Proceedings of the 2011 Winter Simulation Conference
S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, eds.

AN APPROACH TO SEMANTIC-BASED MODEL DISCOVERY AND SELECTION

Claudia Szabo

School of Computer Science
The University of Adelaide
Innova21 Building
Adelaide 5005, AUSTRALIA

Yong Meng Teo

Department of Computer Science
National University of Singapore
13 Computing Drive
SINGAPORE 117417

ABSTRACT

Model discovery and selection is an important step in component-based simulation model development. This paper proposes an efficient model discovery approach and quantifies the degrees of semantic similarity for selection of partially matched models. Models are represented as production strings as specified by an EBNF composition grammar. Together with a novel DHT overlay network, we achieve fast discovery of syntactically similar models with discovery cost independent of the model size. Next, we rank partially matched models for selection using semantic-based model attributes and behavior. Experiments conducted on a repository with 4,000 models show that on average DHT-based model lookup using production strings takes less than one millisecond compared with two minutes using naive string comparisons. Lastly, efficient model selection is a tradeoff between query representation and the computation cost of model ranking.

1 INTRODUCTION

In component-based simulation model development, shared model components are discovered and composed to form larger, more complex models at lower cost than building the model from scratch (Kasputis and Ng 2000, Pidd 2004). Despite shorter development time, flexibility in meeting diverse user needs, and availability of model repositories facilitated by advances in Internet technologies, model component reuse has yet to become mainstream as initially envisaged (Kasputis and Ng 2000). The process of finding the appropriate components for reuse remains an open problem (Aronson and Bose 1999, Oses et al. 2004, Chreyh and Wainer 2009) and there is a need for an efficient discovery service that locates model components from different geographical locations and administrative domains (Davis et al. 2000).

Several challenges make model discovery a non-trivial problem. Firstly, discovery is usually highly dependent on the structure and encoding of models, and becomes unfeasible when models change or are replaced (Uhrmacher et al. 2004). Thus, there is an urgent need for a similarity-based ranking approach in which discovery and selection are performed on a meta-model representation, which is separate from the model encoding (Kasputis and Ng 2000, Uhrmacher et al. 2004). Secondly, similarity is not an exact measure, and partial matches to a user query must be evaluated (Kasputis and Ng 2000). Thirdly, model components are distinguished from stateless software engineering artifacts through their stateful behavior with attached meaning (Henkel et al. 2010). However, this distinction is not found in current information retrieval work which look at the ranking and retrieval of data (Baeza-Yates and Ribeiro-Neto 1999, Li et al. 2003). Thus the discovery of *models*, with respect to their behavior, characteristics, suitability to the objective of the model composer, and inner structure, still poses an important challenge. Lastly, with the advance of Internet technologies, shared model components can reside in large repositories distributed over geographical locations and administrative domains, and a scalable and efficient discovery service is required to locate and rank candidates (Davis et al. 2000, Kasputis and Ng 2000).

Knowledge about components as well as the component query must be represented in an appropriate format to facilitate meaningful discovery. Towards this, ontologies have been proposed in software

engineering to represent knowledge about components and domains in general (Berners-Lee et al. 2001). While the importance of a simulation ontology has recently been highlighted (Turnitsa et al. 2010) and ontologies have been proposed to represent knowledge about application domains (Silver et al. 2009), few works employ ontologies in model discovery and selection (Moradi et al. 2007). In contrast, ontology-based discovery of software artifacts is well studied in software engineering (Tansalarak and Claypool 2005). However, the focus is only on basic programming language constructs such as functions. Lastly, an appropriate model representation that describes both model features (or attributes) and model behavior is required for meaningful discovery (Kasputis and Ng 2000).

In our previous work, we proposed the CoDES (Composable Discrete-Event scalable Simulation) approach for component-based simulation model development (Teo and Szabo 2008). In CoDES, the composed model follows a four-step life-cycle. The user composes a conceptual model by using component icons to draw the model in a GUI. Once the conceptual model is drawn, its syntactic composability is verified, using our proposed Extended Backus-Naur Form (EBNF) composition grammar (Szabo and Teo 2007). If the conceptual model is syntactically correct, model discovery identifies appropriate components from the repository, and the composed model can then be semantically validated (Szabo and Teo 2009).

This paper presents our approach for model discovery and selection. Our main contributions include:

- **DHT-based model discovery**: We propose a novel way to represent the model repository as a Distributed Hash Table (DHT) (Balakrishnan et al. 2003) for fast model discovery. The discovery process is translated into an efficient DHT lookup using the structure of the conceptual model, expressed using a *production string*, as the lookup key.
- **Semantic-based model selection**: Once model components have been discovered, we quantify semantic similarity using our proposed *Matching Index* to rank partial matches between a user query and repository components. Our Matching Index quantifies model similarity using semantically-sugared attribute names, values, and behavior.

This paper is organized as follows. We analyze current challenges and discuss related work in Section 2. Section 3 presents an overview of the CoDES life-cycle, focusing on key aspects such as component abstractions and simulation ontology. Section 4 presents our proposed approach for model discovery and selection. We analyze our approach experimentally in Section 5. Section 6 concludes this paper.

2 RELATED WORK

While discovery and selection is an important issue in modeling and simulation (Aronson and Bose 1999, Chreyh and Wainer 2009), there are few works on this aspect of model reuse. Challenges include the lack of appropriate model representation, the ranking of partial matches to a user query, and meaningful measures of similarity. Adequate model representation that is independent of the model encoding is of great use when models change or are replaced (Uhrmacher et al. 2004) and there is a need for a semantic similarity-based ranking approach that looks at model behavior and is not limited to features such as model author, publication, and model type (Henkel et al. 2010). In particular, ontologies could be used to obtain meaningful measures of model similarity (Kasputis and Ng 2000), but current work only focuses in general on representing domain characteristics (Silver et al. 2009). In this section, we discuss work in modeling and simulation, followed by work in software engineering and computational biology.

Aronson and Bose (1999) describe the simulation model query and selection process with respect to a user specified simulation scenario. A quality function is proposed as a possible solution to partially ordering discovered simulation models, but details are not provided and no follow-up work exists. Current work that employs the Base Object Model (BOM), proposes to discover the appropriate composition of BOMs for a specific simulation scenario (Moradi et al. 2007). Model components are discovered based on a detailed user-specified simulation scenario that includes event names and parameters. Syntactic discovery (based on event name and parameters) as well as semantic discovery (based on entity and data types according to an

ontology) is performed. Several candidates are returned for each BOM. Combinations of all candidates for each component are composed and their composition is executed. A composition of candidate components is valid if its execution conforms to the scenario. As such, composition and semantic validation is performed based on well-specified simulation scenarios with low level details that are costly to define in practice. Furthermore, the discovery-composition-validation cycle is computationally expensive because all possible combinations of candidate components are composed and executed. In contrast, we propose a two-stage discovery and selection process. First, our DHT-based model discovery relies on syntactic information from the conceptual model. Next, in model selection, we propose the semantic-based ranking of partial matches for a user query based on attributes and behavior using our proposed COSMO ontology.

Recent work that uses the DEVS (Discrete-Event System Specification) formalism (Zeigler et al. 2000), CellDEVS++ (CD++) (Chreyh and Wainer 2009), proposes an online repository of CD DEVS models together with their experimental frames. The repository stores atomic and coupled DEVS models with their experiment data and results, including a textual description of the experiment assumptions and constraints. Model discovery is performed based on data from the experimental frames, including information about how the model can be used. However, there are no means yet of ranking partial matches based on a query. Furthermore, the search process compares only string text values without attached semantics and in the absence of linguistic algorithms to determine similarities.

In software engineering, a relevant work is that of Tansalarak and Claypool (2005), which propose a combination of different matching techniques to provide a ranked set of software components from the repository. However, the focus is on the discovery of software components with respect to their syntax in terms of method signatures in interfaces, and not on *semantics*. Nonetheless, the potential of applying information retrieval techniques to model discovery and selection is highly important and remains widely untapped in modeling and simulation. A related domain in which information retrieval techniques have been recently applied is computational biology (Henkel et al. 2010). Here, the retrieval and ranking of computational biology models is proposed based on various mandatory and optional model features used for version control and modeling. Similarity is computed based on a biological ontology. However, discovery based on the semantics of the model, i.e. the model behavior and what the model abstracts is not addressed.

3 LIFE-CYCLE OF COMPONENT-BASED MODEL DEVELOPMENT

The component-based model development life-cycle in CoDES (Teo and Szabo 2008) follows four main steps, namely *conceptual model definition*, *syntactic verification*, *model discovery and selection*, and *semantic validation*. In *Conceptual Model Definition*, a simulation problem is translated into a conceptual model using a graphical editor. Component icons, representing various entities in the problem, are dragged and dropped onto a drawing panel and connected to form the conceptual model. The component icons denote model component stubs and can be viewed as generic building blocks used to compose models in a domain, without explicit syntax and semantics at this stage, other than input/output port. In *Syntactic Verification*, the structure of the conceptual model is verified using our proposed EBNF composition grammar (Szabo and Teo 2007). The conceptual model is translated into a production string describing the linear arrangement of model component types. This production string is then verified by our proposed composition grammar, which describes how components can be connected in that particular domain. If the conceptual model is syntactically correct, model components corresponding to each building block are discovered in the *Model Discovery and Selection* step. Once all components or the composed models have been discovered, their attributes and behavior are instantiated. Lastly, semantic validation is performed and the validated model is then translated into a simulator for experimentation.

We envisage three types of reuse, namely, *standalone models*, which are validated models that are used as-is, *base components*, and *hierarchical components*. Base components represent the basic building blocks that can be used to compose models in a domain. They are represented in CoDES as black-box with in and/or out communication channels. Base components are reused to build models within a domain. For scalability and reuse, hierarchical components are composed from base components from the same domain,

and from hierarchical components from any domain. They must have both in and out communication channels to facilitate composition. Hierarchical components are obtained from validated composed models that are saved into the repository, by removing components to ensure the hierarchical component has both in and out communication channels. For example, for the composed model in Figure 5, components C_1 and one of C_{10}, C_{11} or C_{13} could be removed.

An example of base components for the Queueing Networks domain can be source, server, and sink, which can be used to build queueing network models as shown in Figure 3. The CoDES framework adheres to a component-connector paradigm, in which a *component* is viewed as a black box with an in- and/or an out-channel. Components are interconnected by *connectors* which perform timely message and data passing among them. Connector types include *one-to-one* (*ConO*), *many-to-one* (join or *ConJ*), and *one-to-many* (fork or *ConF*).

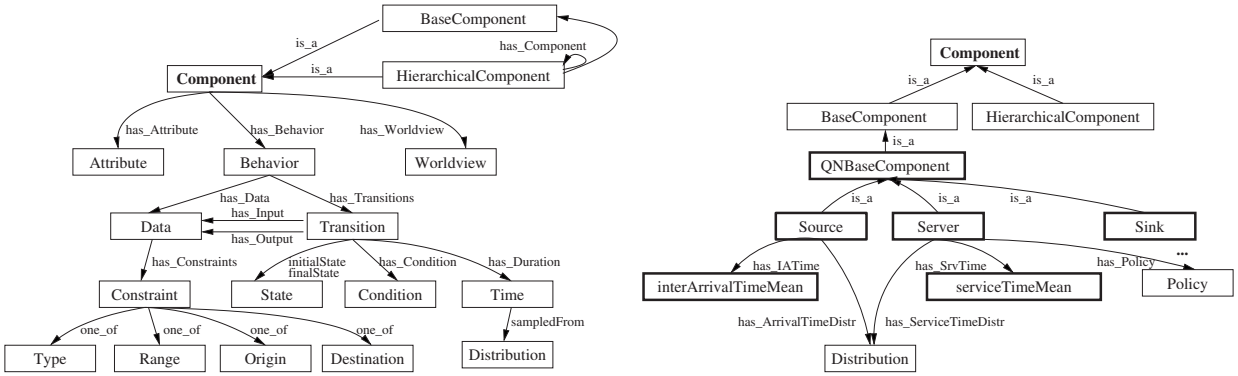
3.1 Component Abstraction and Ontology

Model components are represented in CoDES by *meta-components*, $C_i = \langle R, A_i, B_i \rangle$, which describe the component *required attributes* R, *specific attributes*, A, and *behavior*, B, and are extensively used throughout the life-cycle of a composed model. The required attributes are common to all components and are generally employed for version control, e.g. `author, location, lastUsed`. Examples of specific attributes include `interArrivalTime`, `numJobsServiced`, etc. The component behavior describes the data that it receives and outputs as a set of states. The transitions between states are defined as a set of triggers expressed in terms of input, time and conditions. To represent simulation models in a computer readable format we propose the COML markup language (Teo and Szabo 2008), which is described in a XSD schema using entities from our proposed COSMO ontology.

The data that a component sends and receives must be properly specified to ensure that *data compatibility* between components can be established (Tolk et al. 2007). We describe this data using *data constraints*, i.e. descriptions of data that are semantically enriched by their definition in COSMO. These constraints include the *type* of data, *range* of its values, *origin*, *destination*, and a specific *time* interval. For example, a component can only receive/send data of a given type and in a given interval, arriving from a specific semantically enriched origin, departing to a semantically enriched destination, or arriving at a specific local time. To describe non-primitive data types, we propose a general *class* data constraint. We employ data constraints in discovery to identify the components most suitable to a user query.

As discussed above, capturing domain specific knowledge is of crucial importance both in model discovery, and in semantic validation (Szabo and Teo 2009). We propose the COSMO (COmponent Simulation and Modeling Ontology) ontology for describing component-oriented simulation within and across application domains to support discovery, reuse, and validation of the composed models (Teo and Szabo 2008). The COSMO hierarchy spans two main directions, as shown in Figure 1(a). To achieve generality across domains and at the same time support specific domain requirements, we include a model component oriented hierarchy, followed by a domain-oriented hierarchy. The first level of COSMO describes components with respect to their *attributes* and *behavior*, as discussed in Section 3.1. The classes for attribute, behavior, worldview, transition, state etc., are defined in the ontology. The component repository consists of shared components common to all domains (i.e. *HierarchicalComponents*), and shared components specific to each domain (i.e. *BaseComponents*).

The second level of the COSMO ontology describes base components specific to each domain. The addition of the Queueing Networks domain to CoDES is reflected in COSMO as shown in Figure 1(b). *QN-BaseComponent* is added as a new subclass of BaseComponent, with *Source*, *Server*, and *Sink* as its subclasses. Besides having attributes and behavior like their superclass, the Source component must have an *interArrival-TimeMean* attribute and a arrival time *Distribution*, and the Server component a *serviceTimeMean* attribute, a service time *Distribution*, a *numServiceUnits* attribute, and a service *Policy*. *has_IATime*, *has_SrvTime*, and *has_ArrivalTimeDistr*, *has_ServiceTimeDistr* are subproperties of *has_Attribute* and *has_Distribution* respectively. Every newly added base component in the repository must have a corresponding instance



(a) Scalable COSMO Ontology Structure (b) COSMO Ontology for Queueing Networks

Figure 1: Ontology for Component-based Simulation Development

in the COSMO ontology provided by the component developer. Furthermore, new component attributes must be related to previously existing attributes. For example, a new component could have a different attribute name for the "hasIATime" property, e.g "hasInterArrivalT", which should be defined. The COSMO Ontology is written in OWL DL and is developed using Protégé.

4 PROPOSED APPROACH

We propose a two-stage model discovery and selection approach, as shown in Figure 2. In *model discovery*, we attempt to locate models based on syntax information derived from the conceptual model, by using a fast DHT-based repository overlay. Model discovery returns candidates whose syntax is the same as the query. In semantic-based *model selection*, we rank the candidates based on their semantic similarity to the user query, and consider partial matches as well.

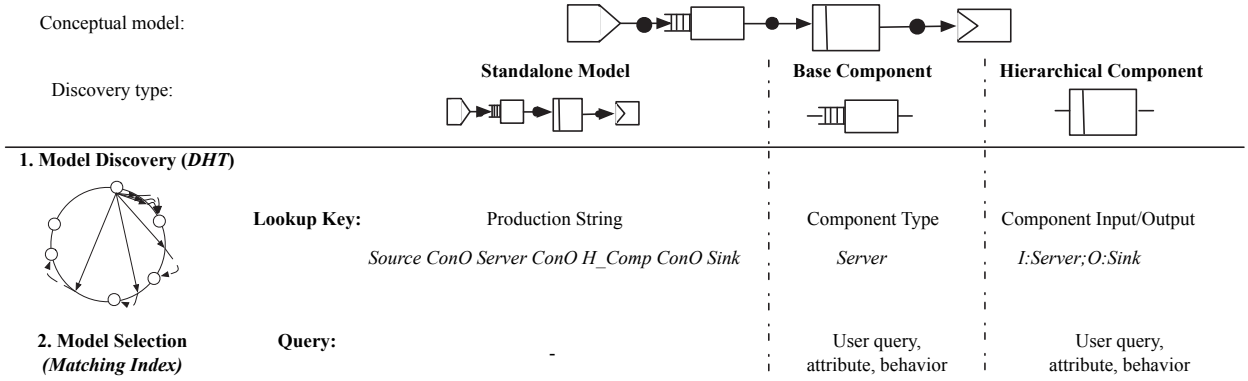


Figure 2: Overview of Model Discovery and Selection

4.1 DHT-based Model Discovery

The advances of Internet-based technologies have made possible repositories where model components reside at different administrative domains and geographical locations. Towards a scalable and efficient discovery process, we propose the use of Distributed Hash Tables (DHT) overlays to represent the CoDES model repository. A DHT is an *overlay network* of nodes responsible for several keys in a keyspace (Balakrishnan et al. 2003). Each key is mapped to a node using some variant of consistent hashing, which results in a structure that is stable to node failure. Each node maintains a set of links to other nodes in a

finger table, with the property that for each key k, the node either has a link to another node responsible for k, or a node that will bring the query closer to key k.

In our proposed approach, a DHT node represents a repository location where various models, i.e. standalone models, base components, and hierarchical components, are stored. The DHT lookup, i.e. the operation of finding models in the repository, is performed based on a key. As shown in Figure 2, we propose a novel way for performing lookup by using the conceptual model to derive the information used to construct the key. As it can be seen, we employ the *production string* that describes the conceptual model in syntactic verification for composed models, the *component type* for base components, and the *component input/output* for hierarchical components. The component input/output for hierarchical components is consistent with the current black-box representation of hierarchical components, but a white-box representation, in which the user provides the inner production string of the hierarchical component, can also be used. The hashcode of the constructed key is used to query the DHT overlay. This results in a fast process since the average lookup time in a DHT is $O(logN)$, where N is the number of nodes in the overlay (Stoica et al. 2001).

For simplicity, consider the single-server queue model in Figure 3 and assume the entire composed model is discovered. The production string for this model is *Source ConO Server ConO Sink*. This production

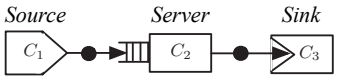


Figure 3: Single-Server Queue Model

string is hashed into a 128-bit key. Assume for simplicity that the hashcode for the production string is `54` and that `node 8` initiates the query in the DHT overlay as shown in Figure 4. A standalone model with this production string exists in the repository, and its hash value (54) is mapped on `node 56`. `Node 8` will forward the query to the nearest node to the key as seen in its finger table, in this case `node 45`. This is shown in Figure 4 with a double line. Since `node 45` is the predecessor of key `54`, `node 45` will forward the query to its successor, `node 56`, which will return the standalone model to `node 8`.

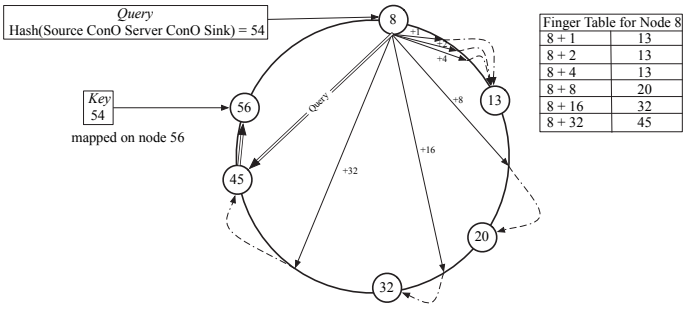


Figure 4: Example Lookup of Production String in a Distributed Hash Table

An important point to highlight is that there can be many models with the same production string but with different model components, in term of attributes and behavior. This is because the production string captures only the structure or *syntax* in terms of the component connection in the conceptual model, and not the *semantics* in terms of characteristics and behavior. If more than one model exists, the list of candidates is returned to the user, who can then select the appropriate candidate based on individual component characteristics as well as validation information stored with the standalone model.

The reduced query time for DHT is appealing for large model repositories. Currently, the CoDES repository contains `3,907` models, where `1,949` are semantically valid standalone queueing networks models, and `1,958` are base and model components for the Queueing Networks domain, where `1,949` are hierarchical components that contain between 4 and 10 base components connected in different ways, and 9 are base components, 3 each for the types *Source*, *Server*, and *Sink* respectively. This repository

models a real-life scenario where we expect the number of base components to be significantly small. We constructed this repository by using the composition grammar to generate syntactically correct models, which were then transformed into semantically valid models by providing behavior and attributes for each component type in the production strings.

We employ FreePastry (Mahajan, Castro, and Rowstron 2003) to build a DHT overlay with ten nodes, which are assigned random ids within the same keyspace as the query. For this example, the average query time over ten runs is runs is `8 milliseconds`. In contrast, a naive string comparison on a centralized repository takes around `2 minutes`. The use of the information from the conceptual model for the DHT lookup key also reduces the number of comparisons performed. For example, the query for component C_2 will locate only components of type *Server*, resulting in 3 candidates, which represents a decrease of `99.99%`. However, the conceptual model is not as helpful in the case of hierarchical components for the above repository. If the user is willing to specify the production string as a key, i.e. in a white-box abstraction, then the number of candidates is smaller, and consequently, selection time is reduced.

4.2 Semantic-based Model Selection

Model selection ranks the candidates returned by discovery based on the attributes and behavior specified in the user query, including exact and partial matches. We propose a *Matching Index* (MI) to quantify the semantic relevance of the repository components with respect to the query component. *Semantic relevance* refers to the relationships between component attributes and the component behavior in terms of states and input/output transformations. *MI* is a weighted sum of partial matching indexes.

Definition 1 (Matching Index) Let $Q = \langle R_q, A_q, B_q \rangle$ denote a query component, and $C = \langle R_c, A_c, B_c \rangle$ a repository component. The matching index $MI(Q,C)$ is defined as

$$MI(Q,C) = w_r * MI_r(R_q,R_c) + w_a * MI_a(A_q,A_c) + w_b * MI_b(B_q,B_c)$$

where $MI_r, MI_a, MI_b \in [0,1]$ are matching indexes for the required attributes, specific attributes and behavior respectively, and $w_r, w_a, w_b \in [0,1]$ are their respective weights.

Definition 2 (Required Attributes Matching Index) Let R_q and R_c be the set of required attributes for the query component Q and the repository component C respectively, where $R = \{r | r = (name, value)\}$. The matching index MI_r of the required attributes is defined as $MI_r(R_q,R_c) = \frac{\sum_{r_q \in R_q} m(r_q,r_c)}{|R_q|}$, where $r_c \in R_c$, with $name(r_q) = name(r_c)$, and $m(r_q,r_c)$ is the matching function between the attributes defined as

$$m(r_q,r_c) = \begin{cases} 1 & \text{if value}(r_q) = \text{value}(r_c) \\ 0 & \text{otherwise} \end{cases}$$

MI_r determines how many of the required attributes values of the repository component are *the same* with the query required attributes values, since the values of these attributes will not be modified. Currently the matching function computes exact matches, but linguistic algorithms could also be employed.

Definition 3 (Component Attributes Matching Index) Let A_q and A_c be the set of component attributes for the query component Q and the repository component C respectively, where $A = \{a | a = (name, value)\}$. The matching index MI_a between the required attributes is defined as $MI_a(A_q,A_c) = \frac{\sum_{a_q \in A_q, a_c \in A_c} m(a_q,a_c)}{|A_q|}$ where $m(a_q,a_c)$ is the matching function between the specific component attributes defined as

$$m(a_q,a_c) = \begin{cases} 1 & \text{if name}(a_q) = \text{name}(a_c) \\ 0.75 & \text{if name}(a_q) = \text{subPropertyOf(name}(a_c)) \\ 0.5 & \text{if subPropertyOf(name}(a_q)) = \text{name}(a_c) \\ 0 & \text{otherwise} \end{cases}$$

MI_a determines if the repository component can be described using similar attributes, by looking at the attribute *name* and not the value as previously, since specific attributes will be parametrized. MI_a compares specific attributes names based on their relation in the COSMO ontology. If a_c, the repository component's attribute, is a *subProperty* of a_q, that is a_c is more "specialized" than a_q, then $m(a_q, a_c)$ is less (0.5) than when a_q is more specialized than a_c (0.75), as shown in Table 1.

Table 1: Example Component Attributes Matching Index Calculation

Query Component Attribute	**Repository Component Attribute**	$m(a_q, a_c)$	**Comments**
$(interArrivalTime, 5)$	$(interArrivalTime, 20)$	1	$name(a_q) = name(a_c)$; same attribute name. Value not important as it can be changed by the user.
$(interArrivalTime, 5)$	$(waitingTime, 40)$	0.75	$name(a_q) = subPropertyOf(name(a_c))$ *interArrivalTime* is more specialized than *waitingTime*
$(interArrivalTime, 5)$	$(interArrivalExponentialTime, 3.4)$	0.5	$subPropertyOf(name(a_q)) = name(a_c)$ *interArrivalExponentialTime* is too specialized.

Ideally, MI_b should perform state-machine matching between the query and the repository component. However, the process of specifying the state-machine query is tedious and error-prone. As such, MI_b considers currently the input/output data constraints of the two components described in Section 3, since this is the basic information the user could provide. In an ideal situation in which the user is willing to provide more complete query information, MI_b can be extended to include state machine similarity, attribute conditions, etc. For example, Mahmood et al. (2009) propose to execute pairs of state machines and evaluate their equivalence based on the sequence of transitions. The notation, $\vee$, represents or, $\subseteq$ represents inclusion for data constraints of type *range*, and subsumption for the rest.

Definition 4 (Behavior Matching Index) Let B_q and B_c be the behavior of the query component Q and the repository component C respectively. Let IC_q, OC_q, IC_c, and OC_c be the set of constraints on the input and output data for the query and repository component, where a set of constraints is defined as $C = \{c = (type, value) | type \in \{range, origin, destination, class\}\}$ is a constraint set. The behavior matching index MI_b is defined as $MI_b(B_q, B_c) = \dfrac{\sum_{c_q \in IC_q, c_c \in IC_c} m(c_q, c_c) + \sum_{c_q \in OC_q, c_c \in OC_c} m(c_q, c_c)}{|IC_q| + |OC_q|}$ where $type(c_q) = type(c_q)$, and $m(c_q, c_c)$ is the constraint matching function defined as

$$m(c_q, c_c) = \begin{cases} 1 & \text{if value}(c_q) = \text{value}(c_c) \text{ and constraint\_type}(c_q) = \text{constraint\_type}(c_c) \\ 0.75 & \text{if typeOf(value}(c_c)) = \text{value}(c_q) \text{ and constraint\_type}(c_q) = \text{constraint\_type}(c_c) \\ & \text{and constraint\_type}(c_q) = (\text{origin} \vee \text{destination}) \\ 0.75 & \text{if value}(c_c) \subseteq value(c_q) \text{ and constraint\_type}(c_q) = \text{constraint\_type}(c_c) = range \\ 0.75 & \text{if type(value}(c_c)) \subseteq value(c_q) \text{ and constraint\_type}(c_q) = \\ & \text{constraint\_type}(c_c) = (type \vee class)) \\ 0.5 & \text{if typeOf(value}(c_q)) = value(c_c) \text{ and constraint\_type}(c_q) = type(c_c) \\ & \text{and constraint\_type}(c_q) = (origin \vee destination) \\ 0 & \text{otherwise} \end{cases}$$

MI_b encapsulates semantic information by evaluating the relations between the various data constraints using the value of the *typeOf* predicate in the COSMO ontology. $m(c_q, c_c)$ returns a higher value (0.75 or 1) when c_q subsumes c_c (c_c is stricter than c_q and C is a "specialized" version of Q) and lower values (0 or 0.5) otherwise, and in a similar manner for constraints of type "range", as shown in Table 2. $m(c_q, c_c)$ can be easily extended to include various data constraint types.

Table 2: Example Behavior Matching Index Calculation

Query Data Constraint	**Repository Data Constraint**	$m(c_q, c_c)$	**Comments**
$(range, [3.5, 5.9])$	$(range, [3.5, 5.9])$	1	Identical types and values.
$(origin, Server)$	$(origin, SingleUnitServer)$	0.75	Identical constraint types, of kind "origin". Repository component data is more specialized; $typeOf(value(c_c)) = value(c_q)$
$(class, Job)$	$(class, CustomerJob)$	0.75	Identical constraint type, of kind "class". Repository component has a more specialized data type;$type(value(c_c)) \subseteq value(c_q)$
$(type, double)$	$(type, int)$	0.75	Identical constraint type, of kind "type". Repository component has a more specialized primitive data type; $type(value(c_c)) \subseteq value(c_q)$
$(destination, SingleUnitServer)$	$(destination, Server)$	0.5	Identical constraint type, of kind "destination". Query requires more specialized data; $typeOf(value(c_q)) = value(c_c)$

4.2.1 Matching Index for Hierarchical Components

As discussed above, hierarchical components are obtained by stripping off some components from a semantically valid composed model at specific cut-off points, such that the resulting hierarchical component has both "in" and "out" communication channels. Moreover, hierarchical components are stored as black-boxes, and are only described by the input/output data constraints of the inner components at the input/output points and the hierarchical component production string. For the ranking of such hierarchical components based on a user query, we propose to look at mandatory attributes, and at input/output data that is received/sent by the component(s) at cut-off points. Thus the formula becomes:

$$MI_h(Q,C) = w_r * MI_r(R_q, R_c) + w_b * MI_b(B_q, B_c)$$

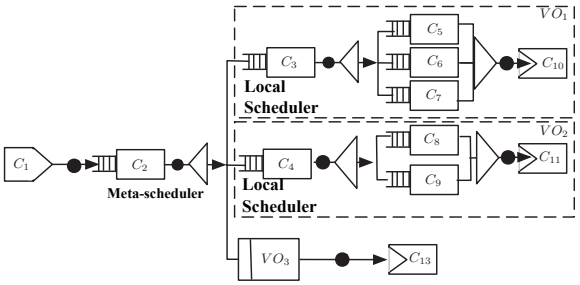
The calculation of MI_r and MI_b is the same as above, except that MI_b considers data constraints at the cut-off points. For example, considering a hierarchical component composed from C_3, C_5, C_6, and C_7 as in Figure 5, the input data constraints are the input data constraints for component C_3, whereas the output data constraints are the reunion of the output data constraints for components C_5, C_6, and C_7.

5 EXAMPLE

This section presents the discovery and selection stage in the development of a model of a grid system with three virtual organizations (Foster et al. 2003). The virtual organizations (VO) share a grid meta-scheduler job queue as shown in Figure 5(a). Each VO consists of a local job scheduler and different types of computational resources. We first study the discovery of the entire model. Next, we present an example of the selection of base component C_1 and evaluate the runtime cost.

5.1 Standalone Model Discovery

The discovery of standalone models employs the conceptual model expressed as a production string to discover composed models with the same structure from the repository. The production string is shown in Figure 5(b). As discussed above, we use the hash of the production string to query a DHT overlay with nodes representing the composed models from the repository. We employ a FreePastry DHT implementation. Our application is built on top of FreePastry and executes on 10 nodes, each a Dell Poweredge Server dual quad core Intel Xeon E5320 @1.86GHz, connected by 100 MB/s Ethernet. Once all nodes are initiated, the hashes of the production string for all the composed models in the repository are published in the DHT overlay made from the 10 nodes. The production strings are assigned to the nodes with the closest hash to the hash of the production string. Once the ring is constructed, it can be queried using the usual DHT



a) Conceptual Model

```
Source ConO Server ConF = (Server ConF (Server ConJ Sink)
                                       (Server ConJ Sink)
                                       (Server ConJ Sink))
                          (Server ConF (Server ConJ Sink)
                                       (Server ConJ Sink))
                          (H_Comp ConO Sink)
```

b) Production String

Figure 5: A Grid Computing Model with Three Virtual Organizations

lookup operation (Stoica et al. 2001). For this example, the hash of the production string is `-43761353`. The query for this hash is unsuccessful and the average query time for 10 queries is `0.09` milliseconds.

An important point to highlight is that this DHT-based method facilitates a scalable discovery service because the query times are not influenced by the size of the composed model in terms of number of components and their attributes. This is because the production string is hashed over the same space regardless of its size. However, because our DHT-based method uses hashing, the discovery process will return only *exact* matches to the query. Partial semantic-based matches are calculated in model selection.

5.2 Semantic-based Model Selection

Since the discovery of standalone models in Section 5.1 did not return any candidate, the user can perform discovery of each individual model component. Table 3 presents the query information provided by the user for component C_1, compared to the component information for two repository components. Our proposed DHT lookup returns only base components whose type matches the *Source* type, which is derived from the conceptual model. Assume that two repository components, R_1 and R_2 are considered. The first repository component, R_1, has the highest matching index and is returned by the discovery service.

Table 3: Query Information for a Source Base Component

Query Component C_1				
Required Attributes	**type:** Source **description:** open source			
Specific Attributes	-			
Behavior: I/O Constraints	Input: $\emptyset$, Output: O_1 $IConstraints = \emptyset$, $OConstraints = \{O_1C_1\} O_1C_1 = \{destination = Server, range = 21:24\}$			
	Repository Component R_1	**Matching Index**	**Repository Component R_2**	**Matching Index**
Required Attributes	**type:** Source **description:** open source	$MI_r = 1$	**type:** Source **description:** open source, two classes of jobs	$MI_r = 1$
Specific Attributes	-	$MI_a = 0$	-	$MI_a = 0$
Behavior: Input/Output Constraints	Input: $\emptyset$, Output: O_1 $IConstraints = \emptyset$, $OConstraints = \{O_1R_1\}$ $O_1R_1 = \{$ *type = int, range =10:15, destination = Server*$\}$	$MI_b = \frac{0+(1+0.75)}{0+2} = 0.875$	Input: $\emptyset$, Output: O_1, O_2 $IConstraints = \emptyset, OConstraints = \{O_1R_2, O_2R_2\}$ $O_1R_2 = \{$ *class = IO_Intensive, destination = Server*$\}$ $O_2R_2 = \{$ *class = CPU_Intensive, destination = Server*$\}$	$MI_b = \frac{0+(1)}{0+2} = 0.5$
		MI(C_1, R_1) = 0.937		**MI(C_1, R_2) = 0.75**

We evaluate the runtime cost for query requests on base component C_1 (query Q_1), and on the hierarchical component VO_3, (query Q_2), measuring the repository size, the number of repository components considered, the average time taken for the calculation of the Matching Index, and the total time. Table 4 shows an average of 10 runs for this experiment. As it can be seen, the average runtime calculation of the Matching Index is similar for both base and hierarchical components. However, the performance is better in the case of base components. This is because the number of candidates returned by model discovery is high, resulting in an increased number of comparisons (around 2,000) in model selection to rank partial matches. This can be improved by using a white-box view of hierarchical components as discussed above.

Table 4: Runtime Evaluation of Queries on Base and Hierarchical Components

Query	Repository Size	# Comparisons	MI Calculation Avg. Runtime (*s*)	Total Runtime (*s*)
Q_1 (base component - *Source*)	1958	3	0.02	0.10
Q_2 (hierarchical component)	1958	1949	0.03	68.21

6 CONCLUSION

In this paper we presented an approach for model discovery and selection to facilitate simulation model composability. We focus on two types of discovery, namely, composed models that are saved into the repository for reuse "as-is", and model components, that are defined as base and hierarchical components. Our proposed approach is divided in two stages and focuses on syntax and semantics. Firstly, *model discovery* is achieved using a DHT overlay to store and lookup models. In contrast with string-based discovery, our production string model representation significantly reduces model lookup time. Secondly, in *model selection*, we distinguish model similarity using a semantic-based *matching index*. The matching index is derived based on model attributes and behavior, defined by our proposed ontology. Experimental analysis conducted on a 1.83 GHz Dell PowerEdge cluster of ten servers with a repository of 4,000 models reveals two key observations. Our model lookup takes less than one millisecond on average as compared to two minutes using a naive string comparison. In model selection, the key tradeoff is between ease of query representation and the computational cost of ranking similar models. While it is easy to specify a query for a black-box component, the number of potential matches is much larger and took one minute to rank when compared with a white-box abstraction of one millisecond. Our proposed approach can be seen as a first step in the discovery and selection of partial matching models, an important step for component-based model development. Other methods of improving lookup performance include the use of heuristics to identify structurally equivalent models and the application of data mining techniques such as clustering. Lastly, the matching index can be further improved to achieve better precision and recall.

REFERENCES

Aronson, J., and P. Bose. 1999. "A Model-Based Approach to Simulation Composition". In *SSR*, edited by M. Jazayeri, A. Mili, and R. Mittermeir, 73–82.

Baeza-Yates, R., and B. Ribeiro-Neto. 1999. *Modern Information Retrieval: The Concepts and Technology behind Search*. ACM Press.

Balakrishnan, H., M. Kaashoek, D. Karger, R. Morris, and I. Stoica. 2003. "Looking Up Data in P2P Systems". *Communications of the ACM* 46:43–48.

Berners-Lee, T., J. Hendler, and O. Lassila. 2001. "The Semantic Web". *Scientific American* 284:28–37.

Chreyh, R., and G. Wainer. 2009. "CD++ Repository: An Internet Based Searchable Database of DEVS Models and Their Experimental Frames". In *Proceedings of the Spring Simulation Conference*, edited by G. Wainer and M. Chinni, 13–21. San Diego, USA.

Davis, P., P. Fishwick, C. Overstreet, and C. Pedgen. 2000, December. "Model Composability as a Research Investment: Responses to the Featured Paper". In *Proceedings of the 2000 Winter Simulation Conference*, edited by J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, 1585–1591. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.: Institute of Electrical and Electronics Engineers, Inc.

Foster, I., C. Kesselman, and S. Tuecke. 2003. *Grid Computing*. John Wiley.

Henkel, R., L. Endler, A. Peters, N. L. Novere, and D. Waltemath. 2010. "Ranked Retrieval of Computational Biology Models". *BMC Bioinformatics* 11:1471 – 1490.

Kasputis, S., and H. Ng. 2000, December. "Composable Simulations". In *Proceedings of the 2000 Winter Simulation Conference*, edited by J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, 1577–1584. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.: Institute of Electrical and Electronics Engineers, Inc.

Li, Y., Z. A. Bandar, and D. McLean. 2003. "An Approach for Measuring Semantic Similarity between Words Using Multiple Information Sources". *Transactions on Knowledge and Data Engineering* 15:871–882.

Mahajan, R., M. Castro, and A. Rowstron. 2003. "Controlling the Cost of Reliability in Peer-to-peer Overlays". In *International Workshop on Peer-to-Peer Systems*, edited by M. F. Kaashoek and I. Stoica, 21–32. Berkley, USA.

Mahmood, I., R. Ayani, V. Vlassov, and F. Moradi. 2009. "Statemachine Matching in BOM Based Model Composition". In *Proceedings of the IEEE Symposium on Distributed Simulation and Real-Time Applications*, edited by S. J. Turner, D. Roberts, W. Cai, and A. E. Saddik, 136–143.

Moradi, F., R. Ayani, S. Mokarizadeh, G. H. A. Shahmirzadi, and G. Tan. 2007. "A Rule-based Approach to Syntactic and Semantic Composition of BOMs". In *Proceedings of the IEEE Symposium on Distributed Simulation and Real-Time Applications*, edited by D. J. Roberts, G. K. Theodoropoulos, and A. E. Saddik, 145–155. Crete, Greece.

Oses, N., M. Pidd, and R. J. Brooks. 2004. "Critical Issues in the Development of Component-based Discrete Simulation". *Simulation Modelling Practice and Theory* 12:495–514.

Pidd, M. 2004, December. "Simulation Software and Model Reuse: A Polemic". In *Proceedings of the 2004 Winter Simulation Conference*, edited by R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, 772–775. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.: Institute of Electrical and Electronics Engineers, Inc.

Silver, G., K. R. Bellipady, J. Miller, and K. J. Kochut. 2009, December. "Supporting Interoperability using the Discrete-event Modeling Ontology (DeMO)". In *Proceedings of the 2009 Winter Simulation Conference*, edited by M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and R. G. Ingalls, 1399–1410. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.: Institute of Electrical and Electronics Engineers, Inc.

Stoica, I., R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. 2001. "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications". In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, edited by J. Crowcroft, 149–160. New York, USA.

Szabo, C., and Y. Teo. 2007. "On Syntactic Composability and Model Reuse". In *Proceedings of the International Conference on Modeling and Simulation (invited paper)*, edited by A. Abraham, 230–237. Phuket, Thailand.

Szabo, C., and Y. Teo. 2009. "An Approach for Validation of Semantic Composability in Simulation Models". In *Proceedings of the 23rd ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation*, edited by P. Reynolds, 3–10. New York, USA.

Tansalarak, N., and K. Claypool. 2005. "Finding a Needle in the Haystack: A Technique for Ranking Matches Between Components". In *Proceedings of the Component-based Software Engineering Conference*, edited by G. T. Heineman, I. Crnkovic, C. Szyperski, and K. Wallnau, 171–186. Ottawa, Canada.

Teo, Y., and C. Szabo. 2008. "CODES: An Integrated Approach to Composable Modeling and Simulation". In *Proceedings of the 41st Annual Simulation Symposium*, edited by H. D. Karatza, 103–110. Ottawa, Canada.

Tolk, A., S. Y. Diallo, and C. D. Turnitsa. 2007. "Model-based Data Engineering: Preparing a Paradigm Shift Towards Self-organizing Information Exchange". In *Summer Computer Simulation Conference*, edited by A. Tolk and L. Yilmaz, 1112–1119. San Diego, USA.

Turnitsa, C., J. J. Padilla, and A. Tolk. 2010, December. "Ontology for Modeling and Simulation". In *Proceedings of the 2010 Winter Simulation Conference*, edited by B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, 643–651. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.: Institute of Electrical and Electronics Engineers, Inc.

Uhrmacher, A. M., D. Degenring, J. Lemcke, and M. Krahmer. 2004. "Towards Reusing Model Components in Systems Biology". In *Proceedings of the Computational Methods for Systems Biology Conference*, edited by V. Danos and V. Schachter, 192–206.

Zeigler, B., H. Praehofer, and T. Kim. 2000. *Theory of Modeling and Simulation*. Academic Press.

AUTHOR BIOGRAPHIES

CLAUDIA SZABO is an Associate Lecturer at the School of Computer Science, The University of Adelaide in Australia. Her email address is claudia.szabo@adelaide.edu.au.

YONG MENG TEO is an Associate Professor with the Department of Computer Science at the National University of Singapore, and a Visiting Professor at the Shanghai Advanced Research Institute, Chinese Academy of Sciences in China.. He heads the Computer Systems Research Group and the Information Technology Unit. His email address is teoym@comp.nus.edu.sg.