Scientific Journal of Impact Factor(SJIF): 3.134
e-ISSN(O): 2348-4470
p-ISSN(P): 2348-6406

International Journal of Advance Engineering and Research Development

Volume 2,Issue 5, May -2015

ROOTS OF WEB SERVICE COMPOSITION

Tapasya Dinkar

Computer And Science Department,Saffrony Institute of Technology ,Linch,Mehsana

ABSTRACT: *Web Services composition has received much interest from both the academic researchers and industry to support cross-enterprise application integration. Promising research projects and their prototypes are being developed. At the same time the web service environment is getting more dynamic as numerous web services are being published by the service providers in the Internet. To meet the users requirements regarding on-demand delivery of customized services, dynamic web service composition approaches have emerged. But still many compositional issues have to be overcome like dynamic discovery of services, compositional correctness, transactional supports etc. In this paper we discuss some of these issues and then investigate some of the representative dynamic web service composition approaches. We evaluate those approaches on the basis of the issues and present how the future research can benefit by addressing those issues of dynamic web service composition.*

KEYWORDS: *WEB services, Service-Oriented Architecture (SOA), Dynamic web service composition.*

I. INTRODUCTION

1.1 WEB SERVICES

WEB services (WSs) are distributed and independent computational elements that solve specific tasks, varying from simple requests to complex business processes, and that communicate using XML messages following the SOAP standard. Current research studies how to specify them (in a formal and expressive enough language), how to (automatically) compose them, how to discover them (on the Internet) and how to ensure their correctness. We focus on service composition.[1]

The web services paradigm has emerged as a powerful mechanism for integrating disparate information technology systems leveraging a concept known as Service-Oriented Architecture (SOA).
A web service can be defined as a self contained, language neutral, platform independent, and loosely coupled software component that encapsulates discrete functionality and is described, published, located, and invoked programmatically over standard internet protocols. Web services are built upon already adopted technology standards namely XML, WSDL , SOAP , and UDDI. The web service architecture comprises three major players:[1]

1. The service provider that creates the web service, defines its description, and advertises it to a service registry.
2. The service requester searches the registry and binds to the desired service and invokes it.
3. The service broker/registry that provides a searchable repository of service descriptions.

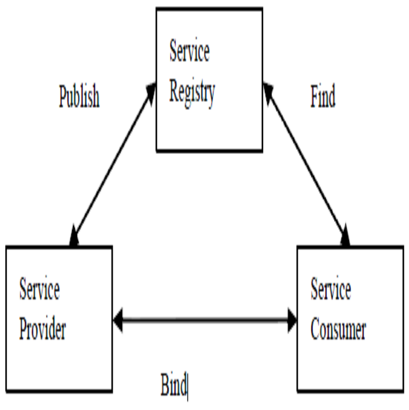

Figure1: Service Oriented Architecture[1]

II. SERVICE COMPOSITION

A main feature of services is the reuse mechanism to build new applications, which often need to be defined out of finer-grained subtasks that are likely available as services again. Composition rules describe how to compose coherent global services. In particular, they specify the order in which, and the conditions under which, services may be invoked. We distinguish syntactic (XML-based) and semantic (ontology-based) service composition.

Web Services Composition is a method to connect together various web services available for creating a high-level business process. It involves compiling of atomic web services to provide functionalities that are not available at design times. As a result a new functionality can be developed through reusing of components that are already available, but unable to accomplish a task on their own.

Web service composition can be classified into two types: **Static and Dynamic**. A static web service composition is performed at the compile time, whereas, the dynamic web service composition occurs autonomously when a user queries for a web service at runtime. However, dynamic web service composition involves much work compared to static web service composition.[2]

Web service composition involves compiling value-added services from elementary or atomic services to provide functionalities that were not available or defined at design times. It enables quick development of new application functionality through reusing components that collaborate to accomplish a task that cannot be provided by any of the existing services. Traditionally, composition is classified into **manual/automatic** and **static/dynamic**. Manual and automatic reflects whether the composition is performed by a human or a software agent. Static composition implies that the compositions is performed at design or compile time. Dynamic service composition, on the other hand, composes an application autonomously when a user queries for an application at runtime. Therefore, dynamic composition involves adapting running applications by changing their functionalities and/or behaviour via the addition or removal of service components at run time. There have been several benefits to dynamic service composition.[2]

1. **Greater flexibility** – the customisation of software, based on the individual needs of a user, can be made dynamic through the use of dynamic composition without affecting other users on the system.
2. **New services can be created at runtime** – the application is no longer restricted to the original set of operations that were specified and envisioned at the design or compile times. The capabilities of the application can be extended at runtime.
3. **Users are not interrupted during upgrades of applications** – instead of being brought offline and having all services suspended before upgrading, through the dynamic composition infrastructure, users can continue to interact with the old services while the composition of new services are taking place. This will provide continuous and seamless upgrading service capabilities to existing applications.
4. **Unlimited set of services** – unlike static composition, where the number of services provided to end users is limited and the services are specified at design time, dynamic composition can serve applications or users on an on-demand basis. With dynamic composition, theoretically an unlimited number of new services can be created from a limited set of service components.

1.2 DYNAMIC WEB SERVICE COMPOSITION

Although dynamic selection and composition of component services at runtime have been put forward as a promising approach to WSC, dramatic challenges are also recognized. First, the core of the dynamic service composition is the ability to identify, select, and integrate appropriate component services from a potentially large set of candidate Web services. That is, it requires an effective mechanism to find a composition of individual services that satisfies a variety of requirements. Second, another fundamental problem in service composition is to specify individual services at an appropriate level that enable precise understanding of services and allow on-demand composition.[3] WSDL does not deal with the dynamic composition of existing services. Although WSDL defines a standard way for service description, it lacks information for reasoning on not just what the inputs and outputs of a service are, but also what the inputs and outputs actually mean. Therefore, when services developed by different organizations use different semantic models to describe services, a compiler that can translate a Web service description language (e.g., DAML-S or OWL-S) into an agreed standard service description language is mandatory.[3] Third, a composition manager is required to control the invocation of individual atomic Web services and the data transfer between them. There has been some research on dynamic Web service composition.

2. Dynamic Service Composition Approaches

This section presents a brief overview of some of the prominent dynamic service composition approaches namely: eFlow , METEOR-S, WebTrans-act, DynamiCoS and SeGSeC.We choose these approaches on the basis of high references in literatures for dynamic service composition. Through this evaluation, we identify which of the features are supported by these approaches[5].

2.1 eFlow[5]

eFlow is a system that supports the specification, enactment, monitoring and management of composite e-services where the composite e-services are modelled as business processes. E services and web services share commonalities so we interchangeably use those terms in this paper. eFlow runs on the top of E-services Platforms (ESPs), such as HP e-speak or Sun Jini which allow the development, deployment and secure delivery of e-services to business and customers .The eFlow model and the overall system provide a flexible, configurable and an open approach to the service composition. The adaptive and dynamic eFlow process model allows processes to adapt the changes in the running environment, perform necessary service execution according to the need of the customer. E-service composition is modeled by a graph that denotes the order of execution among the nodes in the process. The graph is created manually but it can be updated dynamically. The graph may include services (represent the invocation of WS), decisions (specify the alternatives and rules controlling the execution flow) and event nodes (enable service processes to send and receive several types of events). Arcs in the graph denote the execution dependency among the nodes. eFlow includes the notion of transactional region and supports ACID service- level transaction. A transactional region enforces to maintain service level of atomicity. According to the definition of transaction support ,only preserving the ACID property of transaction is not sufficient to maintain transaction support in web services environment. Hence, transactional feature is not supported in eFLow. eFlow supports the modification of the process for dynamic service composition, but it can not guarantee the correctness of the output. eFlow does not provide QoS modeling capabilities. Service processes in eFlow are able to transparently adapt to environmental changes and dynamically configure at runtime. However, the limitation of the eFlow is that it needs too much manual participation to concretize the generic service nodes (a node in eFlow that supports dynamic process definition for composite services) at execution phase and it does not support the automatic generation of composition for the generic nodes. So, there is no support for the automatic composition. In eFlow, the composite services are modeled as processes that are enacted by a service process engine .So, the composition logic is process-driven and a composite service is described as a process schema that composes other basic or composite services.

2.2 METEOR-S[5]

METEOR for Semantic web services (METEOR-S) is a dynamic web service composition framework developed at the University of Georgia which incorporates workflow management for semantic web services. METEOR-S is the follow up research of Managing End-To-End OpeRations (METEOR). METEOR-S uses semantics for the complete life-cycle of the semantic web services. Its annotation framework is an approach to add semantics to current industry standards such as WSDL. METEOR-S uses techniques from the semantic Web, semantic Web services and the METEOR project to deal with the problems of semantic Web service description, discovery and composition. The Figure depicts the architecture of METEOR-S which has two parts: front end and back end.
The front end of METEOR-S is related with annotation and publication of service specifications. The abstract process designer which is related with dynamic composition is a component present at the back end of the METEOR-S. The composition process is initiated by creating the flow of process using the control flow constructs provided by WS-BPEL. The requirements of each service in the process is represented by specifying the service template, which allow to either specify semantic description of the web services or a binding to a known web services. Then, the process constraints for optimization is specified.

In METEOR-S architecture, the The constraint analyzer deals with correctness of the process based on QoS constraints. The support for state machine based verification of WS-BPEL process also contributes on the existence of compositional correctness. METEOR-S uses an extensible ontology to represent the generic QoS metrics and domain specific QoS metrics. The cost estimation module of constraint analyzer represents the QoS support. There composition process is not fully automated. The METEOR-S uses process-driven approach for composition. The coordination of the composite service is based on a BPEL-like centralized process engine.

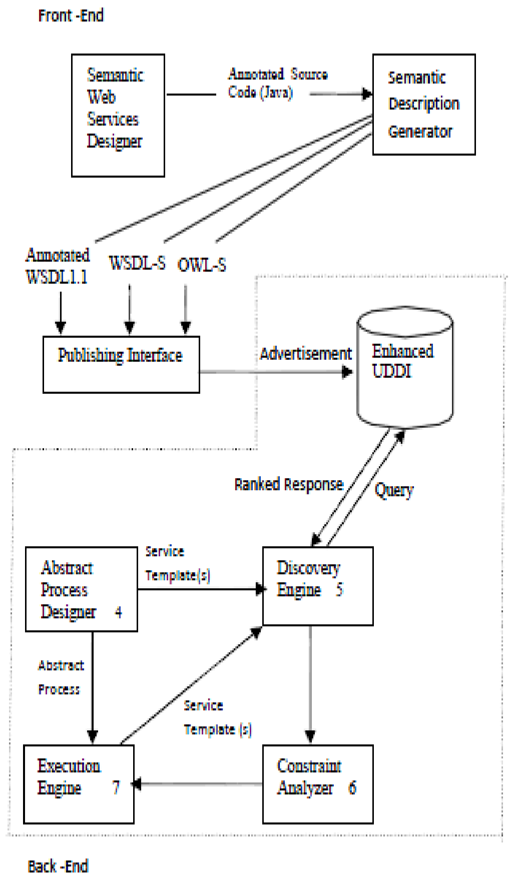

***Figure 2:** METEOR-S architecture[5]*

2.3 WebTransact[5]

WebTransact provides necessary infrastructure for building reliable, maintainable, and scalable web service composition. It is composed of a multilayered architecture, an XML-based language named Web Service Transactional Language and a Transaction model. The multi-layered architecture containing a Service Composition Layer, a Service Aggregation Layer, an Integration layer, and a Description Layer are depicted in Figure.. Based on , we provide a brief explanation of web service composition in WebTransact. Application programs interact with the composite mediator services written by composition developers. Such compositions are defined through transaction interaction patterns of mediator services. Mediator services provide a homogenized interface of semantically equivalent remote services. Mediator services also integrate web services providing the necessary mapping information to convert messages from the particular format of the web service to the mediator format.

In WebTransact, an XML-based language named Web Service Transaction Language (WSTL), is used for describing the transaction support. The transaction model of WebTransact provides an adequate level of correctness guarantees when executing the web services composition built with WSTL. Hence, there is transaction support in WebTransact. The transactional model of the WebTransact exploits the dissimilar transaction behavior of web services and guarantees the correct and safe execution of mediator compositions. The notion of correctness of composition execution is based on both the user needs (composition specification) and the 2L-guaranteed-termination criterion (a weaker notion of atomicity that considers the needs of web service environments). Hence, WebTransact has compositional correctness feature. WebTransact does not provide QoS modeling. The WebTransact approach does not support the dynamic discovery and integration of web services. The Web Services are statically integrated in WebTransact by a developer who plays the role of Web service integrator . So automatic composition is not supported. In WebTransact, web service composition is modelled as composite task by WSTL where a composite task is the combination of atomic task or another composite task. Tasks are identified by its signature, execution dependencies, links and rules. Here rules specify the conditions under which certain event will happen and can be associated with dependencies or to data links and finally evaluating either true or false based on execution. Hence, WebTransact follows the rule-based logic formulation.

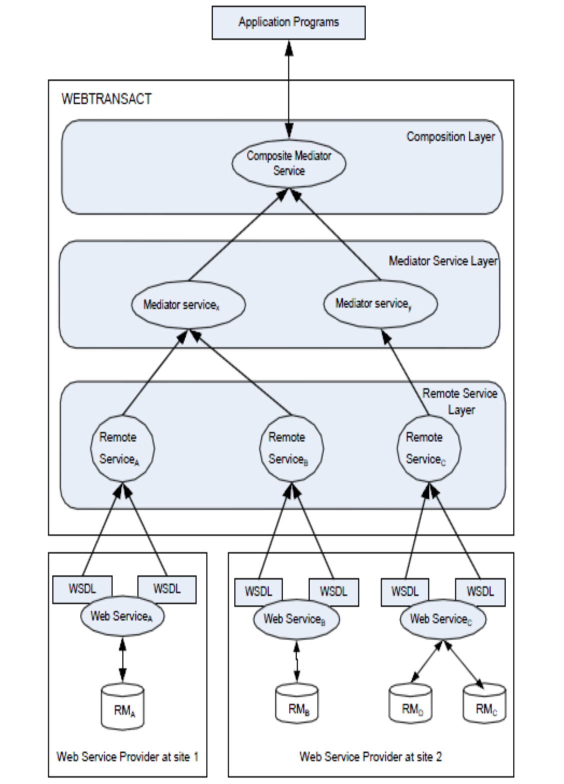

Figure 3: WebTransact Architecture[5]

2.4 DynamiCoS[5]

In, an approach for automated and dynamic service composition named Dynamic Composition of Services (DynamicCoS) is proposed primarily to alleviate the complexity of service composition from the end-users. Upon specifying the service specifications by the users as per their requirements, automatic discovery, matching and the composition of set of services that together fulfil the user's requirement is done by DynamiCoS. The results are then presented to the user who can selects the best suited composition. DynamiCoS represents services in language neutral formalism. A service is represented as a seven-tuple S =<ID,I,O,P,E,G,NF>, where ID is the service identifier, I is the set of service inputs, O is the set of service outputs, P is the set of service preconditions, E is the set of service effects, G is the set of goals the service realises, NF is the set of service non-functional properties and constraint values. DynamiCoS approach consists of following modules: service creation, service publication, service request, service discovery and service composition. Figure depicts the service composition framework of DynamiCoS. To perform composition, it _rst organizes the set of services discovered in service discovery phase in a Casual Link Matrix (CLM). The CLM stores all possible semantic connections, or causal links, between the discovered services input and output concepts. The main aim of Dynam-iCoS is to develop dynamic service composition mechanism to support the end users requirements. Consider-able interest is not given in transaction support in DynamiCoS. The service composition module of DynamiCoS builds compositions from service requests and the compositions are correct-by-construction.The algorithm for composition checks for deadlock and also verify if the composition is in accordance with the goals. In DynamiCoS some simple QoS characteristics can be represented and considered in service compositions. Among four ontologies, NonFunctional.owl in DynamiCoS de_nes non-functional properties for services and hence partially supports QoS modeling. DynamiCoS enbles service creation and publication by service developers at design-time, and automatic service composition by end-users at runtime .The composition is based on semantic graph based composition algorithm, so the composition logic formulation is processdriven.

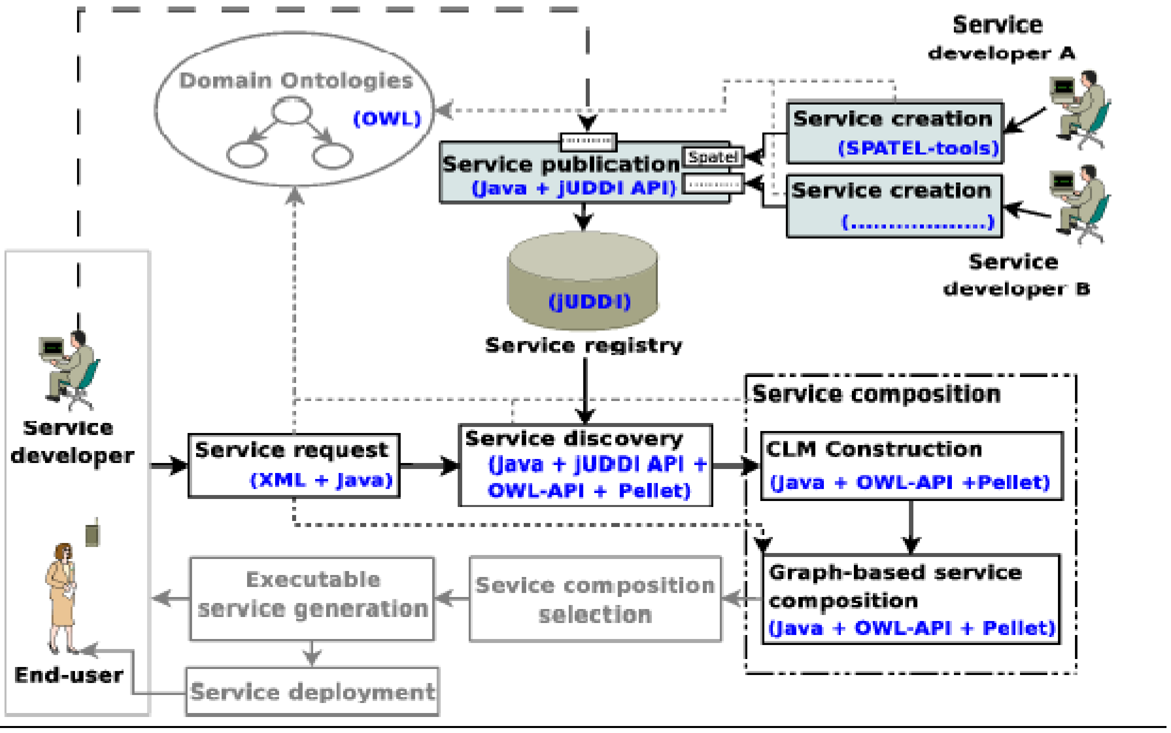

Figure 4:DynamiCoS Architecture[5]

2.5 SeGSeC[5]

In, the authors present a semantic-based dynamic service composition architecture named Semantic Graph-Based Service Composition (SeGSeC) in which the user requests the service in a natural language and the request is then converted into machine-understandable format, i.e. a semantic graph. Based on this semantic graph, SeGSeC composes services. In order to achieve semantic-based dynamic service composition, modeling of the service components and the service composition mechanism itself must support semantics. To satisfy this requirement, SeGSeC is supported by Component Service Model with Semantics (CoSMoS) and Component Runtime Environment (CoRE). The semantic support in the component modeling is achieved by CoSMoS which integrates the semantic and functional information into semantic graph representation. CoRE functions as middleware and provides

functionality to discover and convert different component implementations into a single semantic graph representation. Upon receiving the service request from a user in a natural language, SeGSeC generates the execution path or work flow. The execution path represents the order plan specifying which operations of which component should be accessed in what order. Additionally, SeGSeC also performs the semantic matching to confirm the semantics of the execution plan matches the semantic of the user request. SeGSeC does not provide transactional support and does not guarantee the compositional correctness. Given the user requirements in the natural language the overall approach generates the workflow such that it satisfies the semantics of the requested services. Starting from the service request from the user to the final composition the process is automated. SeGSeC does not provide QoS modeling capabilities. Upon receiving the user request in the form of semantic graph from CoSMoS, the Service Composer component of SeGSeC discovers the components and creates the workflow. In later stage of composition the semantic retrieval rules are applied onto the semantic graph such that the graph models the semantics of the workflow. Hence, SeGSeC has hybrid compositional logic formulation because the workflow is process-driven and later the rules are imposed in composition.

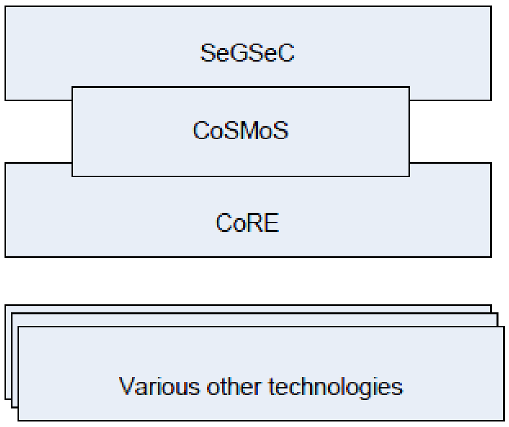

Figure 5: SeGSeC Architecture [5]

III. A FRAMEWORK FOR DYNAMIC WEB SERVICE COMPOSITION

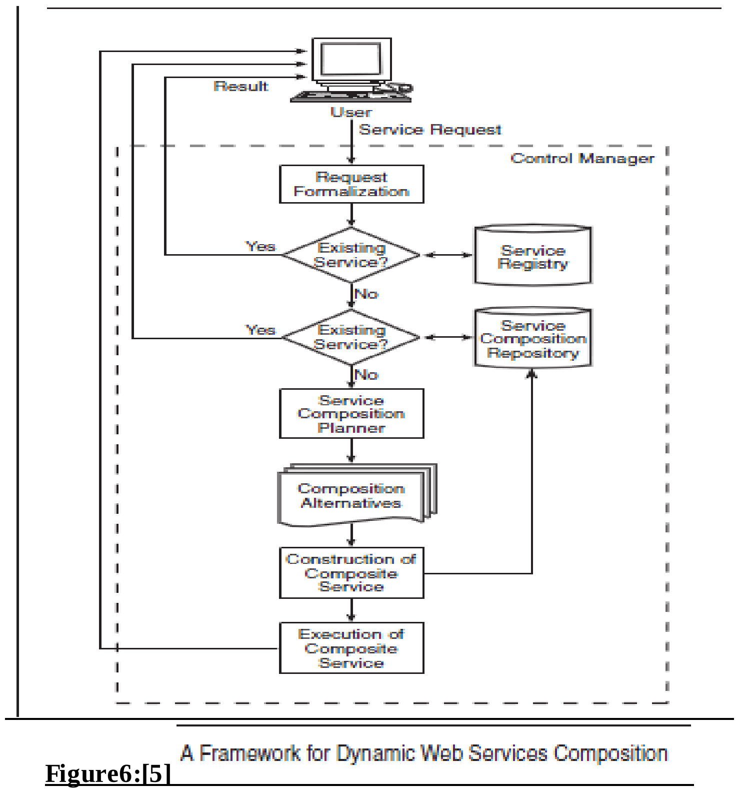
Result
User
Service Request
Control Manager
Request Formalization
Yes
Existing Service?
Service Registry
No
Yes
Existing Service?
Service Composition Repository
No
Service Composition Planner
Composition Alternatives
Construction of Composite Service
Execution of Composite Service
A Framework for Dynamic Web Services Composition

Figure6:[5]

A generic framework for dynamic and personalized Web service composition that highlights an integrated approach to the selection of component services. It takes three factors into consideration: (1) functional attributes of Web services, (2) non functional attributes of services, and (3) consumer preferences. The framework is shown in Figure. After a user's service request is received, it is converted into a logic format. Then, the control manager checks the service registry to see if any existing single Web service can fulfil the request. If yes, then the service is executed and results will be returned to the service consumer. Otherwise, the control manager examines a service composition repository, which stores previously created composite service plans and user requirements, to check if any of them can fulfil the request. If yes, the identified composite service is confirmed and executed, and results will be returned to the service consumer.
Otherwise, the control manager will require the service composition planner to select qualified atomic services and make an appropriate composition plan. Once an optimal service composition plan is determined, it is passed on to the construction phase, where the preparation for composite service execution is performed.
The generated composite service is stored in the composite service repository for possible future reuse. While planning offers the ability to automate the service composition process, finding a solution can be a problem of arbitrary

complexity. Therefore, it is beneficial to reuse existing, verified compositions when possible. Finally, the generated composite service
plan is executed.

Identification of Candidate Web Services

Identification of potential component services is mainly based on functional attributes of services because every candidate component service should satisfy certain functional requirements specified by a service consumer. This is not a trivial process. A practical challenge is that Web services are very likely developed by different organizations, which may use different semantic models to describe services. It is necessary that service consumers and providers use a shared ontology to denote concepts while describing services so that the description of services can be understood appropriately.
The functionality of a Web service should be described by a semantic annotation of what it does and by a functional annotation of how it works. To cope with the heterogeneity of service description, some researchers have attempted to use ontology and Semantic Web
technologies for WSC .
Ontology interoperability is important in service composition because semantic annotations
of Web services have been widely discussed in the Semantic Web community. The Semantic Web relies on ontologies to formalize domain concepts shared among services. Its aim is to enable greater access not only to content, but also to services on the Web. Service consumers should be able to discover, invoke, compose, and monitor Web resources offering specific services and having particular properties in an automated manner. DAML-S is a Web service ontology that provides a core set of markup language constructs for describing the properties and capabilities of Web services. It uses the Service class to model Web services with the properties presents, describedBy, and supports. The properties in turn have classes ServiceProfile, ServiceModel, and Service- Grounding as their respective ranges. The ServiceProfile gives a high-level description of a service that contains inputs, outputs, preconditions, and post-conditions of the service. It has similar functionality to the yellow pages in UDDI, and can be used by clients to select and locate services from registries. The Service Model is a detailed description of a service in which it is modeled as a process. It is similar to the business process model in BPEL4WS. The Service- Grounding provides the binding level information of how a client can access a service.

3.1 Selection of Atomic Services for Composition

Service composition emphasizes the global optimum by aggregating individual services through planning. A set of selected Web services must satisfy additional requirements in order to be composed into a complex service. Such requirements include connectivity, correctness, and scalability.. For example, every composition approach must guarantee connectivity, which indicates which services are composed and what will be input and output messages passed between ports; correctness of composition requires that properties of a composed service must be verified; and scalability demands that composition frameworks scale with the number of composed services, because it is likely that a complex service will involve many existing services in an invocation chain.
Traditionally, a service-selection component filters candidate services only based on the match between the consumer's functional goals and the functional attributes of services. WSC rests upon the coordination and collaboration among multiple Web services. Under-par
performance or failure of any single Web service may result in the failure of entire composed services. The majority of current approaches are based on an assumption that individual Web services are always available and well behaved. In reality, however, this is not always true. While selecting services to generate an optimal service composition plan in order to satisfy a service request, it is essential to develop and use a set of criteria for evaluating individual services. The evaluation criteria should be based on both objective and subjective features of Web services. The former is mostly function oriented, which verifies whether the requested functionality can be fulfilled by a service. The latter is non function oriented (e.g., quality of service, perception, reputation, service charge, trust, response time, etc.), in which services are judged on the basis of service quality and human perceptions. The objective measures are more fundamental, while subjective measures become increasingly important, especially when objective criteria are met or when there are too many options.

IV. A SELECTION OF SERVICE COMPOSITION CHARACTERISTICS

This section, describe the set of characteristics with respect to which we compare the abovementioned approache
s to service composition and the formal approaches that will be described later. We believe that any service composition approach should aim to support these characteristics; we of course do not claim these to be all characteristics of importance

A. Connectivity

Reliable connectivity is needed to reason about service interactions before composition, in order to guarantee the continuity of service delivery after composition. Measures of interest include the following:
1) Reliability: The ability to deliver responses continuously in time (service reliability) and the ability to correctly deliver messages between two endpoints (message reliability).
2) Accessibility: The responsiveness towards service requests.
3) Exception handling/Compensations: What happens in case of an error and how to undo the already completed activities.
The latter two measures in particular are receiving a lot of attention nowadays. Services often make use of external or third-party services (not owned and thus not under control) and hence one must take into account the fact that the latter services can unexpectedly fail. Since services are usually long-running processes that may take hours or weeks to complete, the ability to manage compensations of service invocations is critical.

B. Correctness

Service composition may lead to large, complex systems of concurrently executing services. An important aspect of such systems is the correctness of their (temporal) behaviour. The behavioural properties that a service should satisfy are usually defined by a specification that precisely documents the desired behaviour. Formal methods then provide rigorous mathematical means to guarantee a system's conformance to a specification.

1) Safety/Liveness: Safety properties are assertions that some undesired event never happens in the course of a computation, while liveness properties assert that some event does eventually happen. By verifying such properties, one obtains measures of correctness of a service (composition).
2) Security/Trust: The ability of a service (compositon) to provide proper authentication, authorization, confidentiality and data encryption. This requires the means to validate the credentials of a WS client, to grant, deny and revoke access to services, and to protect certain sensitive information or service functionality. A key property of trust is the assurance that a service (composition) will perform as expected despite possible environmental disruptions, human and operator errors, hostile attacks and design and implementation errors. C. Quality of Services.
There are several measures that determine the quality of service (QoS).
1) Accuracy: The error rate of a service, measured as the number of errors generated by a service in a certain time interval.
2) Availability: The probability that a service is available at any given time, measured as the
percentage of time a service is available over an extended period of time.
3) Performance: The quality of service requests, measured as response time, throughput and latency. Response time is the guaranteed maximum time needed to complete a request, throughput the number of completed requests over a period of time and latency the time needed to process a request.

V. CONCLUSION

Dynamic Web service composition is emerging as a new way of empowering E-businesses.
Building composite Web services can save extensive time and cost for developing new applications and enhance the interoperability and collaboration among E-business partners. Seamless composition of Web services has enormous potential in streamlining businessto- business processes and the integration of enterprise applications. The applications can determine not only what existing services can be bound together, but also how they should be carried out at runtime in order to fulfil users' complicated requests. However, WSC is a very challenging task due to the potentially large number of services that provide the same functionality, as well as the heterogeneity and changing nature of services.

REFERENCES

[1] Yashpalsinh Jadeja, Kirit Modi, and Ankur Goswami "Context Based Dynamic Web Services Composition Approaches: a Comparative Study" International Journal of Information and Education Technology, Vol. 2, No. 2, April 2012.
[2] Mohamad Eid, Atif Alamri and Abdulmotaleb El Saddik " A reference model for dynamic web service composition systems" Int. J. Web and Grid Services, Vol. 4, No. 2, 2008.
[3] Dongsong Zhang, Minder Chen, and Lina Zhou "DYNAMIC AND PERSONALIZED WEB SERVICES COMPOSITION IN E-BUSINESS" Information system management Summer 2005, volume 22.
[4] Maurice H. ter Beek, Antonio Bucchiarone, Stefania Gnesi "Formal Methods for Service Composition" ANNALS OF MATHEMATICS, COMPUTING & TELEINFORMATICS, VOL 1, NO 5, 2007, PP 1-10.
[5] Ravi Khadka1 and Brahmananda Sapkota2," An Evaluation of Dynamic Web Service Composition Approaches"