Semantic Selection for Model Composition using SAMSaaS

Sixuan Wang Gabriel Wainer
Dept. of Systems and Computer Engineering
Carleton University
1125 Colonel By Dr. Ottawa, ON K1S5B6, CANADA
{swang, gwainer}@sce.carleton.ca

ABSTRACT
The reuse of existing models in modeling and simulation software is becoming increasingly important. One way to improve model reuse is through composition. Even though much work has been done for model composition at the syntactic level, research at the semantic level is still under development. In particular, selection for model composition based on semantics is complex, because it is hard for casual users to understand the formal definition of models. Likewise, there are no automated mechanisms to build domain-specific ontologies based on the models. In order to deal with the above issues, we introduced SAMSaaS (Semantic Architecture for Modeling and Simulation as a Service), a layered architecture to deploy and compose M&S resources as services. Here, we focus on the upper layers of the architecture, and present a tag mining method and a tag-tree ontology learning approach to learn domain specific ontology for the models. We show an application to DEVS examples to show its success and possible applications.

KEYWORDS model composition, semantic composition, model selection, tag mining, ontology learning.

1. INTRODUCTION

In recent years, the modeling and simulation (M&S) community has investigated new methods to reuse models in order to save time and reduce costs. One way to increase such reuse is through model composition, defined as the process of selecting and assembling models in various combinations to meet user requirements [1].

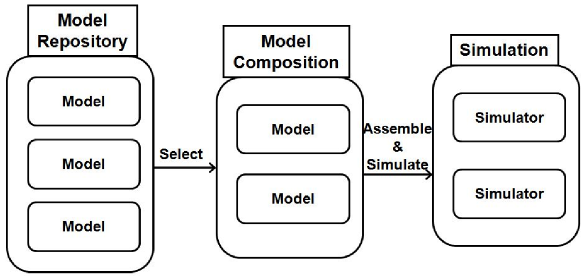

Figure 1. Model composition, repository and simulation

Figure 1 shows the idea of model composition and its relations with repositories and simulation. Using model composition, we can select and assemble models from a repository and simulate the assembled model. Model composition is more than just putting models together: it is to recombine components for different requirements [2]. Model composition can be addressed at the syntactic and semantic levels. Syntactic composition focuses on the actual implementation of composition; while semantic composition focuses on the underlying abstraction and meaning of the models [3]. Even though much work has been done in syntactic composition, semantic composition research is still in early stages. In particular, little research has been done in semantic selection for model composition, which requires the users to understand the meaning of models at the semantic level. The major issues are as follows: 1) users cannot fully understand the formal definitions of models [4]. 2) It is hard to get the "meaning" of the models [5]. 3) There is no way to build domain specific ontologies just based on the models [6].

We want to improve the selection for model composition based on semantics. In order to describe the model better, we propose using a model description in XML with meaningful information. In order to get the meaning of the models, we propose a tag mining method to get the semantics of models by tags. In order to get the domain specific ontologies, we propose an ontology-learning technique to construct a tag tree based on the XML Model Descriptions. We introduced a novel architecture, named Semantic Architecture for Modeling and Simulation as a Service (SAMSaaS), a layered architecture to deploy, identify, discover, compose, and invoke varied M&S resources as services in the Cloud. In this paper, we will discuss the Semantic Selection for Model Composition in the upper layers of SAMSaaS.

Model composition focuses on selection and assembling models. In particular, we focus on the semantic selection for model composition. We will not focus on assembling heterogeneous models, but rather on selection of individual models for composition based on semantics driven techniques. We are particularly interested on the semantic selection of DEVS models [7] and to provide users with meaningful suggestions for composition.

2. BACKGROUND

To compose models we need to select and assemble those that meet user requirements. A model can be defined using different modeling techniques, like Modelica [8] or DEVS [7]. These models can be hard to understand for casual users who are not familiar with the modeling method [4], especially when they are not M&S experts. In addition, each method defines models based on different abstractions, which may cause misunderstandings [9].

In order to deal with these issues and help users understand the models, new technologies, like XML, have been widely used. XML is used because it provides platform independent model descriptions that are robust and extensible [10]. In [5], the authors presented an M&S-based data engineering approach that allows a uniform way of describing distributed resources in XML. Many DEVS tools use XML to describe the models to increase model reusability in different languages and platforms. For instance, DEVSML [11], DML [12] and RISE [13] provide mechanisms using XML to combine different models. CoSMoS [14] supports component-based modeling for DEVS and XML Schema. SiMA [15] provides a similar system using XML for composition. In [4], the authors defined interface descriptions in XML as separate units of model definitions for discrete event models. All these approaches focus on describing platform-independent models in XML, but they do not answer how to compose existing models. Using XML alone is insufficient to deal with the meaning of the models.

Web-based Simulation (WBS) has also helped with model composition (in a simulation interoperability context) by providing different Web Services (WS) for sharing, composing and executing simulation models online. The variety of WS used can be categorized into two main classes: SOAP-based and RESTful. For example, in [17], the authors implements DEVS over an SOA framework. Instead, the RESTful Interoperability Simulation Environment (RISE) [13] introduced RESTful distributed simulation. Cloud computing is also popular for building simulation environments [18]. Most existing WBS efforts focus on simulation interoperability; they cannot provide meaningful model descriptions to help users composing models.

Semantic composition tries to use the meaning of the models and focuses on understanding this meaning. In [19], the authors introduced the Levels of Conceptual Interoperability Model (LCIM), which identifies seven levels of interoperability. In LCIM, the semantic level was defined as the meaning of models, requiring a reference model for common information exchange. In [20], the authors use pragmatics (data use in relation to data structure), semantics (meaning of terms) and syntax (data structure and rules). In [17], the authors proposed a layered M&S architecture. A modeling layer is related to semantic composition (focusing on independent model representation) and a collaboration layer focuses on common understanding. These M&S frameworks focus on the system architecture and they do not address how to get the semantics for these models or how to select the models.

There are many efforts focusing on ontologies for semantic composition. Ontologies can help getting the meaning of the models. The idea is to add a semantic layer to the models and to associate them to an external ontology (which consists of shared terms or vocabularies that are commonly understood, without ambiguity or misunderstanding). In [21], the author presented an ontological representation of the concepts and contexts under DEVS framework. In [22], the author discussed the role of ontologies as a strategy to improve composability. In [5], the authors used a Common Reference Model (CRM) as an ontology for model composition. DeMO [23] is a discrete-event modeling ontology that provides a precise description of simulation models with hard semantics. In [24] the authors presented a method for using the knowledge encoded in ontologies to facilitate the development of simulation models. In addition, some authors use ontologies in the semantic web to discover and match service components. In [25], the authors proposed to use an existing Semantic Web Platform to support their simulation projects. In [26], the authors used OWL-S to define a layered ontology (resource, function, simulation) for composing simulation services. In [4], the authors used the XML Schema Definition (XSD) to check the compatibility of models by applying semantic annotations. Most of these approaches depend on pre-defined ontologies, and defining a single conceptual model to represent all the simulation models is not practical [15]. Most ontologies are developed manually by skilled ontology experts, and ontology building is complex and time consuming. Also, pre-defined ontologies are hard to understand and learn and they may not cover all concepts for the user.

Another way to get the semantics from the models is by using tagging systems. In Web 2.0, tagging systems (also termed as Folksonomies) can be seen as large collections of informal semantics [27] in which many users cooperate to annotate objects with free-form tags of their choice. The more times a tag is used, the more accurate will be to represent the resource semantics; thus, tags will ultimately cover domain concepts [28]. A major problem of traditional tagging systems is their lack of structure/hierarchy. Loosely related tags make it difficult finding and selecting the related resources.

In brief, although semantics and ontologies are being used in M&S, it is still hard to get the meaning of the models for semantic composition. Little has been done about automatic learning domain-specific ontologies for the models. Tagging systems can help getting the model semantics, but there is no method on the use of tagging system and trying to learn tag hierarchy as ontologies in M&S.

Based on these ideas, we propose a novel architecture, named Semantic Architecture for Modeling and Simulation as a Service (SAMSaaS) to manage and compose models based on tag-based ontologies learning. Note that besides these objectives, SAMSaaS intends to improve the reuse of M&S resources (which is out of scope of this paper).

3. SAMSAAS ARCHITECTURE

Figure 2 shows the layered architecture of SAMSaaS. In general, SAMSaaS can deploy, identify, discover, compose, and invoke varied M&S resources as MSaaS. This architecture has the following five layers.

1) Cloud Layer: it is responsible for providing Cloud infrastructure support, like Cloud compute units and Cloud storage units. All resources (including models) are stored and all experiments are executed in this layer.

2) Component Layer: it is responsible for deploying user-provided M&S resources as MSaaS by using the MSaaS middleware (implemented by CloudRISE) on-demand. In addition, this layer can identify and register all the services.

3) Tag Ontology Layer: it is responsible for mining the semantics (by using *Tag Mining* functions) and learning the tag tree ontology (by using a *Tag Tree Learning* algorithm) according to registered model and other services.

4) Composition Layer: it is responsible for semantic composition. It can select from existing models and other services based on their tag semantics (using *Semantic Selection*) and assemble them (using *Semantic Assembling*).

5) Application Layer: it is responsible for developing new applications according to user requirements. This layer can discover and recommend meaningful models and other services for users. It also automates M&S workflows.

The semantic selection for model composition mainly involves the upper layers of SAMSaaS, which are Ontology Layer, Composition Layer, and Application Layer. SAMSaaS allows the users to deploy any user-provided model and related files in the Cloud. SAMSaaS can manage model descriptions by tags, get their semantics by tag mining, and learn domain specific ontologies, after which, we can select models according to the user requirements. In the following sections, we focus on the basic ideas, design and implementation of the upper layers of SAMSaaS in terms of semantic composition of model selection.

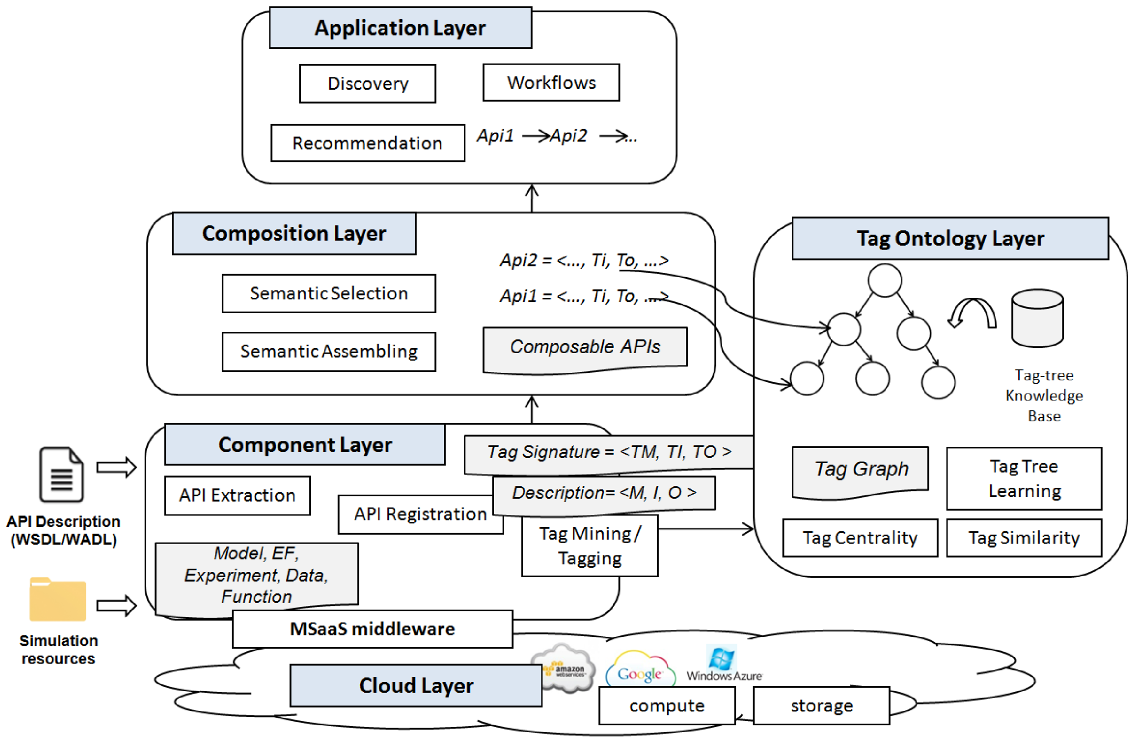

Figure 2. SAMSaaS Architecture

4. SEMANTIC SELECTION FOR COMPOSITION

4.1. Meaningful Description

As discussed above, in order to understand existing models better, we need a meaningful model description containing all necessary information about the model. As an example, we propose an **XML Model Description** for DEVS models, shown in Figure 3. As seen in Figure 3, a DEVS model (<*Models*>) may have different submodels (<*Model*>): atomic (defining behavior) or coupled (defining structure). The basic information includes the model *name*, *type* (e.g. atomic or coupled), and a *description*. If the model contains input/output ports, it also keeps them in <*Ports*>, including their *name*, *type*, and a *description*. The *type* is the message type accepted in that port. In addition, it can store the files used for this model in <*Files*>, including file *name*, *type* and *location*. A model can have multiple files; for instance, a DEVS model can have *CPP* classes, headers, and model configuration files. The *type* is the extension of a file. For example, *EV* means this file is an external events file. The XML Model Description keeps general information, which can be customized.

We can get this XML description file in different ways: 1) the modeler can provide it; and 2) it can be constructed from a well-defined modeling language automatically. For example, CD++ [29] is a simulation environment to execute DEVS models. We can extract most parts of this file from the CD++ model definition file. This kind of description should be separated from its definition and stored independently in order to not to affect the model definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<Models>
  <Model name="" type="" description="">
    <Inputs> <Port name="" type="" descrip-
tion=""/> ...
    </Inputs>
    <Outputs> <Port name="" type="" descrip-
tion=""/> ...
    </Outputs>
    <Files> <File name="" type="" loca-
tion=""/> ...  </Files>
  </Model> ...
</Models>
```

Figure 3. XML Model Description template

4.2. Obtaining the "Meaning" of the Models

As mentioned before, a second issue with semantic composition is how to get the meaning from the models. In our case, we use a tagging system and mining techniques to get tags from the XML Model Descriptions automatically.

We define the **Model Tag Signature** in Definition 1. Each Model Description corresponds to a Model Tag Signature, and each element in the Model Description corresponds to a set of tags in the Model Tag Signature.

Model Tag Signature= **< *TN, TI, TO* >** ***(Definition 1)***

- TN = <t_{model}> is a set of tags with general information of the model, including its name, type and description.

- TI = {<t_{port}>}, is a set of tags for input ports, including port name, type and description for each port.

- TO = {<t_{port}>}, is a set of tags for output ports, including port name, type and description for each port.

We use ***Tag Mining Functions*** to mine tags from the Model Description and build the Model Tag Signature. In general, a tag mining function converts each part of the XML Model Description into a set of corresponding tags, reducing the terms in the description and generating tags that can represent the model better. The Signature is saved in XML and stored together with its XML Model Description. The Tag Mining Function has two steps:

1) Syntactic pre-processing

- Tokenization: the element in the XML Model Description can contain multiple terms consisting of text and delimiters used to separate each term. For example, *the new_customer* will be tokenized into *{new, customer}.*

- CamelCase: they are compounded words that are personalized by users, (e.g. *BarberShop*), we separate them into different terms (e.g. *{Barber, Shop}*).

- StopWordFilter: some words are not needed for semantic composition, like *"the", "a", "and", "it", "its"*. We include them in a filter list.

2) Semantic pre-processing

- POS (Part of Speech): we consider that meaningful tags should be nouns or verbs. We use the POS concept of Natural Language Processing and an online API (e.g. *http://www.merriam-webster.com/dictionary/*) to automatically get the POS of each term and keep noun/verb terms.

- Abbreviations: we use an online API (like *http://www.abbreviations.com/*) to replace the abbreviation term with its full version automatically.

- Plural nouns and verb conjugation: after knowing the term's POS, we keep only the single form of the noun and the infinitive form of the verb (e.g. *customers* is replaced by *customer*, and *finishes* by *finish*).

4.3. Learning Domain Specific Ontology

After we have all the Model Tag Signatures, we define the tag tree ontology for the tags in these tag signatures. We first define the Tag Tree Ontology as follows:

Tag Tree Ontology ***TR = (T, E)*** ***(Definition 2)***

TR *is a Directed Acyclic Graph.* ***T*** *is a set of vertex represent tags {t1,t2,...,tn}, and* ***E*** *is the set of edges represented "subTag" relationship between two tags (formally t1* $\prec$ *t2).*

Currently, the only relationship we considered is *subTag*, which is learned from the tag co-occurrence graph of the Model Tag Signatures. The subTag relationship shows the semantically equivalence of tags. It implies that if a model can be described by a child tag, it will also be correct if described by its parent tag. The tree positions of the tags reflect the semantics of the model. Therefore, this kind of tag tree serves as an ontology for the given models.

The tag-tree ontology acts as a simplified domain-specific semantics, representing knowledge using tags agreed by users. Unlike the complex structures and relationships in normal ontologies, a tag-tree ontology only includes tags and their hierarchy, which makes it easy to understand and modify. The tags in higher levels are more general and abstract, while the tags in lower levels are more detailed and concrete. A tag tree can be a subset of an ontology/terminology for a domain that is defined by semantic experts. Users are also free to define their own tag trees.

Now, let us see how to learn the tag-tree ontology based on the models with mined tags. We defined a **Tag Tree Learning Algorithm** based on [28], whose idea is that tags tend to express the same concepts if they occur together frequently. Heymann's algorithm cannot be directly used for the model composition because it only works for connected tag graphs. Heymann's algorithm assumes that there should always be a path between two tags in the graph, and that the weight of two connected tags should always be 1. Therefore, we cannot use it (i.e. because the tags mined from models can be anything; the tag graph could be weighted and unconnected). Heymann's algorithm uses the cosine similarity, and it does not consider all kinds of relationships between tags, including frequency, syntactic or semantic similarities. Instead, our algorithm deals with these issues. The process is iterative. On each iteration, it does followings: 1) select a right tag to be added into the tree; and 2) find the tag's position in the tree.

The algorithm initials the tree *TR* with a root node. Then, it calls the *Centrality Function* to get the closeness centrality list of given tags in descending order. This centrality list indicates how central the tag is in the Co-occurrence Graph. Next, it selects each tag in the closeness centrality set. The *Similarity Function* tells where we should place the tag, it checks the tag with each tag in the tree, and it finds a tag to be attached with highest similarity. If the highest similarity is greater than the predefined threshold, it adds the tag as a child; otherwise, it adds the tag as a child of the root node.

$$closeness(i) = \sum_j \frac{1}{Dij} \qquad \textbf{\textit{(Definition 3)}}$$

$$where\ Dij = \begin{cases} \frac{1}{Fij}, & if\ tag\ i\ and\ tag\ j\ are\ connected\ directly \\ shortest\ distance\ between\ them, & else \end{cases}$$

For selecting the right tag, we modified the closeness centrality function (the inverse value of the sum of the distances to all other nodes) [28]. We calculate the closeness as the sum of the inverse distances to all other nodes (Definition 3). The edge of the tag graph is now weighted by the distance of tags (not always 1). This also handles cases with unconnected graphs. For instance, if two tags (e.g. *a* and *b*) are not connected, their distance will be infinite (e.g. $F_{ab}= 0$, so $D_{ab}\rightarrow\infty$), these two tags will be calculated as *0* (e.g. $1/D_{ab}=1/\infty\rightarrow 0$). Therefore, the unconnected tags will contribute nothing to the closeness of each tag, which makes the centrality more accurate.

For placing the tag in the right position into the tree, we use a similarity function to check each tag in the existing tree and to find the tag with highest similarity. The similarity function considers the distance of the two tags in the co-occurrence graph; the syntactic similarity of the two tags is based on the editing distance; and the semantic similarity of the two tags is based on the Levenshtein edit distance, and if the two tags are synonyms. We use WordNet [30] as a linguistic database for finding synonyms.

4.4. Semantic Model Selection

Our tag-based semantic composition can provide **Composable Models** that can be linked (Definition 4). Let us assume there are two Model Tag Signatures M1 and M2. If for any tag *t2* in the tag set T_i of a M2's input port there is a tag *t1* in the tag set T_o of an output port of M1, and *t1* is equivalent to or a "subTag" of *t2* (formally $t1 \prec t2$), then M1 can be composed with M2 (formally M1→M2). We view the tags as semantically equivalent if they have a relation in the tag tree. If the tags produced by a model are semantically equivalent to the tags accepted by another model, these two models could be composed.

M1 -> M2 Composable Models ***(Definition 4)***
$M1=<T_{m1},\{T_{i1}\},\{T_{o1}\}>$ and $M2=<T_{m2},\{T_{i2}\},\{T_{o2}\}>$ are two Model Tag Signatures for a given a tag tree ontology **TR = (T, E)**. *M1 -> M2* are said **composable** if they satisfy that $\forall t2 \in T_{i2};\ \exists t1 \in T_{o1};\ \exists\ t1 = t2\ or\ t1 \prec t2\ in\ T$.

Based on the composable models, we can recommend the users the models that they may be interested in, suggesting semantically equivalent models (i.e. tags are identical or have a *subTag* relation). For DEVS models, the composable models suggest potential coupling linkings of the models (i.e. linking input and output ports of two models).

Besides the model recommendation for a given model, our approach could also help users discover other models. They can query models by tags of their name/inputs/out-puts, and we return models that meet their query semantically. For example, if a user wants to find a model with an output with the tag *password*, and no model has that output, but there is one with output *pin*, and the tree includes a relation *pin*≺*password*, we will suggest that model.

5. IMPLEMENTATION

We developed **CloudRISE,** a middleware to manage all the resources related to SAMSaaS in the Cloud, and a Python project **PyCom** that implements the tag-based model composition logic. CloudRISE implements the concept of MSaaS (exposing all kinds of M&S resources as services). CloudRISE is an extended version of RISE [13] for reproducing experiments for either the simulations or supported functions. CloudRISE uses a resource-oriented design via RESTful WS in which M&S resources are identified through URIs in the Cloud.

CloudRISE works as a repository interface for users to share and manage the M&S resources. There are six main branches: *models, data, semantics, instances, simulations,* and *functions*. The ones related to the tag-based model composition are *models* (i.e. model files, Model Descriptions and Model Tag Signatures), and *semantics* (i.e. tag-tree ontology). Users can upload and modify these files by using the HTTP methods GET/PUT/POST/DELETE to the corresponding URIs. For instance, we use PUT to *../models /{approach}/{model}* to give an XML Model Description for the model; POST to *../models/{approach}/{model}/files* to append its Model Tag Signature; DELETE to *../models/ {approach}/{model}* to remove a model, or GET *.../models/ {approach}/{model}* to retrieve the files related to a model.

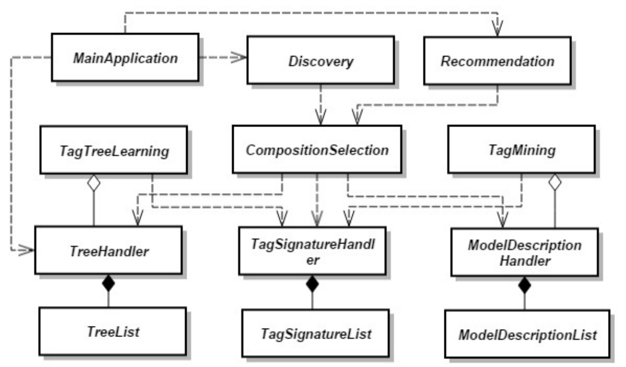

Figure 4. PyCom Class Diagram for semantic composition.

The tag-based model composition is implemented in a tool called PyCom, described partially in Figure 4. PyCom uses two kinds of classes: *data type* and *logic*. The data type classes include *ModelDescription, ModelDescriptionHandler, TagSignatureList, TagSignatureHandler, TreeList,* and *TreeHandler*. Since our XML files are stored in the Cloud (through CloudRISE), all data type classes in PyCom are for storing data in corresponding XML files. The handler classes load and save these XML files into a list. The logic classes include *TagMining, Tag TreeLeaning, Model Composition, Discovery, Recommendation,* and *Main-Application*. *TagMining* implements the functions discussed in Section 4.2 to mine tags for the models. *TagTree-Learning* has functions to learn a tag tree ontology based on the algorithm of Section 4.3. The Model composition can check the relations of two tags (whether they has *subTag* relations in a given ontology tree), and check whether two models are composible. The *Discovery* can find models that have semantically equivalent tags in their inputs and outputs. The *Recommendation* can suggest users with models that they may be interested in. Note that PyCom can be extended for specific functions or other modeling formalisms.

6. TEST AND CASE STUDY

In this section, we test the success of the proposed tag-based semantic model composition, and then we show a case study to select models for user requirements.

6.1. Success Rate Testing

We tested our approach using an existing repository of CD++ models. We chose 16 model samples at random (with a total of 103 sub-models). These are DEVS models and cover a wide range of domains, such as network protocols, mobile phones, alarm systems, secure area access, computer networks, and transportation. Each model has different submodels that could be reused for new purposes.

In order to test the effect of the Model Description for model composition, we used two different Model Descriptions. In the first version, we obtained the model names and port names from its model definition file, and built the Description automatically. Note that in this version the ports only have name but no description. For the second version, we improved these with meaningful text, studying each model carefully and writing detailed descriptions.

Then, we used the tag mining functions to get tags for these models and built a tag tree from these tags following our tag-tree learning algorithm. The tag hierarchies of the two trees obtained were reasonable. Most *subTag* relations were correct. For example, *feedback* $\prec$ *notify* $\prec$ *model; wireless* $\prec$ *network* $\prec$ *data* $\prec$ *system* $\prec$ *model* $\prec$ *treeRoot*. Note that the trees are domain-specific and can be modified by the users. We customized the second tree (e.g. removing a few tags that were made nonsense, like *access* $\prec$ *area*).

The tests were based on the idea that for the existing DEVS models, we know how the models can be composed, and we use these as the **expected set**. Using our approach, we get a **learnt set** that suggests models. The idea of this test is to compare how similar these two sets are. We measured the overall success of the two sets using popular metrics in graph similarity analysis: *sensitivity (S), precision (P),* and *F-measure (F)*, which are defined in Definition 5.

Composition Metrics: ***(Definition 5)***

$$S=C/(C+M); \quad P=C/(C+E); \quad F=2/(1/S + 1/P)$$

Where, C (Common) is the common number of couplings in both sets, E (Extra) is the number of couplings that only exists in the learnt set; and M (missing) is the number of couplings that only in the expected set.

In particular, expected set is the sum of common models (C) and missing models (M); while learnt set is the sum of command models (C) and extra models (E). Sensitivity (S) represents the ratio of common models in expected set, i.e. S=C/(C+M); while Precision (P) represents the ratio of common models in learnt set, i.e. P=C(C+E). Usually, sensitivity and precision are combined into F-measure (F), which is defined as their harmonic mean F=2/(1/S +1/P)).

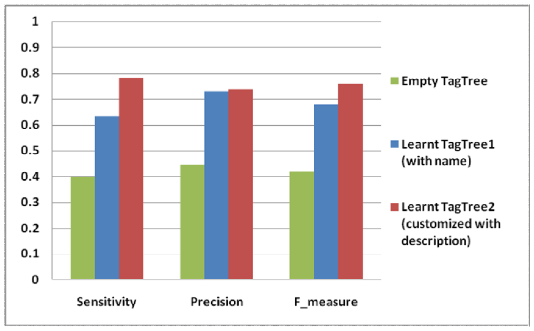

Figure 5. Model composition success rates.

Figure 5 shows the test results. We show the results of three different tests. The first one used an empty tree (i.e., if two models can link, they must have the same tag). The other two tests used the two trees discussed above. As we can see, using tag-trees has a higher success rate. Tree 1 is the one learnt from the port names, and Tree 2 is customized with more descriptions. Tree 2 has a better success with an average of 75% for each criterion, which shows how the description improves semantic selection. The empty tree is always worse: it can only find around 40% of all the three metrics. A possible reason for this is that this tree is only based on a *keyword* search, in which two models that can be linked must have the same tag, without any semantic considerations. In our tag tree approach, more semantic factors are considered, and we can find more couplings matching the best case (i.e., our approach will use tags that are not exactly the same but semantically equivalent).

6.2. Case Study

Let us assume that we want to build an occupancy model for a public service office. During the day, people come into the office at random, and there are different officers serving them throughout the day. We want to evaluate the work efficiency of officers and to improve their service. To do that, we want to know: 1) the **time** of the day, and 2) the total **number** of clients who have been served. If we search existing models looking for the keywords *time* and *number*, we obtain an empty result. Instead, using PyCom with semantic model selection, we obtain results based on Figure 6, which shows parts of Tree 2. In this tree, we can see that there are several subTag relations, such as *count* $\prec$ *number, clock* $\prec$ *time,* and *passenger* $\prec$ *customer* $\prec$ *user*.

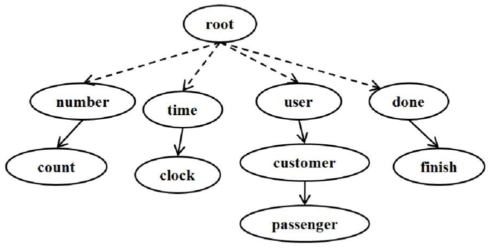

Figure 6. A part of the learnt tag tree ontology.

Initially, we know that the occupancy model should have two output ports: one with tag *number* (number of clients who have been served) and one with tag *time* (the time of the day). By searching *time* as an output tag, PyCom suggests models with those ports, and we can choose the one that meets our demands best. For instance, it can suggest model *Clock* since in the tag tree there is *clock* $\prec$ *time*, which means that the output *clock* generated from the *Clock* model can also be the output *time* for the target model (see Figure 7). *Clock* is an atomic model included in the *Alarm* model sample. *Clock* can take inputs of hour and minutes, set the current time and keep updating the time as a wall clock in the output clock port.

Similarly, when searching *number* as an output tag, since in the *count* $\prec$ *number*, PyCom will suggest model *Counter*, which is included in the *Garage Door* model sample. The model takes an input in *done,* and it *count*s how many inputs were received. PyCom can be used to suggest models that can link to *Counter* as inputs. Since we have *finish* $\prec$ *done,* which means that the *input* done of the *Counter* model can be linked with the output *finish* of the *Processing* model. The *Processing* model is from the *Airport Boarding* model and it can process an input *user*, and after serving the user, it will generate an output *finish*. Similarly, since *customer* $\prec$ *user*, PyCom can suggest *Queue* from the *Panama Canal* model as a sample for *Processing*, which takes a *customer* in a buffer. Then PyCom can suggest *Generator* from the *Train* model sample since there is *passenger* $\prec$ *customer*, which can generate new *passenger*s to be served.

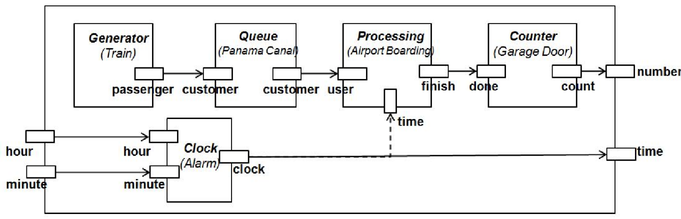

Figure 7. Semantic model composition for an Occupancy case using the tag-based composition approach.

Now, the model selection is done, we can link them as a new coupled model. This model will take the input of *hour* and *minute*, and generate the output of *time* and the *number* of users that have been served. Inside this model, it will generate clients, queue them in the reception, serve them and count the number that have been served. As we can see, we have reused models from 5 different samples, and saved development time by selecting and reusing existing models. Please note that these suggested models can be reused fully or partially. This depends on the users' responsibility to customize these models. In fact, they can do whatever they want to improve the logic inside each reused model. For example, the queuing logic in *Queue* may be FIFO; but in this case, we need a priority-based order. If so, we can change it accordingly. In addition, the serving time in the processing model maybe dependent on the time of day, so we can add a new port *time* in the processing model and link it as an output with the *clock* in the *Clock* model (the dashed line added in Figure 7). So far, the model is complete and we can assemble the found models together in model definition file and execute it in either standalone simulator (i.e. CD++) or on our web-based simulation middleware (i.e. CloudRISE).

This example shows the effectiveness of our approach for selecting models using the semantic of the tags by the leant tag-tree ontology, which can provide meaningful models to meet our requirements. Please note that there are only "subtree" relations between tags in current tag tree ontology. However, the tag tree could be complicated with more relations and constrains. For instance, the learning algorithm can be improved by analyzing more semantic relations of tags, and ontology experts can improve the tag trees with specific constrains. Furthermore, if the models are very close, our approach may suggest users more models. In that case, users are responsible to decide which models to use, since this paper focuses on recommending models to users instead of making decision for the users. This issue could be handled if users provide more accurate model descriptions; so close models can have different tags and could be suggested in different cases.

7. CONCLUSION

We presented a new architecture, called SAMSaaS, for semantic selection for model composition. It uses a tag-based approach to help users to select existing models in a semantic way. In particular, we proposed a XML Model Description with meaningful information for models. We presented a tag mining method to get their semantics as tags. We proposed a tag-based ontology learning approach to learn domain specific tag-tree ontology for the models. In addition, we implemented the CloudRISE middleware for model repository and remote simulation execution. We also implemented the PyCom tool for tag-based model composition. We used different DEVS models as examples to test our approach, showing a higher success when compared to the traditional "keyword" selection.

The tests presented here are based on homogenous DEVS models in CD++. In the near future, we will explore the semantic assembling of heterogeneous models. Since the models selected could be from different formalisms / implementations, after selecting the models to be composed, we might need data exchange or type transformation in order to let the message transfer from one model to the other.

REFERENCES

1. Petty, M.D.; Weisel, E.W. 2003. A composability lexicon. In 2003 Spring Simulation Interoperability Workshop.
2. Petty, M.D.; Kim, J.; Barbosa, S.E.; Pyun, J. J. 2014. Software Frameworks for Model Composition. Modelling and Simulation in Engineering. 2014 (1), 1-18.
3. Davis, P. K.; Anderson, R. H. 2004. Improving the composability of DoD models and simulations. The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology. 1(1), 5-17.
4. Röhl, M.; Morgenstern, S. 2007. Composing simulation models using interface definitions based on web service descriptions. In the 39th conference on Winter simulation.
5. Tolk, A.; Diallo, S. Y. 2005. Model-based data engineering for web services. Internet Computing, IEEE. 9(4), 65-70.
6. Wang, S.; Wainer, G. 2014. Semantic mashups for simulation as a service with tag mining and ontology learning. In the Symposium on Theory of Modeling & Simulation.
7. Zeigler, B. P.; Praehofer, H.; Kim, T. G. 2000. Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems. Academic press.
8. Elmqvist, H.; Mattsson, S.E.; Otter, M. 2000. Object-oriented and hybrid modeling in Modelica. In 4th International Conference on Automation of Mixed Processes.
9. Davis, P. K.; Tolk, A. 2007. Observations on new developments in composability and multi-resolution modeling. In Proceedings of the 39th conference on Winter simulation.
10. Harold, E. R. 2002. Processing XML with Java. Addison-Wesley Longman Publishing Co., Inc.
11. Mittal, S.; Risco-Martín, J. L.; Zeigler, B. P. 2007. DEVSML: automating DEVS execution over SOA towards transparent simulators. In Spring Simulation Multiconference
12. Touraille, L.; Traoré, M.K.; Hill, D.R. 2009. A mark-up language for the storage, retrieval, sharing and interoperability of DEVS models. In Spring Simulation Multiconference.
13. Al-Zoubi, K.; Wainer, G. 2013. RISE: A General Simulation Interoperability Middleware Container Journal of Parallel and Distributed Computing. 73(5), 580–594.
14. Sarjoughian, H.S.; Elamvazhuthi, V. 2009. CoSMoS: a visual environment for component-based modeling, experimental design, and simulation. In Proceedings of the 2nd international conference on simulation tools and techniques.
15. Alpdemir, M. N. 2012. SiMA: a discrete event system specification-based modelling and simulation framework to support model composability. JDMS, 9(2), 147-160.
16. Röhl, M.; Uhrmacher, A.M. 2008. Definition and analysis of composition structures for discrete-event models. In Proceedings of the 2008 Winter Simulation Conference.
17. Mittal, S.; Zeigler, B. P.; Martin, J. L. R. 2009. Implementation of formal standard for interoperability in M&S/systems of systems integration with DEVS/SOA. International Command and Control C2 Journal, Special Issue: M&S in Support of Network-Centric Approaches and Capabilities, 3(1).
18. Cayirci, E. 2013. Modeling and simulation as a cloud service: A survey. In Winter Simulation Conference.
19. Tolk, A.; Muguira, J. A. 2003. The levels of conceptual interoperability model. In the 2003 Fall Simulation Interoperability Workshop.
20. Zeigler, B. P.; Hammonds, P. E. 2007. Modeling & simulation-based data engineering: introducing pragmatics into ontology for net-centric information exchange. Academic Press.
21. Yilmaz, L. 2004. On the need for contextualized introspective models to improve reuse and composability of defense simulations. Journal of Defense M&S. 1(3), 141-151.
22. Yilmaz, L. 2006. On improving dynamic composability via ontology-driven introspective agent architectures. In Proc. of World Multi-Conference on Systemics, Cybernetics and Informatics.
23. Miller, J. A.; Baramidze, G. T.; Fishwick, P. 2004. Investigating ontology for simulation and modeling. In Proceedings of the 37th Annual Simulation Symposium.
24. Silver, G.A.; Hassan, O. H.; Miller, J.A. 2007. From domain ontology to modeling ontology to executable simulation models. In Winter Simulation Conference.
25. Rabe, M.; Gocev, P. 2012. Applying Semantic Web technologies for efficient preparation of simulation studies in manufacturing. In 2012 Winter Simulation Conference.
26. Li, T.; Chai, X.; Hou, B.; Li, B. 2013. Research and application on ontology-based layered cloud simulation service description framework. In the 2013 ACM SIGSIM conference on Principles of advanced discrete simulation.
27. Wal, T. V. 2014. Accessed by March 10. Folksonomy coinage and definition. http://vanderwal.net/folksonomy.html.
28. Heymann, P.; Garcia-Molina, H. 2006. Collaborative creation of communal hierarchical taxonomies in social tagging systems. Technical Report. Stanford.
29. Wainer, G. 2009. Discrete-Event Modeling and Simulation: A Practitioner's Approach.CRC/Taylor Francis.
30. Fellbaum, C. 1998. WordNet. Blackwell Publishing Ltd.