2012 IEEE Ninth International Conference on Services Computing

A Survey on Automated Service Composition Methods and Related Techniques

Yang Syu
Department of Computer Science and Information Engineering
National Taipei University of Technology
Taipei, Taiwan
a29066049@gmail.com

Shang-Pin Ma
Department of Computer Science and Engineering
National Taiwan Ocean University
Keelung, Taiwan
albert@mail.ntou.edu.tw

Jong-Yih Kuo
Department of Computer Science and Information Engineering
National Taipei University of Technology
Taipei, Taiwan
jykuo@ntut.edu.tw

Yong-Yi FanJiang
Department of Computer Science and Information Engineering
Fu Jen Catholic University
Taipei, Taiwan
yyfanj@csie.fju.edu.tw

***Abstract*—As a promising, low-cost, and agile way to develop software, in recent years automatic service composition has been a popular research topic receiving a lot of attentions. For this topic, upon our long-term study and paper reviewed, we present technical survey and observation in this paper, including indispensable background and preliminary knowledge. The survey assumes under traditional composition context. Moreover, following the survey and observation, we suggest two approach patterns and point out possible future challenge as well as direction, especially to the influence of the mature of mobile devices and environment.**

Keywords- Service-Oriented Architecture; Automatic Service Composition; Semantic Web Service;

I. INTRODUCTION

In recent years, many researchers have concentrated on automatic service composition, as it is a remarkable and promising solution to software engineering. The basic idea is depending on Service-Oriented Architecture (SOA) triangle model [1] and mostly implemented via Web Service (WS) related technologies, composing reusable off-the-shelf services into value-added composite service (CS) to satisfy requester. A exhausted introduction to SOA model, WS, and service composition can be found in [2]. Comparing with manual composition and conventional "programming from scratch" development, automatic composition has many benefits and advantages, such as low-cost, time-saving, risk-reducing, and agility. Those motivate adopting automatic composition to fulfill software requirement.

In traditional composition, an implicit assumption is that services are running on heavyweight enterprise servers and they usually supply computation-intensive functions. That is traditional context of service composition. However, the assumption and context may be broken by modern mobile devices equipped with various sensors and powerful computation-power, such as smart phones.

Automatic composition is not a new topic; it has been studied for several years and already producing hard-to-counted researches. Hence, systematic survey and analysis are doubtless required to construct this topic's architecture and overview for people struggling or interesting in the topic to understand and refer to. The main contribution of this paper is to satisfy the demand. Because this is a large-scoped topic involving in several branches and sub-concerns, researchers in this field can use this paper to position their works in the topic (what research concerns are considered by their efforts). Moreover, through this paper beginners can find their interested subject and understand the topic's outline and basic knowledge. Otherwise, we also discuss possible future challenge and direction, suggesting two approach patterns. When mobile devices popularize, future service composition will not be limited under traditional context anymore, it should be more flexible and complex.

The rest of this paper is organized as follows. Section II describes preliminary knowledge and background for following discussion. Section III is description in detail for each identified research concern. Ultimately, section IV states future challenges and directions of automatic composition as well as our approach patterns.

II. PRELIMIRARY OF TRADITIONAL SERVICE COMPOSITION

This section presents preliminary and background of traditional service composition, preparing for, in next section, our technical survey and observation to automatic service composition. Comparing with automatic composition, manual composition done by human composers is a mature technique and already has been applied universally. To compose services by labor force, traditionally there are two distinct design approaches: top-down and bottom-up [3]. Bottom-up is that, at first, identifying potential partner WSs (they are concrete executable services) and then connecting them with specific process logic. Another one, top-down design, is entirely different. It starts from specifying business process (workflow) consisted of abstract non-executable activities and, subsequently, choosing a fittest concrete service for each activity [3]. As a fatal drawback, manual composition relatively demands for much higher cost. Therefore, currently a large proportion of research efforts are dedicated to automations instead of costly and time-consumed manual composition, trying to thoroughly eliminate human

978-0-7695-4753-4/12 $26.00 © 2012 IEEE
DOI 10.1109/SCC.2012.91

intervention. The research concerns of automatic composition are various technical aspects regarding how to automatically and efficiently generate composite services that exactly meet the expectations of requesters.

Automatic service composition is a large-scoped academic subject, involving in and profiting from many computer science's topics (e.g. semantic web, Artificial Intelligence) and techniques (e.g. AI-planning, ontology) [2]. The subject has several branches such as service discovery, interface and process description, and runtime repairing. In terms of research concern, it is quite difficult to clearly partition these branches into isolated sub-theme as part of them are tangled and interleaved with one another. With top-down composition in mind (e.g. planning abstract workflow first, and then selecting a service for each abstract activity to concretize abstract workflow) [3] [4], some of sub-themes, in terms of research concern, can be seen as horizontally independent. The others cut across them (e.g. service description manner). In this paper, to have a systematic and well-structured presentation, the architecture of survey will follow "separation of concerns" and Aspect-Oriented principles, identifying crosscutting as well as core research concerns for this subject. We refine automatic composition efforts into three core concerns, namely, *service classification*, *combination* (*planning* [5]), *selection*. With these core concerns, concerns crosscutting them are identified (*service description* and *matchmaking*) and will be discussed alone.

Even with a fully-automated composition approach, before composition begins, it still needs least human intervention, e.g. specifying the formulation of composition request [2]. According to our survey, thus far most intuitive and common way to define composition request follows "query by example" approach [3]. More specifically, it means, upon the specification of service description used, accurately assigning a service specification (or interface) that a service with same specification is desired. For instance, in [5], each service is described through its Input as well as Output and that is identical with the formulation of composition goal (request). Our previous work [4] complies with the approach, defining "*provided service*" and "*required service*", they identically have same 7-tuple.

Each time the outcome of composition is composite service (CS), which is an ordered service set with specific process structures and logic. In other words, a CS consists of element (component) services as well as a workflow of them. To automatic composition, workflow consisting of data and control flows is a research emphasis, the definition of workflow can be found in section 3 of [6] [2].

Generally the architecture of CS is analogous with "pipe and filter" architecture pattern introduced in chapter 6 of [7]. Comparatively, each CS's element service acts like filter providing data transformation, and the data (output and input) of element services flow between them through, for example, the exchanges of SOAP messages (pipe). Most enterprise SOA-based systems also conform to this architecture pattern. Moreover, workflow (sometimes called "business process" or "orchestration" in different contexts, but conceptually they are interchangeable), which unambiguously indicates the invocation order of element services, normally offers various types of process structure for process designers, such as sequence, loop, and parallel. The types provided depend on process language used. In industry and academia, currently a widely adopted standard is WS-BPEL, it totally has more than seven structure types (structured activities) [8]. The readers can refer to section 2.1 of [9] for a quick understand to it. Besides WS-BPEL, there still are many other languages or standards to define workflow, such as Petri net [2] [10], YAWL [11], UML [12], and Mashups [13] [14].

Nevertheless, in academia, researchers only interest in conceptual logic of execution path rather than generating a real implementation of executable workflow (e.g. a bpel file that can be executed on a BPEL engine). Thus, many works like [15] [16] consider and utilize only the semantic meaning of top-common structure types (or part of them). They are, presented by BPEL's term, *sequence*, *flow*, *while* and *if*, which have been imported in our previous work [4] and are introduced by most process languages using disparate symbols. For example, in [16], *XOR-split* as well as *XOR-join* pair is, in semantic meaning, *if*, and *AND-split* as well as *AND-join* pair equates *flow*. For more information, an introduction to industrial and formal workflow languages is presented in section 3 of [2]. How to create workflows without any human influence and effort is concern of *service combination* (*planning*) that will be exhaustedly discussed later.

In practical implementation, as mentioned earlier, most common manner to realize SOA and service composition is WS technology [2]. Nevertheless, like workflow modeling language, in academia people principally concern with conceptual services. In academia, how services are being implemented in reality (e.g. SOAP WS) is not so concerned. One thing that they care about is how services are described in theoretical level, and that will be discussed thoroughly in *service description* later. To conceptual services, a difference in the assumptions of each works is cardinality (in UML, multiplicity) of service-to-operation. In research works like [12] (especially in those that they concern with *service combination* [5] [17] and *selection* [5] [18] [19] [11] [20] [21]), they usually assume that, for simplicity and concentration, the cardinality of service-to-operation is one-to-one, as explained in introduction of [22] as well as section 4.1 of [23]. To works concerning with *service combination* and *selection*, the cardinality is a trivial assumption because, without any substantial influence, they can arbitrarily set service or operation as finest function unit or easily viewing each operation as a service (e.g. as defined in section 2 of [24], its workflow is a collection of operations rather than services). However, this assumption largely affects the processing of works belonging to *service classification*. We will further explain that in *service classification*.

From viewpoint of CS, its life cycle goes through design, execution, monitoring, and reengineering [22], or more general, design-time and runtime stages. According to our observation and paper reviewed, most automation efforts implicitly have an assumption that they aim at CS's

design-time stage. During CS's runtime, a necessity is that the processing must be "react in real-time". That is quite difficult to achieve by most automations since they target on really designing and composing CS from scratch, such as concretizing entire abstract workflow or generating CS (workflow) by connecting services one by one. These are hard to be real-time as some tasks, e.g. semantic reasoning on domain ontology, are time-consumed.

III. Concerns of Traditional Service Composition

As remarked in previous section, we separated and categorized whole automatic composition into three core research concerns (*service classification*, *combination*, and *selection*) as well as two crosscutting research concerns (*service description* and *matchmaking*). In this section, we present crosscutting concerns first and then core concerns. Crosscutting concerns cut across all core concerns, widely affecting and involving in them. For instance, how services are described and expressed (the concern of *service description*) extensively influences the rationales of how to match, classify, combine, and select services (another four concerns we identified). Thus, it is better to discuss crosscutting concerns prior to core concerns.

Before starting the discussion for each concern, a crucial component in academic research of automatic composition, i.e. ontology, should be briefly introduced as it largely benefits this subject. It is a backbone of the Semantic Web—the second generation of the web—trying to share and reuse data (knowledge) across diverse boundaries. The term *ontology* is originally from philosophy, and in computer and information sciences it is used to model and represent knowledge within specific domain. By capturing a domain's concepts and the relationships between them, it can for computer doing logical semantic reasoning. The readers can find more about it in Introduction of [25], along with exhausted introduction regarding its usage in software engineering. Considering theoretical researches of automatic composition, we have recognized that, in this field, ontology has two principal purposes. One is to enhance services with syntactic interoperability as semantic services having machine-processable interface, which facilitates automation. Another purpose is for generating workflow, upon domain knowledge modeled within it. We will explain them in *service description* and *combination*, respectively.

A. Service Description

As above explained, this concern widely twists with the others. Under SOA, services' reusing is black-box, entirely different from component-based reusing that designers can alter and extend components through modifying their source code. The only way to understand services is via their descriptions of external specifications. That is the reason for why it vastly cuts across other four concerns and why we present it with top priority.

For service description, currently most widely-accepted and de-facto standard is WSDL, but just like drawbacks remarked in section 2.2 of [1] and section 4 of [3], WSDL is barely enough to build syntactical interfaces for connecting and invoking services. WSDL lacks formal semantic [2] [26] and metadata [1] of service, thus semantic community has provided service specification ontologies, such as OWL-S [27], WSMO, and SAWSDL [28] [3], for accurately and semantically describing service. Together with appropriate domain ontology defining the terms within applied domain, they enable and facilitate automatic manipulation of services.

Both WSDL as well as semantic service ontologies have complex specifications and implementation detail but, in academia, that are not the primary concern of researchers. Hence, for simplicity researchers usually represent services in terms of tuple, neglecting trivial specifics. For example, in abstract level WSDL simply describes a service (operation) by Input and Output (IO) tuple, namely the SOAP messages intended and produced by the service. With semantic interface, the tuple could be richer including Input, Output, Pre-condition, Effect (IOPE), Capability (C), and Non-functional properties (NF). IOPEC are functional tuple that they together express the functionality of a service. Non-functional properties are service's attributes like QoS. Their detailed definition can refer to [2] [4]. In addition, as mentioned by [29] and its related work, a recent trend, QoS-aware WS, is widely followed. Hence, semantic specification for services' QoS attributes is necessary and has been proposed. In [29] it defined a comprehensive ontology scheme for semantically describing classes, QoS, and domain knowledge of services. Ontology defining terms about services' non-functional features will be more important when we consider more and more criteria in *service selection*.

In these tuple, especially in functional tuple, an implicit assumption of most works is that a tuple, e.g. Output of a service, usually comprises only several coordinative simple elements. Each element refers to or is an instance of a concept (class) defined on applied domain ontology. However, in reality, this assumption does not work. In WSDL each IO element could be a complex SOAP message element having deep and nested structure definition. A corresponding semantic version of WSDL is SAWSDL [3], it is identical with WSDL in structure and component, with additional semantic annotation for each WSDL component. Although semantic services largely benefit automatic composition, it still has several serious problems, as remarked in [17], causing its unreality. One of the problems is that the semantic reasoning on ontology is quite time-consumed task. Thus, to each tuple, most works did not assume containing complex and plethoric elements (for instance, in [5] simply the Input element of S_8 only has "PID" and the Output is "Review"). That is a gap between academia and industry. Efforts like [17] [23] proposed using pre-processing to services as well as specific data structure to store services' information, in order to accelerate semantic-related tasks and enhance the quality of composition.

It is not true that every research works identically utilize all of the tuple abovementioned in their service description. Which tuple should be included in description is decided by what are a work's core concerns. Automations do not need service information that is not concerned by them. For example, in [11] the authors came up with

algorithm to select services for concretizing abstract workflow into workable CS with intended composite transactional and QoS properties. In [18] it focuses on the uncertainty of available time of composed CS and each service is expressed by its name and mobile prediction (time-availability) information. They both concern with non-functional aspect (*service selection*), therefore the functional tuple did not be utilized in this two cases. Our previous work [4] covers 7-tuple because we need functional and non-functional information of services.

B. Service Matchmaking

As before mentioned, in academia, since tuple is ideal way to understand and express services, *service matchmaking* concerns with functional similarity or compatibility calculation between tuple, rather than other service's information. We have identified that *service matchmaking* roughly has two types, but conceptually they are, in essence, equivalent. They are matchmaking between services as well as between service junctions.

More specifically, first type could be calculation between service and service, between service and abstract activity, or between service and composition request. Essentially they are all same. Assuming that, in certain context, IOPE tuple are adequate and enough to unambiguously describe a service. An abstract activity represents a place that needs a service to stuff and satisfy it, thus an activity usually is also described by tuple used for representing service (in this case, IOPE). As to composition request, we already have explained that most efforts conform to "query by example", so a request usually is presented by tuple for service (IOPE) as well. Therefore, no matter what are compared (service, activity, or request), all of them can be seen as identical specification, having same tuple. The matchmaking is corresponding, e.g. a service's Input is calculated with another service's Input. Corresponding matchmaking is also a critical concern to other service-related research topics such as *service discovery* introduced in section 5.5 of [2] and *service retrieve* [17], as they must make sure whether services found meet expectation and requirement.

Second matchmaking type is between service's junctions, namely the calculation between a service's Output (Effect) and its successor service's Input (Pre-condition). The purpose of it is to clarify whether two services are eligible to be connected together. For sake of simplicity, here we assume that dataflow among CS is stateless instead of stateful assumed in previous work [4], and the discussion will concentrate on O/I junctions matchmaking. Basically, to figure out whether two services are connectable with each other, a question must be clarified is that, whether a service's Output can be link with another service's Input. It is to calculate if the elements contained in Output are exactly identical or compatible with elements included in Input. Thus, actually this matchmaking is same with corresponding matchmaking, which calculates the similarity or compatibility of two element groups contained in two identical tuple (e.g. two services' Input tuple). In addition, junction matchmaking is also a crucial basis for *service combination* that we will discuss later.

So far, we have explained that corresponding matchmaking and junction matchmaking are equal in essence. The core of them is comparison of two elements comprised in two distinct service tuple. Below we will focus on similarity and compatibility of two elements. In *service description*, we already discussed that current research trend is semantic service, where a service is expressed by service ontology or conceptual tuple. However, that still is not enough. To be fully semantic, each term appeared in service ontology or conceptual tuple should be specified on an ontology modeling interested domain [21]. How to precisely define these terms is another question because the design of ontology and the annotation of terms are pretty time consuming [17] [24].

Most primitive manner like previous work [4] and [30] uses keywords to define terms (both focus on junctions matchmaking). The calculation is poorly letter-by-letter string comparison, which is far away from enough. Explanation to why keyword is insufficient as well as its drawbacks is in Introduction of [31]. There are several efforts trying to enhance this primitive comparison. First, in [31], it proposed clustering algorithm to cluster keywords, they are associated in semantic meaning, into same concept. However, it still is not as precise as domain ontology modeled by knowledge expert since the relationships between concepts (e.g. sub-concept) cannot be recognized by this algorithm. Next, in first part of [3], it exploits a general lexical database WordNet (a general English word ontology) to reason semantic relationship and distance (similarity) between two non-semantic keywords (terms). [28] is similar with [3], but it further considers the relations between terms contained in same theme and adopts search engine instead of WordNet. It claims that, in this way, it is better than [3] and [31]. Another work exploiting WordNet is [17], it uses WordNet to specify semantic definitions for semantic-less terms. Otherwise, it also proposed formulations for semantic calculation across heterogeneous ontologies. Similarly, in [24], it proposed method to automatically generate semantic annotations for operations' parameters. Nevertheless, despite it is expensive and costly, defining by human experts or designers still is a better and more accurate way to annotate terms.

With semantic-annotated terms, the calculation of similarity and compatibility is more accurate and complex. There are several relationships between semantic-annotated terms, such as exact match, plugin, subsume, intersection, and disjoint [21]. The place that terms exist absolutely affects the result. For example, in junction matchmaking, an Input including "Food" is compatible with Output containing "Salad" or any other sub-concept of "Food". Thus, this is a legal connection called causal link (or semantic link) [21] [23] or robust link [5]. The relationship is not symmetric, for example, an Input including "Salad" is incompatible with Output containing "Food". As we discussed in *service description*, most of approaches assume that services' IO structures are very simple. Semantic

variant of [3] addresses corresponding matchmaking considering realistically complex and nested IO structures.

C. Service Classification

Although it does not directly attend composition, *Service classification* still is an important concern to the subject. Basically, it can be seen as a helper for *service combination* and *selection*. To the former, in *bottom-up service combination* (we will explain it in next subsection) it can help to improve the performance and efficiency of composer by decreasing the number of services. For example, in [5], it clusters and treats services having identical functionality as a single unit, and then the composition is to these units rather than services. To *selection*, for each abstract activity comprised in a workflow, classification can aggregate services having needed function but posing different non-functional features, and then selector can choose most appropriate service from them. Otherwise, for runtime environments and frameworks, it can prepare services for replacement [5] or substitution [3].

Here we identify two classification types: coarse-grained and fine-grained. The unit of coarse-grained classification is service and, to fine-grained, the unit is operation. For automatic composition, as we already discussed in section II, operation is most suitable function unit. In real world, each service usually offers plural operations they are, in terms of functionality, entirely independent and non-associated. For example, a service exposed by a bank may provide query of currency exchange rate, financial deal functions, and in-house operations internally used by bank's employees. To assist automatic composition, fine-grained classification is fittest. Another difference is that the class labels are given in advance or not. In [29], user must pre-define class labels by self. Otherwise, some efforts adopt standard common taxonomy system such as [32] using UNSPSC. Yet, to automatic composition, the labels defined in taxonomy systems or by users are too coarse. To aid composition automations, in our opinion, a perfect classification is like functional clustering in [5]. Without pre-defined labels, it uses functional IO parameters to semantically cluster functionality-identical operations together (it is fine-grained, viewing an operation as a service).

Most primitive classifications like [29] use only service name to classify. In example of [29], the user defines ten classes in advance and services are classified to proper class according to its name. In this way, the classification is quite inexact and coarse. Advanced approaches, like [32], rely on services' text information (in WSDL and UDDI) as well as parameter terms. Many of them are variants of term-frequency based manner.

D. Service Combination

In our opinion, toward automatic composition, most important concern is *service combination* because it works out skeleton of CS (workflow) to meet functional requirement of requester. In efforts concerning with *service combination*, despite each effort's solution and algorithm detail is disparate, but roughly there are two different mechanisms. First mechanism is that a composer somehow creates abstract non-executable workflow template, and then forwarding the template to approaches concerning with *service selection*. Or, second mechanism, a composer directly synthesizes available and known component services as an executable CS implying a workflow. Comparatively, first mechanism is akin to top-down manual composition earlier mentioned and second mechanism is similar with bottom-up.

In top-down process, designer receives composition request and then analyzes it relying on knowledge to applied domain and workflow design (sometimes, like requirement engineering, designer must work with domain expert as he does not have sufficient domain knowledge). Subsequently, based on domain know-how and professional sense, the designer works out a procedure to fulfill and match the request. The procedure is a collection of tasks in which every task must be done by something. The procedure in designer's mind can be written via workflow definition language discussed in section II. In this way, as you can see, evidently the key in top-down design is domain knowledge. Therefore, to automate that, there must be a knowledge representation model, i.e. ontology, to capture domain knowledge from domain expert. The works belonging to first mechanism like [26] [2] ([2] is a special case, we will discuss it in detail later) leverage knowledge contained in domain ontology to generate an abstract workflow template. The biggest obstacle of approaches following this mechanism is the lack of a worth-to-trust domain ontology modeling sufficient domain knowledge. For the obstacle, [26] proposed borrowing well-defined ER-model to construct domain ontology to overcome the obstacle.

In bottom-up process, designer definitely knows which services are available as well as the detailed specification of them. Those are stored in a service repository. When receiving a request, the designer struggles to filter, pick, and link appropriate services into a feasible service chain (CS) written by, for example, WS-BPEL. The linkages between component services depend on dataflow, connecting and matching by "junction matchmaking" introduced in *service matchmaking*. By this mechanism, workflow is implied in composed CS and it is called data-driven workflow because it is constructed by dataflow connections, a formal definition of data-driven workflow is in section 2 of [24]. According to our survey, most efforts concerning with *service combination* work with bottom-up mechanism as, in our opinion, it is more feasible than top-down mechanism. The construction of a domain ontology having comprehensive and sufficient coverage in applied domain is a great challenge, involving many stakeholders and exertions [17] [26] that hinders top-down mechanism. To automate handcrafted bottom-up process, many efforts following this mechanism exploit AI planning technique (a introduction to works employing AI planning can refer to section 3 of [2]), using forward or backward chaining [17] [23]. Complying with bottom-up mechanism, our previous work [4] and [30] leverage genetic algorithm to evolve CS. Contrast to top-down manner, a virtue of bottom-up mechanism is that the outcome of composition is a real workable CS.

An special case, [2], proposed a theoretical goal-driven approach adopting top-down mechanism with the assistance of bottom-up manner. Unlike efforts using "query by example", in this work composition demand is well-formulated goal defined using formal goal description language. In addition, another works [33] [34] are able to present and process composition request in forms of nature language. The services they illustrated are real world services (e.g. microphone) rather than only software services. In [2] abstract workflow is generated by goal decomposition (top-down). The decomposition can be based on dedicated goal ontology or domain ontology if they comprehensively and sufficiently describe respective domain. However, if they are not enough to adequately decompose a goal, the composer can discover real services, acquiring their service ontologies (semantic specifications) to aid the decomposition (bottom-up).

Besides data-driven workflow, there is another type of workflow that it is not constructed by dataflow links between services. An example is workflow introduced in [35], it is built with services handling MPEG video stream and each service is described by its capability and state instead of functional or non-functional tuple. The composition is quite different since some services can be arranged at any place among workflow and the others have to be placed before or after certain service.

E. Service Selection

Giving a workflow template consisted of abstract activities, *service selection* concerns with how to efficiently select an appropriate service for each activity (going through *service classification*, each activity may have several function-identical candidate services that they have different non-functional characteristics), in order to concretize an executable CS satisfying intended and restricted non-functional criteria. In other words, the main concern of *service selection* is diverse non-functional aspects. In addition, it is second phrase of top-down design introduced in *service combination*, after first phrase creates proper workflow.

Service selection could occur during CS's design time, runtime, or both. There are several possible scenarios. The scenario maybe that selecting proper services at design-time, and then rebinding (dynamic binding) inadequate or failed services during runtime. Or, it is completely dynamic binding at runtime (later binding), without any selection at design-time. Dynamic binding is one of the most important advantages promised by original SOA model, but practically the model does not succeed [1]. The discussion here will purely concentrate on design-time selection because it is regarding selection for every activity contained in workflow, instead of only re-selecting for one or two activities. Normally runtime selection (dynamic binding) merely repairs and selects for a workflow's segment as it must react in real-time. Thus far, according to our study there are several non-functional aspects (criteria) including Quality of Service (QoS) [19], transactional property [20], reliability [15], mobility prediction [18], and machine loading (resource consumption) as well as distance between services [35]. Below we briefly discuss them.

In automatic composition, QoS-aware selection has been a critical concern for years. A lot of works are purely dedicated to it, such as [19] [16] [36] [37] [38]. Its goal is that, upon a workflow template, rapidly choosing services to construct a CS posing acceptable QoS attributes. The QoS features held by composed CS or asked by requester can be a single representative value, which is calculated via certain QoS aggregation function. Generally, there are two different policies to QoS-aware selection, local and global optimization. The global optimization is preferred and adopted by most efforts because it is more accurate than local optimization. Common approach types can be roughly divided as three: linear programming, genetic algorithm, and heuristic algorithm. This order also presents the performance comparison of them, from best to worse. Yet there still are many other approach types. An exhausted introduction to QoS-aware selection can refer to [39].

Although transactional property (transaction-aware) selection is not as widely followed as QoS-aware selection, it is, in fact, a quite important concern especially to traditional composition that it composes CS for enterprise business systems. In these business systems, assuring transactional operations is necessary and inevitable (e.g. ACID check). Since a CS is a distributed system in essence, the topic is similar with two-phrase distributed transaction trying to ensure the consistency of distributed units. For example, if an ending component of CS is failed, the services they have been successfully executed must be able to recover the works that are already done, or the failed component must be invoked iteratively until it is success. [11] is an special case that it concerns with both local QoS optimization and transaction property.

Reliability-aware selection like [15] struggles to deal with a multi-objective problem. In this type of problem, each objective conflicts with one another. For instance, using *active parallel* strategy to an activity will receive relatively short response time, but it damages another objective: lowering price. The reason is that this strategy simultaneously runs plural services for an activity. In contrast, using *standby parallel* strategy will have lower price but relatively longer response time as it uses only one service at a time. The challenge is that, for each activity, selecting proper strategy and services in order to meet user request and constraint.

A newly emerging non-functional aspect is mobility prediction mentioned in [18]. With the popularization and advance of smart mobile devices, providers of services contained in a composition could be mobile devices. They move constantly and cause composition crash when they are out of communication range. The goal of this selection is to make an "as dependable as possible" composition, relying on devices' mobility prediction information. In addition, for simplicity, selection efforts like [15] [18] often assume that only sequential workflows are considered.

So far, to service systems, most researches have an implicit assumption that the communications between services are just small message packages (e.g. SOAP

messages). But in some application types (e.g. multimedia applications), the communications could be large data streams consuming machine's CPU resource enormously. Long distance and overloading machine probably result in negative effect to the application, such as video delay. Thus, works like [35] consider distance between services (can be measured by number of routers passed) and machine loading (on a machine, how many services are running and how many data streams are being processed).

Notices that the non-functional aspects discussed are not complete; there still are many other aspects, determined by requesters' concerns and application characteristics.

According to our observation, virtually every research work covers more than one concern because there are crosscutting concerns involving in all of them. For example, [17] concerns with not only *service combination* but also *service description* and *matchmaking*. Another example, [11] mainly focus on *service selection*, but it still has certain manner to *describe* services' non-functional properties (in this case, QoS and transactional behavior) despite the manner is not the emphasis of this work. Other than crosscutting concerns, there are works trying to cover more than one core concern. For example, our previous work [4] simultaneously concerns with *service combination* and *selection*. In [5], it covers all of three core concerns. Nevertheless, upon our observation, most efforts only highlight one core concern. A large part of efforts belonging to this type is QoS-aware *service selection* such as [19] [16] [36] [37] [38] [40]. Otherwise, [18] purely concentrates on *service selection*.

IV. FUTURE CHALLENGE AND DIRECTION

In this section, we discuss future challenges and possible directions of automatic service composition, suggesting approach pattern for top-down and bottom-up composition design, respectively.

At present, in our opinion, the biggest challenge of traditional composition is propose a comprehensively fully-automated methodology. In section 3 of [2], it states that "*In a fully-automated approach the role of a human is limited to formulation of a composition request which contains the specification of initial state, goal state and possibly optimization criteria*". As remarked in our previous work [4], mostly efforts concerning with *service selection* like [18] [11] [16] [15] assumed that the workflows are given in advance. In other words, they cannot generate workflows. The analysis of shortcomings of them can refer to section 2.3.3 of [34]. On the other hand, efforts focusing on *service combination* did not consider non-functional aspects (*optimization criteria*), such as [17] [30]. Thus, as the quoted statement described, to become fully-automated inevitably an approach comprehensively covering *combination* and *selection* is required. In [4] we have a genetic algorithm (GA) based approach covering workflow generation, QoS, and transactional properties, but it still is not good enough. First, it needs time to evolve (this is inevitably congenital defect of GA) and, as we introduced in *service selection*, there still are many other possible non-functional properties. We think that a fully-automated methodology should be as flexible and automatic as possible, providing workflow generation as well as all possible combinations of non-functional properties. Requesters must be able to choose freely concerned properties.

Here we suggest two approach patterns: For top-down design, after that abstract workflow had been created and classification for each activity was done by composer, it can use proper selection approach to concretize the workflow as a CS having desired non-functional features. For bottom-up design, suggested pattern is akin to [5]. First, it must cluster services with same function into same group unit and then properly chaining these units as a concrete workflow. Concrete workflow is distinct from abstract workflow in that every node comprised in concrete workflow already has several candidate services. Both of suggestions are three-stage pattern (generation of abstract workflow, classification, and selection; clustering, chaining, and selection). Unsolved issue is at selection stage. When selecting services, there are "precedence rule" in non-functional properties. In [11] [4], the rule that the priority of transactional property is higher than QoS was proved. Nevertheless, there still are many other non-functional properties. Therefore, clarifying non-functional properties and precedence rules between these properties are critical research issue to suggested patterns and fully-automated composition approaches.

In the future, with advancement of technologies and newly emerging concepts, we predict that service composition will be quite different. Traditionally, services are mostly running on enterprise servers. But along with modern smart devices' progress, now they can become service providers as well. They can provide certain services that cannot be supplied by traditional providers. Services provided by mobile providers will be quite different from conventional computation-intensive services. They could be moving location-based or context-aware services sensing and providing, through their sensors, immediate real world information. Furthermore, they also can act as intermediaries for people to enable them becoming movable "human provided services" [40]. Forced by these new trends, defining a new service model to substitute obsolete triangle SOA model [1] is required. Next, to composition, non-functional aspects uniquely belonging to mobile devices or applications, such as mobility prediction [18], may be real concern of future mobile service composition, instead of QoS or transaction property that are interested by traditional composition works. Identifying and analyzing unknown mobile non-functional aspects is significant because, to modern software users, non-functional requirement usually is more important and prior than functional requirement [7]. In the future, a CS probably may blend traditional as well as mobile services, and the architecture of CS may not always be typical "pipe and filter". Imaginably, how to define process and aggregate non-functional features will be baffling problem.

ACKNOWLEDGMENT

This research is partially sponsored by National Science Council (Taiwan) under the grant NSC 100-2221-E-030-017 and NSC 100-2221-E-019-037.

REFERENCES

[1] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "End-to-End Support for QoS-Aware Service Selection, Binding, and Mediation in VRESCo," *Services Computing, IEEE Transactions on,* vol. 3, pp. 193-205, 2010.

[2] D. Zhovtobryukh, "A Petri Net-based Approach for Automated Goal-Driven Web Service Composition," *Simulation,* vol. 83, pp. 33-63, 2007.

[3] P. Plebani and B. Pernici, "URBE: Web Service Retrieval Based on Similarity Evaluation," *Knowledge and Data Engineering, IEEE Transactions on,* vol. 21, pp. 1629-1642, 2009.

[4] S. Yang, Y.-Y. FanJiang, J.-Y. Kuo, and S.-P. Ma, "Towards a Genetic Algorithm Approach to Automating Workflow Composition for Web Services with Transactional and QoS-Awareness," in *2011 IEEE World Congress on Services*, Washington, DC USA 2011, pp. 295-302.

[5] F. Wagner, F. Ishikawa, and S. Honiden, "QoS-Aware Automatic Service Composition by Applying Functional Clustering," in *Web Services (ICWS), 2011 IEEE International Conference on*, 2011, pp. 89-96.

[6] M. B. Blake and D. J. Cummings, "Workflow Composition of Service Level Agreements," in *Services Computing, 2007. SCC 2007. IEEE International Conference on*, 2007, pp. 138-145.

[7] I. Sommerville, *Software Engineering, 9/E*: Addison-Wesley, 2010.

[8] A. Alves, "OASIS Web Services Business Process Execution Language (WSBPEL) v2.0," *OASIS Standard* 11 April 2007 2007.

[9] L. An, L. Qing, H. Liusheng, and X. Mingjun, "FACTS: A Framework for Fault-Tolerant Composition of Transactional Web Services," *Services Computing, IEEE Transactions on,* vol. 3, pp. 46-59, 2010.

[10] A. Portilla, G. Vargas-Solar, C. Collet, J.-L. Zechinelli-Martini, and L. García-Bañuelos, "Contract Based Behavior Model for Services Coordination," in *Web Information Systems and Technologies*. vol. 8, J. Filipe and J. Cordeiro, Eds., ed: Springer Berlin Heidelberg, 2008, pp. 109-123.

[11] J. El Hadad, M. Manouvrier, and M. Rukoz, "TQoS: Transactional and QoS-Aware Selection Algorithm for Automatic Web Service Composition," *IEEE Transactions on Services Computing,* vol. 3, pp. 73-85, Jan.-March 2010

[12] M. B. Blake, "Decomposing Composition: Service-Oriented Software Engineers," *Software, IEEE,* vol. 24, pp. 68-77, 2007.

[13] Sébastien Mosser, Franck Chauvel, Mireille Blay-Fornarino, and Michel Riveill, "Web Services Composition: Mashups Driven Orchestration Definition," presented at the *2008 International Conferences on Computational Intelligence for Modelling, Control and Automation*; *Intelligent Agents, Web Technologies and Internet Commerce; and Innovation in Software Engineering*, , 2008.

[14] W. Qian and R. Deters, "SOA's Last Mile-Connecting Smartphones to the Service Cloud," in *Cloud Computing, 2009. CLOUD '09. IEEE International Conference on*, 2009, pp. 80-87.

[15] W. Chien-Min, W. Shun-Te, C. Hsi-Min, and H. Chi-Chang, "A Reliability-Aware Approach for Web Services Execution Planning," in *Services, 2007 IEEE Congress on*, 2007, pp. 278-283.

[16] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints," *ACM Trans. Web,* vol. 1, p. 6, 2007.

[17] R. Kaijun, X. Nong, and C. Jinjun, "Building Quick Service Query List Using WordNet and Multiple Heterogeneous Ontologies toward More Realistic Service Composition," *Services Computing, IEEE Transactions on,* vol. 4, pp. 216-229, 2011.

[18] W. Jianping, "Exploiting Mobility Prediction for Dependable Service Composition in Wireless Mobile Ad Hoc Networks," *Services Computing, IEEE Transactions on,* vol. 4, pp. 44-55, 2011.

[19] R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz, "Heuristics for QoS-aware Web Service Composition," presented at the *Proceedings of the IEEE International Conference on Web Services*, 2006.

[20] L. Li, L. Chengfei, and W. Junhu, "Deriving Transactional Properties of CompositeWeb Services," in *Web Services, 2007. ICWS 2007. IEEE International Conference on*, 2007, pp. 631-638.

[21] F. Lecue and N. Mehandjiev, "Towards Scalability of Quality Driven Semantic Web Service Composition," presented at the *Proceedings of the 2009 IEEE International Conference on Web Services,* 2009.

[22] W. Gaaloul, S. Bhiri, and M. Rouached, "Event-Based Design and Runtime Verification of Composite Service Transactional Behavior," *Services Computing, IEEE Transactions on,* vol. 3, pp. 32-45, 2010.

[23] F. Lecue and A. Leger, "Semantic Web Service Composition through a Matchmaking of Domain," presented at the *Proceedings of the European Conference on Web Services*, 2006.

[24] K. Belhajjame, S. M. Embury, N. W. Paton, R. Stevens, and C. A. Goble, "Automatic Annotation of Web Services Based on Workflow Definitions," *ACM Trans. Web,* vol. 2, pp. 1-34, 2008.

[25] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani, "An Approach for QoS-aware Service Composition Based on Genetic Algorithms," presented at the *Proceedings of the 2005 conference on Genetic and evolutionary computation*, Washington DC, USA, 2005.

[26] D. A. Menascé, E. Casalicchio, and V. Dubey, "A Heuristic Approach to Optimal Service Selection in Service Oriented Architectures," presented at the *Proceedings of the 7th International Workshop on Software and Performance*, Princeton, NJ, USA, 2008.

[27] D. A. Menascé, E. Casalicchio, and V. Dubey, "On Optimal Service Selection in Service Oriented Architectures," *Perform. Eval.,* vol. 67, pp. 659-675, 2010.

[28] R. G. Daniel Schall, C. Dorn, and S. Dustdar, "Human Interactions in Dynamic Environments through Mobile Web Services," presented at the *IEEE International Conference on Web Services (ICWS 2007)*, , 2007.

[29] Z. Yajing, D. Jing, and P. Tu, "Ontology Classification for Semantic-Web-Based Software Engineering," *Services Computing, IEEE Transactions on,* vol. 2, pp. 303-317, 2009.

[30] X. ChengZhi, L. Peng, W. Taehyung, W. Qi, and P. C. Y. Sheu, "Semantic Web Services Annotation and Composition Based on ER Model," in *Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC), 2010 IEEE International Conference on*, 2010, pp. 413-420.

[31] M.B.D. Martin, J. Hobbs, O. Lassila, D. McDermott, and S.N. McIlraith, M. Paolucci, B. Parsia, T. Payne, et al., *OWL-S: Semantic Markup for Web Services*: W3C Recommendation, 2004.

[32] L. Fangfang, S. Yuliang, Y. Jie, W. Tianhong, and W. Jingzhe, "Measuring Similarity of Web Services Based on WSDL," in *Web Services (ICWS), 2010 IEEE International Conference on*, 2010, pp. 155-162.

[33] A. Moraru, C. Fortuna, B. Fortuna, and R. R. Slavescu, "A Hybrid Approach to QoS-aware Web Service Classification and Recommendation," in *Intelligent Computer Communication and Processing, 2009. ICCP 2009. IEEE 5th International Conference on*, 2009, pp. 343-346.

[34] L. Aversano, M. D. Penta, and K. Taneja, "A Genetic Programming Approach to Support the Design of Service Compositions," *International Journal of Computer Systems Science & Engineering,* vol. 21, pp. 247--254, 2006.

[35] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang, "Similarity Search for Web Services," presented at the *Proceedings of the Thirtieth International Conference on Very Large Data Bases* - Volume 30, Toronto, Canada, 2004.

[36] L. Qianhui, L. Peipei, P. C. K. Hung, and W. Xindong, "Clustering Web Services for Automatic Categorization," in *Services Computing, 2009. SCC '09. IEEE International Conference on*, 2009, pp. 380-387.

[37] K. Fujii and T. Suda, "Semantics-based Context-aware Dynamic Service Composition," *ACM Trans. Auton. Adapt. Syst.,* vol. 4, pp. 1-31, 2009.

[38] K. Fujii and T. Suda, "Semantics-based Dynamic Web Service Composition," *International Journal of Cooperative Information Systems,* pp. 293 - 324, 2006.

[39] J. Jin and K. Nahrstedt, "Resource- and Quality-aware Application-level Service Multicast," in *Distributed Computing Systems, 2003. FTDCS 2003. Proceedings. The Ninth IEEE Workshop on Future Trends of*, 2003, pp. 198-204.

[40] Q. Li, A. Liu, H. Liu, B. Lin, L. Huang, and N. Gu, "Web Services Provision: Solutions, Challenges and Opportunities (invited paper)," presented at the Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication, Suwon, Korea, 2009.