Towards Effort Estimation for Web Service Compositions using Classification Matrix

Zheng Li
NICTA and UNSW
School of CSE
Sydney, Australia
Zheng.Li@nicta.com.au

Liam O'Brien
CSIRO and ANU
School of CS
Canberra, Australia
Liam.OBrien@csiro.au

Abstract — **Within the service-oriented computing domain, Web service composition is an effective realization to satisfy the rapidly changing requirements of business. Although the research into Web service composition has unfolded broadly, little work has been published towards composition effort estimation. Since examining all of the related work in this area becomes a mission next to impossible, the classification of composition approaches can be used to facilitate multiple research tasks. However, the current attempts to classify Web service composition are not suitable for the research into effort estimation. For example, the contexts and technologies of composition approaches are confused in the existing classifications. This paper firstly proposes an effort-oriented classification matrix for Web service composition, which distinguishes between the context and technology dimension. The context dimension is aimed at analyzing the environmental influence on the effort of Web service composition, while the technology dimension focuses on the technical influence on the effort. Therefore, different context types and technology categories can be treated as different effort factors. Based on the classification matrix, this paper also builds an effort-estimation-checklist table by applying a set of qualitative effort estimation hypotheses to those effort factors. The table can then be used to facilitate comparing the qualitatively estimated effort between different composition approaches.**

Keywords - service-oriented architecture (SOA); classification matrix; Web service composition; effort hypotheses; effort estimation

I. INTRODUCTION

Web services have been widely accepted as the preferred standards-based way to implement Service-Oriented Architecture (SOA) in practice. Since "only when we reach the level of service composition can we realize all the benefits of SOA" [16], the research into composing Web services has grown significantly along with the increasing necessity of reusing existing resources. Over the past decade, numerous works for composing Web services have been developed and reported in the literature. However, little work can be found towards the cost and effort estimation for Web service compositions. Meanwhile, it is difficult to investigate different composition effort by exhausting all the published composition approaches. However, we can inductively classify the existing Web service composition works, and thereby to facilitate the comprehension of related knowledge and the effort estimation work.

Existing classification work of Web service composition can be found in several survey papers [17, 19]. These classifications are either incomplete or ambiguous, which brings many issues when using them to categorize and analyze new composition approaches. Firstly, none of the existing classifications distinguishes between the composition technologies and the composition contexts. For example, Dustdar and Schreiner [17] list model-driven approaches as a separate composition category, while combining AI planning approaches with the automatic design process and ontology environment. Secondly, the terminology is vague in some composition classifications. For example, Rao and Su [19] use "static composition" to cover those approaches having manual workflow generation, even though the component Web service selection and binding are accomplished automatically. Finally, the lack of clear classification targets is the most significant weakness of existing classification work of Web service composition. Current classification work generally surveys composition types through subjective identification without objective constraints. The resulting classification is then hardly associated with other specific research topics such as software cost and effort estimation. For example, the declarative service composition class [17] focuses on its irregular composition architecture that is almost irrelevant to the composition effort and cost.

In this paper, we first present a novel classification matrix aimed at the influence on the effort of Web service composition. This matrix uses clarified terminology, and differentiates the classifications between the *Context* and *Technology* dimensions. The *Context* dimension includes major effort related contexts that are *Pattern*, *Semiotics*, *Mechanism*, *Design Time* and *Runtime*. When considering different composition *Patterns* for the same target, orchestration deals with a central mediator while choreography is a collaboration of all the participant Web services. Within the *Semiotics* context, semantic Web services have more descriptions than syntactic Web services, which can facilitate service discovery and matchmaking. *Mechanism* context comprises SOAP-based and RESTful composition. RESTful Web service compositions are relatively lightweight compared with SOAP-based compositions. According to the manipulation procedure before generating a real composite Web service, there can be manual, semi-automatic, or automatic compositions at *Design Time*. During *Runtime*, the dynamic and static compositions are differentiated by the adaptability of Web service composition. On the other hand, the Technology dimension is divided into well defined *Workflow-based*, *Model-driven*, and *AI Planning* technology categories. In fact, one composition approach can be classified into one technology category and some context categories at the same time. For example, the approach in [5] uses model-driven technology and is under the contexts: Orchestration, Semantics, SOAP, Manual, and Static. Therefore, a matrix is suitable to represent this kind of cross-classification.

Considering the different influences on the composition effort, different context types and technology categories can be viewed as different effort factors of Web service compositions. After applying a set of effort estimation hypotheses to these factors, we can get a checklist that qualitatively defines their effort influences. By using several assistant symbols and rules, an effort score is further assigned to each factor to reflect its influence on composition effort. By associating the effort scores with the applied hypotheses, we can then build an effort-estimation-checklist table based on our previously proposed effort-oriented classification matrix of Web service composition [1]. Supposing the effort scores of different factors across two dimensions can be multiplied to reflect their combined influence on composition effort, the multiplied result are also specified in the corresponding cross area in this table. Eventually, the effort-estimation-checklist table facilitates comparing the qualitatively estimated effort of different composition approaches listed in the classification matrix.

The contributions of this research are manifold. Firstly, the complete classification matrix can help researchers explore the knowledge space in service composition domain, and help developers choose suitable techniques when composing Web services. Secondly, since different technology categories and context types can be regarded as different effort factors when composing Web services, a set of effort estimation hypotheses are proposed and a checklist is generated to qualitatively define these factors' influences on composition effort. Thirdly, an effort-estimation-checklist table is built, which can further help researcher and developers compare the qualitative effort between different composition approaches. Last but not least, new research opportunities could be revealed when comparing and analyzing those different composition approaches.

This paper is organized as follows. Section II justifies the necessity of the research into effort estimation for Web service composition. The two following sections try to identify effort factors of Web service composition by building up a classification matrix. Section III introduces the context-based classification through specifying every type of context. Section IV presents the technology-based classification, and explicitly defines different technology categories. In addition, a part of our work is demonstrated in Appendix I as an example of classification matrix of Web service composition. Section V introduces a set of effort estimation hypotheses, and applies these hypotheses to different composition effort factors. The result then constitutes an effort-estimation-checklist table, as illustrated in Appendix II. The conclusion is drawn, and some potential research opportunities are identified in Section VI.

II. Necessity of Effort Estimation for Web Service Composition

As previously mentioned, service composition has increasingly become a significant type of SOA project. In SOA, composition of services is the concept with which we provide support for business processes in a flexible and feasible way. Through this way of business support, business processes in SOA are essentially a composition of service invocations in a certain order with rules that influence the execution and other constructs, such as parallel invocations, transformations of data, dependencies, and correlations. As organizations move to having more and more services, and business application software will increasingly rely on subscribing services [49], then the major problem in SOA implementation will be service composition and may be less on development of new services.

Consequently, we can concentrate on the service composition as a breakthrough in effort estimation for SOA implementations that is crucial for properly balancing the benefit and cost in SOA system investment or project bidding. In practice, contemporary SOA is intrinsically reliant on Web services, and meanwhile Web service concept and technology used to actualize service-orientation have influenced and contributed to a number of the common SOA characteristics [50]. Therefore, Web service can be viewed as the de facto implementation of service concept, and we can then focus on the effort estimation for Web service compositions.

To the best of our knowledge, unfortunately, there is little work published about estimating effort of composing Web services. Through literature review, we believe the challenges of effort estimation for Web service composition are mainly twofold:

- ***The complexity of Web service composition.*** Following general principles of SOA, composing Web services may comprise distributed processes because component Web services are loosely coupled and could scatter in different locations. Josuttis [46] has pointed out that distributed processing would be inevitably more complicated than non-distributed processing, and any form of loose coupling would increase complexity.
- ***The diversity of Web service composition.*** Existing works [1, 17, 19] have revealed that numerous solutions to Web service composition have been proposed during the past decade. Different techniques and contexts may result in different influence on the final effort of an instance of Web service composition.

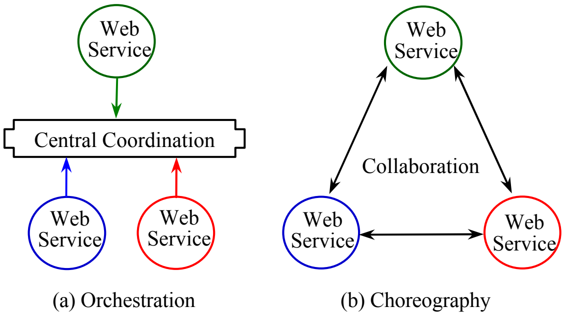
Limited to these two challenges, it is nearly impossible to collect enough development data to estimate effort of various complex compositions quantitatively. For a particular Web service composition project, nevertheless, qualitative effort comparison between different composition approaches can still facilitate developer's decision making. Therefore, this paper is to investigate such a method to realize the qualitative comparison between composition effort estimates.

III. Contex-based Classification of Web Service Composition

The context discussed here refers to the environment and different stages involved in composing Web services. Through analyzing the lifecycle of Web service composition, we have identified several contexts: *Pattern, Semiotics, Mechanism, Design Time, and Runtime* that have the most influence on composition effort.

A. Pattern: Orchestration and Choreography

According to the methods of cooperation among component Web services, the Web service composition patterns can be distinguished between *orchestration* and *choreography*.



(a) Orchestration (b) Choreography

Figure 1. Web Service Orchestration and Choreography.

Orchestration, as shown in Figure 1(a), describes and executes a centralized process flow that normally acts as a coordinator to the involved Web services. The central coordinator explicitly specifies the business logic and controls the order of invocation of Web services. As a result, the coordination defines a long-term, cross-organization, transactional process. The involved Web services, on the other hand, need not be aware of their involvement in an orchestrated process. Orchestration represents coordination from the perspective of a single participant that can be another Web service.

Choreography, as shown in Figure 1(b), describes collaboration between web services that focuses on the peer-to-peer message exchange. The collaboration is decentralized where all participating Web services work equally and do not rely on a central controller. Each Web service involved in choreography understands its contribution to a business process: operation, timing of operation, and the interaction with other participants. Choreography represents collaboration from a global perspective.

In brief, orchestration and choreography describe two aspects of Web service composition for creating business processes [38]. Orchestration concentrates on the interactions of a single Web service with its environment, while choreography concentrates on the exchange of messages among all the involved Web services. Consequently, an orchestration can be broken down into a series of primitive workflow logic activities, which invokes Web services following the determined execution sequence based on the central controller's enactment; whereas a choreography can be broken down into a series of message exchanges, which is not to control but to make autonomous participants cooperate based on their agreement.

In most cases, the pattern to which Web service composition belongs can be identified easily through the adopted standards or flow languages. For example, the current de facto standard for Web service orchestration is the Business Process Execution Language also known as BPEL. BPEL is an executable business process modeling language that can be used to describe the execution logic by defining the control flow and prescribing the rules for managing the non-observable data. The BPEL engine can then execute the description and orchestrate the pre-specified activities. Whereas one of the most widespread W3C recommended protocols for choreography is Web Services Choreography Description Language (WS-CDL). WS-CDL is designed to describe the common and collaborative observable behavior of multiple Web services that interact with each other to achieve their common goal. In other words, WS-CDL description offers the specification of collaborations between the participants involved in choreography.

Therefore, we can conveniently identify that the BPEL description related Web service compositions normally have orchestration context, e.g. [22], while WS-CDL description involved Web service compositions generally have choreography context, e.g. [23]. Nevertheless, the Web service composition pattern should not be judged merely through these keywords, because the technique can be adapted to satisfy different scenarios. For example, some people advocate the use of abstract BPEL as a choreography language. Consequently, the most reliable judgment should be still based on the understanding of the Web service composition process.

B. Semiotics: Syntactic and Semantic Compositions

The semiotic environment is becoming a more significant context for Web service composition as the Web evolves. Semiotics is the general science of signs, which studies both human language and formal languages. *Syntax* and *Semantics* are two of fundamental components of semiotics. Syntax relates to the formal or structural relations between signs and the production of new ones, while semantics deals with the relations between the sign combinations and their inherent meaning.

Currently, the World Wide Web can be mainly considered as syntactic Web that uses Hyper Text Markup Language (HTML) to compose documents and publish information. When it comes to Web services, the syntactic level XML standards, for example Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI) have been used extensively to address corresponding e-business activities and research issues in industry and academia. By using human-oriented metadata, SOAP is designed to provide descriptions of message transport mechanisms; WSDL is for describing the interfaces of Web services; while UDDI registers Web services by their physical attributes such as name, address and functional categorization. However, the syntactic Web was designed primarily for human interpretation and conveying information, a syntactic web page does not contain special tagging and the meaning of information is not readable by a computer program. The lack of machine-readable semantics then requires human intervention for Web service discovery and composition, and therefore hampers the usage of Web services in complex business environment.

To overcome the obstacles of interpretability and interoperability between traditional systems and applications, the semantic Web was proposed through incremental and information-added adjustments. These adjustments make the Web ontological. Ontology was originally developed to facilitate knowledge sharing and reuse [37]. Benefiting from ontology, greater ability of expression is provided for knowledge modeling and communicating knowledge between heterogeneous and distributed application systems. Therefore, the semantic Web can be viewed as a version of a Web of ontological contents and services, which includes machine-readable and human-transparent descriptions to the existing data and documents on the syntactic Web. In addition, the semantic Web supplies the necessary infrastructure and techniques for publishing, resolving and reasoning ontological descriptions of the contents and services.

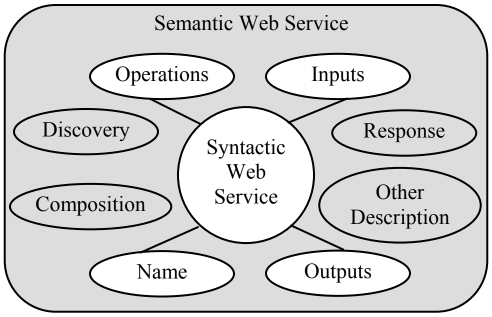


Figure 2. Syntactic Web Service and Semantic Web Service.

As for semantic Web services, besides the syntactic description, the information needed to select, compose, and respond to services are also encoded with semantic markup at the service Web sites. These efforts of service augmentation can then facilitate automated service discovery, composition, dynamic invocation and binding without human assistance or highly constrained agreements on protocols. Figure 2 illustrates the differences between syntactic and semantic Web services. Informally, a Web service can be characterized by its required inputs, the produced outputs, and the operations it will take [36]. The inputs and outputs may be further subject to pre-conditions and post-conditions respectively. With only descriptions in the syntactic level, as shown in the unfilled nodes of syntactic Web service in Figure 2, it is difficult for service providers and consumers to represent or interpret the meaning of inputs, outputs and other applicable constraints. A semantic Web service relaxes such limitation by augmenting the service description with a rich set of formally semantic annotations of the service's capabilities, as shown in the grey nodes of semantic Web service in Figure 2. Accordingly, new standards and languages of semantic markup, like Web Ontology Language for Web Services (OWL-S) and Web Service Modeling Ontology (WSMO), should be investigated and used to give meaning to Web services.

Overall, the XML-based standards are for syntax, whilst the ontology-based standards are for semantics. Both share unified Web infrastructure and together provide capability for developing Web applications that deal with data and semantics. Nevertheless, one of the most important characteristics of ontology-based techniques is that they allow a richer integrability and interoperability of data in communications between domains. As previously mentioned, driven by the semantic markup and agent technologies, semantic Web service discovery, selection, composition, and execution are all supposed to be automatic tasks. Although fully automating these processes is still a challenge, accomplishing parts of this goal can still be achieved. For example, the semantic description is useful for the translation between Web service composition problems and AI-planning systems [13], while the semantic matchmaking can be used to facilitate the automatic Web service discovery [2]. Considering these outstanding characteristics, Web service compositions can be categorized according to syntactic and semantic context, while the context can be also identified through employed standards and techniques.

C. Mechanism: RESTful and SOAP-based Compositions

Concentrating on the technologies and architectures, nowadays there are two main mechanism paradigms of building composite Web services, namely *RESTful* composition and *SOAP-based* composition.

Basically, REpresentational State Transfer (REST) and Simple Object Access Protocol (SOAP) are not directly comparable with each other and not necessarily opposite. REST is an architectural style originally designed for building large-scale distributed hypermedia systems, whereas SOAP is a general protocol used as one foundation of numerous WS-* technologies. Within the REST environment, the Web is considered as a universal storage medium for publishing globally accessible information. In contrast, SOAP treats the Web as the universal transport mechanism for message exchange. When building Web services, traditional SOAP/WS-* environment requires relatively heavyweight open standards than that are being used in RESTful context. Although the SOAP vs. REST debate has been an ongoing discussion for some time, there is an implicit consensus that REST is more suitable for basic, ad-hoc, client-driven scenarios, while SOAP/WS-* are more suitable to address the quality of services requirements in highly interactive Web applications.

However, RESTful and SOAP-based Web services are indeed comparable. We can identify the differences between RESTful and SOAP-based Web services mainly through their interfaces, the operations and Message Exchange Patterns (MEPs) behind interfaces, and their QoS support techniques.

1) Interface differences. The interface of a RESTful Web service comprises a variable set of Uniform Resource Identifiers (URIs). Each URI uses a globally unique address to identify a specific resource. Unfortunately, to the best of our knowledge, there is no standard and machine-processable way of describing RESTful interfaces. Using WSDL 2.0 description to wrap the RESTful Web services has been revealed as a burden for service consumers [32]. The Web Application Description Language (WADL) and other dedicated interface definition languages for RESTful services like RESTful Interface Definition and Declaration Language (RIDDL) [33] are not yet widely employed. Consequently, most of the time the interfaces of RESTful Web services are described through natural, informal, and more human-oriented documentations. When it comes to SOAP-based Web services, as mentioned previously, WSDL has gained widespread adoption to syntactically define the service interfaces. In a WSDL document, SOAP-based Web services are described as collections of network endpoints, or ports. A port associates a network address with a reusable binding. The reusable WSDL binding contains the concrete transport protocol and data format specifications for a particular port type. A port type is a set of abstract operations that are related to some abstract messages representing the data for exchange. Benefiting from the abstract interfaces described by WSDL, technical details of SOAP-based Web services can be concealed, for example, the implementation language, deployment platform and underlying communication protocol.

2) Operation differences. Since "REST is in many ways a retrospective abstracting of the principles that make the World Wide Web scaleable" [34], RESTful Web services requires little technology support apart from well accepted HTTP and XML infrastructures. As a result, the manipulations of resources are completely constrained in the RESTful environment through a fixed set of four operations associated with HTTP: GET, PUT, DELETE, and POST. GET is used to retrieve a representation of the current state of a resource. PUT can either update the state of existing resource or create a new resource with the request URI if it does not previously exist. DELETE is used to delete a URI-identified resource and also invalidate the URI itself. POST creates subordinate resources to which new URIs are assigned by service provider. In contrast to the standard operations among RESTful Web services, the operations provided by SOAP-based Web services are ad hoc. Various APIs defined in different WSDL documentations represent different sets of operations for communication and interaction between service providers and consumers. The operations of SOAP-based Web services essentially are functional components that are located on remote machines and can be invoked through APIs over the network.

3) MEPs differences. MEPs are patterns or templates that abstract the sequences of message transmission in the Web service context. Since REST is associated closely with HTTP, and HTTP is stateless request-response application protocol, RESTful Web services only have the synchronous request-response pattern under the HTTP mechanism. SOAP-based Web services allow rich patterns ranging from traditional request-response to broadcasting and sophisticated message exchanges. The latest WSDL 2.0 has been published with supporting eight MEPs [35]. Each MEP describes a bilateral message exchange between two involved services from a service point's perspective.

- *In-Only* – The service receives a message.
- *Robust In-Only* – The service receives a message and will return a fault message only when meeting a fault.
- *In-Out* – The service receives a message and returns a response message.
- *In-Optional-Out* – The service receives a message and optionally returns a response message.
- *Out-Only* – The service sends a message.
- *Robust-Out-Only* – The service sends a message and will receive a fault message only when its partner service meets a fault.
- *Out-In* – The service sends a message and receives a response message.
- *Out-Optional-In* – The service sends a message and optionally receives a response message.

4) QoS support technique differences. Quality of Service (QoS) indicates a certain performance level of services that will be delivered to consumers, and can be evaluated through corresponding parameters like response time, throughput, cost, etc. As REST is usually used in conjunction with HTTP, the QoS of RESTful services are supported generally through basic protocols and techniques. For example, services' interactions can be secured at the transport layer using the Secure Sockets Layer (SSL) protocol, while the security of messages can be guaranteed by encryption and digital signatures. On the contrary, SOAP-based Web services adopt more complicated mechanisms to cover QoS features. On the one hand, the header of an SOAP document contains message-layer infrastructure information that can be used for QoS configurations. On the other hand, the WS-* technology stack is employed to satisfy the large scope of QoS requirements such as transactions, security, and reliability. Benefiting from SOAP and WS-* technologies, QoS aspects of SOAP-based Web services are protocol transparent and independent. In other words, the QoS of Web service can be provided end to end without taking into account the variety of middleware systems transported.

All these differences between RESTful and SOAP-based Web services make the problem of RESTful Web service composition fundamentally different from the composition problem of SOAP-based Web service. SOAP-based Web service composition is a collection of related, structured activities or tasks that produce a specific service or product for a particular customer. Within the relatively complex SOAP-based environment, a large number of standards and tools have been developed to facilitate the service composition activities. Dissimilarly, RESTful Web service composition integrates normally disparate Web resources to create a new application. These resources can be the exposure of pure data or traditional application functionality. With the constraint of lightweight technologies adopted in RESTful environment, service compositions mainly focus on the Web 2.0 Mashups that usually imply simple and fast integration of data/content from different sources on the Internet.

D. Design Time: Manual, Semi-Automatic and Automatic Compositions

Generally, there are four fundamental activities when composing a Web service, namely *Planning, Discovery, Selection*, and *Execution* [18]. *Planning* is to determine a composition plan including the execution sequence of tasks. Each task corresponds to either the functionality or activity of a service. *Discovery* is to find all the candidate services that can satisfy the tasks in the plan. The aim of *Selection* is to choose optimal subset from all the discovered services by using non-functional attributes. *Execution* builds a real composite Web service. In practice, the sequence of *Planning, Discovery*, and *Selection* can be diverse. For example, the theorem proving approach in [13] is based on the pre-determined Web services to generate the composition plan. Moreover, during the service composition procedure, the network configurations and non-functional factors may change, and existing Web services may be updated or terminated. As a result, some pre-identified services may not be available, and the new ones need to be re-selected or re-discovered. In other words, *Discovery* and *Selection* can also take place during or even after *Execution*. Therefore, we can define a potential *Adaptation* activity at the end of the procedure of Web service composition.

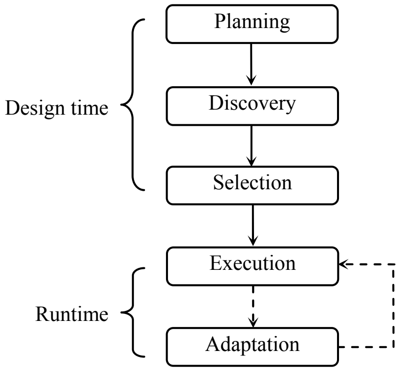


Figure 3. Stages of a Web Service Composition Scenario.

Based on the previous analysis, the process of Web service composition can be separated into design time and runtime stages. Figure 3 shows one of the possible composition scenarios. Depending on the real practices, the design time stage comprises various activities from only *Planning* to the combination of *Planning, Discovery*, and *Selection.* According to the extent to which human intervention is involved, the design time procedure can be *manual*, *semi-automatic*, and *automatic*. Considering that there is still a long way to realize the complete automation of Web service composition even at design time, we mainly concentrate on the *Planning* activity when unfolding classification. Therefore, we can draw the outline of these three types of composition approaches during design time as:

1) Manual approach. In general, the manual *Planning* activity implies manual Web service composition. Two different scenarios of manual approaches can be further identified respectively as primitive level and abstract level respectively. In primitive manual composition approaches, developers have to specify every detailed activity in the composition processes. The resulting specifications are executable composition programs. For example, we can use BPEL to describe the procedure of Web service composition following the logic of corresponding business process, and the finalized description is executable with the support of BPEL engine. As for the manual composition approaches at an abstract level, the Web service composition plans are usually drawn into abstract workflows or models instead of specific programs. In such approaches the manual planning results cannot be executed directly, but can be transformed into executable specifications and finally executed by some tools or engines. Examples can be found in most of the UML related model-driven approaches.

2) Automatic approach. In general, the automatic *Planning* activity implies automatic Web service composition. In manual approaches discussed above, although we can decrease the effort of Web service composition through abstraction rather than programming, the planning phase still has to be realized manually. How to automatically generate the composition model or workflow then becomes a subsequent research topic. The current trend is to use Artificial Intelligence (AI) planning to satisfy the automation of the generation of a Web service composition plan. Benefiting from existing AI planning systems, the prerequisite effort of Web service composition is only to encode the requirements into dedicated, formal, and mathematical expressions.

3) Semi-automatic approach. We treat an instance of Web service composition as semi-automatic approach, if one of the following cases is met: (1) there are specifically automatic *Discovery/Selection* activities to facilitate manual *Planning*; or (2) there are specifically manual *Discovery/Selection* activities that constrain automatic *Planning*. Taking [2] as an example of the former case, semantic matchmaking technique is used to realize the semi-automatic approach by automatically filtering and presenting matching services to the user at each step of a composition. An example of the latter case can be found in [13], the theorem proving technique requires manually pre-determining Web services before automatically generating the composition plan.

E. Runtime: Static and Dynamic Compositions

The *Execution* and potential *Adaptation* activities remain at the runtime stage of Web service composition. By focusing on the *Adaptation* activity, we can define that the Web service composition is *dynamic* at runtime if it is adaptive with minimal user intervention, otherwise it is *static*. In detail, static Web service composition re-discovers and re-selects new services manually when adapting the environment. In the worst case, static composition does not have adaptability at all. On the contrary, dynamic composition can re-discover and re-select new services at runtime without requiring any human assistance. Moreover, we also define a dynamic Web service composition if services can be discovered and selected during *Execution* activity, for instance eFlow [3].

Benefiting from the division between the design time and runtime of Web service compositions, we can clearly distinguish the two concepts: automatic and dynamic compositions that are confusing in the existing literature. Furthermore, it can be found that there is no relationship between automatic composition at design time and dynamic composition at runtime. On the one hand, automatic composition does not imply dynamic composition, for example, most of the AI planning approaches only concentrate on the automatic *Planning* process while leaving the planning result executed statically. On the other hand, static composition does not require automatic composition, for example, the visual language UML Profile for Web Service Composition (UML-WSC) [7] supports dynamically composing Web services although the composition model is still built manually.

IV. Technology-based Classification of Web Service Composition

Technology refers to the techniques used in the approaches to implement Web service composition. It is difficult to enumerate all kinds of composition techniques, although different technique can contribute different composition effort. However, we can identify three groups of techniques: Workflow-based, Model-driven, and AI planning techniques.

A. Workflow-based Techniques

Workflow is a virtual representation of actual work including a sequence of operations. Workflow-based Web service composition uses the workflow perspective to describe the normally complex collaboration among Web services and implement the composition procedure. There are two ways to describe the Web service composition workflow:

- *To program the executable workflow directly*: Obviously, the composition process can be programmed from scratch by using traditional languages and standards. However, the current universal technique is to use the dedicated, process-oriented language, for example the current de facto executable business process modeling language BPEL, to specify the transition interactions among Web services at a macro-level state.
- *To draw the abstract workflow without programming*: Supported by some tools and engines, the workload of Web service composition can be relieved by drawing the abstract workflow without programming. For example, the semantic matchmaking based approach [2] uses the GUI panel of composer to construct an abstract flow, while eFlow [3] adopts a graph-oriented method to define the interaction and order of execution among the nodes in an abstract composition process.

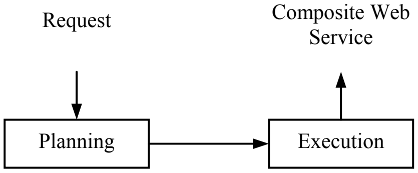


Figure 4. One-Stop Process of Web Service Composition.

If we only focus on the two main activities (*Planning* and *Execution*) in the Web service composition approaches, workflow-based techniques generally follow the One-Stop process, as shown in Figure 4. In the One-Stop process, the *Planning* activity happens just after receiving the composition requirement, and delivers the executable composition specification directly. In most cases of One-Stop based approaches, during the planning stage the user must provide inputs at choice points, decide the interoperation among component Web services, and specify the composition procedure.

B. Model-driven Techniques

In model-driven approaches of Web service composition, models are used to describe user requirements, information structures, abstract business processes, component services and component service interactions. The models are independent of, but can be tranformed into, executable composition specifications. Generally, there is also modeling work in several workflow-based techniques. Whereas the model-driven techniques discussed here merely follow the standards provided by the Object Management Group (OMG). The standards mainly refer to the Unified Modeling Language (UML) and Model-Driven Architecture (MDA).

Numerous discussions related to UML-based modeling of Web service composition can be found in the literature. Through analysis and abstraction, we can further identify two basic scenarios of model-driven approaches for composing Web services.

- *To build executable composition model*. A typical example of this particular scenario is the UML-WSC profile [7]. The UML-WSC profile is a well-defined UML extension, which uses a static model and extended variant of activity diagrams to define the process-oriented Web service composition. The static model describes the available Web services and components, while the extended variant of activity diagrams describes the composition processes. The composition model specified through UML-WSC profile can be executed automatically by a process engine. Therefore, the UML-WSC profile is also considered as an alternative to non-visualized languages like BPEL.

- *To build transformable composition model*. This generic scenario is to use UML class diagrams to represent the state parts of compositions, while the behaviour parts are represented through UML activity diagrams. The state parts can be Web service interface [4], the structure of composite Web service [5] and QoS characteristics [6]. On the other hand, the behaviour parts describe the composition operations, interactions of component Web services, and control and data flow. Furthermore, since BPEL is widely accepted for composing Web services, UML has been designedly to extend BPEL to include common aspects of Web service composition. Therefore, the modeling results can be conveniently transformed into executable BPEL specifications to eventually realize Web service compositions.

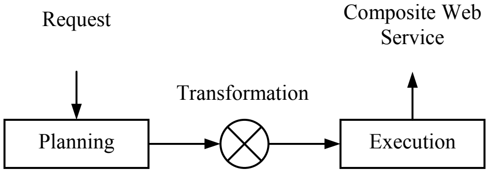


Figure 5. Bridge Process of Web Service Composition.

Although the former, particular scenario of model-driven approach still employs the One-Stop process for Web service composition, most of the existing modeling techniques adopt the Bridge process when composing Web services, as illustrated in Figure 5. The Bridge process can be viewed as an evolution from the One-Stop process, which describes such approaches that plan Web service compositions at an abstract level, while the planning results cannot be directly executed and have to be transformed into executable specifications. Therefore, unlike the first scenario of model-driven approaches employing the One-Stop process, any Web service composition approach adopting the Bridge process uses a transformation procedure for the mapping between the planning result and executable specification. The notion of the Bridge process is that the planning phase of Web service composition does not need to be tied to any particular composition language and execution engine, and thereby the same planning result can be transformed into more than one executable description.

C. AI Planning Techniques

AI planning seeks to use intelligent systems to generate a plan that can be one possible solution to a specified problem, while a plan is an organized collection of operators within the given application domain. AI planning is essentially a search problem. The underlying basis of planning relies on state transition system with states, actions and observations. Benefiting from the state transition system, the planner explores a potentially large search space and produces a plan that is applicable to bridge the gap between the initial state and goal when run. AI planning in Web service composition normally comprises of five attributes, they are (1) all the available services, (2) the initial state, (3) the state change functions, (4) all the possible states, and (5) the final goal. The initial state and final goal are specified in the requirements for composing Web service. The state change functions define the preconditions and effects when invoking Web services.

A large amount of research has been reported about the AI planning related Web service composition. These works apply techniques ranging from Situation Calculus [8], Automata Theory [9], Rule-based Planning [10], Query Planning [12], Theorem Proving [13], Petri Nets [14], to Model Checking [15]. Generally, these techniques convert the problems of composition into generating execution workflows using the dedicated expression. The workflows can then be transformed into executable specifications like BPEL documents or other XML-based descriptions, and executed through the corresponding engines.

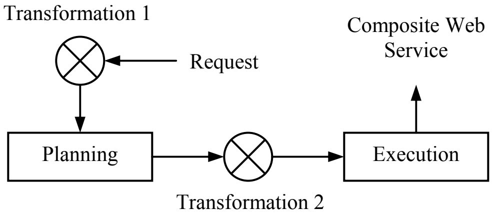


Figure 6. Double-Bridge Process of Web Service Composition.

Therefore, we can find that the Web service composition approaches using AI planning techniques normally contain the Double-Bridge process, as shown in Figure 6. The Double-Bridge process can be treated as further evolution from the Bridge process. The *Planning* activity is settled between two transformation procedures in a Double-Bridge process. In detail, since AI planning systems generally adopt dedicated, formal, and mathematical techniques, the initial information and composition requirement must be transformed for input into a planning system, and the planning result should be transformed again into an executable specification to build a composite Web service.

V. Qualitative Discussion about Effort Estimation for Web Service Composition

Through categorizing Web service composition approaches along Context and Technology dimensions, a classification matrix can be established, as demonstrated in

Appendix I. Considering the different influences of different contexts and techniques on the composition effort, those technology categories and context types in the classification matrix can be viewed as effort factors when composing Web services. Therefore, we can use the classification matrix to facilitate the cost and effort estimation for different Web service composition approaches. Since the data we collected here are all based on qualitative descriptions, it is not suitable to do quantitative work for composition effort estimation. Through analyzing these qualitative descriptions, however, we can further build a checklist for experts to judge qualitatively the effort when implementing Web service compositions. Before building the qualitative effort estimation checklist, some effort related hypotheses should be investigated.

A. Qualitative Effort Estimation Hypotheses

In the context of software engineering, effort of a task is generally accounted by calculating how long and how many workers are needed to finish the task, and the unit can be person-day, person-month, or person-year. In brief, the amount of human activities in a project is proportional to the amount of effort required to finish the project. Therefore, for a certain software project, we can hypothesize:

H1. The increase of human activities in a project will have a proportional impact on the final effort.

Human activities include both physical and mental activities. Since software engineering is a knowledge-intensive domain, the effort of a software project is mainly composed of mental activities. Unfortunately, within a given time span people have limited mental capability to deal with information [39]. For every single person, the increased amount of information beyond a certain point may even defeat his/her mental ability, and hence result in errors [41]. As a result, the more information that exists in a project, the more people and human activities will be required to perform accurate manipulations. Together with H1, therefore, we can hypothesize:

H2. The increase of information in a project will have a proportional impact on the required human activities.

H2'. The increase of information in a project will have a proportional impact on the final effort.

Moreover, complexity has been proved to be a significant and non-negligible factor that influences software development and maintenance [42]. Meanwhile, the more complexity involved in a system, the more difficulty the designers or engineers have to understand the implementation process and thus the system itself [40], and hence the greater mental effort people have to exert to solve the complexity [39]. To summarize, we can further hypothesize:

H3. The increase of complexity in a project will have a proportional impact on the final effort.

When it comes to project complexity, one of the main contributors is the complexity of the methods that regard achieving the project goals [43]. The methods mentioned herein generally consist of processes, tools, and techniques that are used to complete the corresponding project [44]. In particular, processes and techniques have been viewed as internal environment of a system (organization), while the system's complexity is considered a response to the environmental complexity [45]. Consequently, the complexity of processes and techniques involved in a software project will positively influence the complexity of the project. As for the tools, although the adoption of sophisticated tools usually implies a complex project, tools are essentially developed and used to save human activities. For a certain project, the more work the tools can fulfill, the less human activities the project will require. Overall, we can also hypothesize:

H4. The increase of process complexity in a project will have a proportional impact on the project complexity.

H4'. The increase of process complexity in a project will have a proportional impact on the final effort.

H5. The increase of difficulty of techniques in a project will have a proportional impact on the project complexity.

H5'. The increase of difficulty of techniques in a project will have a proportional impact on the final effort.

H6. The increase of work that tools can fulfill in a project will have an inversely proportional impact on the human activities.

H6'. The increase of work that tools can fulfill in a project will have an inversely proportional impact on the final effort.

B. Qualitative Effort Estimation Checklist for Web service composition approaches

As mentioned earlier, we treat technology categories and context types in the classification matrix as effort factors of Web service composition approaches. After applying different effort estimation hypotheses to different but comparable factors, a set of qualitative effort estimation statements will be generated. These statements can then constitute a checklist for developers and engineers to qualitatively judge and compare the effort and cost of different composition strategies. In fact, using a checklist has been considered a simple way of utilizing experience and advocated as an efficient method of improving expert judgment processes when doing estimation [48]. To facilitate building this qualitative effort estimation checklist, some symbols and rules are also proposed:

For one certain task of Web service composition, we use $E_{F\text{-}H}$ to represent the effort E determined by factor F when applying hypothesis H. Moreover, a score S will be set for $E_{F\text{-}H}$ to flag different effort determined by different but comparable factors when applying some hypothesis. For convenience of calculation, the rules of score setting can be:

$$\begin{cases} S(E_{F1\text{-}H}) = 2,\ S(E_{F2\text{-}H}) = 1 \quad \text{if } E_{F1\text{-}H} > E_{F2\text{-}H} \\ S(E_{F1\text{-}H}) = 1,\ S(E_{F2\text{-}H}) = 1 \quad \text{if } E_{F1\text{-}H} \approx E_{F2\text{-}H} \\ S(E_{F1\text{-}H}) = 1,\ S(E_{F2\text{-}H}) = 2 \quad \text{if } E_{F1\text{-}H} < E_{F2\text{-}H} \end{cases} \qquad (1)$$

Note that if we use E_F to represent the effort E determined by factor F under all the different but applicable hypotheses, then all the scores for E_F under corresponding hypotheses can be summed up and represented as $S(E_F)$.

We can hereby build the effort estimation checklist following the sequence of building the classification matrix.

1) For Orchestraton and Choreography: As analyzed previously, orchestration stands for a central coordination, while choreography represents multiparty collaborations. Since distributed processing would be inevitably more complicated than non-distributed processing [46], for a same Web service composition project choreography requires more effort than orchestration if applying H3. Meanwhile, as the current de facto standard of orchestrating Web services, BPEL stemmed from existing languages and tools and has been widely accepted, whereas the choreography language WS-CDL was developed without any prior implementation and is still far from maturity [47]. Considering this technical influence, the implementation of choreography will be more difficult than that of orchestration. By using *For* for representing the effort factor Orchestration and *Fch* for Choreography, the effort compare and scores can be listed in Table I.

TABLE I. EFFORT COMPARE BETWEEN ORCHESTRATION AND CHOREOGRAPHY

Applied Hypotheses	Compare	Scores
H3	$E_{For\text{-}H3} < E_{Fch\text{-}H3}$	$S(E_{For\text{-}H3})$=1, $S(E_{Fch\text{-}H3})$=2
H5’	$E_{For\text{-}H5’} < E_{Fch\text{-}H5’}$	$S(E_{For\text{-}H5’})$=1, $S(E_{Fch\text{-}H5’})$=2
Total	$E_{For} < E_{Fch}$	$S(E_{For})$=2, $S(E_{Fch})$=4

2) For Syntactic and Semantic Compositions: Since semantic Web and semantic Web services are proposed to automate service discovery, selection, composition and execution by adding the inherent meanings, human activities within semantic compositions will be decreased while the involved information will be increased. Considering the increased information is for machine interpretation rather than human intervention, however, hypothesis H2 is not applicable here. Meanwhile, syntactic and semantic Web services share the unified Web infrastructure and both use markup language based techniques to describe information. It can then be stated that the difficulty levels of techniques adopted in both syntactic and semantic service compositions are similar. Therefore, by using *Fsy* for representing the effort factor Syntax and *Fse* for Semantics, the effort compare and scores can be listed in Table II.

TABLE II. EFFORT COMPARE BETWEEN SYNTACTIC AND SEMANTIC COMPOSITION APPROACHES

Applied Hypotheses	Compare	Scores
H1	$E_{Fsy\text{-}H1} < E_{Fse\text{-}H1}$	$S(E_{Fsy\text{-}H1})$=1, $S(E_{Fse\text{-}H1})$=2
H5’	$E_{Fsy\text{-}H5’} \approx E_{Fse\text{-}H5’}$	$S(E_{Fsy\text{-}H5’})$=1, $S(E_{Fse\text{-}H5’})$=1
Total	$E_{Fsy} < E_{Fse}$	$S(E_{Fsy})$=2, $S(E_{Fse})$=3

3) For SOAP-based and RESTful Compositions: Compared with RESTful Web service compositions, SOAP-based compositions employ more sophisticated techniques including heavyweight protocols, a set of WS-* stack, and more MEPs, which can satisfy more QoS requirements while also deal with more information. Therefore, the hypotheses H2’ and H5’ are both applicable. Incidentally, although the SOAP/WS-* related techniques indeed are complex, they should still be adopted when addressing advanced requirements especially in the enterprise computing scenarios. However, here we only focus on the implementation effort without considering other tradeoffs. By using *Fso* for representing the effort factor SOAP and *Fre* for REST, the effort compare and scores can be listed in Table III.

TABLE III. EFFORT COMPARE BETWEEN SOAP-BASED AND RESTFUL COMPOSITION APPROACHES

Applied Hypotheses	Compare	Scores
H2’	$E_{Fso\text{-}H2’} > E_{Fre\text{-}H2’}$	$S(E_{Fso\text{-}H2’})$=2, $S(E_{Fre\text{-}H2’})$=1
H5’	$E_{Fso\text{-}H5’} > E_{Fre\text{-}H5’}$	$S(E_{Fso\text{-}H5’})$=2, $S(E_{Fre\text{-}H5’})$=1
Total	$E_{Fso} > E_{Fre}$	$S(E_{Fso})$=4, $S(E_{Fre})$=2

4) For Manual, Semi-Automatic, and Automatic Compositions: During the design time of Web service compositions, the more automated the design processes are, the less human activities the compositions will require, and the less detailed information developers need be concerned with. Considering the realization of automation usually requires assistant tools and more techniques, for example the Semantic Matching approach [2], the hypotheses H5’ and H6’ are both applicable together with H1 and H2’. By using *Fma* for representing the effort factor Manual, *Fsa* for Semi-Auto and *Fau* for Auto, the effort compare and scores can be listed in Table IV.

TABLE IV. EFFORT COMPARE BETWEEN MANUAL, SEMI-AUTOMATIC AND AUTOMATIC COMPOSITION APPROACHES

Applied Hypotheses	Compare	Scores
H1	$E_{\text{Fma-H1}} > E_{\text{Fsa-H1}}$ $E_{\text{Fma-H1}} > E_{\text{Fau-H1}}$ $E_{\text{Fsa-H1}} > E_{\text{Fau-H1}}$	$S(E_{\text{Fma-H1}})=2+2=4$ $S(E_{\text{Fsa-H1}})=1+2=3$ $S(E_{\text{Fau-H1}})=1+1=2$
H2’	$E_{\text{Fma-H2’}} > E_{\text{Fsa-H2’}}$ $E_{\text{Fma-H2’}} > E_{\text{Fau-H2’}}$ $E_{\text{Fsa-H2’}} > E_{\text{Fau-H2’}}$	$S(E_{\text{Fma-H2’}})=2+2=4$ $S(E_{\text{Fsa-H2’}})=1+2=3$ $S(E_{\text{Fau-H2’}})=1+1=2$
H5’	$E_{\text{Fma-H5’}} < E_{\text{Fsa-H5’}}$ $E_{\text{Fma-H5’}} < E_{\text{Fau-H5’}}$ $E_{\text{Fsa-H5’}} < E_{\text{Fau-H5’}}$	$S(E_{\text{Fma-H5’}})=1+1=2$ $S(E_{\text{Fsa-H5’}})=2+1=3$ $S(E_{\text{Fau-H5’}})=2+2=4$
H6’	$E_{\text{Fma-H6’}} > E_{\text{Fsa-H6’}}$ $E_{\text{Fma-H6’}} > E_{\text{Fau-H6’}}$ $E_{\text{Fsa-H6’}} > E_{\text{Fau-H6’}}$	$S(E_{\text{Fma-H6’}})=2+2=4$ $S(E_{\text{Fsa-H6’}})=1+2=3$ $S(E_{\text{Fau-H6’}})=1+1=2$
Total	$E_{\text{Fma}} > E_{\text{Fsa}} > E_{\text{Fau}}$	$S(E_{\text{Fma}})=14$, $S(E_{\text{Fsa}})=12$, $S(E_{\text{Fau}})=10$

5) For Static and Dynamic Compositions: If we emphasize the adaptation in both static and dynamic compositions during runtime, we can draw the same conclusions through the similar analysis as above. Therefore, by using *Fst* for representing the effort factor Static and *Fdy* for Dynamic, the effort compare and scores can be listed in Table V.

TABLE V. EFFORT COMPARE BETWEEN STATIC AND DYNAMIC COMPOSITION APPROACHES

Applied Hypotheses	Compare	Scores
H1	$E_{\text{Fst-H1}} > E_{\text{Fdy-H1}}$	$S(E_{\text{Fst-H1}})=2$, $S(E_{\text{Fdy-H1}})=1$
H2’	$E_{\text{Fst-H2’}} > E_{\text{Fdy-H2’}}$	$S(E_{\text{Fst-H2’}})=2$, $S(E_{\text{Fdy-H2’}})=1$
H5’	$E_{\text{Fst-H5’}} < E_{\text{Fdy-H5’}}$	$S(E_{\text{Fst-H5’}})=1$, $S(E_{\text{Fdy-H5’}})=2$
H6’	$E_{\text{Fst-H6’}} > E_{\text{Fdy-H6’}}$	$S(E_{\text{Fst-H6’}})=2$, $S(E_{\text{Fdy-H6’}})=1$
Total	$E_{\text{Fst}} > E_{\text{Fdy}}$	$S(E_{\text{Fst}})=7$, $S(E_{\text{Fdy}})=5$

6) For Workflow-based, Model-driven and AI Planning Compositions: To simplify the effort analysis in the Technology dimension, we constrain that workflow-based approaches strictly follow the One-Stop process, model-driven approaches strictly follow the Bridge process, and AI planning approaches strictly follow the Double-Bridge process. Considering that the One-Stop process delivers executable specifications, the Bridge process focuses on the abstract modeling, and the Double-Bridge process focuses on the composition requirement, workflow-based approaches have to deal with the most information while AI planning approaches deal with the least information for one certain task of Web service composition. Meanwhile, AI planning approaches have the longest processes while workflow-based approaches have the shortest. However, we can imagine that both One-Stop and Bridge processes also contain two transformation procedures as well as the Double-Bridge process does. The intangible transformation procedures essentially take place as mental activities, while the tangible ones can be supported by tools. Therefore, it can be found that AI planning approaches require less human activities and use more tools, workflow-based approaches require more human activities and use fewer tools, while model-driven approaches are in the middle. When it comes to techniques, it is nearly impossible to compare the difficulty levels of workflow, modeling and AI planning with each other. Consequently, here we simply treat their difficulties similarly. After applying all the suitable hypotheses and using *Fwf* for representing the effort factor Workflow-based, *Fmd* for Model-Driven and *Fai* for AI Planning, the effort compare and scores can be listed in Table VI.

TABLE VI. EFFORT COMPARE BETWEEN WORKFLOW-BASED, MODEL-DRIVEN AND AI PLANNING COMPOSITION APPROACHES

Applied Hypotheses	Compare	Scores
H1	$E_{\text{Fwf-H1}} > E_{\text{Fmd-H1}}$ $E_{\text{Fwf-H1}} > E_{\text{Fai-H1}}$ $E_{\text{Fmd-H1}} > E_{\text{Fai-H1}}$	$S(E_{\text{Fwf-H1}})=2+2=4$ $S(E_{\text{Fmd-H1}})=1+2=3$ $S(E_{\text{Fai-H1}})=1+1=2$
H2’	$E_{\text{Fwf-H2’}} > E_{\text{Fmd-H2’}}$ $E_{\text{Fwf-H2’}} > E_{\text{Fai-H2’}}$ $E_{\text{Fmd-H2’}} > E_{\text{Fai-H2’}}$	$S(E_{\text{Fwf-H2’}})=2+2=4$ $S(E_{\text{Fmd-H2’}})=1+2=3$ $S(E_{\text{Fai-H2’}})=1+1=2$
H4’	$E_{\text{Fwf-H4’}} < E_{\text{Fmd-H4’}}$ $E_{\text{Fwf-H4’}} < E_{\text{Fai-H4’}}$ $E_{\text{Fmd-H4’}} < E_{\text{Fai-H4’}}$	$S(E_{\text{Fwf-H2’}})=1+1=2$ $S(E_{\text{Fmd-H2’}})=2+1=3$ $S(E_{\text{Fai-H2’}})=2+2=4$
H5’	$E_{\text{Fwf-H5’}} \approx E_{\text{Fmd-H5’}}$ $E_{\text{Fwf-H5’}} \approx E_{\text{Fai-H5’}}$ $E_{\text{Fmd-H5’}} \approx E_{\text{Fai-H5’}}$	$S(E_{\text{Fwf-H5’}})=1+1=2$ $S(E_{\text{Fmd-H5’}})=1+1=2$ $S(E_{\text{Fai-H5’}})=1+1=2$
H6’	$E_{\text{Fwf-H6’}} > E_{\text{Fmd-H6’}}$ $E_{\text{Fwf-H6’}} > E_{\text{Fai-H6’}}$ $E_{\text{Fmd-H6’}} > E_{\text{Fai-H6’}}$	$S(E_{\text{Fwf-H6’}})=2+2=4$ $S(E_{\text{Fmd-H6’}})=1+2=3$ $S(E_{\text{Fai-H6’}})=1+1=2$
Total	$E_{\text{Fwf}} > E_{\text{Fmd}} > E_{\text{Fai}}$	$S(E_{\text{Fwf}})=16$, $S(E_{\text{Fmd}})=14$, $S(E_{\text{Fai}})=12$

To reflect the combined influences of different factors on the composition effort, we further define that the scores for different effort factors are accumulable in the same dimension, while they are multipliable across different dimensions. After filling the applicable hypotheses and scores to the classification matrix, we can achieve an effort-estimation-checklist table, as shown in Appendix II. Note that the numbers do NOT indicate any count of the amount of effort. These quantitative scores are only used to facilitate qualitatively contrasting the effort of different composition approaches, as demonstrated in Table VII.

Through Table VII, we can conveniently compare the estimated effort between different Web service composition approaches: one composition approach requires more effort than another does if the former’s effort score is bigger than the latter’s. Moreover, by investigating the result and procedure of calculation of the effort scores, we can find that the amount of applicable hypotheses implies the times of comparisons, while the times of consistent comparisons is proportional to the resulting effort score. Here we regard different comparisons are consistent when the same conclusion can be drawn in these comparisons by applying different hypotheses. For example, there are two consistent comparisons when applying hypotheses H3 and H5' to the compare between Orchestration and Choreography in Table I. Since the consistent comparisons can help to confirm and reinforce the comparison result, the effort scores also reflect the extent of our confidence in the effort estimation result. Therefore, the larger difference between two approach effort scores, the more confidence we will have in the comparison result.

TABLE VII. EFFORT COMPARE BETWEEN DIFFERENT COMPOSITION APPROACHES

Composition Approaches	**Approach Effort Scores**
BPEL Programming	$S(E_{Fwf})\times(S(E_{For})+S(E_{Fsy})+S(E_{Fso})+ S(E_{Fma})+ S(E_{Fst}))$ $=16\times29=464$
Semantic Matching [2]	$S(E_{Fwf})\times(S(E_{Fch})+S(E_{Fse})+S(E_{Fso})+ S(E_{Fsa})+ S(E_{Fst}))$ $=16\times30=480$
SA-REST + Smashup [21]	$S(E_{Fwf})\times(S(E_{For})+S(E_{Fse})+S(E_{Fre})+ S(E_{Fsa})+ S(E_{Fst}))$ $=16\times26=416$
RESTfulBP [28]	$S(E_{Fwf})\times(S(E_{Fch})+S(E_{Fsy})+S(E_{Fre})+ S(E_{Fma})+ S(E_{Fst}))$ $=16\times29=464$
UML + MDA [4]	$S(E_{Fmd})\times(S(E_{For})+S(E_{Fsy})+S(E_{Fso})+ S(E_{Fma})+ S(E_{Fst}))$ $=14\times29=406$
UML + OCL [5]	$S(E_{Fmd})\times(S(E_{For})+S(E_{Fse})+S(E_{Fso})+ S(E_{Fma})+ S(E_{Fdy}))$ $=14\times28=392$
UML + QoS Support [6]	$S(E_{Fmd})\times(S(E_{For})+S(E_{Fse})+S(E_{Fso})+ S(E_{Fsa})+ S(E_{Fst}))$ $=14\times28=392$
UML + IHE framework [22]	$S(E_{Fmd})\times(S(E_{For})+S(E_{Fsy})+S(E_{Fso})+ S(E_{Fma})+ S(E_{Fdy}))$ $=14\times27=378$
Petri Net [23]	$S(E_{Fai})\times(S(E_{Fch})+S(E_{Fse})+S(E_{Fso})+ S(E_{Fau})+ S(E_{Fst}))$ $=12\times28=336$
Interface Automata [11]	$S(E_{Fai})\times(S(E_{For})+S(E_{Fse})+S(E_{Fso})+ S(E_{Fau})+ S(E_{Fst}))$ $=12\times26=312$
AIMO [24]	$S(E_{Fai})\times(S(E_{Fch})+S(E_{Fse})+S(E_{Fso})+ S(E_{Fau})+ S(E_{Fdy}))$ $=12\times26=312$
…	…

In fact, the calculation rule here for counting the effort scores of different Web service composition approaches are mainly inspired by the Addition and Multiplication principles in Combinatorics: (1) We apply an Addition-principle-like method to the effort factors in the Context dimension of the classification matrix, considering that different partial efforts of one Web service composition within different contexts are mutually exclusive, while different contexts are accumulable. (2) We apply a Multiplication-principle-like method to the effort factors across those two dimensions of the classification matrix, considering that the Technology dimension is independent of the Context dimension, and one technique can be used to compose Web services within any combination of contexts. However, this calculation rule still suffers from intuition, and will be further validated and revised through empirical study in our future work.

VI. CONCLUSION

The territory of Web service composition has been researched so broadly that it becomes difficult to analyze and estimate the composition effort by exploring every existing composition approach. However, we are able to deliver a general classification of Web service composition to facilitate the effort estimation work through investigating limited approaches inductively. Unlike existing classification work, this paper proposes an effort-oriented classification matrix of Web service composition through a systematic review. Some of the reviewed composition approaches are then classified according to their published descriptions, as demonstrated in Appendix I. The matrix uses two dimensions, Context and Technology, to classify different compositions. Several pairs of effort-related contexts are selected in the Context dimension, while three technology categories are paralleled in the Technology dimension. Moreover, this paper also builds an effort-estimation-checklist table by applying a set of effort estimation hypotheses to different context types and technology categories that are viewed as different composition effort factors. The combined influences of factor pairs across Context dimension and Technology dimension on the composition effort are also represented in this table. The effort-oriented classification matrix can be used to facilitate exploration and comprehension in the research area of Web service composition, while the effort-estimation-checklist table can be used to facilitate the qualitative effort compare between different composition approaches. Furthermore, based on our current work, some new research opportunities in the Web service composition area can also be identified. For example, the gap between automatic composition at design time and dynamic composition at runtime should be bridged.

Overall, the work described in this paper not only brings a new perspective of classification of Web service composition, but also introduces a new method to compare the qualitatively estimated effort between different composition approaches. The prominent characteristic of the proposed classification matrix is of our primary objective - aiming at the influence on software development effort required for different Web service compositions. As such, the classification matrix is eventually developed into an effort-estimation-checklist table, while the effort-estimation-checklist table should be applied closely with the classification matrix. Our future work is to continue filling this classification matrix and to use the effort-estimation-checklist table to establish the basis of the research into cost and effort estimation for Web service composition.

ACKNOWLEDGMENT

This paper is based on our previous work in collaboration with Jacky Keung and Xiwei Xu. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

REFERENCES

[1] Z. Li, L. O'Brien, J. Keung, and X. Xu, "Effort-Oriented Classification Matrix of Web Service Composition," Proc. the Fifth International Conference on Internet and Web Applications and

Services (ICIW 2010), IEEE Computer Society, June 2010, pp. 357-362, doi: 10.1109/ICIW.2010.59.

[2] E. Sirin, J. Hendler, and B. Parsia, “Semi-automatic Composition of Web Services using Semantic Descriptions,” Web Services: Modeling, Architecture and Infrastructure workshop in ICEIS 2003, Apr. 2003.

[3] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M. Shan, “Adaptive and Dynamic Service Composition in EFlow,” Proc. 12th International Conference on Advanced Information Systems Engineering (CaiSE*00), Springer, Jun. 2000, pp. 13-31, doi: 10.1007/3-540-45140-4_3.

[4] D. Skogan, R. Groenmo, and I. Solheim, “Web Service Composition in UML,” Proc. Eighth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004), IEEE Computer Society, Sept. 2004, pp. 47-57, doi: 10.1109/EDOC.2004.1342504.

[5] J. T. E. Timm and G. C. Gannod, “Specifying Semantic Web Service Compositions using UML and OCL,” Proc. 2007 IEEE International Conference on Web Services (ICWS 2007), IEEE Computer Society, Jul. 2007, pp. 521-528, doi: 10.1109/ICWS.2007.168.

[6] R. Grønmo and M. C. Jaeger, “Model-driven Semantic Web Service Composition,” Proc. 12th Asia-Pacific Software Engineering Conference (APSEC '05), IEEE Computer Society, Dec. 2005, pp. 15-17, doi: 10.1109/APSEC.2005.81.

[7] S. Thone, R. Depke, and G. Engels, “Process-Oriented, Flexible Composition of Web Services with UML”, Proc. Third International Joint Workshop on Conceptual Modeling Approaches for E-business: A Web Service Perspective (eCOMO 2002), Springer, Oct. 2002, pp. 390-401, doi: 10.1007/b12013.

[8] V. R. Chifu, I. Salomie, and E. St. Chifu, “Fluent Calculus-based Web Service Composition — From OWL-S to Fluent Calculus,” Proc. 4th International Conference on Intelligent Computer Communication and Processing (ICCP 2008), IEEE Computer Society, Aug. 2008, pp. 161-168, doi: 10.1109/ICCP.2008.4648368.

[9] S. Mitra, R. Kumar, and S. Basu, “Automated Choreographer Synthesis for Web Services Composition Using I/O Automata,” Proc. IEEE International Conference on Web Services (ICWS 2007), IEEE Computer Society, Jul. 2007, pp. 364-371, doi: 10.1109/ICWS.2007.47.

[10] B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid, “Composing Web services on the Semantic Web”, The VLDB Journal, vol. 12, Sept. 2003, pp. 333-351, doi: 10.1007/s00778-003-0101-5.

[11] S. V. Hashemian and F. Mavaddat, “A Graph-based Approach to Web Services Composition,” Proc. The 2005 Symposium on Applications and the Internet, IEEE Computer Society, Jan.-Feb. 2005, pp. 183-189, doi: 10.1109/SAINT.2005.4.

[12] S. Thakkar, C. Knoblock, and J. Ambite. “A View Integration Approach to Dynamic Composition of Web Services,” Proc. 2003 ICAPS Workshop on Planning for Web Services, AAAI Press, 2003.

[13] J. Rao, P. Küngas, and M. Matskin, “Composition of Semantic Web Services using Linear Logic Theorem Proving,” Information Systems, vol. 31, Jun.-Jul. 2006, pp. 340-360, doi: 10.1016/j.is.2005.02.005.

[14] V. Gehlot and K. Edupuganti, “Use of Colored Petri Nets to Model, Analyze, and Evaluate Service Composition and Orchestration,” Proc. 42nd Hawaii International Conference on System Sciences (HICSS’09), IEEE Computer Society, Jan. 2009, pp. 1-8, doi: 10.1109/HICSS.2009.487.

[15] P. Traverso and M. Pistore, “Automated Composition of Semantic Web Services into Executable Processes,” Proc. Third International Semantic Web Conference (ISWC’04), Nov. 2004, pp. 380-394.

[16] P. Sarang, F. Jennings, M. Juric, and R. Loganathan, SOA Approach to Integration: XML, Web services, ESB, and BPEL in real-world SOA projects. Birmingham: Packt Publishing, 2007.

[17] S. Dustdar and W. Schreiner, “A Survey on Web Services Composition,” International Journal of Web and Grid Services, vol. 1, Aug. 2005, pp. 1-30, doi: 10.1504/IJWGS.2005.007545.

[18] J. Cardoso and A. P. Sheth, Semantic Web Services, Processes and Applications. New York: Springer, 2006.

[19] J. Rao and X. Su, “A Survey of Automated Web Service Composition Methods,” Lecture Notes in Computer Science, vol. 3387/2005, Jan. 2005, pp. 43-54, doi: 10.1007/b105145.

[20] F. Rosenberg, F. Curbera, M. J. Duftler, and R. Khalaf, “Composing RESTful Services and Collaborative Workflows: A Lightweight Approach,” IEEE Internet Computing, vol. 12, Sept.-Oct. 2008, pp. 24-31, doi: 10.1109/MIC.2008.98.

[21] J. Lathem, K. Gomadam, and A. P. Sheth, “SA-REST and (S)mashups : Adding Semantics to RESTful Services,” Proc. First IEEE International Conference on Semantic Computing (ICSC 2007), IEEE Computer Society, Sept. 2007, pp. 469-476, doi: 10.1109/ICSC.2007.94.

[22] R. Anzboeck and S. Dustdar, “Semi-Automatic Generation of Web Services and BPEL Processes - A Model-Driven Approach,” Lecture Notes in Computer Science, vol. 3649/2005, Sept. 2005, pp. 64-79, doi: 10.1007/11538394_5.

[23] V. Valero, M. E. Cambronero, G. Díaz, and H. Macià, “A Petri Net Approach for the Design and Analysis of Web Services Choreographies,” Journal of Logic and Algebraic Programming, vol. 78, May-Jun. 2009, pp. 359-380, doi: 10.1016/j.jlap.2008.09.002.

[24] S. G. H. Tabatabaei, W. M. N. Kadir, and S. Ibrahim, “Semantic Web Service Discovery and Composition Based on AI Planning and Web Service Modeling Ontology,” Proc. IEEE Asia-Pacific Services Computing Conference (APSCC '08), IEEE Computer Society, Dec. 2008, pp. 397-403, doi: 10.1109/APSCC.2008.126.

[25] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau, “HTN Planning for Web Service Composition using SHOP2,” Web Semantics: Science, Services and Agents on the World Wide Web, vol. 1, Oct. 2004, pp. 377-396, doi: 10.1016/j.websem.2004.06.005.

[26] H. Zhao and P. Doshi, “Towards Automated RESTful Web Service Composition,” Proc. IEEE International Conference on Web Services (ICWS 2009), IEEE Computer Society, Jul. 2009, pp. 189-196, doi: 10.1109/ICWS.2009.111.

[27] F. Casati, M. Sayal, and M. Shan, “Developing E-Services for Composing E-Services,” Lecture Notes in Computer Science, vol. 2068/2001, Jan. 2001, pp. 171-186, doi: 10.1007/3-540-45341-5_12.

[28] X. Xu, L. Zhu, Y. Liu, and M. Staples, “Resource-Oriented Architecture for Business Processes,” Proc. 15th Asia Pacific Software Engineering Conference (APSEC 2008), IEEE Computer Society, Dec. 2008, pp. 395-402, doi: 10.1109/APSEC.2008.52.

[29] S. Mosser, “Web Services Composition: Mashups Driven Orchestration Definition,” Proc. 2008 International Conference Computational Intelligence for Modeling Control & Automation, IEEE Computer Society, Dec. 2008, pp. 284-289, doi: 10.1109/CIMCA.2008.96.

[30] Y. Xu, S. Tang, Y. Xu, and Z. Tang, “Towards Aspect Oriented Web Service Composition with UML,” Proc. 6th Int’l. Conf. Computer and Information Science (ICIS 2007), IEEE Computer Society, Jun. 2007, pp. 279-284, doi: 10.1109/ICIS.2007.185.

[31] J. Pathak, S. Basu, R. Lutz, and V. Honavar, “MoSCoE: A Framework for Modeling Web Service Composition and Execution,” Proc. 22nd International Conference on Data Engineering Workshops, IEEE Computer Society, Apr. 2006, pp. x143, doi: 10.1109/ICDEW.2006.96.

[32] C. Pautasso, “RESTful Web Service Composition with BPEL for REST,” Data and Knowledge Engineering, vol. 68, no. 9, Mar. 2009, pp. 851-866, doi: 10.1016/J.DATAK.2009.02.016.

[33] J. Mangler, E. Schikuta, and C. Witzany, “Quo Vadis Interface Definition Languages? Towards a Interface Definition Language for RESTful Services,” Proc. 2009 IEEE International Conference on Service-Oriented Computing and Applications (SOCA ‘09), IEEE Computer Society, Dec. 2009, pp. 1-4, doi: 10.1109/SOCA.2009.5410459.

[34] M. zur Muehlen, J. V. Nickerson, and K. D. Swenson, “Developing Web Services Choreography Standards – the Case of REST vs. SOAP,” Decision Support Systems, vol. 40, no. 1, July 2005, pp. 9-29, doi: 10.1016/j.dss.2004.04.008.

[35] A. A. Lewis, “Web Services Description Language (WSDL) Version 2.0: Additional MEPs,” W3C Working Group Note, June 2007, http://www.w3.org/TR/wsdl20-additional-meps/.

[36] S. Kona, A. Bansal, M. B. Blake, and G. Gupta, “Generalized Semantic-based Service Composition,” Proc. IEEE 2008 International Conference on Web Services (ICWS’08), IEEE Computer Society, Sept. 2008, pp. 219-227, doi: 10.1109/ICWS.2008.118.

[37] T. R. Gruber, “A Translation Approach to Portable Ontology Specifications,” Knowledge Acquisition, vol. 5, no. 2, June 1993, pp. 199-220, doi: 10.1006/knac.1993.1008.

[38] L. Liu and M. T. Özsu, Encyclopedia of Database Systems. New York: Springer, 2010.

[39] T. Globerson, “Mental Capacity, Mental Effort, and Cognitive Style,” Developmental Review, vol. 3, no. 3, Sept. 1983, pp. 292-302, doi: 10.1016/0273-2297(83)90017-5.

[40] J. Cardoso, “How to Measure the Control-Flow Complexity of Web Processes and Workflows,” Workflow Handbook 2005, Lighthouse Point: Layna Fischer, Apr. 2005, pp. 199-212.

[41] G. A. Miller, “The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information,” Psychological Review, vol. 63, no. 2, Mar. 1956, pp. 81-97, doi: 10.1037/h0043158.

[42] C. Francalanci and F. Merlo, “The Impact of Complexity on Software Design Quality and Costs: An Exploratory Empirical Analysis of Open Source Applications,” Proc. 16th European Conference on Information Systems (ECIS 2008), June 2008, pp. 1442-1453, Galway, Ireland.

[43] J. R. Turner and R. A. Cochrane, “Goals-and-Methods Matrix: Coping with Projects with Ill-defined Goals and/or Methods of Achieving them,” International Journal of Project Management, vol. 11, no. 2, May 1993, pp. 93-102, doi: 10.1016/0263-7863(93)90017-H.

[44] A. Camci and T. Kotnour, “Technology Complexity in Projects: Does Classical Project Management Work?,” Proc. Technology Management for the Global Future (PICMET 2006), IEEE Computer Society, vol. 5, July 2006, pp. 2181-2186, doi: 10.1109/PICMET.2006.296806.

[45] K. Dooley, “Organizational Complexity,” International Encyclopedia of Business and Management, M. Warner (ed.), London: Thompson Learning, Oct. 2001, pp. 5013-5022.

[46] N. M. Josuttis, SOA in Practice: The Art of Distributed System Design. Sebastopol: O'Reilly Media, Inc., 2007.

[47] A. Barros, M. Dumas, and P. Oaks, “Standards for Web Service Choreography and Orchestration: Status and Perspectives,” Proc. Business Process Management Workshops, Sept. 2005, pp. 61-74, doi: 10.1007/11678564_7.

[48] K. M. Furulund and K. Moløkken-Østvold, “Increasing Software Effort Estimation Accuracy Using Experience Data, Estimation Models and Checklists,” Proc. Seventh International Conference on Quality Software (QSIC '07), IEEE Computer Society, Oct. 2007, pp. 342-347, doi: 10.1109/QSIC.2007.4385518.

[49] H. Demirkan, R. J. Kauffman, J. A. Vayghan, H. G. Fill, D. Karagiannis, and P. P. Maglio, “Service-Oriented Technology and Management: Perspectives on Research and Practice for the Coming Decade,” Electronic Commerce Research and Applications, vol. 7, no. 4, Dec. 2008, pp. 356-376, doi: 10.1016/j.elerap.2008.07.002.

[50] T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design, Crawfordsville: Prentice Hall PTR, 2005.

APPENDIX I: A SAMPLE OF CLASSIFICATION MATRIX OF WEB SERVICE COMPOSITION

Technology		**Context**										
Category	***Detailed Technique***	***Pattern***		***Semiotics***		***Mechanism***		***Design Time***			***Runtime***	
		Orchestration	***Choreography***	***Syntax***	***Semantics***	***SOAP***	***REST***	***Manual***	***Semi-Auto***	***Auto***	***Static***	***Dynamic***
Workflow-based	BPEL Programming	√		√		√		√			√	
	Semantic Matching [2]		√		√	√			√		√	
	eFlow [3]	√		√		√		√				√
	Bite [20]		√	√			√	√			√	
	SA-REST + Smashup [21]	√			√		√		√		√	
	CSDL [27]	√		√		√		√			√	
	RESTfulBP [28]		√	√			√	√			√	
Model-driven	UML + MDA [4]	√		√		√		√			√	
	UML + OCL [5]	√			√	√		√				√
	UML + QoS Support [6]	√			√	√			√		√	
	UML-WSC [7]	√		√		√		√				√
	UML + IHE framework [22]	√		√		√		√				√
	MD Mashup [29]		√	√			√	√			√	
	UML-AOWSC [30]	√		√		√		√				√
	MoSCoE [31]	√			√	√			√		√	
AI planning	SHOP2 [25]	√			√	√				√	√	
	Petri Net [23]		√		√	√				√	√	
	Situation Calculus [8]	√			√	√				√	√	
	I/O Automata [9] *		√	√	√	√				√	√	
	Rule-based Planning [10]	√			√	√				√	√	
	Interface Automata [11]	√			√	√				√	√	
	Query Planning [12] *	√		√	√	√				√	√	
	Linear Logic Theorem Proving [13]	√			√	√			√		√	
	Colored Petri Net [14]	√		√		√				√	√	
	Model Checking [15]	√			√	√				√	√	
	AIMO [24]		√		√	√				√		√
	Situation Calculus for REST [26]	√			√		√		√		√	

* The approaches in [9] and [12] are independent of the Semiotics context.

APPENDIX II: EFFORT-ESTIMATION-CHECKLIST TABLE FOR WEB SERVICE COMPOSITION

Technology		**Context**										
Category		***Pattern***		***Semiotics***		***Mechanism***		***Design Time***			***Runtime***	
		Orchestration	***Choreography***	***Syntax***	***Semantics***	***SOAP***	***REST***	***Manual***	***Semi-Auto***	***Auto***	***Static***	***Dynamic***
	Applied Hypotheses --------------- ***Score***	H3, H5’ --------------- $S(E_{For})=2$	H3, H5’ --------------- $S(E_{Fch})=4$	H1, H5’ ------------ $S(E_{Fsy})=2$	H1, H5’ ------------ $S(E_{Fse})=3$	H2’, H5’ ------------- $S(E_{Fso})=4$	H2’, H5’ ------------ $S(E_{Fre})=2$	H1, H2’, H5’, H6’ ------------ $S(E_{Fma})=14$	H1, H2’, H5’, H6’ ------------ $S(E_{Fsa})=12$	H1, H2’, H5’, H6’ ------------ $S(E_{Fau})=10$	H1, H2’, H5’, H6’ ------------ $S(E_{Fst})=7$	H1, H2’, H5’, H6’ ------------ $S(E_{Fdy})=5$
Workflow-based	H1, H2’, H4’, H5’, H6’ --------------- $S(E_{Fwf})=16$	H1, H2’, H3, H4’, H5’, H6’ ------------- $S(E_{Fwf})$ $\times S(E_{For})$ $=32$	H1, H2’, H3, H4’, H5’, H6’ ------------- $S(E_{Fwf})$ $\times S(E_{Fch})$ $=64$	H1, H2’, H4’, H5’, H6’ ------------- $S(E_{Fwf})$ $\times S(E_{Fsy})$ $=32$	H1, H2’, H4’, H5’, H6’ ------------ $S(E_{Fwf})$ $\times S(E_{Fse})$ $=48$	H1, H2’, H4’, H5’, H6’ ------------- $S(E_{Fwf})$ $\times S(E_{Fso})$ $=64$	H1, H2’, H4’, H5’, H6’ ------------ $S(E_{Fwf})$ $\times S(E_{Fre})$ $=32$	H1, H2’, H4’, H5’, H6’ ------------ $S(E_{Fwf})$ $\times S(E_{Fma})$ $=224$	H1, H2’, H4’, H5’, H6’ ------------ $S(E_{Fwf})$ $\times S(E_{Fsa})$ $=192$	H1, H2’, H4’, H5’, H6’ ------------ $S(E_{Fwf})$ $\times S(E_{Fau})$ $=160$	H1, H2’, H4’, H5’, H6’ ------------ $S(E_{Fwf})$ $\times S(E_{Fst})$ $=112$	H1, H2’, H4’, H5’, H6’ ------------ $S(E_{Fwf})$ $\times S(E_{Fdy})$ $=80$
Model-driven	H1, H2’, H4’, H5’, H6’ --------------- $S(E_{Fmd})=14$	H1, H2’, H3, H4’, H5’, H6’ ------------- $S(E_{Fmd})$ $\times S(E_{For})$ $=28$	H1, H2’, H3, H4’, H5’, H6’ ------------- $S(E_{Fmd})$ $\times S(E_{Fch})$ $=56$	H1, H2’, H4’, H5’, H6’ ------------- $S(E_{Fmd})$ $\times S(E_{Fsy})$ $=28$	H1, H2’, H4’, H5’, H6’ ------------ $S(E_{Fmd})$ $\times S(E_{Fse})$ $=42$	H1, H2’, H4’, H5’, H6’ ------------- $S(E_{Fmd})$ $\times S(E_{Fso})$ $=56$	H1, H2’, H4’, H5’, H6’ ------------ $S(E_{Fmd})$ $\times S(E_{Fre})$ $=28$	H1, H2’, H4’, H5’, H6’ ------------ $S(E_{Fmd})$ $\times S(E_{Fma})$ $=196$	H1, H2’, H4’, H5’, H6’ ------------ $S(E_{Fmd})$ $\times S(E_{Fsa})$ $=168$	H1, H2’, H4’, H5’, H6’ ------------ $S(E_{Fmd})$ $\times S(E_{Fau})$ $=140$	H1, H2’, H4’, H5’, H6’ ------------ $S(E_{Fmd})$ $\times S(E_{Fst})$ $=98$	H1, H2’, H4’, H5’, H6’ ------------ $S(E_{Fmd})$ $\times S(E_{Fdy})$ $=70$
AI planning	H1, H2’, H4’, H5’, H6’ --------------- $S(E_{Fai})=12$	H1, H2’, H3, H4’, H5’, H6’ ------------- $S(E_{Fai})$ $\times S(E_{For})$ $=24$	H1, H2’, H3, H4’, H5’, H6’ ------------- $S(E_{Fai})$ $\times S(E_{For})$ $=48$	H1, H2’, H4’, H5’, H6’ ------------- $S(E_{Fai})$ $\times S(E_{Fsy})$ $=24$	H1, H2’, H4’, H5’, H6’ ------------ $S(E_{Fai})$ $\times S(E_{Fse})$ $=36$	H1, H2’, H4’, H5’, H6’ ------------- $S(E_{Fai})$ $\times S(E_{Fso})$ $=48$	H1, H2’, H4’, H5’, H6’ ------------ $S(E_{Fai})$ $\times S(E_{Fre})$ $=24$	H1, H2’, H4’, H5’, H6’ ------------ $S(E_{Fai})$ $\times S(E_{Fma})$ $=168$	H1, H2’, H4’, H5’, H6’ ------------ $S(E_{Fai})$ $\times S(E_{Fsa})$ $=144$	H1, H2’, H4’, H5’, H6’ ------------ $S(E_{Fai})$ $\times S(E_{Fau})$ $=120$	H1, H2’, H4’, H5’, H6’ ------------ $S(E_{Fai})$ $\times S(E_{Fst})$ $=84$	H1, H2’, H4’, H5’, H6’ ------------ $S(E_{Fai})$ $\times S(E_{Fdy})$ $=60$