Constructing a Creative Software with Services

Faisal Fahmi [a,b], Pei-Shu Huang [c], Feng-Jian Wang [c], Hongji Yang [d]

[a] EECS Int'l Graduate Program, National Yang-Ming Chiao-Tung University, Hsinchu City, Taiwan
faisalfahmiy@gmail.com

[b] Information and Library Science, Airlangga University, Surabaya City, Indonesia
faisalfahmiy@gmail.com

[c] Dept. of Computer Science, National Yang-Ming Chiao-Tung University, Hsinchu City, Taiwan
{pshuang, fjwang}@cs.nctu.edu.tw

[d] Department of Informatics, University of Leicester, Leicester, England
hongji.yang@leicester.ac.uk

***Abstract*— Service Oriented Architecture (SOA) and Creative Computing can be applied to construct a creative service software by utilizing various domain knowledges, where the software contains a solution that not only effective, but also novel, useful and surprising. In creative computing, several theoretical evaluation metrics and verification approaches have been proposed for system design in several domains. However, a solid methodology for development of creative service software is rarely considered in current researches. In this paper, we propose a method composed of requirement specification and service design phases to develop creative software with SOA, where each phase applies a specification model based on semantic web. Inside the development, the models containing XML structures and the associated directed graphs are constructed in both phases to improve machine readability for automatic information processing in creative computing and reduce communication work among development participants with different knowledges, respectively. The graph models defined also can improve the traceability of the specifications and support machine processing. After the model resulted in the second phase is validated for consistency and completeness, the creative service software is well-constructed and can be implemented and reused effectively. Besides, a real example is adopted to demonstrate the workings of the method.**

Keywords*— *Creative Computing, Semantic Web, Service-Oriented Architecture

I. Introduction

The creativity of computing software has been defined as the properties of the software: novel, astonishing, and useful [1]. Theoretically, its scope of creativity has been claimed to be more powerful than the capability of current Artificial Intelligence (AI) techniques [2, 3, 4], where an AI-based software commonly works deeply on a specific domain only, and a creative computing works on broader domain to create a new domain which is the combination of two different domains, or the transformation or exploration of existing domains. Applying creative computing, the software constructed can also be self-adaptive for its creativity. There are many theoretical results for creative computing [5, 6,7]. However, it is still lack of a practical result built based on current software techniques such as, object-oriented languages and Service-Oriented Architecture (SOA).

Recently, SOA is one of the most acceptable software development techniques, due to the extensive open libraries and better integration capability of web-based services [8]. In this paper, we developed a framework to help constructing a creative service-based software with semantic web. Our framework of creative software development contains methodology of two phases, domain-creative requirement specification and semantic-based service design, and their associated specification models.

In the first phase, the participants are domain experts, familiar with this application domain, and software engineers, familiar with creativity, semantic web, and SOA. The first step in this phase is to construct a primary (non-creative) requirements specification, defined by both of them. In the next step, software engineers add the creativity properties into the specifications and the acceptability of this part are validated by the domain experts. This step works iteratively until all the creativities are accepted (i.e., consistent and complete) by both participants.

In the second phase, the participants are software engineers. The first step of this phase is to filter out system features and functions in the requirement specifications that will not implemented as services, e.g., functions that should be performed by legacy systems. The next step is an iterative service design for a domain-aware layered service model to improve services reusability and integration [9]. After each service and the associated service composition constructed, including the external services and legacy system (if any), are validated for the consistency and completeness, service design specifications can be implemented effectively.

Besides, a practical example for each phase is introduced to demonstrate feasibility of the method. However, our work does not concern software implementation, testing, and maintenance parts still. Our next work is to complete the implementation with SOA, in order to find the characteristics of constructive software developed based on our approach. There are several issues can be discussed further. For example, a static anomaly detection method [10] can be applied during the specification to help remove the redundancy.

The rest of this paper is organized as follows: In Section II, the background introduces creative computing, SOA, and semantic web. In Section III, we introduce specification models base on semantic web used for development of creative service software. In Section IV, we present the series of works need to be done and their expected results in the requirement and design phases respectively. Section V gives demonstration with a practical example. Finally, Section VI gives a conclusion.

II. Background

A. Creative Computing

The software construction applying creative computing has been discussed widely in recent years, targeted to produce novel, innovative, and valuable software [11]. Creative computing attempts to consolidate the objectivity of computer system with the subjectivity of human creativity resulting a creative algorithm that solve practical problems, where its computing has all creativity factors: fresh, surprising, and useful [1].

It is observed that new knowledge created through creative ways can be used to innovate new products or services in various domains. For some products or services in a domain, an innovative product or service tends to be more successful in the competitive market than the others [6]. Besides, there are three creative ways to build new knowledge: exploratory, combinational, and transformational [2]. For the exploratory way, new knowledge can be found by researching an existing conceptual space. For the transformational way, new knowledge can be created by transforming an existing conceptual space into another. For the combinational way, new knowledge can be created by combining similar ideas, thoughts, or knowledge.

Applying creative computing, the computations for the represented knowledge can be defined as an inference engine or machine learning algorithm, based on exploratory, transformational, and/or combinational creativities, to create a novel, surprising, and useful solution for a given problem. However, the current approaches of creative computing [5, 6] are only applied on regular application development which is not concerned with the reuse of existing services or microservices to improve the software development and its performance.

B. *Service-Oriented Architecture*

Service-Oriented Architecture (SOA) promotes the reuse or creation of services which are applied intra- and inter-enterprise to improve enterprise agility, ability to respond to a change (industry change). Services designed with SOA commonly have the following characteristics: interoperable, loosely coupled, reusable, composable, and autonomy [8]. Although SOA technology provides several benefits in service design, the implementation and maintenance are complex for small scale organizations. Microservice technologies are further introduced to improve the performance and maintainability of an application, where a microservice has additional characteristics: bounded by context, smaller in size, and operationally independent.

However, the models of service structure used in current software development methodologies with SOA [8, 9, 12, 13] commonly separate the service description and service composition (or integration) which enable an automatic engine for service discovery to be created but introduce complexity. Besides, it also becomes more complex to review the completeness if both descriptions are separated. In this paper, we develop a model based on Semantic Web to help designing services or microservices of a creative software with SOA.

C. *Semantic Web and Its Usages*

The web, i.e., World Wide Web (WWW), is an information system where documents and other web resources are identified by Uniform Resource Locators (URLs, such as https://example.com/resource) and accessible over the Internet [14]. The Semantic Web [15] improve the representation of data relation in conventional web to support better readability of machine, data integration, navigation, and automation of tasks. The Semantic Web [16] provides a common framework allowing data and link(s) between data to be shared and reused across application, enterprise, and community boundaries through a web.

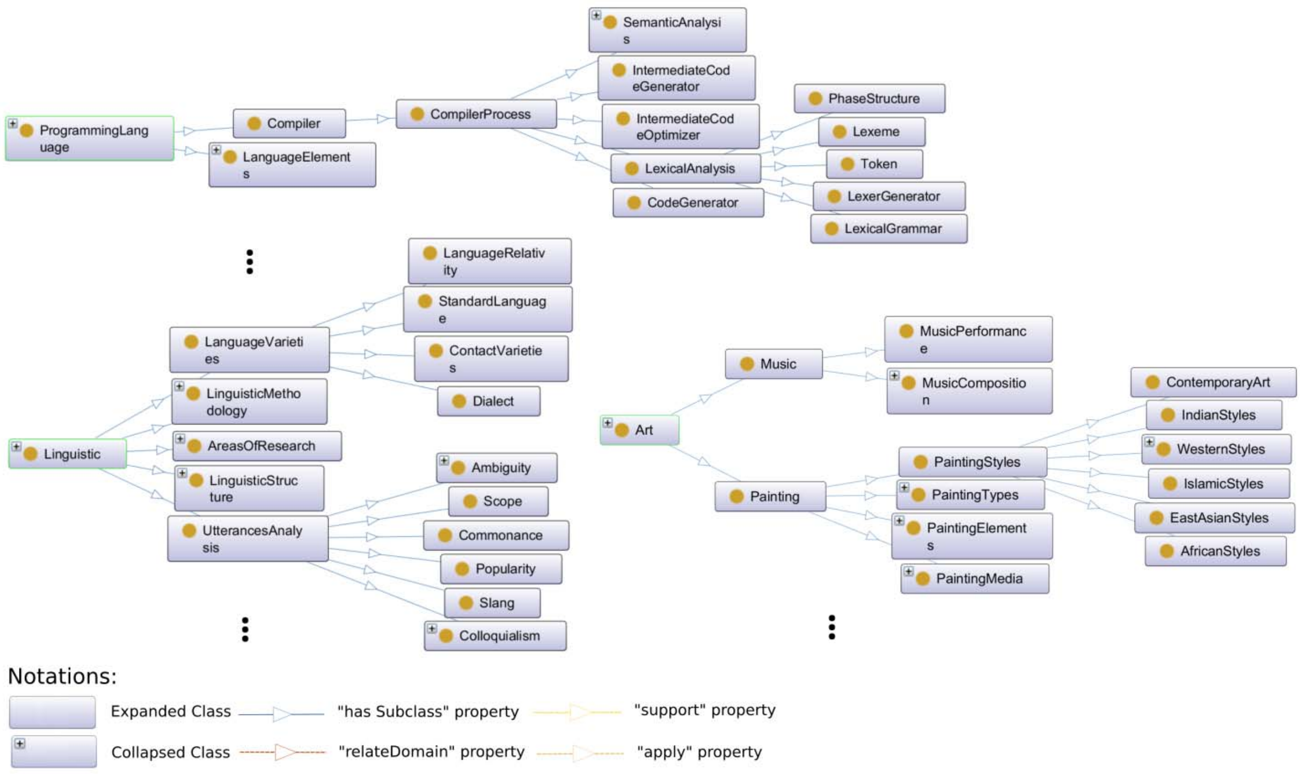


Fig. 1: Parts of knowledge represented in Semantic Web using Protégé

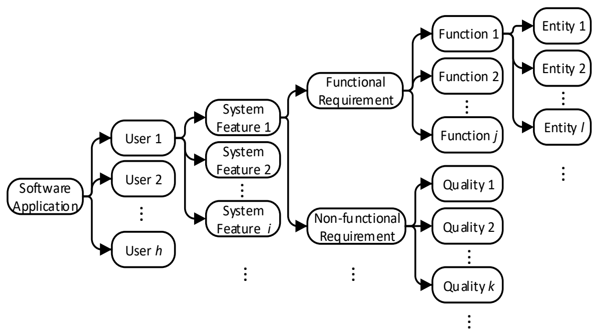


Fig. 2: A directed graph for requirement specification

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:ars="ApplicationRequirementSpecifications.owl">
<ars:SystemUser rdf:about="User 1">
   <ars:SystemFeature>Feature 1</ars:SystemFeature>
   <ars:SystemFeature>Feature 2</ars:SystemFeature>
   … (continue for SystemFeature)
   <ars:SystemFeature>Feature i</ars:SystemFeature>
</ars:SystemUser>
<ars:SystemUser rdf:about="User 2">
   … (SystemFeature accessed by User 2)
</ars:SystemUser>
… (continue for SystemUser)
<ars:SystemUser rdf:about="User h">
   … (SystemFeature accessed by User h)
</ars:SystemUser>
</rdf:RDF>
```

Fig. 3: An example XML structure describing parts of requirement specification in Fig. 2

The architecture of Semantic Web contains seven layers named as identifier and character set layer, syntax layer, data interchange layer, ontology, query and rule layer, logic layer, proof layer, and trust layer from bottom to top, where each layer utilizes the capabilities of the layers below. There are two core standards to construct a document of Semantic Web: Resource Description Framework (RDF) in data interchange layer and vocabularies representation language, e.g., Web Ontology Language (OWL) and Resource Description Framework Schema (RDFS) in ontology, query, and rule layer.

In software development, Semantic Web has been studied to be used as knowledge representation and data modeling when constructing creative software [17], where the knowledge representation is used to construct creativity resource and the data modeling is used to generate creativities automatically.

Using Semantic Web, a knowledge can be represented as a set of RDF triples (subject, predicate, object), where a given (subject, object) pair represents a certain resource and its attribute respectively, and the corresponding predicate represents a relationship between them [18]. For instance, Fig. 1 shows a collection of various knowledges represented in Semantic Web and modified from a Knowledge Base (KB) in [6], where a rectangle represents a subject or object and an arc represents a predicate in RDF. However, such a knowledge representation is generally too complex to help the software development because they contain too many vocabularies unrelated to the development at least.

For the data modeling, semantic web can be used to model requirement specifications and service structures that can effectively utilize any knowledges represented in Semantic Web during the development of creative service-based software [17]. The data model of requirement specification [17], called DMRS, consists of directed graph and XML structure constructed based on specialized semantic web with specific ontology and RDF, where the ontology contains common terms for requirement specifications in scenario-based method [19] and the RDF is used to describe the properties of better requirement specifications based on that method. Fig. 2 shows a directed graph describing a structure of requirement specifications in DMRS and Fig. 3 shows the XML structure for parts of the directed graph in Fig. 2. The directed graph and XML structure can be transformed into each other since they are constructed based on the same ontologies and RDF. The directed graph is used to simplify the discussion works between domain experts and software engineers. The XML Structure is used by a machine of creative computing to add creativity into the DMRS by processing the knowledges represented in KB, where the result is called Creative-DMRS.

Similar to DMRS, the data model of service structure [17], called DMSS, also consists of directed graph and XML structure based on specialized semantic web with specific ontology and RDF, where the ontology contains all terms used commonly for service structure and communication mechanism between services [13], and the RDF is used to describe the properties of a layered services model, e.g., service layer and service composition. Fig. 4 shows a directed graph for the service structure with three layers architecture in DMSS, where the directed-arcs indicate the communication from the tail to the

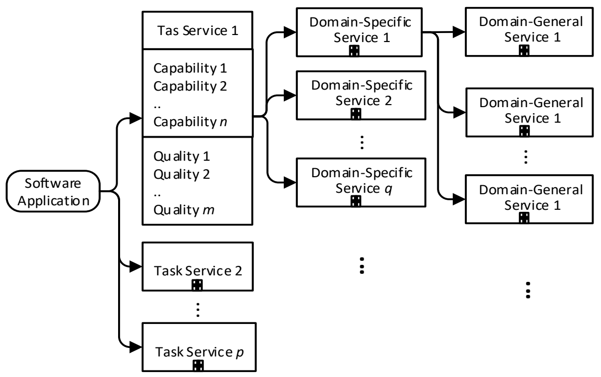
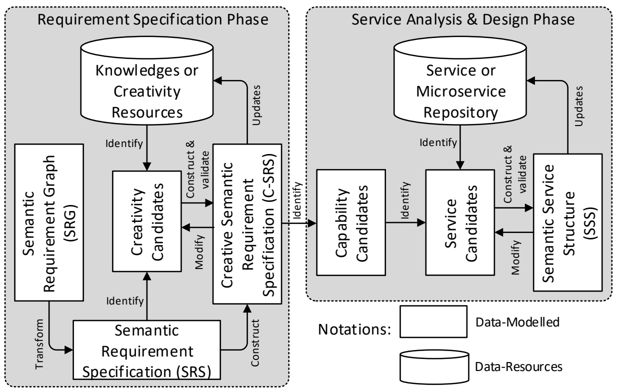


Fig. 4: A directed graph for service structure in DMSS

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:schema="http://schema.org/"
   xmlns:ssm="ServiceStructure.owl">
<schema:SoftwareApplication rdf:about="Application Name">
   <rdfs:label>Application Description</rdfs:label>
   <ssm:ServiceComposition>
      <ssm:TaskService rdf:about="Task Service 1">
         <rdfs:label>Description of Task Service 1</rdfs:label>
         <ssm:QualityAssuranceList> … </ ssm:QualityAssuranceList>
         <ssm:ServiceCapabilityList> … </ ssm:ServiceCapabilityList>
         <ssm:ServiceComposition> … </ssm:ServiceComposition>
      </ssm:TaskService>
      <ssm:TaskService rdf:about="Task Service 2"> … </ssm:TaskService>
      … (continue for TaskService)
      <ssm:TaskService rdf:about="Task Service p"> … </ssm:TaskService>
   </ssm:ServiceComposition>
</rdf:RDF>
```

Fig. 5: An example XML structure describing parts of service structure in Fig. 4


Requirement Specification Phase
Service Analysis & Design Phase
Knowledges or Creativity Resources
Updates
Identify
Construct & validate
Semantic Requirement Graph (SRG)
Creativity Candidates
Creative Semantic Requirement Specification (C-SRS)
Modify
Identify
Transform
Semantic Requirement Specification (SRS)
Construct
Service or Microservice Repository
Updates
Identify
Capability Candidates
Identify
Service Candidates
Construct & validate
Semantic Service Structure (SSS)
Modify
Notations:
Data-Modelled
Data-Resources
Fig. 6: Structure of DDCS

head and their response (if any), and Fig. 5 shows for the XML structure for parts of the directed graph in Fig. 4. Since each term in DMSS and DMRS is defined clearly, software engineers can construct a DMSS from a Creative-DMRS effectively without losing the creativity properties. However, the data to be processed during the construction of Creative-DMRS and DMSS when developing a creative software may be too big, e.g., the knowledges represented in KB can be too many. In this paper, we introduce two models, called as creativity candidate and service candidate, to reduce the data to be processed during the construction of Creative-DMRS and DMSS respectively, where the consistency and completeness of the specification can be maintained effectively.

III. Data Models based on Semantic Web

In this paper, we extend the data models based on semantic web to develop creative service software from [17], called Data-model for Development of Creative Service-software (DDCS), as shown in Fig. 6. The detailed contents of DDCS and their usage are described in Subsections A and B, respectively.

A. *Data Model for Requirement Specifications*

1) *Semantic Requirement Graph (SRG)*

We adopt the directed graph of DMRS [17] shown in Fig. 2, called SRG, containing the users and their expected system features of which each represents some distinct functionalities of the system. There are six layers in SRG, which are named from high to low as follows: application, user, system feature, functional and/or non-functional requirements, function and/or quality, and entity layers, where each layer may contain one or more nodes. A node in each layer can be defined by a distinct name associated with descriptions. A directed arc from a node to one in a next (lower) layer represents a relationship between both nodes. In SRG, the vocabularies for each node and the relationship(s) associated with all output arc(s) are defined in Application Requirement Specification (ARS) ontology [17]. Besides, the usage of ontology can improve the traceability of the specifications [19].

2) *Semantic Requirement Specification (SRS)*

Different from common AI techniques that rely on deep-knowledge perspective in a specific domain to generate effective solution, creative computing techniques rely on broad-knowledge perspective to generate a new knowledge, as a solution of a given problem, which is novel, surprising, and useful by processing the input of existing knowledges. The creative computing technique may need to read and process a large number of knowledges represented from different domains to result better creativities. In this paper, we adopt the XML Structure of DMRS [17] shown in Fig. 3, called SRS, to simplify the knowledge acquiring process during the execution of creative computing algorithms.

3) *Knowledge and Creativity Resources (KCR)*

KCR contains a set of existing knowledge-based ontologies, represented in XML structure applying semantic web standards, used for the creativity resources of the computation in creative computing. Each element can contain a concept, properties of this concept, and the relationships between the concept and properties, where: (1) a property can be either an attribute, instance, or sub-concept, (2) a concept may have zero or more than one attributes, instances, or sub-concepts, and (3) each sub-concept can have its own relationships and properties recursively, as shown in Fig. 7. There are two types of relationships for a concept: direct and indirect. The direct one is a relationship represented by the concept's output arc whose target is its property. The indirect one is a relationship represented by an output arc of its sub-concept, where the target of this arc is the property of the sub-concept.

4) *Creativity Candidates*

KCR may contain some knowledges that are not appropriate when used as creativity resources, according to the context and constraint of the application domain. The set of creativity candidates is a subset of KCR, of which each candidate contains the elements of two levels, where level 1 contains one concept or sub-concept in KCR which is similar to the concepts in application domain and level 2 contains a set of properties and direct relationship of the former concept in KCR, as shown in Fig. 8. The application domain AD can be derived from system features and their functional requirements in SRS, where the technology-centric elements can be filtered out. An example of technology-centric requirement can be a function to access a database management system, email notification, etc.

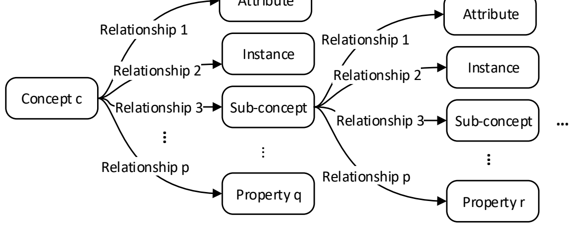


Fig. 7: Structure of each element in KCR

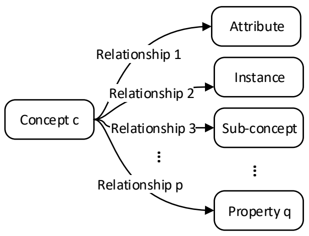


Fig. 8: Structure of creativity candidate

5) Creative Semantic Requirement Specification (C-SRS)

Since SRS is constructed in XML structure with semantic web standards, a machine of creative computing can be used to automatically or semi-automatically generate creative requirement specifications in SRS model, called C-SRS. The candidates of C-SRS can be constructed by iteratively merging the functional requirements in SRS and creativity candidates. A candidate with the highest creativity score [5] and above the user-defined threshold can be selected as C-SRS.

B. Data Model for Service Structures

1) Capability Candidates

The functionalities associated with a service can also be named as service capabilities. Capability candidates can be identified from AD and its associated quality requirements (if exist) in C-SRS. The capability candidates identified can be further decomposed into functions with finer granularity to increase the reusability [8].

2) Service Candidates

A service candidate contains a group of capability candidates with a defined context [8]. A service-based software may be composed of more than one service or microservices, of which each may have different contexts. To reduce the management complexity, we adopt a layered model for the services, called Domain Layered Service Model (DLSM), to separate the services based on the context of reusability levels. DLSM contains three (logical) layers: Task, Domain-Specific, and Domain-General Service layers from top to bottom as shown in Fig. 9, where a service in a higher layer can be composed of services in lower layer(s) and/or external services. Task Service Layer contains services with non-reusable context that corresponds to single-purpose business process logic. A task service can encapsulate service compositions that invoke other services to complete its capabilities. Domain-Specific Service Layer contains reusable services applied in a specific domain only. Domain-General Service Layer contains reusable services that can be requested from different domains. It can encapsulate low-level technology-centric functions, such as notification, logging, and security processing.

In Fig. 9, the white arrows indicate the allowed access between services in different layers, where services in the same layer can communicate with each other. The black arrows indicate the allowed access to external services or legacy systems, where Message/Event Broker works to transform and route a message or event that can be in different formats or protocols between the sender and receiver with both static or dynamic network address. A service model from the directed graph of DMSS [17], shown in Fig. 4, is adopted to model a service candidate in DLSM, where each service candidate is simplified to contain a set of capabilities and the quality requirements associated with these capabilities (if any) without concerning the runtime environment such as deployment machine, hardware and software infrastructures, etc.

3) Service or Microservice Repository (SMR)

To speed up the development, a service can be constructed by composing the capabilities from the existing services. SMR contains the description of existing services and their associated contract including the access method and quality assurance. SMR can be used to identify capability candidates which can be replaced with the capability provided by existing services. The existing services from the Domain-Specific and Domain-General layers of a service software applying DLSM can be adopted to improve SMR since the services are reusable.

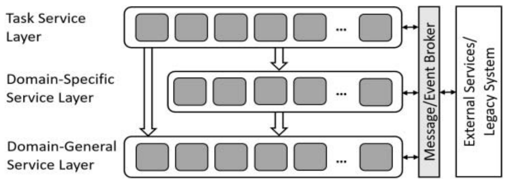

Task Service Layer
Domain-Specific Service Layer
Domain-General Service Layer
...
Message/Event Broker
External Services/ Legacy System

Fig. 9: Domain Layered Service Model (DLSM)

4) Semantic Service Structure Model (SSS)

Jolie [20] is claimed as a service-oriented programming language which combines service description and its implementation to simplify the development. However, a service modeled in Jolie can increase the complexities since the service also contains its implementation details. In this paper, we adopt a model, similar to Jolie but without the implementation details, from XML Structure of DMSS [17], called Semantic Service Structures (SSS), to model a service structure in DLSM containing its capabilities and the service composition of these capabilities (if any), as shown in Fig. 5. Since SSS is an XML Structure with semantic web standards and machine-readable, an automatic machine for service discovery or selection can be constructed (if necessary).

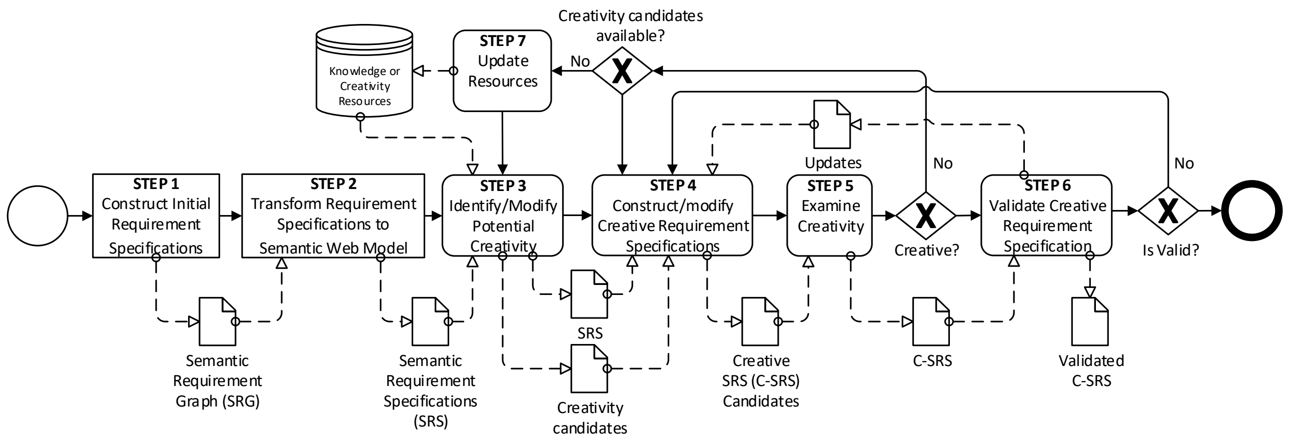
In SSS, a service is described to contain a set of capabilities, of which each is described by message mechanism, endpoint address, and input and/or output message. The message in the communication can be exchanged through SOAP, REST, or Event-Driven to support both orchestration and choreography service integration methods [13]. The operation in the communication can be a one-way or request-response, where in a one-way communication, a capability only contains input message, and in a request-response one, a capability contains both input and output messages. If a capability needs to invoke other services, in the lower layers or external services, to complete its operations, the service composition contained series of services to be invoked are described within the description of that capability.

IV. A Method of Constructing Creative Service Software

In this section, we present a method to construct requirement and design specifications for creative service software using DDCS without concerning the implementation, testing, deployment, and maintenance. The method contains two phases: 1) domain-creative requirement specification and 2) semantic-based service analysis and design. For each phase, there is a distinct method designed to derive the specification, where the consistency and completeness of the specification can be maintained effectively.

A. Domain-Creative Requirement Specification

The work in the first phase include two parts: (1) identify initial requirement specifications (non-creative) and (2) an iterative work to construct and validate creative requirement specifications, with semantic web, based on the initial one. The specification constructed in this phase contains users, system



Creativity candidates available?
No
STEP 7 Update Resources
Knowledge or Creativity Resources
STEP 1 Construct Initial Requirement Specifications
STEP 2 Transform Requirement Specifications to Semantic Web Model
STEP 3 Identify/Modify Potential Creativity
STEP 4 Construct/modify Creative Requirement Specifications
Updates
STEP 5 Examine Creativity
No
Creative?
STEP 6 Validate Creative Requirement Specification
No
Is Valid?
Semantic Requirement Graph (SRG)
Semantic Requirement Specifications (SRS)
SRS
Creativity candidates
Creative SRS (C-SRS) Candidates
C-SRS
Validated C-SRS

Fig. 10: Activities of domain-creative requirement specifications

features, functional requirements with the entities accessed or manipulated, and non-functional requirements. Fig. 10 shows activities of this phase, where Steps 1 and 2 (rectangle) contain the activities in Part (1) and iterations of Steps 3~7 (rounded rectangle) describe the activities in Part (2).

Step 1: Construct Initial Requirement Specification. In this step, software engineers need to discuss with users or domain experts to understand the application domain, where the requirement specifications are constructed by software engineers based on the viewpoints of users or domain experts. To construct better requirement specification, software engineers also need to analyze documentations of the corresponding legacy system (if any).

Initially, the requirements are written in natural language to be understood and validated by the users or domain experts effectively. The requirements include goal(s) of the software, scope of the problem, and interaction between the users and the desired system. Each problem that its topic is not in the scope of software or it cannot be implemented inside a software is filtered out, e.g., it associated with manual business process.

To reduce the communication work, SRG is adopted to construct requirement specification containing the users and their expected system features of which each represents some distinct functionalities of the system. The software engineers discuss with the users or domain experts and construct detailed scenarios for each system feature. A scenario describes a list of events defining the interactions between the system and the users who can be a human or an external system, such as hardware, service, microservice, etc. A system feature is described by name, users, scenarios, and quality requirements [19], where scenarios are used to derive functional requirements and the entity(ies) accessed or manipulated by the functions, and quality requirements are used to derive non-functional requirements. If there exist non-functional requirements conflicting with each other, they need to be prioritized for the consistency during the design of the system.

Step 2: Transform Requirement Specifications to Semantic Web Model. To enable a machine of creative computing understand the requirement specification quickly, the input of SRG is transformed to Semantic Web model, where the model adopted is SRS. An algorithm from [21] is employed to transform SRG into SRS model.

Step 3: Identify or Modify Potential Creativity. In this step, Creativity Candidates are discovered from the knowledges represented in KCR and AD, where AD is derived from the input of SRS. Algorithm 1 is defined to discover creativity candidates based on syntactic and semantic similarities between the concepts in KCR and AD, where syntactic similarity computes the distance between two strings and semantic similarity computes the similarity of the meaning between two strings. Line 1 resets the Creativity Candidates discovered in previous iteration (if any). In Line 4, α and β are normalized weights for syntactic and semantic similarities respectively, where $\alpha + \beta = 1$. In Line 5, the threshold is defined by user. The label in Lines 8 and 10 is used to recognize a creativity candidate similar to which concept(s) in AD.

Algorithm 1: Discover Creativity Candidates
Input: AD A, KCR K, threshold
Output: Creativity Candidates CC

```
1   Clear CC
2   For each concept ca in A
3     For each concept ck in K
4       aggregate similarity = α × syntactic similarity(ca, ck) + β × semantic
            similarity(ca, ck)
5       If aggregate similarity > threshold then
6         new creativity candidate nc = ck and its direct relationship and properties
7         If nc exist in CC then
8           Add new label of ca into nc I nside CC
9         Else
10          Add label of ca into nc
11          Put nc into CC
12        End If
13      End If
14    End For
15  End For
16  Return CC
```

Step 4: Construct or Modify Creative Requirement Specifications. This step works to construct C-SRS candidates by iteratively merging the functional requirements inside the input of SRS with the selected properties of concepts inside the input of Creativity Candidates, where the merging options are discovered by the machine of creative computing. The machine is constructed based on inference engines or machine learning algorithms applying explorational, transformational, and/or combinational creativities employed from [2] and [22], respectively. For each iteration, the C-SRS candidate is modified, based on different merging options, to be examined for its creativity in Step 5.

The inference engines [2] adopted for the machine of creative computing contain 12 inference rules, where Rules 1~6 are used for explorational creativity, Rules 7~9 are used for transformational creativity, and Rules 10~12 are used for combinational creativity. For instances, in Rule 7, for an input concept C or property of concept p, if the following conditions are met, its reasoning result is another concept D. The conditions include: (1) Input D is not C, (2) D is not a sub-concept of C, (3) D is not an empty, and (4) If an input is an instance, p ∈ C.

Algorithm 2 contains the work for the machine of creative computing based on machine learning algorithms [20]. Since a concept in the input of AD only similar to some concepts in the input of CC, to maintain the relevancy of creativity resulted, Lines 2-6 select the concepts in CC similar to the current concept in AD to be processed. The merging_option in Lines 8 and 11 only applies specific creativity techniques to simplify the calculation because the merging is done between functional requirements and selected properties of concepts. In Line 8, the exploration creativity technique adopts Self-Organizing Map (SOM) algorithm to categorizes creativity candidates, where each group of creativity candidates with the same categorize is defined as a distinct merging option. For the transformation creativity technique in Line 8, Principle Component Analysis (PCA) is adopted to discover the merging options from the concepts in creativity candidates with the highest and lowest similarities to a functional specification of the input system feature, where this functional requirement has the biggest variation of similarity to the creativity candidates (Principal Component). In Lines 8 and 11, the combination creativity technique adopts Apriori algorithm to select the most and least similar concepts among the creativity candidates to construct merging options. The highest similarity in transformational and combinational techniques resulting the merging options with better usefulness property of creativity. On the other hand, the least similarity resulting those with better novelty and surprising properties.

```
Algorithm 2: Construct Creative Requirement Specification
Input: AD A, CC C
Output: Candidates of C-SRS CR
1   For each concept ca in A
2     For each creativity candidate cr in C
3       If label of cr = ca then
4         Put cr into similar_concept
5       End If
6     End For
7     If ca is a system feature then
8       merging_option = combinational(similar_concept) ∪ transformational(ca,
                 similar_concept) ∪explorational(ca, similar_concept)
9       Put the result of merge between ca's properties and merging_option into CR
10    Else //ca is a functional requirement
11      merging_option = combinational(similar_concept)
12      Put the result of merge between ca and merging_option into CR
13    End If
14  End For
15  Return CR
```

Step 5: Examine Creativity. To examine the creativity of C-SRS candidate inputted, the creativity scores are measured based on the creativity factors: novel, surprising, and useful, where the novel factor includes originality and paradigm relatedness, the surprising factor includes unexpectedness and unusualness, and the usefulness factor includes relevance, acceptability, implementability, implicational explicitness, and completeness [5]. The C-SRS candidate with the highest creativity score and above the user defined threshold is selected to be the C-SRS.

Step 6: Validate Creative Requirement Specifications. In this step, domain experts and software engineers check the completeness and consistency of the input of C-SRS in term of users, system features, functional requirements and entity(ies) manipulated or accessed by the functions, and non-functional requirements. If any error is found, go back to Step 4, perform necessary updates, and continue.

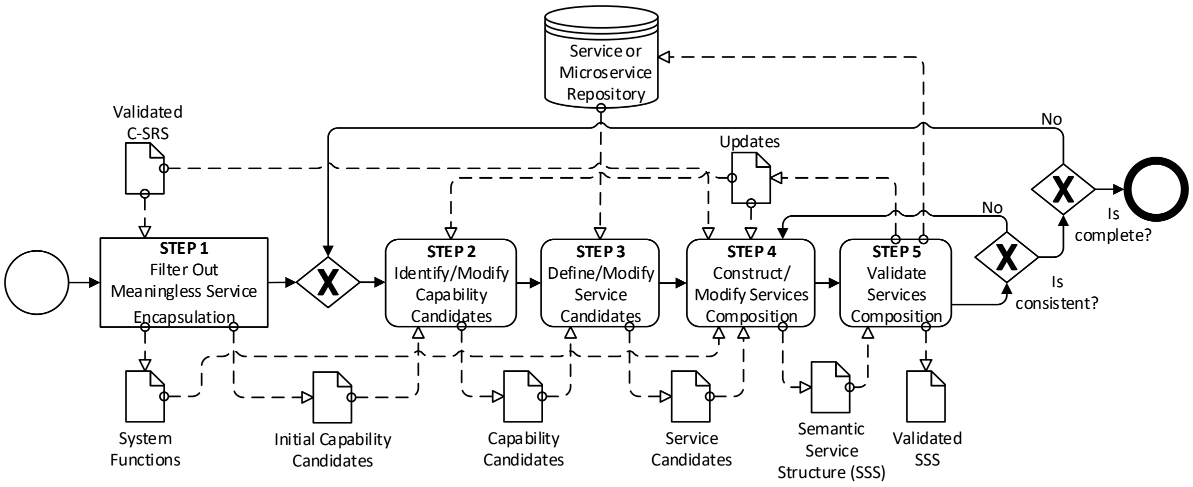
Step 7: Update Resources (if necessary). If all C-SRS candidates cannot pass the examination of creativity in Step 5, KCR needs to be updated with new knowledge extracted from textual documents with the method defined in [2]. This method contains four steps: sentence abstraction, vocabulary abstraction, domain vocabulary extraction, and ontology mapping. Then, the domain ontology extracted can be added to KCR. Other methods are adopted to extract new knowledge from different domains, including service descriptions [23] and web documents [24, 25].

B. *Semantic-Based Service Analysis and Design*

The works in the second phase include two parts: (1) filtering out system features and functional requirements that will not be implemented as services and (2) iterative work on (detailed) service analysis and design. Since a service software is composed of multiple services or microservices, the SSS model defined in Section III, is adopted to reduce the management of reusability. The design specifications derived in this phase contains services definition and composition in SSS model, where each service definition contains service capabilities and quality requirements. There are five steps of activities in this phase, as shown in Fig. 11, where Step 1 (rectangle) is an activity for Part (1) and iterations of Steps 2~5 (rounded rectangle) contains the activities in Part (2).

Step 1: Filter Out Meaningless Service Encapsulation. The requirement specification elements in the input of validated C-SRS contain system features, functional requirements, and their associated quality requirements. These elements are divided into two categories: 1) Those that will be implemented as services are categorized into Initial Capability Candidates and 2) the rest are named as System Functions. Such a categorization is calculated based on its benefits and drawbacks toward the goals of the desired software [26]. For example, if the goal includes performance, the requirement specification whose implementation needs low latency real-time communication, i.e., strictly enforced response times, it is meaningless to be encapsulated within a service.

Step 2: Identify or Modify Capability Candidates. In this step, the functionalities to be implemented as the capabilities of services in the desired software, are identified. There are two inputs for this step, i.e., Initial Capability Candidates and Updates. To increase the reusability, the Initial Capability Candidates are decomposed recursively if they are too coarse. The granularity of the candidates represents the operational scope of a distinct function, i.e., a fine-grained function usually has less work to be completed than a coarse-grained one during the design and implementation [13]. The modification of Capability Candidates is based on the Updates, which are


Service or Microservice Repository
Validated C-SRS
Updates
No
No
Is complete?
Is consistent?
STEP 1 Filter Out Meaningless Service Encapsulation
STEP 2 Identify/Modify Capability Candidates
STEP 3 Define/Modify Service Candidates
STEP 4 Construct/ Modify Services Composition
STEP 5 Validate Services Composition
System Functions
Initial Capability Candidates
Capability Candidates
Service Candidates
Semantic Service Structure (SSS)
Validated SSS
Fig. 11: Activities of semantic-based service analysis and design

derived in Step 5 and contain the System Functions to be accessed for service composition, and the capability candidates used to fix the error of completeness, if required. System Functions existing in the Updates indicate to adopt one or more capability candidates to handle the communication to these System Functions. For instance, a capability based on circuit-breaker pattern [13] is used to prevent a long waiting time for a failure response.

Step 3: Define or Modify Service Candidates. In this step, the service candidates for each layer of DLSM, are specified to contain a set of capabilities and their quality requirements (if any). The elements in the input of Capability Candidates are grouped based on one or more logical contexts of which each represents a service candidate. The contexts defined depend on the layer of the services in DLSM being concerned. For example, the layer of task services requires that a service has a context specific to a business process of an enterprise or organization, and the layer of domain-general services requires that a service has a multi-purpose context applied on several domains. The Service Candidates are modified for each iteration if the content of the input updated. The Capability Candidates replaced with those provided by the existing services or microservices stored in SMR are also identified in this step to increase the reusability.

Step 4: Construct or Modify Services Composition. In this step, the most common interactions are identified among the capabilities inside the input of Service Candidates, including those provided by existing services or microservice (if any), and System Functions derived in Step 1. The interactions are discovered based on the success and failure scenarios of possible activities sequences derived from the relationships among system features and functional requirements inside the input of Validated C-SRS. The next work is to construct Service Composition by generalizing all interactions discovered. The Updates from Step 5 are used to modify Service Composition, where the Updates contain the redundant or conflicted Capability Candidates (i.e., consistency error) to be fixed. Similar to Service Candidates, the Service Composition is modified for each iteration if the content of the input updated. Finally, the Service Composition derived is used to derive SSS.

Step 5: Validate Service Composition. In this step, software engineers validate the completeness and consistency of the services definition and composition in DLSM from the input of SSS model. If any error is found, go back to the corresponding Step, perform necessary updates, and continue. For instance, in the service composition, if a capability of task or domain-specific service has a conflict with one or more capabilities of domain-general services, the former capability is removed to increase reusability. After validation succeeds, the validated SSS is stored into SMR to help the development of creative service software and increase reusability in the future.

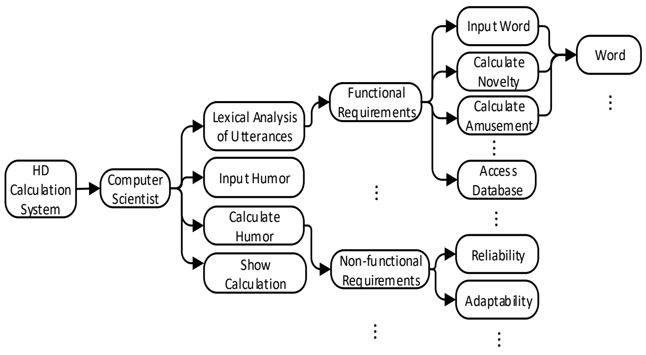


Fig. 12: An example SRG of HD Calculation System

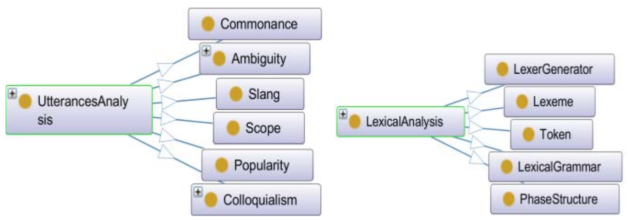


Fig. 13: Creativity candidates discovered from KCR in Fig. 1

```
1  <?xml version="1.0"?>
2  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
4    xmlns:schema="http://schema.org/"
5    xmlns:rsm="ApplicationRequirementSpecifications.owl">
6  <schema:SoftwareApplication rdf:about="HD Calculation System">
7    <rdfs:label>An application to describe how humorous an item is</rdfs:label>
8    <rsm:UserList>
9      <rsm:SystemUser rdf:about="Computer Scientist">
10       <rsm:SystemFeature>Lexical Analysis of utterances</rsm:SystemFeature>
11       <rsm:SystemFeature>Input humor</rsm:SystemFeature>
12       <rsm:SystemFeature>Calculate Humor</rsm:SystemFeature>
13       <rsm:SystemFeature>Show Calculation</rsm:SystemFeature>
14     </rsm:SystemUser>
15   </rsm:UserList>
16   <schema:featureList>
17     <rsm:SystemFeature rdf:about="Lexical Analysis of utterances">
18       <rdfs:label>A feature to measure a value related to a word</rdfs:label>
19       <rsm:FunctionalRequirementList>
20         <rsm:SystemFunction rdf:about="Input word">
21           <rsm:SystemEntity type="input">Word</rsm:SystemEntity>
22           ... (continue for SystemEntity)
23         </rsm:SystemFunction>
24         <rsm:SystemFunction rdf:about="Calculate Ambiguity">
25           ... (detail of SystemFunction)
26         </rsm:SystemFunction>
27         <rsm:SystemFunction rdf:about="Calculate Popularity">
28           ... (detail of SystemFunction)
29         </rsm:SystemFunction>
30         ... (continue for SystemFunction)
31   </schema:featureList>
32 </schema:SoftwareApplication>
33 </rdf:RDF>
```

Fig. 14: An example C-SRS for HD Calculation System

V. A Practical Example Applying the Method

A real-world creative application, called Humor Degree (HD) Calculation System [27], is adopted to evaluate the feasibility of the presented method. Due to the space limitation, all requirement specifications in the example are implemented as services and there is no problem of consistency and completeness found during construction process. HD Calculation System is an application to describe how humorous an input of verbal joke containing no more than two utterances. The users of the system are computer scientists which attempt to provide a quantitative description of humor during the humor recognition using computer techniques. Two general features of humor, i.e., novelty and amusement, are selected as the computation elements, where each of them can be assigned proper values according to their contribution to the humor. The users also expect the HD calculation in the system can be reliable and adaptable for humor exploration.

In the first phase, the domain experts and software engineers work on Step 1 to construct SRG for requirement specifications

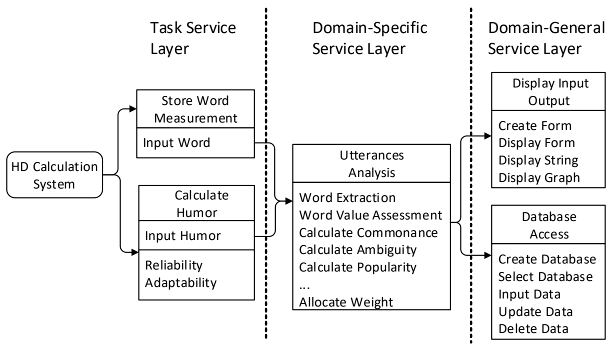


Fig. 16: An example of service description and composition for HD Calculation System

- Store word measurement
- Input Humor
- Calculate Humor
- Show Calculation
- Input Word
- Calculate Commonance
- Calculate Ambiguity
- Calculate Popularity
- Access Database
- Word Value Assignment
- Allocate Weights
- Word extractions

...

...

Fig. 15: Example capability candidates of HD Calculation System

of HD Calculation System. Here, we use lexical analysis to calculate HD of utterances, assuming that certain lexical features exhibit correlation with the occurrence of humor. The utterances consist of words, thus the properties associated with the words can, to some extent, reflect the corresponding utterances. To perform lexical analysis, the approach adopts a way which allows the users to store words with associated novelty and amusement values in order to calculate HD of the words derived from the inputs of humor. The result of SRG is shown in Fig. 12.

After the SRG is transformed into SRS model in Step 2 (the SRS model is similar to Fig. 14, but with different contents), Step 3 discovers the Creativity Candidates. Fig. 13 shows the Creativity Candidates resulted in Step 3 by calculating syntactic and semantic similarities between the concepts, extracted from AD of the SRS, and the concepts from knowledges represented in KCR shown in Fig. 1. Based on the execution of a machine of creative computing in Step 4, the *Utterances Analysis* concept is selected as Merging Options from the Creativity Candidates. Lines 24~30 of a C-SRS in Fig. 14 show the result of merging between the functional requirements for *Lexical Analysis of Utterances* containing *Calculate Novelty* and *Calculate Amusement*, shown in Fig. 12, and properties of Utterances Analysis concept containing *Ambiguity*, *Slang*, *Colloquialism*, *Commonance*, *Popularity*, and *Scope*, shown in Fig. 13. This C-

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:swss="ServiceStructure.owl">
<swss:DomainSpecificService rdf:about="Utterances Analysis">
  <rdfs:label>An analysis of a group of words</rdfs:label>
  <swss:ServiceCapabilityList>
    <swss:ServiceCapability rdf:about="Word Extraction">
      <rdfs:label>Extract each word from an input</rdfs:label>
      <swss:MessageMechanism>REST</swss:MessageMechanism>
      <swss:ServiceEndPoint>http://example.com/utterance/extract
</swss:ServiceEndPoint>
      <swss:MessageInput>
        <xs:element name="Humor" type="xs:String"/>
        ... (continue for xs:element)
      </swss:MessageInput>
      <swss:ServiceComposition>
        <swss:GenericService rdf:about="Display Input/Output">
          <rdfs:label>A service to make and display input/output user
interface</rdfs:label>
          ... (continue for ServiceCapability)
        </swss:GenericService>
      </swss:ServiceComposition>
      <swss:MessageOutput>
        <xs:element name="Status" type="xs:String"/>
        ... (continue for xs:element)
      </swss:MessageOutput>
    </swss:ServiceCapability>
    <swss:ServiceCapability rdf:about="Word Value Assessment">
      ... (details of ServiceCapability)
    </swss:ServiceCapability>
    ... (continue for ServiceCapability)
  </swss:ServiceCapabilityList>
</swss:DomainSpecificService>
</rdf:RDF>
```

Fig. 17: An example SSS for HD Calculation System

Table 1: Activities of the presented method and their associated input(s) and output(s)

Step	Activities	Input	Output
Phase 1: Domain-Creative Requirement Specification			
1	Construct Initial Requirement Specification	The result of discussion between software engineers and domain experts, and documentations of legacy system.	SRG
2	Transform Requirement Specifications to Semantic Web Model	SRG	SRS
3	Identify or Modify Potential Creativity	SRS and KCR	SRS and Capability Candidates
4	Construct or Modify Creative Requirement Specifications	SRS, Capability Candidates, and Updates	C-SRS Candidates
5	Examine Creativity	C-SRS Candidates	C-SRS
6	Validate Creative Requirement Specifications	C-SRS	Validated C-SRS or Updates
7	Update Resources (if necessary)	Ontology-based Knowledges	KCR
Phase 2: Semantic-Based Service Analysis and Design			
1	Filter Out Meaningless Service Encapsulation	Validated C-SRS	System Function and Initial Capability Candidates
2	Identify or Modify Capability Candidates	Initial Capability Candidates and Updates	Capability Candidates
3	Define or Modify Service Candidates	Capability Candidates and SMR	Service Candidates
4	Construct or Modify Services Composition	Service Candidates, System Functions, Validated C-SRS, and Updates	SSS
5	Validate Service Composition	SSS	Validated SSS or Updates

SRS is then examined for its creativity in Step 5 and validated for consistency and completeness in Step 6.

In the second phase, the software engineers extract the requirement specifications from the validated C-SRS to identify Initial Capability Candidates in Steps 1, which are defined as Capability Candidate in Step 2 and shown in Fig. 15. In Step 3, these Capability Candidates are grouped based on the contexts defined in the layer of services in DLSM. For example, the Capability Candidates of *Calculate Commonance*, *Calculate Ambiguity*, *Calculate Popularity*, …, and so on, are grouped as *Utterances Analysis* service candidate in domain-specific layer since the capabilities are reusable for a specific domain only. Step 4 identifies the Service Composition for Service Candidates, resulted in Step 3, to construct the SSS model. Fig. 16 shows an example of service definition, containing a set of capabilities and quality requirements (if any), and service composition. Fig. 17 shows an example of SSS model based on Fig. 16. After the SSS is validated in Step 5, the creative service software is well-constructed and can be implemented effectively without losing the creativities generated in Phase 1.

The development activities inside the example above show the feasibility of our presented method and its associated specification models to construct a creative service software. On the other hand, the detailed activities of the method with their associated input(s) and output(s) are shown in Table 1. The model constructed during the first phase can be used to generate necessary creativity successfully. In the second phase, the model of services can organize the services with no reusability, reusability in a specific domain, and reusability in multi domain, into different layers of service. Besides, the validated SSS can also be stored into SMR to help the development of creative service software and increase reusability in the future. For example, if SMR contains necessary SSS models, a machine of service discovery can automatically and effectively discover the available existing services to be reused from an input of C-SRS.

VI. Conclusions and Future Work

The development of creative software not only seeks an effective solution, but also a solution with novel, surprising, and useful properties by integrating multi-domain knowledges. In this paper, we present an effective method to construct a creative software with SOA in two phases, i.e., requirement specification and service design, where each phase applies a specification model based on semantic web, including SRG, Creativity Candidates, SRS, C-SRS, Capability Candidates, Service Candidates, and SSS, correspondingly. The creative service software constructed based on the presented model contains not only the intrinsic characteristics as that in SOA, but also contain the factors as that in creative computing, which can improve the novelty and reusability of the services. We further provide a practical example to demonstrate the feasibility of the methodology and its associated specification models.

Future researches focus on the validation of reusability of the services constructed, the reduction of communication between services to improve efficiency, and the development patterns to utilize the creativity resulted in our methods. The implementation, testing, and maintenance issues in the development of creative service software will be further discussed also.

References

[1] L. Zhang and H. Yang, "Definition, research scope and challenges of creative computing," in *Proceedings of 19th IEEE International Conference on Automation and Computing*, UK, pp. 1-6, 2013.

[2] D. Jing and H. Yang, "Domain-Specific 'Ideation': Real Possibility or Just Another Utopia?", *Applied Science Journal*, pp. 68-99, 2015.

[3] L. Zhang, H. Yang, C. Zhang and N. Li, "A New Way of Being Smart? Creative Computing and Its Applications in Tourism," in *Proceedings of 42nd IEEE Annual Computer Software and Applications Conference (COMPSAC)*, pp. 45-50, 2018.

[4] Y. Qi, W. Ren, M. Shi and Q. Liu, "A Combinatorial Method based on Machine Learning Algorithms for Enhancing Cultural Economic Value," International Journal of Performability Engineering, vol. 16, no. 7, pp. 1105–1117, 2020.

[5] D. Jing, H. Yang, L. Xu and F. Ma, "Developing a Creative Idea Generation System for Innovative Software Reliability Research," in *Proceedings of 2nd International Conference on Trustworthy Systems and Their Applications*, pp. 71-80, 2015.

[6] H. Yang, D. Jing and L. Zhang, "Creative Computing: An Approach to Knowledge Combination for Creativity?" in *Proceedings of IEEE*

Symposium on Service-Oriented System Engineering (SOSE), Oxford, UK, pp. 407-414, 2016.

[7] [7] C. Zhou, N. Li, C. Zhang and X. Yang, “Evaluation of Text Semantic Features using Latent Dirichlet Allocation Model”, International Journal of Performability Engineering, vol. 16, no. 6, pp. 968–978, 2020.

[8] T. Erl, *Service-Oriented Architecture: Analysis and Design for Services and Microservices*, Prentice Hall, 2016.

[9] F. J. Wang and F. Fahmi, "Constructing a Service Software with Microservices", in *Proceedings of 2018 IEEE World Congress on Services (SERVICES)*, pp. 43-44, 2018.

[10] P.S. Huang, F. Fahmi and F.J. Wang, "Improving the Detection of Artifact Anomalies in a Workflow Analysis," *IEEE Transactions on Reliability*, doi: 10.1109/TR.2020.3048612, 2021.

[11] A. Hugill and H. Yang, “The Creative Turn: New Challenges for Computing”, *International Journal of Creative Computing*, Inderscience, UK, vol. 1, no. 1, pp. 4–19, 2013.

[12] T. Erl, *SOA: Principles of Service Design*, Prentice Hall, 2008.

[13] S. Newman, *Building Microservice: Designing Fine-Grained Systems*, O'Reilly Media, 2015.

[14] Wikipedia, “World Wide Web”, https://en.wikipedia.org/wiki/World_Wide_Web

[15] World Wide Web Consortium (W3C), “Semantic Web”, https://www.w3.org/standards/semanticweb/

[16] W. Hall and K. O'Hara, "Semantic Web," *Encyclopedia of Complexity and Systems Science*, R. A. Meyers, Ed., ed New York, NY: Springer New York, pp. 8084-8104, 2009.

[17] P. S. Huang, F. Fahmi and F. J. Wang, “A Model to Helping the Construction of Creative Service-Based Software”, Accepted in *45th IEEE Annual Computer Software and Applications Conference (COMPSAC)*, 2021.

[18] World Wide Web Consortium (W3C), “RDF 1.1 Concepts and Abstract Syntax”, https://www.w3.org/standards/semanticweb/, 2014.

[19] B. Brueggle and A.H. Dutoit, *Object-Oriented Software Engineering: Using UML Patterns and Java*, Prentice Hall, 2004.

[20] Jolie: The Service-Oriented Programing Language, http://www.jolie-lang.org/.

[21] Z. Wu et al., "Implementing an Inference Engine for RDFS/OWL Constructs and User-Defined Rules in Oracle," in *Proceedings of IEEE 24th International Conference on Data Engineering*, pp. 1239-1248, 2008.

[22] Q. Liu, H. Zhou, H. Yang, and W.C.C. Chu, Solution Generation through Hybrid Intelligence and Creativity based on Investment Portfolio, *International Journal of Performability Engineering*, vol. 14, pp. 1641-1650, 2018.

[23] N. Zhang, J. Wang and Y. Ma, "Mining Domain Knowledge on Service Goals from Textual Service Descriptions," *IEEE Transactions on Services Computing*, vol. 13, no. 3, pp. 488-502, 2020.

[24] H. Alani et al., "Automatic ontology-based knowledge extraction from Web documents," *IEEE Intelligent Systems*, vol. 18, no. 1, pp. 14-21, 2003.

[25] V. Storey, R. Chiang, and G. Chen, "Ontology Creation: Extraction of Domain Knowledge from Web Documents," in *Proceedings of the 24th International Conference on Conceptual Modelling*, pp. 256-269, 2005.

[26] S. Newman, *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*, O'Reilly Media, 2019.

[27] T. Liu, H. Yang, and F. J. Wang, “A Creative Approach to Humour Degree Calculation for Utterances,” in *Proceedings of 20th IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pp. 650-656, 2020.