A Decision Support System for QoS-enabled Distributed Web Services Architecture

Chen Wu, Elizabeth Chang
School of Information Systems
Curtin University of Technology
Perth, Western Australia 6845
AUSTRALIA
{chen.wu, elizabeth.chang}@cbs.curtin.edu.au

Patricia Thomson
School of Information Systems
University of Tasmania
Hobart, Tasmania 7001
AUSTRALIA
patricia.thomson@utas.edu.au

Abstract – Service selection is crucial for fulfilling the requirements of service requestors. While in the real service-oriented environment, Quality of Services (QoS) is one of the greatest concerns for consumers during service selection. Existing web services' standards do not tackle the QoS issue adequately when service discovery and selection are performed. In this paper, we argue that the process of services selection is a kind of decision making – to decide which service should be selected dependent on their QoS and trustworthiness values as well as their functional capabilities. Hence, we propose a service selection solution which utilizes the Decision Support Systems Module (DSS Module) to select the most appropriate service. In DSS module we introduce *Service Trust* to carry out the service QoS measurement based on the *Context*-specific *Quality Aspects*. The architecture of DSS module is presented in detail and the solution is also integrated into one of the components – domain-broker – in our proposed distributed web services architecture. The contributions of this paper are two fold. Firstly, we apply DSS module into web services, thus opening a new, fertile ground for DSS research in web services literature and secondly, we provide a novel and feasible solution for QoS-based service selection.

I. INTRODUCTION

Decision Support Systems (DSS) are computer systems used to facilitate human decision making in solving complex problems, where at least some stage is semi-structured or un-structured. DSS was promoted in the late 1970s from the Management Information Systems (MIS), and continued evolving until today where is appears in the web-based DSS [1] and open DSS [2]. In general, by 1) analyzing transaction data; 2) gathering information from external sources; and 3) reasoning from a decision model, a DSS is able to promptly present useful decision making information in an appropriate format that is easily understood and manipulated by humans.

Today DSS have been extensively utilized in various applications within one enterprise or across different business organizations. For instance, airlines use DSS to assist analysis of flight ticket pricing and route selection. Supply chain DSS can be used for planning schedules and forecasting changes from the up-stream suppliers and down-stream retailers. In this paper, we mainly investigate the application of DSS in one of the significant e-logistics scenarios where logistics customers select the logistics service provider based on Quality of Services (QoS).

Meanwhile, in recent years, web services have increasingly gained significant attention from both industry and academia. As a promising solution for distributed system integration, web services provide programmatic interfaces that are used for flexible connectivity among heterogeneous applications [3]. A large number of enterprises are now starting to adopt emerging web services technologies such as WSDL, SOAP and UDDI, to build their service-oriented systems [4]. Hence, we believe it is a natural trend in the IT industry that all inter-organization systems, where flexible and reliable integration is highly pursued, will be enabled by web services technology in the near future.

When thinking of how DSS can facilitate web service-enabled service-oriented architecture (SOA), we would see the well-known issue of service selection [5] can be effectively tackled by employing certain DSS solutions for the following two reasons. Firstly, existing web services standards (e.g. UDDI) failed to provide a flexible, dynamic, and reliable mechanism to allow service requestors to choose the right quality service instance based on non-functional attributes such as QoS and trust. Secondly, in service-oriented environments, service selection can be seen as a process of decision making, i.e. the service requestor should make a decision on which service provider is currently offering the most appropriate service to fulfil the requirement among abundant function-relevant service provider candidates. This argument is partly based on the assumption that DSS 'support' rather than 'automated' decision making [6], a task which cannot be completely achieved without human participation and final decision making. Hence, while some research attempts to automate the service selection without human intervention, it is our belief that such thorough automation will not occur in the real logistics transaction due to the complexity of service selection [7].

Consequently, in this paper, we mainly deal with the issue of how to support the decision making process with regards to selecting the most appropriate web services providers using DSS modules. We integrate into existing distributed web services architecture a DSS module which would assess the QoS of web services based on trust. An architecture based on such conceptualization is also given.

The remainder of this paper is organized as follows: Section II introduces the preliminary concepts of Trust, QoS, Service Trust, Context, and Quality Aspects; Section III explains the design of DSS module; Section IV describes the overall QoS-enabled distributed web services architecture which incorporates the DSS module and also provides the detailed architecture of Domain-Broker where DSS module resides; and Section V concludes our research in this paper and visions for future work.

II. PRELIMINARY CONCEPTS

Since our research in this paper incorporates some of our previous work on QoS and trust [8], before we discuss the module design and architecture we would like to introduce some preliminary concepts on QoS and trust, which would facilitate understanding of our work in service selection DSS in Service-Oriented Environments (SOE).

A. QoS and Service Trust

In Chang, Dillon and Hussain [8], 'Trust' is defined in a service-oriented environment to mean 'quality'. Hence, we believe the quality of services (QoS) can be reflected by the trustworthiness of the service. The 'belief' of Trust in a service-oriented environment is meant the 'quality assessment'. In general, trust is realized by the concept of a Trust relationship and Trust Value. Each relationship denotes a trust value from the trusting party to the trusted party and it is only for a particular context and timeslot. A context can be decomposed into several aspects (dimensions) and that can be used to derive the criteria for quality evaluation or measurement. In a particular SOE, trust is defined as the *Service Trust* that the *Customer* has in the Service Provider's ability to deliver Quality of Service in the given service context and timeslot. In summary, the service trust conceptualization is depicted in Fig. 1.

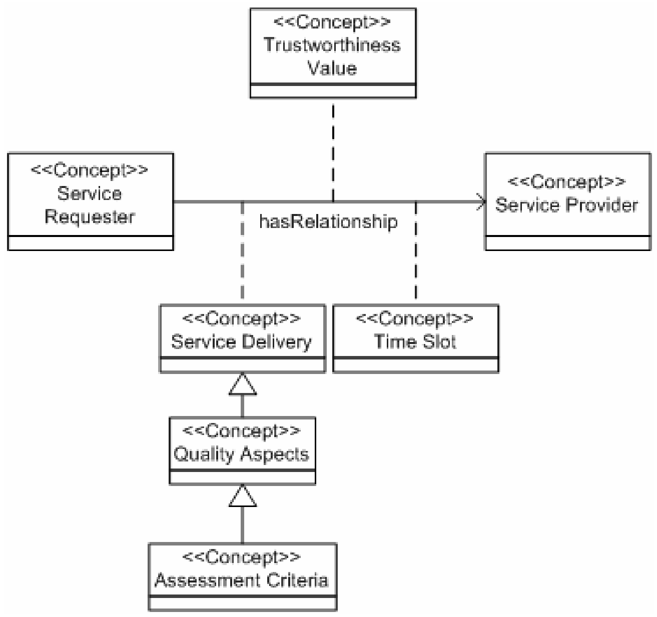
<<Concept>> Trustworthiness Value
<<Concept>> Service Requester
hasRelationship
<<Concept>> Service Provider
<<Concept>> Service Delivery
<<Concept>> Time Slot
<<Concept>> Quality Aspects
<<Concept>> Assessment Criteria

Figure 1. Service Trust Conceptualization

We identity the following concepts encompassed by *Service Trust*.

- ***Service Requester*** is a Trusting Party who has the trustworthiness value of the QoS of the service provider.
- ***Service Provider*** is a Trusted Party whose Quality of Service is being considered by the service requester.
- ***Context*** is a specific service or service function.
- ***Quality Aspects*** defines the quality of service.
- ***QoS Criteria*** are metrics that are used for quality assessment of the trustworthiness of the services.
- ***Timeslot*** is the timeframe for which the trust value holds. That is, during this period, the trust value remains constant.
- ***Trustworthiness*** is a measure of the trust against a trustworthiness scale.

B. Context and Quality Aspects

Here, we provide the definition of *Context*: Context defines the nature of the service and service functions, and each context has a name, a type and a functional specification [8]. A *Context* can be decomposed into several quality aspects, and with each quality aspect, there is always a set of assessment criteria that can be used to measure the quality of the Context Aspects. Fig. 2 depicts the formal representation of a context hierarchy in the SOE.

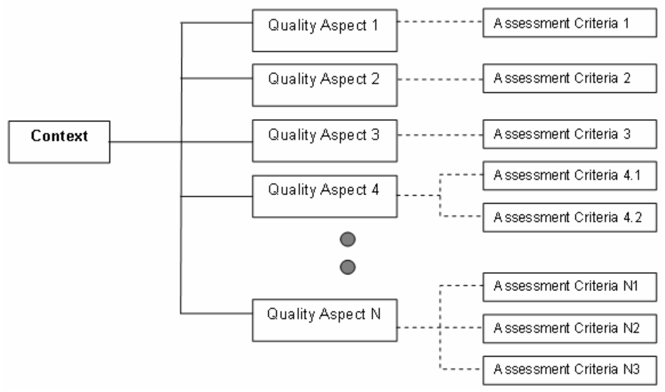

Figure 2. Context with Quality Aspects

In Fig. 2, context such as 'a logistics company providing delivery service in the state of New Southern Wales' can be seen. Each context should be further decomposed into a set of 'Quality Aspects'. *Quality Aspects* decompose the *Context* into several dimensions for the purpose of quality assessment or measurement [8]. To continue with the example, the NSW delivery service context can be further decomposed into three *Quality Aspects*: intact delivery, on-time delivery, and tracking capability. One should be aware that *Quality Aspects* could be defined by the Service Level Agreement (SLA), quality standards, or contracts established via bilateral negotiation, etc. In the far right, there is a set of 'Assessment Criteria' which are associated with each of the *Quality Aspects* defined in the Context. Assessment Criteria define the quality metrics for each *Quality Aspect* of the Context for the purpose of measuring the delivered quality (aspects) against the defined quality (criteria) [8]. For 'on-time delivery', we can setup *Criteria*, for example: if never late, assign Excellent - if rarely late, assign Good – and so forth. For clear illustration of the metrics, the trustworthiness level for each of the above criteria can be depicted in Table 1 below.

C. DSS Request

Decision Support System (DSS) Request is initiated by the end customer. DSS request is the problem definition for the DSS module. In our proposed architecture, DSS request is a small subset of *Service Subscription Request (SRS)* and is the default input for DSS module. A typical DSS request contains Service *Context*, QoS requirement, and/or QoS report. The QoS requirement specifies the QoS level that the service consumer expects from the desired service providers. The QoS report contains the QoS experience information collected from service consumers.

Table 1. Trustworthiness level for each criteria

Trustworthiness Scale (Ordinal Scale)	Semantics for Logistics Service (Linguistic Definitions)	Alternate Semantics for Logistics Service
Level -1	Unknown	Unknown
Level 0	Very Untrustworthy	Awful
Level 1	Untrustworthy	Poor
Level 2	Partially Trustworthy	Average
Level 3	Largely Trustworthy	Fair
Level 4	Trustworthy	Good
Level 5	Very Trustworthy	Excellent

In this section, we defined the basic concepts of QoS, Service Trust, and metrics to measure the trustworthiness of a service provider in the service-oriented environment. In the following sections, we illustrate how to integrate such concepts into the architectural design of DSS module and overall architecture of distributed web services.

III. DSS MODULE DESIGN

In this section, we explain our design of DSS module in facilitating the QoS based service selection. The DSS module is deployed in the Domain-Broker which is described in detail in next section from the overall architecture perspective.

Although different DSS might have different purposes and are working in diverse scopes, all DSS should have similar technical components: Model, Database, Communication, and User Interface [6] as illustrated in Fig. 3. The Database component contains collections of both structured and unstructured data from a number of sources that have been organized for easy access and analysis. Recently, as the roaring development of data warehouses and data mining techniques, the database component is increasingly central in large enterprise decision making. In a DSS, a decision problem is usually formulated as a model. The model component is utilized and operated unswervingly by decision makers to manage and exploit these models so that they can simulate, describe and solve the actual problems using appropriate solvers. The User Interface is also very important since DSS does not generally automate decision making; rather it gives support to the human to make decisions. For instance, many problems are too complex to be handled by humans but not so well defined that they would be entirely performed by computers. This is where the user interface can come into play. Good user interface facilitates the interactions between the decision maker and the DSS.

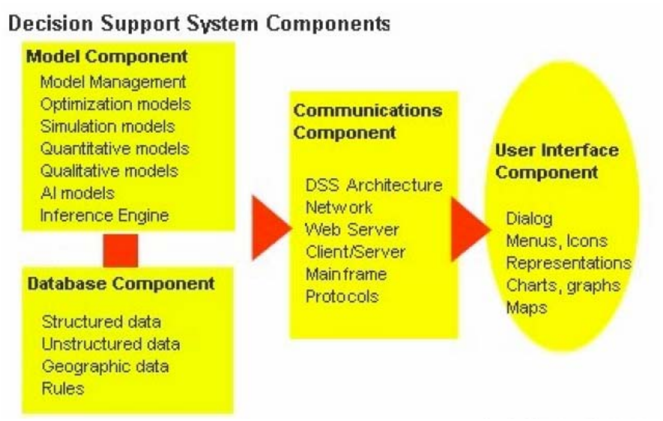

Figure 3. DSS Components (source [6])

Based on this components diagram, we therefore provide the DSS module for web services selection in our architecture design illustrated in Fig. 4 below.

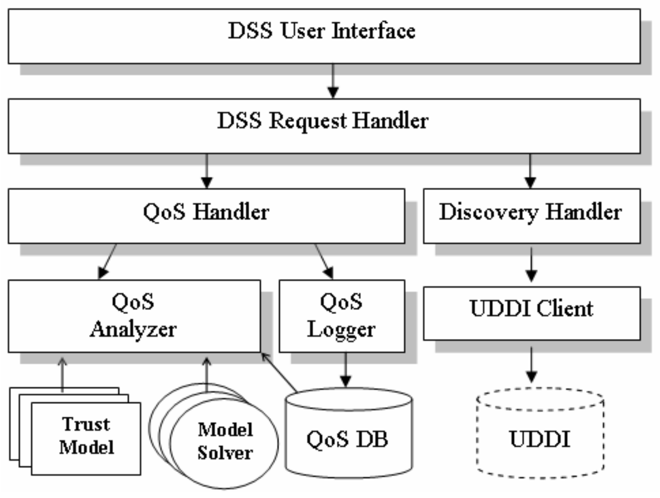

Figure 4. DSS Module Design

As we can see, the DSS module consists of the following components: DSS User Interface, DSS Request Handler, QoS Handler, Discovery Handler, UDDI Client, QoS Analyzer, QoS Logger, and QoS DB. The dotted-line around the UDDI indicates that the UDDI is the external component. These components work together to provide decision support in selecting appropriate web services.

User Interface – User Interface essentially fulfils two functionalities:

Firstly, collecting the *DSS request* which may include: Service *Context* (reference), QoS requirement, or QoS report. The service function information, which is conveyed by the *Context*, is passed to *Discovery handler* as query parameters for searching UDDI, while QoS requirement along with the service *Context* are delivered to *QoS handler* for afterwards service selection processing, if necessary. The entity that interacts with User Interface (i.e. the DSS module) could be end customers, service requestors as well as other internal modules in the *Domain-Broker*. If the *DSS request* directly comes from end customers, Graphical User Interface (GUI) could be offered as interaction tools. Application Programming Interface (API) is also provided for programmatic level integration with the DSS module. In this research, all the API-level interfaces are represented in the form of XML message documents such as WSDL.

Secondly, returning a list of service candidates in rank order with the most appropriate service placed at the top of the list. The representation of the list could be GUI web-based tabular catalogue or pure XML data structure defined by the particular schemas or some domain standards.

In a word, *User Interface* component can be regarded as I/O sub-system for the DSS module.

DSS Request Handler – This component receives the *DSS request* from the *User Interface*, handles the logic control to parse the request, and splits it into two parts which are further processed by two sub-handlers respectively: *QoS Handler* and *Context Handler*.

The ***Discovery Handler*** processes the service functional requirements associated with the *Context* embedded in the *DSS request* and posts the query parameter to *UDDI Client* for initial service discovery.

UDDI Client – For the purpose of discovering function-relevant web services, the UDDI client is used for interacting with and searching the specified UDDI registry to find those services that could meet the functional requirements indicated in the service context information enclosed in the *DSS request*. In general, UDDI client operates the UDDI inquiry API (e.g. *find_service*(), *get_serviceDetail* etc.). The invoked API parameters are populated by the *Context Handler*, which sends those functional requirements to the UDDI client as each DSS request arrives.

QoS Handler mainly deals with the part of the *DSS request* related to the QoS: either QoS requirement or QoS report. The QoS requirement is restructured into DSS compliant representation format which is sent as input parameter for the QoS Analyzer component. The QoS report is however forwarded to the QoS Logger for recording the history data related to QoS.

QoS Analyzer – QoS Analyzer is the most important component. It facilitates the actual decision making by taking as input the QoS requirement, Context, relevant Trust Model, appropriate Model Solver, and History QoS data from the QoS Database, meanwhile generating the output of trustworthiness values for involved web services. The result is sent back via QoS Handler to the DSS Requester handler, which produces the final ranking list of the services in preferred representation format. For the purpose of supporting an open architecture of DSS module, our design fosters a flexible, multiple trust model loading mechanism such that different models can be chosen for different service *contexts* according to the nature of that particular service. Consequently, choosing an appropriate model is a key design issue for *QoS analyzer* since each *Trust Model* might have a specific purpose. Meanwhile, as every *Trust Model* more often than not employs fairly complex mathematical vehicles such as the Markov Trust Model [9], the *QoS Analyzer* has to select compatible (syntactically and semantically) solvers i.e. the model solving algorithms, for a model and adequately applying it to the model. In doing so, the *QoS Analyzer* should be able to match the model parameters to the solver parameters. In this paper, the Trust Model and the corresponding solver algorithm will not be described in detail. Interested readers are referred to Hussain, Chang and Dillon [9] to obtain further extensive studies on Markov Trust Model.

QoS Logger – After a web service finishes a service function defined in the *Context*, the current service consumer, i.e. service requestor, may wish to file the QoS experience and send them to the DSS Module via the *DSS User Interface*, which hands over the QoS report down to the *QoS Logger* via QoS Handler. The *QoS Logger* collects and inspects the raw data in the report and ultimately produces the QoS history data from which trustworthiness value and service behaviour can be inferred through QoS Analyzer. Such QoS data is then saved to the *QoS database* for further query.

QoS Database – The design of a QoS database is based on our previous work of 'Trust database for Quality of Services' proposed [8]. For the reason of simplicity and architectural description, we assume that the QoS measure is based on the service standard specified by certain consortium in a particular domain rather than against the Service Level Agreement or contract established by both service provider and service consumer. As a result, we consider the following table structures in the *QoS database* schema.

1. *Service Requestor* (<u>Requestor ID</u>, RName.QoS Measure#)
This table represents those trusting party who share their direct experience or trust value to other potential service requestors.

2. *Service Provider* (<u>Provider ID</u>, PName, …..QoS Index#)
This table is used to keep the records of trusted entity in service-oriented environment. Please note that the Provider_ID shall be consistent with the counterpart provider identifier registered in the UDDI component.

3. *Service Context* (<u>Service Context #</u>, QoS Aspects#, QoS Aspects)
Since our assumption of standard service context is made by standard organization or industry consortium, the Service Context table is essentially a look-up dictionary which is predefined tables, also called look-up tables in database terms, and the data is normally persistent, i.e. no edit or delete operations etc.

4. *QoS Criteria* (<u>QoS Aspect#, Context#</u>, Criteria1, Criteria1, Criteria2 ….CriteriaN)

The QoS Criteria table stores criteria for each *Quality Aspect*, and this is also a look up dictionary, and the criteria database is a weak entity and must associate with one of QoS context database.

5. *QoS Measure* (<u>Customer ID, QoS Assessment#</u>, QoS Index#, Provider ID, Context#, Trustworthiness Value)

The QoS Measure table contains each customer's feedback i.e. trust values and opinions. Note that each customer feedback is represented by a trusting party's trust value and opinion in the trustworthiness scheme.

IV OVERALL ARCHITECTURE

In this section, we provide an explanation of how the DSS module, elaborated above, can be integrated into the full distributed web services architecture designed for a logistics network.

A. Architectural Topology

As depicted in Fig. 5, the primary components of this architecture are: *service-peer*, *domain-peer*, *super-peer*, *alliance-peer*, and *domain-broker* and *global-broker*. *Service-peers*, which provide logistics service such as transportation and warehousing, scatter in the *global service space* - the cloud in Fig. 5 – the broker-based web services [10] environment where service requestors and providers register with a single global UDDI. Ideally, logistics customers are able to automatically discover and benefit from these services without human intervention by exploiting standard web services protocols (i.e. UDDI, SOAP, and WSDL). Nevertheless, to our best knowledge, such thorough dynamic automation does not occur in actual logistics industry partially due to service consumers' fear of services QoS and trust. This motivates us to introduce into the existing web services architecture the concept of *domain* - the light-grey circle in Fig. 5. A *domain* represents a virtual society where well related (functionality or vicinity) web services gather together in attempt to offer quality and added-value services to potential end customers. The domain-specific knowledge is essential in defining the service trust criteria [8], thus significantly facilitating the quality of service assessment, without which the service selection decision cannot be made. In particular, we utilize *domain-broker* to include the DSS module to perform the service selection. The *domain-broker* provides QoS-based service selection within the scope of a particular domain. In general, the *domain-broker* is responsible for managing (e.g. register, matchmaking, etc.) *domain-peers*. The *domain-peer* is a type of *service-peer* within a particular *domain*. Based on capabilities and willingness, a *domain-peer* is prepared to be joined with other selected *domain-peers* to offer quality logistics services in response to the dynamic requirements from external customers. Such dynamic relationship gives rise to the *alliance-peer*. The *alliance-peer* is a special domain-peer within a certain *dynamic-alliance*. The dynamic-alliance (dotted line circle) is a smaller community established in an ad-hoc manner amongst QoS-trusted web services inside the same domain. When detailed requirements are presented to a specific domain-peer, who alone is unable to suffice such requirements, this domain-peer attempts to initiate a dynamic-alliance by sending a service cooperation request to selected domain-peers from its local matching table where QoS partner services are logged and updated. *Alliance-peers* work autonomously by exchanging messages with each other to fulfil the end user requirements introduced by the alliance-initiator. A *super-peer* is an alliance-peer that maintains a particular dynamic-alliance composed of selected domain-peers. A s*uper-peer*'s major responsibility includes initiating the alliance by propagating a service composition request and re-arranging the formation of the alliance in accordance with changing requirements. *Super-peer* interacts and coordinates with *Super-peers* representing dynamic-alliance from other domains, thus coordinating two or more alliances across domains.

Due to the space limit, we focus on the *Domain-Broker* which integrates the *DSS Module*.

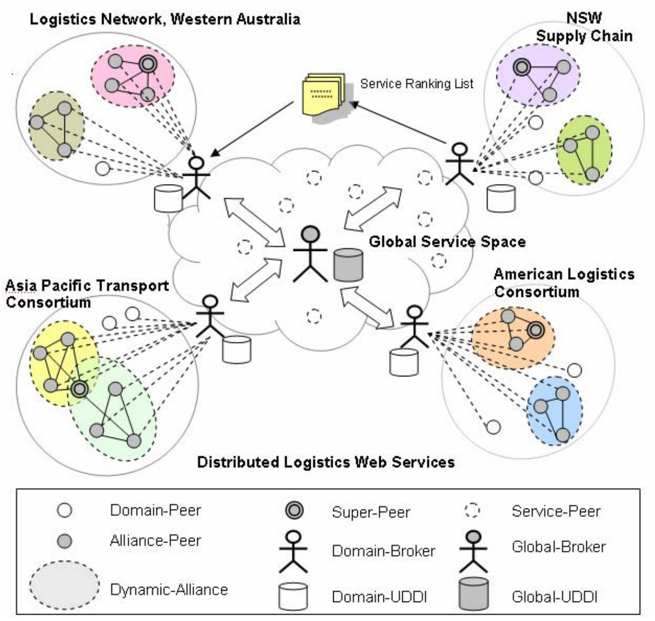

Figure 5. Overall Architecture

B. Domain-Broker Architecture

As mentioned earlier, *Domain-Broker* manages *Domain-Peers* as well as provides some crucial add-on services to *Domain-Peers* inside the domain. It handles the joining and leave request from domain-peers, generates matching tables for each *Domain-Peer*, and maintains the transaction history data for *Domain-Peers* as well.

Formative domain protocol is employed inside each domain to allow *Service-Peer* (service provider and/or consumer) 'join' and 'leave' a particular domain for some reason. Once a *Service-Peer* $\boldsymbol{SP_{new}}$ turns into a new *Domain-Peer* $\boldsymbol{DP_{new}}$, it is firstly granted privilege to register (i.e. apply the UDDI Publication API set) its detailed service metadata with the *Domain-UDDI*. If the metadata is found entirely new to this domain, the *Domain-Broker* creates new QoS entry for $\boldsymbol{DP_{new}}$ in the *QoS database* maintained by the *DSS module*. Otherwise, related QoS and trustworthiness value can be obtained directly from existing records stored in the *QoS database* for future service selection processing.

Suppose set $DP = \{dp_1, \quad, dp_n\}$ where DP represents all the *Domain-Peers* in the current domain. The *Domain-Broker* then propagates $\boldsymbol{DP_{new}}$'s service metadata (mainly high-level data such as name, interface, classification, etc.) to a set of *Domain-Peers* $IDP \subset DP$, where

$$IDP = \left\{dp_i \ \mid dp_i \in DP \,\&\, dp_i \textit{ is a potential consumer of } DP_{new}\right\}$$

The *Domain-Broker* needs to calculate the potential consumer list for such propagation by comparing $\boldsymbol{DP_{new}}$'s service metadata with *IDP's Service Request Subscriptions* (SRS) which include QoS requirement as well as Service *Context*, i.e. the functional requirements. Each $idp_i \in IDP$ is able to check the detail service metadata by querying *Domain-UDDI* and *DSS module* before $\boldsymbol{DP_{new}}$ can be appended to its local matching table. For instance, it may have specific QoS requirement at different time slot which is not publicly stated in their *SRS*. Meanwhile, the *Domain-Broker* also generates the matching table for $\boldsymbol{DP_{new}}$ itself. This matching table stores a set of *Domain-Peers* $PDP \subset DP$ where

$$PDP = \left\{dp_i \mid dp_i \in DP \,\&\, dp_i \textit{ is a potential provider of } DP_{new}\right\}$$

The potential provider is meant each $pdp_i \in PDP$'s published service matches with $\boldsymbol{DP_{new}}$'s intent of consumption and potential requirements – both *Context* and QoS – embedded in its *SRS*. $\boldsymbol{DP_{new}}$ can furthermore choose the GUI-based matching list so that the human (i.e. the decision maker) can mediate the service selection process and update the matching table interactively. As a result, each one of the involved $idp_i \in IDP$ as well as $\boldsymbol{DP_{new}}$ obtains an updated local matching table which is afterwards used for service interaction as a peer-to-peer routing mechanism. On leaving the domain, the $\boldsymbol{DP_{new}}$ notifies the *Domain-Broker*, who will then take the following steps 1) retrieve transaction history data (e.g. QoS report) from the involved $idp_i \in IDP$ and report them to DSS module for further review; 2) propagate the leaving message to all involved $idp_i \in IDP$ and $pdp_i \in PDP$, which will in turn perform certain routine operations accordingly (e.g. removing the entry from the matching table); 3) unregistered the $\boldsymbol{DP_{new}}$ from the *Domain-UDDI*.

The architectural design of Domain-Broker consists of three core components: the subscription queue, the matching engine, and the UDDI client. The domain interface corresponds to the domain-broker protocol, while the service interface corresponds to the global-broker protocol.

V. CONCLUSION and FUTURE WORK

In this paper, we proposed a new solution for QoS-based web services selection. Our solution is based on our observation that the service selection can be deemed as a process of decision making. Hence, the solution makes decisions for selecting the most suitable services by leveraging a DSS module, which relies on measuring the *Services Trust* against certain *Quality Criteria* defined in *Context*-specific *Quality Aspects*. To realize our solution, we place the DSS module into the *Domain-Broker*, one of the most important architectural components in the distributed web services architecture.

Currently, proof-of-concept prototyping work is ongoing. For the future work, we will focus on the implementation of the DSS module prototype embedded in the Domain-Broker. In addition, the QoS selection simulation is also need to be considered and the effectiveness measurement metrics should be formulated in a formative way.

VI. REFERENCES

[1] Shim, J.P., Warkentin, M., Courtney, J.F., Power, D.J., Sharda, R. & Carlsson, C., 2002, 'Past, present, and future of decision support technology', *Decision Support Systems*, no. 931.

[2] Gregg, D.G. & Goul, M., 1999, 'A proposal for an open DSS protocol', *Communication of the ACM*, vol. 42, no. 11

[3] Kreger, H., 2003, 'Fulfilling the web services promise', *Communication of the ACM*, vol. 46, no. 6.

[4] Cimetiere, J.C., 2003, 'Web services adoption and technology choices – Analysis of survey results', Technical Report, Group SQLI, *TechMetrix Research*.

[5] Maximilien, E.M. & Singh, M.P., 2004, 'Toward autonomic web services trust and selection', *Proceedings of the 2nd International Conference of Service-Oriented Computing*, November 15-19, New York, USA.

[6] Power, D.J., 2000, *Decision Support Systems Hyperbook*. Cedar Falls, IA: DSSResources.COM, HTML version, Fall 2000, accessed on (23rd Aug, 2005) at URL http://dssresources.com/dssbook/

[7] Alonso, G., Casati, F., Kuno, H., Machiraju, V., *Web Services – Concepts, Architecture and Applications*, Springer-Verlag Berlin Heidelberg New York 2004, ISBN 3-540-44008-9

[8] Chang, E., Dillon, T.S. & Hussain, F.K., 2005, *Trust and Reputation for Service-Oriented Environments: Technologies for Building Business Intelligence and Consumer Confidence*, John Wiley & Sons, ISBN: 0-470- 01547-0

[9] Hussain, F.K., Chang, E. & Dillon, T.S., 2005, 'Markov model for modelling and managing dynamic trust', *Proceedings of 3rd IEEE International Conference on Industrial Informatics*, August 10-12, Perth, Australia.

[10] Wu, C. & Chang, E., 2005, 'Comparison of web service architectures based on architecture quality properties', *Proceedings of 3rd IEEE International Conference on Industrial Informatics*, August 10-12, Perth, Australia.