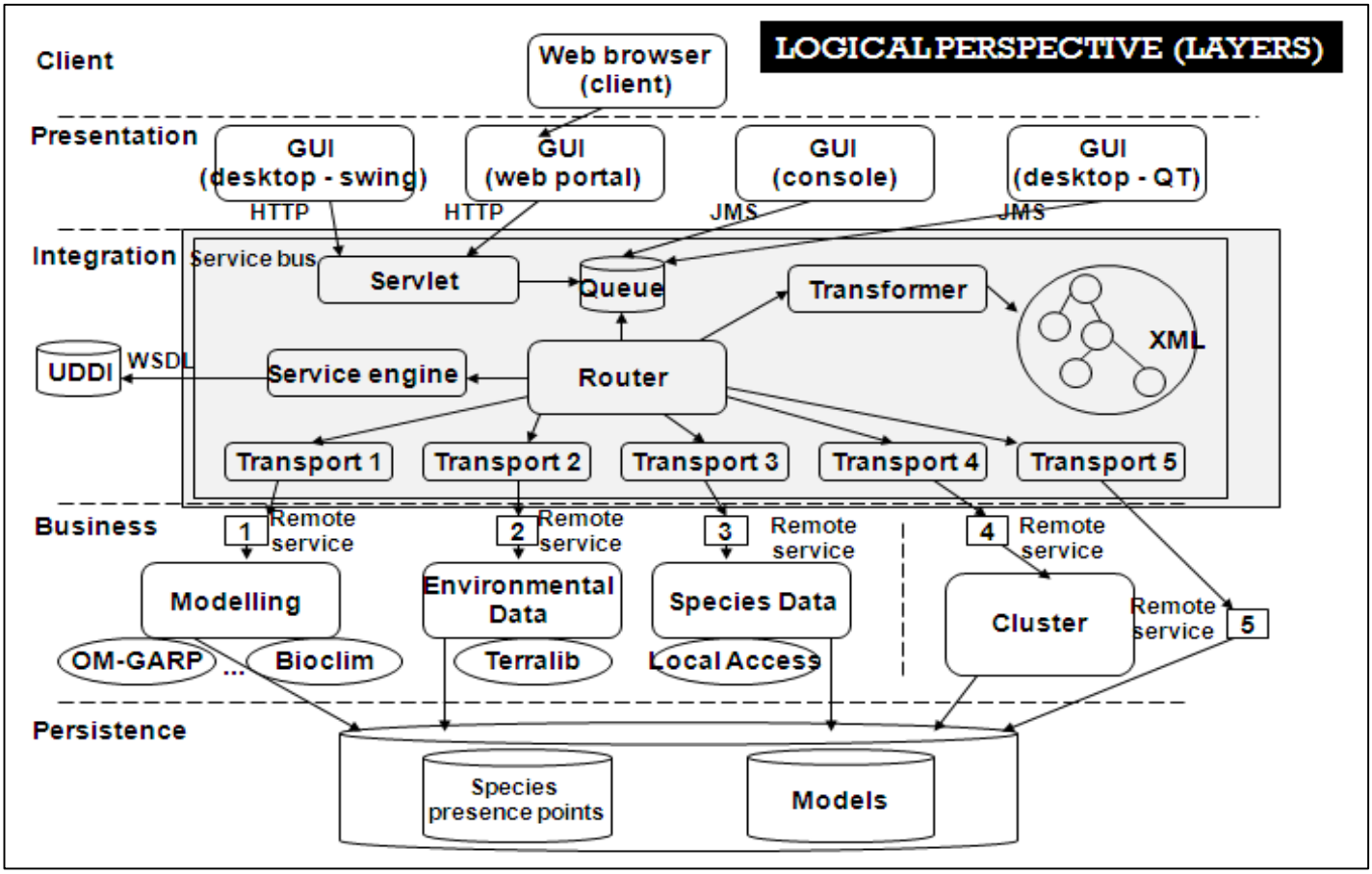

Fig. 3 – Reference architecture for Ecological Niche Modelling System.

the flexibility of constructing more than a single table, so as to be able to attend application from different domains. Finally, as the number of adaptive decision tables is not limited, by definition, an *Adaptive Services Repository* should be incorporated to the Service Engine. The result would be a Service Engine as presented in Fig. 4.

Other ESB solutions, such as openESB [https://open-esb.dev.java.net/], supported by SUN Microsystems, also have functionalities similar to the proposed Service Engine, despite presenting some architectural differences and, nowadays, representing a more complete solution, since many other features were implemented. In such a case, it is named *Application Monitor*. However, as ESB principles are the same, since ESB is mainly an architectural concept, a similar architectural proposal may be proposed to each ESB-based solution created for supporting SOA and SOC in a similar way as the presented before.

VI. Discussion and Future Works

As criteria definition may be an empirical process, conflicts and inaccuracy in the services ranking may occur. Therefore, it will probably be necessary to dedicate time and effort to construct a relatively reliable adaptive decision table to each different domain of problems that the SOA architecture intends to treat.

On the other hand, the adoption of an adaptive decision tables implemented onto ESBs may bring many choices and facilities to the user. It also may avoid misinterpretation of who is guilty when an SOA-based

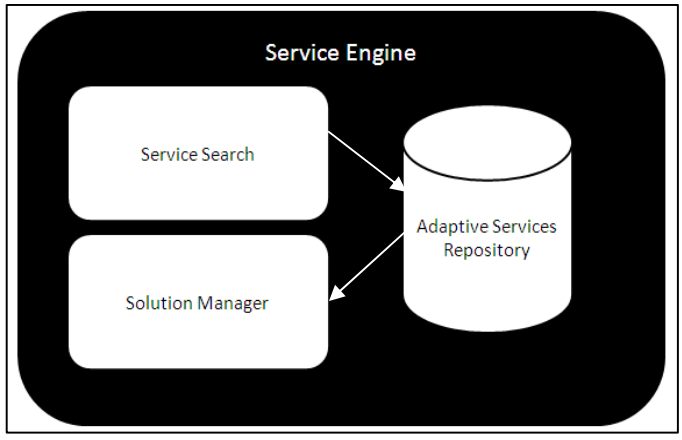

Fig. 4 – A new architectural proposal to a Service Engine

service application does not offer the desirable results, in terms of performance, security, reliability or whatever other quality criteria which be important in the specific context.

An additional advantage is that an adaptive decision table implemented onto ESBs may be used to compare the results of the execution of similar services furnished by different providers. In the situations where service providers have a significant cost, this data may be an important factor to be considered by IT executives.

Besides, as services may be executed in parallel, by using different providers, it is possible to evaluate and, if interesting, to present to the user all results obtained. This is important, for instance, in modelling applications based on non-deterministic algorithms [19].

However, to implement the proposals presented in this paper is not an easy task and, probably, will demand some effort, even after the architectural constraints were solved (and implemented). So, many future works may be proposed, starting from the problem definition.

At first, as the service ranking and selection is nowadays a very hot research area, to study and formalize an architectural proposal to incorporate the new concept of Service Engine to the ESB conceptual definition is a main challenge and many effort will be applied to this.

Solved the architectural problem, a second important point is to define the data structures to implement adaptive decision tables in ESBs. Since services are registered on UDDI or similar devices distributed on the Internet, search mechanisms need to be incorporated to maintain the portability of the solution. Then, an internal language must be defined, using some specific technique (e.g.: ontology, metadata) to enable the service inclusion in the adaptive decision tables.

Other very interesting problem is, supposing services found, how to rank them, considering services attributes. At first, it is necessary to exploit the concepts of attributes of services and, perhaps, this may require some kind of categorization. Then, after solving this problem, the next step is to find an adequate function or algorithm to service ranking or selection. Many proposals were made, but they are usually based on untrue hypothesis, such as all UDDI are oriented to ontology or only QoS factors are relevant. Nevertheless, some approaches are very interesting, and the necessary solution may be, in fact, a combination of a set of proposals, instead of choosing only one. This may depend of several factors, including each problem domain.

After designing a whole scenario, the next step is to propose a solution for a specific problem, so as to present a concept proof. In such a case, as the matter is related to other works, application in the biodiversity field may be the chosen domain, since there are already collaborators interested in evolve software techniques and whose contribution may not be disregarded.

Since a concept proof is implemented, it will need to be validate and then, only them, it is possible to imagine offering this solution as part of a real ESB, for practical research and/or commercial application.

However, as the set of problems is open in this paper and many of them may be solved in parallel, maybe contributions may improve the velocity to reach such a solution, and they will be very welcome.

VII. CONCLUSION

The implementation of adaptive decision tables associated to ESBs is a highly complex problem, but it is also a highly interesting challenge. If the proposals presented in this paper were all achieved in order to adopt the solution, probably a big step will be have given into the direction of the definitive solution to the service selection and ranking problem.

Exclusively from the adaptive viewpoint, the most relevant contribution is that the usage of adaptive decision tables to service ranking and selection. It is not a dream to imagine that, over time, only better services will be selected among many similar available solutions, and the service providers will have to review their priorities. Note that, however, the "better services" may change, since adaptive technologies will make regular re-rankings in the services organization and evaluation criteria.

Finally, adaptive techniques may completely transform the architectural concept of ESB and SOA and, perhaps, even the SOC concept itself, since the constant re-evaluation of the services may lead to a way to evolve the services over time.

ACKNOWLEDGMENTS

Authors are grateful to FAPESP – Fundação de Amparo à Pesquisa do Estado de São Paulo – Brazil, for the support to the openModeller (04/11012-0) and BioAbelha (04/15801-0) projects.

REFERENCES

[1] BASS, L., CLEMENTS, P., KAZMAN, R., 2003. Software Architecture in Practice. Second Edition, Addison-Wesley Professional.

[2] BRAVO, C., NETO, J.J., SANTANA, F.S., SARAIVA, A.M., 2007. *Towards and adaptive implementation of genetic algorithms*. Latin American Workshop on Biodiversity Informatics – INBI 2007 / CLEI 2007 – XXXIII Conferência Latino Americana de Informática. 9–12 Outubro. San José, Costa Rica.

[3] BROOKS, F. P. 1987. No Silver Bullet, IEEE Computer, Vol. 20-4. 1987.

[4] CASATI, F., CASTELLANOS, M., DAYAL , U., SHAN, M. 2004. Probabilistic, Context-Sensitive, and Goal-Oriented Service Selection. ICSOC'04, November 15–19, 2004, New York, New York, USA, pp 316-321.

[5] ENDREI, M. et al., Newling, T. 2004. Patterns - Service-Oriented Architecture and Web Services. IBM Redbook.

[6] GILDERSLEEVE, T. R. 1970. *Decision Tables and Their Practical Application in Data Processing*. Englewood Cliffs, N. J., Prentice Hall.

[7] HOLLAND, J. H. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.

[8] HUGHES, M. L., SHANK, R. M., STEIN, E. S. 1968. Decision Tables. Midi Publications, Management Development Institute, Divisions of Information, Industries, Inc., Wayne, Pennsylvania.

[9] HUHNS, M., SINGH, M.P., 2005. Service-Oriented Computing: Key Concepts and Principles. IEEE Internet Computing, v.9, n.1, p.75-81.

[10] LIU, Y., NGU, A. H.H., L. 2004. QoS Computation and Policing in Dynamic Web Service Selection. WWW2004, May 17–22, New York, New York, USA. ACM 1-58113-912-8/04/0005, pp 66-73.

[11] MACKENZIE, C. M. et al., 2006. Oasis Reference Model for Service Oriented Architecture 1.0.

[12] MAXIMILIEN, E. M., SINGH, M. P. 2005. Multiagent System for Dynamic Web Services Selection. In Proceedings of the AAMAS Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE), Utrecht, July.

[13] MENASCÉ, D. A. . QoS Issues in Web Services. 2002. IEEE Computer, pp 72-75.

[14] MENGE, F. 2007. Enterprise Service Bus. Free and open source software conference, pp 01-06

[15] NETO, J. J. 2001. *Adaptive Rule-Driven Devices – General Formulation and Case Study*. Lecture Notes in Computer Science. Watson, B.W. and Wood, D. (Eds.): Implementation and Application of Automata 6th International Conference, CIAA. Vol. 2494, Pretoria, South Africa, July 23-25, Springer-Verlag, 2001, pp. 234-250.

[16] NETO, J. J. 2007. *Um Levantamento da Evolução da Adaptatividade e da Tecnologia Adaptativa***.** Revista IEEE América Latina. Vol. 5, Num. 7, ISSN: 1548-0992, Novembro, pp. 496-505.

[17] POLLACK, S.L., HICKS, H.T., and HARRISON, W.J. 1971. *Decision Tables: Theory and Practice*, Wiley, New York.

[18] SANTANA, F. S, MURAKAMI, E., SARAIVA, A. M., BRAVO, C., CORREA, P. L. P. 2007. Uma arquitetura de referência para sistemas de informação para modelagem de nicho ecológico. In: 6º Congresso Brasileiro de Agroinformática – SBIAgro.

[19] SANTANA, F. S, SIQUEIRA M. F., SARAIVA, A. M. & CORREA, P. L. P. 2008. *A reference business process for ecological niche modeling*. Ecological Informatics v. 3, n. 1, p. 75-86.

[20] STAL, M., 2002. Web Services: Beyond Component-Based Computing. Communications of the ACM. Vol. 45, No. 10, 71-76.