2010 Fifth IEEE International Symposium on Service Oriented System Engineering

A Community-Centric Model for Service Publication, Discovery, Selection, Binding, and Maintenance†

W. K. Chan ‡
Department of Computer Science
City University of Hong Kong
Tat Chee Avenue, Hong Kong
wkchan@cs.cityu.edu.hk

Lijun Mei
Department of Computer Science
University of Hong Kong
Pokfulam Road, Hong Kong
ljmei@cs.hku.hk

Zhenyu Zhang
State Key Laboratory of Computer Science
Institute of Software Chinese Academy of Sciences
Beijing, China
zhangzy@ios.ac.cn

Xiaopeng Gao
School of Computer Science and Technology
Beihang University
Beijing, China
gxp@buaa.edu.cn

***Abstract*— Services discovery, selection, composition, verification, and adaptation are important in service-oriented computing. Existing researches often study techniques to maximize the benefits of individual services. However, following the power laws, a small fraction of quality services offers their executions to support a significant portion of all service requests. We argue that locating and maintaining such a small and significant set of services is important to the development of service-oriented computing. In this paper, we propose the notion of *adaptive service-oriented community*. A community consists of peer-reviewed services, and only those operations of member services that the community collectively exceeds a significance threshold are discoverable and bondable. Services also select such communities to bind to its requested operations primarily based on their significance. Our proposal essentially raises a service ecosystem from pursuing the benefits of individual services to that of the community as a whole. Our model also has features to make a namespace or a web service privacy-aware.**

Keywords—service community, significance, adaptation, privacy-awareness

I. INTRODUCTION

A web service is a process with public interface in a standardized format. It sends messages to, and receives messages from, other web services over standardized protocols. To compose an application, multiple web services may collaborate to form a *service composition* [17]. To ease our presentation, this paper also refers a service composition to a web service.

A precursor of such a collaboration is a criterion to locating a set of candidate web services [11][13][21] followed by matchmaking of their interfaces and protocols [9][18], ranking of these candidate web services [8][11][22], and finally identifying (i.e., selecting) a subset of these web services to build a target application. *Service discovery* is the usual name to describe such a mechanism.

Traditionally, process engineers conduct service discovery at design time, and codify their service discovery decisions as, for example, locators in WS-BPEL applications [20]. However, design-time service discovery makes resultant service compositions harder to respond to unplanned changes. For instance, supposed that a trip planner service having a lower brokerage fee than that of a participating service of an application has published its interface publicly, a design-time service discovery could be unable to alter the service composition dynamically to use this newly published service. A simple and existing solution is for a process engineer to maintain the configuration or the code of a web service to implement a required change, publish the interface of the new service, and withdraw the preceding version of the interface.

A natural step toward automation is to let the service deployment, publication of new interfaces and withdrawal of existing interfaces be handled by a web service environment. For instance, developers may add an interception level at the underlying middleware of such applications so that a service request can be diverted to a web service locator selectable at run time [17]. Although such a mechanism dynamically binds a service request to various web services, yet such an underlying interception approach still requires the process engineers to provide a designated web service (locator) in advance. In other words, the problem of dynamic service discovery is still outstanding.

† This research is supported in part by the General Research Fund of the Research Grants Council of Hong Kong (project no. 123207 and 717308), the Strategic Research Grant of City University of Hong Kong (project 7002464), and the National High Technology Research and Development Program of China (project no. 2007AA01Z145).

‡ Correspondence author.

978-0-7695-4081-8/10 $26.00 © 2010 IEEE
DOI

To design a dynamic service discovery mechanism is however challenging, which notably includes solving the service matchmaking problem and the service selection problem. Concisely, the service matchmaking problem is to determine whether a pair of web services can collaborate, and the service selection problem is to identify a candidate service from a set of matched web services.

To address the matchmaking problem, researchers propose techniques to determine whether the syntactic interface protocols of web services are type-compatible, and if feasible, to generate interface adapters to chain up these web services [5]. Some techniques associate web services with semantics information, usually in the form of an ontology, which describes how a tag inscribed in a public syntactic interface is related to some tags (through hypernym, hyponym, synonym, antonym, and aggregation relations) inscribed on the public syntactic interface of another web service.

Syntactic matchmaking [9][13] is more general than the one using an ontology because the former kind does not assume the existence of any semantic relations among the tags used by different web service interfaces. Nonetheless, multiple operations or web services with different behaviors may share the same syntactic interface. Using pure syntactic matchmaking may result in selecting candidate web services with unintended behaviors.

To use syntactic matchmaking effectively, one needs a follow-up service selection technique to address this problem of incompatible behaviors or a similar kind. For instance, one of the existing service selection techniques is to apply the notion of majority voting strategy [19] as in fault tolerant systems: All candidate web services execute the same set of test cases. Their corresponding results are mutually compared by stages, where at each stage, the current subset of candidate web services with a non-majority result is discarded. Finally, the technique identifies a pool of candidate web services with identical test results. One may then select a web service from this restricted candidate pool randomly or according to certain functional or emerging requirements. Nonetheless, such a web service may not match the expected behavior of the service to be discovered.

Semantic matchmaking addresses the above-mentioned unintended behavior problem, yet having an ontology or a precise pre- and post-conditions for every operation of every web service may severely compromise large-scale deployments of web services. For instance, many successful medium to large-scale software applications have no formal specification, and existing web services APIs (such as the API provided by Google, Amazon or Microsoft) are documented informally.

We observe that real-world web services usually provide their clients a set of cohesive services, such as medical services, trip planning services, or zip code lookup services. In theory, the kinds of services provided by one web service can be unlimited. In practice, a vast majority of web services is domain-specific. That is, the kinds of functions offered by a web service usually belong to the same domain. For instance, http://www.webservicelist.com shows that there are service categories for Stock Quotes, Payment API, Retail API, Healthcare, Search, Content, Address, Access, to name a few. We also observe that some generic services may merely aim at passing the service test suites provided by Sun or Microsoft.

In each domain, each web service provides a kind of services meaningful to the domain. For instance, on a Yellow Page directory, we do not expect to find a hairdresser in the list of medical doctors. Indeed, Yellow Page enforces this kind of classification. Similarly, there are many research communities. Each research community has its own signature conferences. Each signature community talks in the same jargons, yet different communities may use the same terminology to refer to different concepts.

A namespace is a domain that can be codified in software artifacts. Existing web services already use namespaces to resolve various name references. Moreover, following the notion of power laws, a small fraction of quality services offers their executions to support a significant portion of all service requests. Motivated by this widely accepted real-world practice, in this article, we propose the notion of *adaptive service-oriented community*, which is a namespace-centric strategy, for service-oriented computing.

A community contains peer-reviewed services that they may vote to accept a candidate service to join the community, but are free to leave. Each service offers the community its set of operations, which have been peer-reviewed by the current or former member services of the community. On the other hand, only those operations of member services that the community collectively exceeds a significance threshold can be discoverable by other services and bondable to service requests. Services also select communities to bind to its requested operations based on their significance, relevance and quality requirements. Our proposal essentially raises a service ecosystem from pursuing or maximizing the benefits of individual services to that of a community.

Apart from using the notion of significance as an indicator of the quality of a namespace, our model has a number of unique features. A namespace may contain many web services that have previously been validated to be compatible to existing web services in providing a function with the same interface. Our model probabilistically verifies the namespace in the spirit of regression testing of software programs whenever there is any change in its constitution.

Moreover, unlike many existing proposals essentially treat a web services repository as a passive collection of web services, a namespace in our model has its own behavior, which is a collection of the non-deterministic choice compositions of the compatible operations of the namespace. Web services may rely on a namespace to select a requested operation from a non-deterministic choice composition of the namespace. Hence, rather than individual services being benefited from the community, the services that support the same non-deterministic choice compositions can collectively benefit from such a design.

Moreover, the privacy of web services under a namespace is protected by a few ways in our model. For instance, web services may hide some operations when they join a namespace. They may also choose to join a non-deterministic choice composition for the namespace to resolve a web service to fulfill a web request. Moreover, operations of a web service that are not significant enough will remain unknown to other web services or namespaces.

The main contribution of this paper is to propose a significance-led model to formulate the notion of adaptive service-oriented community.

We organize the rest of the paper as follows. In Section II, it presents our model, followed by a review of related work and a conclusion in Section III and Section IV, respectively.

II. MODEL

A. Namespace, Web Services, and Operations

In this section, we define some basic terminologies and present our model.

Namespace: The basis of our model is a *namespace*. A namespace models a community that a web service may join and offers its ("professional") behavior on behalf of the community to other web services. We abstractly identify each namespace by a unique identifier (i.e., an alphabet).

A namespace query q is a characterization of the intended namespaces. We suppose that a global namespace repository is available for q to operate. For instance, all namespaces can be registered as entries in a UDDI repository, and one can use an XQuery q to extract the matched namespaces kept in the repository. To activate a query, there are many approaches such as using an event-condition-action (ECA) approach where the action is the query. Alternatively, it may on the process of a web service to invoke such a query.

We proceed to define what a web service is and come back to present additional properties of a namespace.

Web Services: We model a web service as a triple $\langle w, \Psi, \Omega \rangle$. In the triple, w is a unique identifier of the web service, Ψ is a set of operation signatures provided by the web service, and Ω is a set of triple in the form of $\langle q, \Delta, o \rangle$. Each triple $\langle q, \Delta, o \rangle$ is a namespace query q, an operation signature Δ, and the operation o (of a particular web service) to bind to this operation request. A triple $\langle q, \Delta, o \rangle$ models that w requests an operation with an operation signature Δ from a namespace that is retrievable by the query q (which can be a wildcard, denoted by $*$), and finally resolves to use (i.e., bind to) o by w. To ease our reference, we say such o as a requested operation.

When o is a wildcard, it means that no service has been selected to bind to this service operation. A few possible scenarios may lead to this condition. For instance, the service w has not discovered any concrete service to implement this requested operation. Alternatively, the requested operation has been depreciated or is no longer available, and therefore, the original bond defined in the above-mentioned triple in Ω is outdated. If the requested operation is no longer available, our model treats it as an undefined operation implementation as in the first scenario.

To ease our reference, we use the notation *namespace*(o) to refer to the namespace of the web service that binds to this operation o. We refer to the set of concrete namespace captured in Ω to be *namespace*(Ω).

Abstraction Operations of Namespaces: Whenever a web service w joins a namespace n, in our model, the signature α of each provided operation o of w is also appeared as an abstract operation (signature) α of n. It is this abstract operation α to be matched with a requested operation signature Δ specified in Ω component of a web service.

In our model, a namespace may use its set of available abstract operations to determine whether their requested operations are accessible by web services, and to select one of the operations to communicate with the requested web service. We will present our idea in the section "*Web Services Joining/Leaving Namespaces*".

We recall that the namespace represents a community of its joined web services. We further define the behavior of a namespace (aka community) n as a collection of compositions (in the sense of process algebra), each of which is a non-deterministic choice composition of the compatible provided operations of its member web services. Therefore, the behavior of the namespace n changes as web services join n or leave it.

Definition of Namespace: Put it all together, a namespace is a triple $\langle n, A, M \rangle$ where n is the unique identity (alphabet) of the namespace, A is the set of abstract operations currently available at the namespace, M is the set of web services currently joined the namespace.

Web Services Joining/Leaving Namespaces: A web service w may join a namespace $\langle n, A, M \rangle$ or leave n dynamically. To join n, the namespace employs a scenario-based voting strategy to determine whether the community M accepts w to join the namespace. Specifically, a web service $\langle w, \Psi, \Omega \rangle$ has a set of provided operations Ψ, each of which shares an abstract operation α ($\in A$) with an existing set X_α of web services that currently join n. For each such abstract operation α, the web service w submits a non-empty set of service composition scenarios (denoted by SC) to the namespace n. Each such service composition scenario is an operation sequence (or a process) that w offers α to collaborate with a set of "*significant*" web services from various namespaces (see Section II.B for more details about significance.)

For each submitted service composition sc in SC, the namespace executes the given scenario sc to ensure that the scenario is also feasible with respect to w. Moreover, the namespace randomly selects, with a uniform distribution, zero or more web services from X_α. Such a selected web service w' is to replace the position of w in sc so that the namespace can

"run" the service composition scenario $sc[w'/w]$ to determine whether w is behaviorally compatible with w' in supporting sc. (Note that $sc[w'/w]$ stands for a service composition that every occurrence of w is replaced by w'). To ease our reference, we call w' as a replacement candidate of w.

Each replacement candidate (say w') in X_α votes on behalf of the namespace on whether the namespace may accept w to provide α. Moreover, the namespace also selects a subset of existing service composition scenarios that offer α (each of which is submitted by a current web service of the namespace) and test w to provide α in executing these scenarios. These current web services then vote to accept w to provide α or decline w. We have obtained two voting decisions. The namespace accepts w to provide α only if the results of both votes are for w to provide α. We further note that this bidirectional testing scenarios provide a continuous monitoring and assurance on the quality of the namespace.

For instance, a resultant service abortion of $sc[w'/w]$ may indicate that w is incompatible to w', or the resultant quality as specified by sc is unacceptable to w'. In the former case, it reveals that w does not provide compatible services as required by the community M. In the latter case, the quality of w has not reached the perceived *average* quality of the community that offers α. Our model chooses to use the average quality rather than the minimal quality of the community because our model uses a random approach to selecting the set of replacement candidates in the above-mentioned testing phase. In either case, service abortion may result, which leads the replacement candidate to vote for rejection.

As described above, a namespace uses a voting strategy (e.g., majority voting strategy) to determine whether the corresponding community accepts w to provide α. The web service w is accepted by n only if every provided operation in a given subset Ψ_S of Ψ is accepted by n. To ease our presentation, we use a predicate $vote(n,w)$ to denote whether or not the namespace n accepts w.

Note that we choose to allow a web service to expose a subset Ψ_S of Ψ for the web service to join a namespace. It is because a web service may choose to hide some of its operations from a namespace.

Adding an operation $\alpha 1$ by a web service $w1$ to a namespace $n1$ is handled by the same scenario testing-voting strategy to evaluate $vote(n1,w1)$. Revoking an operation $\alpha 1$ of $w1$ from $n1$ can be done by $w1$ without any precondition. A revision of $\alpha 1$ of $w1$ is modeled by revoking $\alpha 1$ from $n1$ followed by adding $\alpha 1$ to $n1$ (or the other way round). To leave a namespace $n1$, a web service must remove all its operations from $n1$.

Namespace Maintenance: To maintain the community of a namespace automatically, we observe that a web service may cease to exist, an existing operation of a web service may be obsolete, or the quality of an operation of a web service may evolve and become below the average quality of the community in the above-mentioned bidirectional continuous testing.

To handle the first case, a namespace may invoke each web service occasionally. To handle the remaining two cases, our model uses the scenarios provided by "new" members as test cases to validate existing members. We recall that adding an operation α by a web service w to a namespace n must accomplish with a non-empty set of service composition scenarios SC. Whenever an operation α is accepted by the namespace $\langle n, A, M\rangle$, the model randomly selects a subset (probably empty) SC_X of SC, and randomly selects a subset M_X of M such that each web service in M_X has an operation corresponding to the abstract operation $\alpha (\in A)$. It then uses the above-mentioned testing-voting strategy to decide whether each selected web service $w1$ in M_X can lead $vote(n,w1)$ to be evaluated to be true.

If $vote(n,w1)$ is false, every operation α of this web service $w1$ will be revoked from n. This mechanism is similar to regression testing to assure software programs. Intuitively, all requested operations specified in each web service will be delinked (see *Web Services* section for more details).

There are several outstanding problems to be addressed. First, a web service may add a provided operation with a new signature to a namespace. Therefore, in principle, many namespaces may offer operations sharing the same signature. Existing web services may aim to limit the amount of competing services in the same namespace. Second, if a web service decides to offer an operation publicly, the web service may wish to join a namespace that may maximize the potential usage of the web service. We address these problems in the next section.

B. Selection of Services by Significance

In Section II.A, each namespace $\langle n, A, M\rangle$ contains a set of web services M, and each web service w ($\in M$) links its set of requested operations to a set of operations of some namespaces. Conversely, each abstract operation α of n is bound by a set of operations of web services that currently join n or other namespaces. Using this link information, our model computes the citation significance of the namespace, and uses these significance indices to guide service selections.

Namespace Significance: Let us denote the number of web services bound to an abstract operation α of the namespace n by $|\alpha(n)|$. To know the significance of a namespace n, we apply the notion of ***h*-index** $h(n)$ [10]: A namespace has an index h if h of its abstract operations have at least h number of web services bound to, and each of the remaining abstract operations has at most h web services bound to. That is, $h(n) = |X|$ and $X = \{\alpha \in A \mid \langle n, A, M, N\rangle$ is a namespace $\wedge\ |\alpha(n)| \geq h(n) \wedge\ |\{\beta \in A \backslash X \mid h(n) \geq |\beta(n)|\}| = |A| - |X|\}$.

A necessary condition of an abstract operation α of n to contribute to the h-index $h(n)$ of the namespace is $|\alpha(n)| \geq h(n)$. Our model restricts the *selectable* web services to be

those having $|\alpha(n)| \geq h(n)$. In this connection, no abstract operation χ having $h(n) > |\chi(n)|$ would be selectable until at least $h(n)$ competing web services offers χ in the same namespace. This restriction aims to assure any selectable abstract operations to be relatively significant (with respect to the abstract operations of the namespace).

In our model, every *selectable* abstract operation α of a namespace n (having $|\alpha(n)| \geq h(n)$) will share the same h-index $h(n)$. In other words, this h-index value becomes a threshold for a service to be discoverable, selectable and bondable by web services.

Hence, if the namespace n votes to reject a new candidate operation α of a web service to join n aggressively, the h-index of α of n may remain small in value. If there are multiple namespaces providing the same abstract operation, our service selection scheme (see below) will select a namespace for the service discovery of the requested operation based on their relative significance. Therefore, a namespace having a smaller h-index value will receive fewer binding opportunities than another namespace having a larger h-index value. On the other hand, if a namespace is non-selective to allow web services to join the namespace, many service compositions using the operations of the namespace may experience service transaction problems. Our model also has a special feature in service selection to cope with this issue (see *Service Selection* section for detail).

Service Selection: To locate a namespace to provide an operation α, a web service w first discovers a series of namespaces $\langle n_1, n_2, \ldots, n_k \rangle$, each of which offers α with a parallel series of h-index values $H_{w,\alpha} = \langle h_1, h_2, \ldots, h_k \rangle$. We define $H_{w,\alpha}[i] = h_i$ to ease our subsequent presentation.

Our model uses the normalized ratio of these h-index values as the first estimate of the probabilities of selecting individual namespaces to provide a requested operation: the first estimate of the probability p_i to select n_i is $H_{w,\alpha}[i] \; / \sum_{j=1\ldots k} H_{w,\alpha}[j]$. We call this estimate as the significance component of our service selection assessment.

However, the operation provided by a namespace may be irrelevant to w. Our model addresses this problem by using the popularity of the requested link statistics of other web services who share the same namespace with w in addition to the above-mentioned probability.

Suppose that $\langle n1, A, M \rangle$ is a namespace such that $w \in M$. Let a web service in $M \setminus \{w\}$ be a triple $\langle w2, \Psi, \Omega \rangle$, in which the third component of each triple in Ω is an operation of a web service.

Our model collects the set of namespaces from all such web services in $M-\{w\}$ that each web service has replied the namespace that the web service is willing to fulfill the web request. To simplify our presentation, we simply use $M-\{w\}$ rather than its subset under the above-mentioned restriction to describe our model. In other words, we collect the set $Z = \cup_{j \in Y}$ *namespace*(j) where $Y = \{ o \mid \langle w2, \Psi, \Omega \rangle \in M-\{w\}$ and $\Omega = \langle q, \Delta, o \rangle\}$. A namespace then computes the number of occurrences of each namespace in the set Z, and let us denote such a number for a namespace n_i as r_i. Our model further computes the probability q_i to select the namespace n_i by $r_i / \sum_{j=1\ldots|\nabla|} r_j$, and we call it the relevance component of our service selection assessment.

Our third component is the service quality experienced by w. Each web service keeps the history of successful and failed service collaborations (i.e., service transaction) with each invoked service. We recall that each web service that provides a requested operation joins exactly one namespace. Our model computes the total number of successful service transactions and failed service transactions for each namespace that w perceives. Our model further uses the following ratio to stand for the perceived execution quality of the namespace of w: *Number of successful service transactions* ÷ (*Number of failed service transactions* + *Number of successful service transactions*). A web service w can maintain one such ratio for each namespace. We use s_i to denote the normalized ratio for the namespace n_i and use it as the probability to select the namespace as the quality component of our service selection assessment.

Our model finally computes the *geometry mean* of the above-mentioned three probability values, that is, the cubic root of $p_i \times q_i \times s_i$ to denote the relative probability of the service to select the namespace to provide the operation. Nonetheless, a relevant namespace may not be in the list of namespaces whose provide the required operation. To address the zero probability problem, in case that q_i is zero but not p_i, our model substitutes this zero by a small positive constant ε, which is defined as one-tenth of the $min_i\{$ non-zero $sqrt(p_i \times q_i)\}$ (i.e., the smallest non-zero geometric mean obtained) for all discovered namespaces by the web service w for the operation α.

The web service w finally uses this modified geometric mean as the relative probability to select a namespace among the discoverable namespaces to bind its α in its $\langle q, \Delta, \alpha \rangle$ to a web service in the namespace, which is determined by the namespace internally. Once this bond is established, the web service uses this operation to serve its transaction.

C. Discussion

In this sub-section, we discuss various design decisions made in our model.

Significance: The first aspect is the usage of significance. H-index is based on the notion of power law. Apart from using the current significance component formula, one may use $\exp(H_{w,\alpha}[i]) \; / \sum_{j=1\ldots k} \exp(H_{w,\alpha}[j])$ or other strategies that align with the power law property. Moreover, apart from using h-index, there are quite a number of significance indexes such as g-index. Our model can use these indexes instead of using h-index. Finally, all available operations of the same namespace share the same h-index value. This reduces the complexity of

our model and its presentation. A more dynamic design may allow individual operations of each namespace to have their own h-indexes.

Model Setting: The second aspect is about our model setting. To estimate the relative probability of a namespace is selected, this paper presents a model having three components, namely, significance, relevance, and quality. Our model uses geometric mean to integrate these three components to yield one value. This approach can be extended in a similar way to handle different combinations of components. Moreover, one may replace geometric mean by other integrators.

However, to cope with some practical issue, we use a modified geometric mean, which introduces an artificial parameter in our model. Currently, we arbitrarily choose it to be one-tenth of the smallest non-zero uses geometric mean of two chosen components. It is interesting to replace it by a non-parametric approach. However, our model has not formulated it yet.

In the quality component, we use the ratio of two types of service transactions as a criterion. We may further factor in the Quality of Services (QoS) if there is a scalar metric to measure the corresponding QoS dimension (e.g., performance). Integrating multiple QoS dimensions can be done by the geometric mean approach. Let us consider an example. Suppose that x is a QoS random variable and x_0 is the required QoS threshold. we may compute the change in quality between successful service transactions and failed ones: *Probability*($x \geq x_0$| service transactions are successful) − *Probability*($x \geq x_0$| service transactions are failed). Many formulas can be set up to assess the quality. It is interesting to explore them.

Second, our model assumes that after having the bond is determined, services may use the bound services and evaluate their suitability to serve the objectives of the former services. We tend to believe that the best (i.e., the most significant) slice of the namespaces as a whole would attract those widely used, highest quality web services. At the same time, because these widely used services are successful. They naturally attract many other web services to offer similar functionality.

Our setting is designed to encourage providers of small web services to offer individual (but compatible) functionalities to compete with these widely used services. In this case, the h-indices of these web services or the threshold of the namespace may improve.

Efficiency and Effectiveness: The third aspect is about the efficiency and the effectiveness of the model. Currently, to maintain a community, there is a bidirectional testing, which incurs many overheads. Our current solution is to apply a sampling approach to selecting services to be "audited". For a large namespace or many test scenarios being available, the approach can still be inefficient. It is interesting to study how to improve the effectiveness of the model maintenance. One of the concerns is our aggressive way to enforce an existing web service to leave a namespace. Currently, the model chooses to use the perceived average quality of the namespace in a testing-voting mechanism for benchmarking. One may choose other levels such as the 25% percentile or the minimum rather than the perceived average quality.

Privacy-Awareness: The fourth aspect is about *privacy-awareness*. In our model, there are a number of privacy controls. First, an operation of a web service cannot be seen by other web services until the corresponding abstract operation reaches the significance threshold of the namespace. Therefore, the web services that provide less significant operations will remain unknown (i.e., undiscoverable) to the other web services or namespaces.

Second, in fulfilling an operation request, our model chooses to use a non-deterministic choice composition. It does not allow a caller to identify a particular web services. Rather, our model provides the namespace in question an opportunity to resolve the web services to fulfill the web request. For instance, the web request from a client web service can be forwarded to individual web services for them to reply to the namespace whether they allow the client web services to be selectable. Based on the replies, the namespace may resolve the non-deterministic choice composition internally. In this design scenario, the web services involved in the choice composition can be invisible to the client. Even the set of web services can be somehow determined by the client, the non-deterministic choice composition makes the client unable to determine whether their target web service says "no" and whether it is simply the choice resolution selects another web service to fulfill the web request.

Third, a web service may hide some operations to be visible to a namespace to be joined. This design further protects the privacy of the web services under a namespace.

III. RELATED WORK

WS-Agreement [9] aimed to describe the (syntactic) interface of service compositions. Oldham et al. [18] enhanced it with semantics information. Based on such schemes or its kind, researchers proposed to generate composition plans.

For instance, Mokhtar et al. [16] constructed such a plan based on the automata descriptions of services. However, even with such a plan, applications still need mechanisms to select services to implement them. Casati et al. [4] pointed out that a static service binding is often too rigid to adapt changes in user requirements, to decouple service selection from process definition, and to dynamically discovering the best available service that satisfies a specific process definition. They used rules and policies to guide the selection of services.

Other researchers also studied the problem of service selection using static approaches such as pattern recognition [24], Petri nets [23], and graph network analysis [8], or using dynamic approaches to support diverse types of application [4][13][16]. These ideas focused on the benefits of the service consumers; whereas, our model focuses on the benefits of the community of web services as a whole.

Many existing techniques selected services by using nonfunctional quality attributes [1][2][11][18][21]. There are also proposals to select services based on the functional aspect such as the perceived successful executions [13] .

For instance, Bonatti and Festa [2] designed algorithms to optimize the service matchmaking procedure at the service level under different selection criteria. In our model, we propose using a significance index as the primary criterion. Our model also associates this criterion to operations, web services, and namespaces. It seems to us that their algorithms may be adaptable to operate on our criterion.

Ardagna and Pernici [1] proposed a flexible approach to optimizing individual service compositions, in which they modeled policies as user-provided utility functions. Similar to [1], Xiong and Liu [21] studied a similar problem from the trustworthiness perspective. Our model has not extended to handle policies.

In service matchmaking, Lamparter et al. [11] captured constraints in ontology and formulated efficient algorithms based on their ontology proposal. In our model, we select services based on their community significances, which can be classified as a syntactic approach.

Like [13], we use a kind of link analysis technique to formulate significance. Unlike [13], we do not use a random walk approach to computing the popularity of node. Rather, we use significance indexes directly with the aim of reducing the computation overheads.

Bonatti and Festa [2] examined the service selection problem as a multi-attribute decision problem. Unlike their proposal, we use the notion of geometric mean to combine multiple attributes to make decisions. Zeng et al. [22] developed a middleware platform to select web services. They performed service selection at the task level and globally, which aims at maximizing user satisfactions. We have not explored the efficient support from middleware yet.

Zhang et al. [23] proposed to use a Petri-net based specification model for web services to facilitate verification and monitoring of web service integration. In our model, collaboration scenarios can be reused not only in testing, but also in joining or monitoring web services of a namespace.

None of the above-reviewed studies considered service activities at the community level. With the advance of cloud computing [12][15], we may virtually consider that all web services are kept in a cloud. Thus, distribution of significance and binding information can be handled by the underlying cloud infrastructure. This paper has not explored the execution model of a service in a cloud [7].

Bucchiarone et al. [3] reported a survey in 2007. We observed from their surveys and our own studies [6][14] that many existing testing, analysis, and verification techniques focused on checking individual service compositions or services. It is interesting to know how to verify and validate a service community. In our model, we test a community whenever there is a change in membership.

IV. Concluding Remarks and Future Work

In this paper, we have proposed a new kind of model to formulate the notion of *adaptive service-oriented community*. Our model essentially promotes to raise a service ecosystem from pursuing or maximizing the benefits of individual services to these of a community as a whole.

A pool of peer-reviewed web services may form a community, in which compatible operations may be grouped together by the community. We refer to a community as a namespace. When a critical mass is reached, these operations can be discoverable, selectable and bondable under the namespace by other web services. We have proposed to use a significance-centric metric to determine the threshold of attaining the critical mass for individual namespaces.

In our model, a namespace has its own behavior, which is the collection of all non-deterministic choice compositions of the compatible operations of the namespace. Individual web services may rely on the namespace to select a requested operation from the non-deterministic choice Composition. We have proposed to combine the namespace significance, the relevance of the namespace of the web service to the namespace that provides the operation, and the perceived quality of the web services on the operation into one value, and use this value to select a namespace probabilistically and select a web service of a selected namespace non-deterministically. In particular, the services that support the same non-deterministic choice composition can benefit from such a design. The significance-centric operation discovery and the non-deterministic resolution of web services in a choice composition provide opportunities for namespace and web services to be privacy-aware.

Moreover, a namespace may contain web services that have historically been verified to be compatible to the current or former member web services of the namespace. Our model probabilistically verifies the namespace in the sense of continuous regression testing whenever there is any change in its constitution.

Our model can be extended in multiple dimensions and still have many rooms for improvements. It is interesting to conduct experimentation to evaluate our proposal, and simplify our model so that the model can be more usable in practice. We are aware that we have not incorporated QoS or web services semantics in a significant way. It is still unclear how web services can be reasoned in our model.

References

[1] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *IEEE Transactions on Software Engineering* (*TSE*) 33 (6): 369–384, 2007.

[2] P. A. Bonatti, and P. Festa. On optimal service selection. In *Proceedings of 14th International Conference on World Wide Web (WWW2005),* pages 530–538, 2005.

[3] A. Bucchiarone, H. Melgratti, and F. Severoni. Testing service composition. In *Proceedings of the 8th Argentine Symposium on Software Engineering* (*ASSE 2007*). Mar del Plata, Argentina, 2007.

[4] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M.-C. Shan. Adaptive and dynamic service composition in eFlow. In *Advanced Information Systems Engineering*, volume 1789 of Lecture Notes in Computer Science (LNCS), pages 13–31. Springer, Berlin, Germany, 2000.

[5] G. Castagna, N. Gesbert and, L. Padovani. A theory of contracts for Web services. *ACM Transsactions on Programming Languages and Systems (TOPLAS),* 31 (5): Article 19, 2009. http://doi.acm.org/10.1145/1538917.1538920

[6] W. K. Chan, S. C. Cheung, and K. R. P. H. Leung. A metamorphic testing approach for online testing of service-oriented software applications. *International Journal of Web Services Research*, 4 (2): 60–80, 2007.

[7] W.K. Chan, L. Mei, and Z. Zhang. Modeling and testing of cloud applications. To appear in *Proceedings of 2009 IEEE Asia-Pacific Services Computing Conference (APSCC 200*9), 2009.

[8] J. Gekas and M. Fasli. Automatic Web service composition based on network analysis metrics. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, volume 3761 of LNCS, pages 1571–1587. Springer, Berlin, Germany, 2005.

[9] Global Grid Forum. *Web Services Agreement Specification (WS-Agreement)*. Available at www.ogf.org/documents/GFD.107.pdf (Last accessed Feb 10, 2010)

[10] J.E. Hirsch. An index to quantify an individual's scientific research output. *Proceedings of the National Academy of Sciences of the United States of America* (*PNAS*), 102 (46): 16569–16572, 2005.

[11] S. Lamparter, A. Ankolekar, R. Studer, and S. Grimm. Preference-based selection of highly configurable web services. In *Proceedings of 16th International Conference on World Wide Web (WWW2007)*, pages 1013–1022, 2007.

[12] L. Mei, W.K. Chan, and T.H. Tse. A tale of clouds: paradigm comparisons and some thoughts on research issues. In *Proceedings of 2008 IEEE Asia-Pacific Services Computing Conference (APSCC 2008),* pages 464–469, 2008.

[13] L. Mei, W.K. Chan, and T.H. Tse. An adaptive service selection approach to service composition. In *Proceedings of the 6th IEEE International Conference on Web Services* (*ICWS 2008*), pages 70–77, 2008.

[14] L. Mei, W.K. Chan, and T.H. Tse. Dataflow testing of service choreography. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundation of Software Engineering (ESEC 2009/FSE-17),* pages 151–160, 2009.

[15] L. Mei, Z. Zhang, and W.K. Chan. More tales of clouds: software engineering research issues from the cloud application perspective. In *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC 2009),* vol. 1, pages 525–530, 2009.

[16] S. B. Mokhtar, D. Fournier, N. Georgantas, and V. Issarny. Context-aware service composition in pervasive computing environments. In *Rapid Integration of Software Engineering Techniques*, volume 3943 of LNCS, pages 129–144. Springer, Berlin, Germany, 2006.

[17] O. Moser, F. Rosenberg, and S. Dustdar. Non-intrusive monitoring and service adaptation for WS-BPEL. In *Proceedings of Proceedings of 17th International Conference on World Wide Web (WWW2008)*, pages 815–824, 2008.

[18] N. Oldham, K. Verma, A. Sheth, and F. Hakimpour. Semantic WS-agreement partner selection. In *Proceedings of 15th International Conference on World Wide Web (WWW2006),* pages 697–706, 2006.

[19] W.T. Tsai, Y. Chen, Z. Cao, X. Bai, H. Huang, and R. Paul. Testing Web services using progressive group testing. In *Content Computing*, volume 3309 of Lecture Notes in Computer Science, pages 314–322. Springer, Berlin, Germany, 2004.

[20] Web Services Business Process Execution Language *Version 2.0*. OSAIS. Available at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel/ . (Last accessed 10 Feb 2010)

[21] L. Xiong and L. Liu. Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 16(7):179–194, 2004.

[22] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for Web services composition. *IEEE Transactions on Software Engineering (TSE)*, 30 (5): 311–327, 2004.

[23] J. Zhang, C. K. Chang, J.-Y. Chung, and S. W. Kim. WS-Net: a Petri-net based specification model for Web services. In *Proceedings of 2nd International Cofnerence on Web Servces* (*ICWS 2004)*, pages 420–427, 2004.

[24] L.-J. Zhang, S. Cheng, Y.-M. Chee, A. Allam, and Q. Zhou. Pattern recognition based adaptive categorization technique and solution for services selection. In *Proceedings of 2007 IEEE Asia-Pacific Services Computing Conference (APSCC 2007)*, pages 535–543, 2007.