A rule-based semantic matching of base object models

Farshad Moradi*

Swedish Defence Research Agency (FOI),
164 90 Stockholm, Sweden
E-mail: farshad@foi.se
*Corresponding author

Rassul Ayani and Shahab Mokarizadeh

School of Information and Communication Technology,
Royal Institute of Technology (KTH),
Electrum 229, S-164 40 Kista, Sweden
E-mail: ayani@kth.se
E-mail: shahabm@kth.se

Gary Tan

School of Computing,
National University of Singapore,
13 Computing Drive, Singapore 117417
E-mail: gtan@comp.nus.edu.sg

Abstract: Creating simulation models via composition of predefined and reusable components is an efficient way of reducing costs and time associated with the simulation model development. However, to successfully compose models one has to solve the issues of syntactic and semantic composability of components. The Base Object Model (BOM) standard is an attempt to ease reusability and composition of simulation models. However, the BOM does not contain sufficient information for defining necessary concepts and terms to avoid ambiguity, and neither does it have any method for dynamic aspects matching conceptual models (i.e., their state-machines). In this paper, we present our approach for enhancement of the semantic contents of BOMs and propose a three-layer model for syntactic and semantic matching of BOMs. The enhancement includes ontologies for entities, events and interactions in each component. We also present an OWL-S description for each component, including the state-machines. To test our approach, we specify some simulation scenarios and implement BOMs as building blocks for development of those scenarios, one of which is presented in this paper. We also define composability degree, which quantifies closeness of the composed model to a given model specification. Our results show that the three-layer model is promising and can improve and simplify the composition of BOM-based components.

Keywords: semantic matching; BOMs; base object models; composability of simulation models.

Reference to this paper should be made as follows: Moradi, F., Ayani, R., Mokarizadeh, S. and Tan, G. (2009) 'A rule-based semantic matching of base object models', *Int. J. Simulation and Process Modelling*, Vol. 5, No. 2, pp.132–145.

Biographical notes: Farshad Moradi is a senior scientist at the Swedish Defence Research Agency (FOI). He acts as the programme manager for Modelling and Simulation, and is the leader of the Simulation and Distributed Systems competence group at the Department of Informatics. He has been working on modelling and simulation for the past 14 years. He holds an MSc in Computer Science and Engineering from Chalmers University of Technology, Gothenburg, Sweden, and a PhD in Distributed Simulations from the Royal Institute of Technology (KTH). His research interests are in distributed systems, distributed and web-based modelling and simulation, embedded simulations, service-oriented architectures, computer generated forces, and logistics.

Rassul Ayani is Professor of Computer Science at the School of Information and Communication Technology (ICT), Royal Institute of Technology (KTH), Stockholm, Sweden. He received his first degree from Technical University of Vienna, his MSc from the University of Stockholm and his PhD from KTH in Stockholm. He has been working on parallel and distributed systems for the past 20 years, and his current research interests are in distributed systems, performance

analysis of computer and communication systems, distributed simulation and composability of simulation models. He is an area editor of the ACM *Transactions on Modeling and Computer Simulation* (TOMACS).

Shahab Mokarizadeh is a PhD student at the School of Information and Communication Technologies (ICT), the Royal Institute of Technology (KTH) in Sweden. He received his BSc from Ferdowsi University, Iran, and his MSc from KTH. His research interests are on SOA and distributed systems, focusing on semantic web, WS composition, and simulation model composability.

Gary Tan is an Associate Professor at the Department of Computer Science, School of Computing, National University of Singapore. He is also an Assistant Dean of Corporate Communications with the School. He received his BSc from the National University of Singapore and his MSc and PhD from the University of Manchester, UK. His research interests are in parallel and distributed machines, and parallel and distributed simulation, and he has published over 50 journal and conference papers. He is currently concentrating on crisis management and symbiotic simulation.

1 Introduction

Creating simulation models via composition of predefined and reusable components is a way to reduce the costs and time associated with the simulation model development process. This approach has been successfully deployed in manufacturing industry and software engineering. However, to successfully compose models one has to solve the issues of syntactic and semantic composability of components. Composability has been defined as

> "the capability to select and assemble reusable simulation components in various combinations into simulation systems to meet user requirements." (Weisel et al., 2003; Petty et al., 2004)

Syntactic composability is concerned with the compatibility of implementation details, such as parameter passing mechanisms, external data accesses, and timing mechanisms. It is the question of whether a set of components can be combined (Hu et al., 2003; Szabo and Teo, 2007). Semantic composability, on the other hand, is concerned with whether the models that make up the composed simulation system can be composed in a meaningful way and the composition is valid. (Weisel et al., 2003; Petty et al., 2004)

HLA is the most widely used architecture for distributed simulations today (https://www.dmso.mil). It provides a simulation environment and standards for specifying simulation parts via Simulation Object Models (SOMs) and interactions between simulation parts via Federation Object Models (FOMs). A HLA simulation is named Federation, which is composed out of Federates, or simulation parts. Through SOMs and FOMs, HLA intends to formalise how federates function and how they interact. However, SOMs and FOMs do not contain enough semantic information about what they intend to simulate and hence, have little support for semantic composability. The simulation community has recently formulated a standard, the Base Object Model (BOM), to ease reusability and composability (SISO, 2005).

In this paper, we investigate how BOMs can be used to develop simulation models in a component-based fashion and suggest a process for component-based simulation development using BOMs. We argue that even though the BOM standard looks promising and exhibits good capabilities for reuse and composability, through, e.g., its conceptual model, it lacks the required semantic information for semantic matching and composition. Moreover, BOM provides little support for defining necessary concepts and terms to avoid ambiguity, and there is no method for matching dynamic aspects of conceptual models (i.e., their state-machines). We also discuss utilisation of Semantic Web and Web Service (WS) (http://www.w3.org/2001/sw/; http://www-106.ibm.com/developerworks/webservices/) technologies for further refinement of the process and improving the semantic composition of BOMs.

The main contributions of this paper are:

- enhancement of BOM by a Semantic BOM Attachment (SBA)
- proposing a three-layer method for BOMs matching
- defining composability degree to measure goodness of the composed models
- implementation, test and analysis of the SBA and the three-layer matching method.

The rest of this paper is organised as follows. Section 2 discusses the approach we adopted for the Model composition, while Section 3 discusses the SBA. The overall architecture of our approach is presented in Section 4 and Section 5 provides a case study of a restaurant scenario. The conclusion is given in Section 6.

2 Model composition approach

To compose a simulation out of components, the components need to contain (and expose) some information about their internal structure and how they can be used.

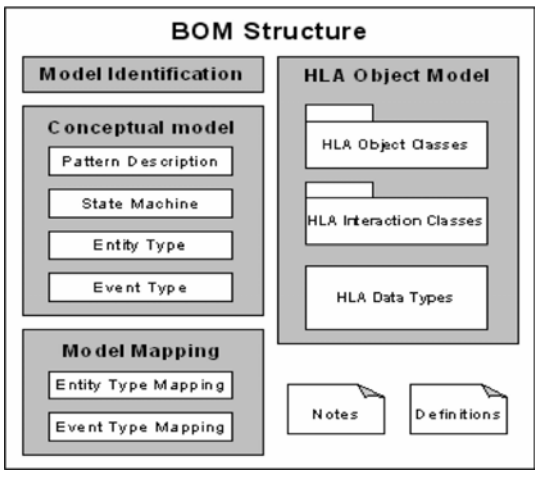
This information is called metadata and contributes to simplified use of a component by others (Morse et al., 2004).

Generally, the concepts and terminologies used in various components may vary substantially and thus can lead to misunderstanding. Hence, the concepts and terminologies should be defined in an unambiguous way to avoid misunderstandings, particularly if the composition process is automated. Ontology is used to help create a common understanding among components and to improve communication among them (Gruber, 1993). In the computer science context, ontology is a description of terminologies and frames of references between entities that interact with each other. Thus, ontology creates a shared understanding of entities and events, and contributes to reaching an agreement on meanings of what is communicated between the components. This shared understanding is the key to discover semantic mismatch despite syntactically correct matching. By adding axioms to the ontology we can use them to narrow the selection criteria and detect semantically mismatching items (Gruber, 1993).

2.1 BOM structure

A BOM is an XML document that encapsulates the information needed to describe a simulation component. The BOM concept is based on the assumption that piece-parts of simulations and federations can be extracted and reused as modelling building blocks or components. The interplay within a simulation or federation can be captured and characterised in the form of reusable patterns. These patterns of simulation interplay are sequences of events between simulation elements. BOMs are structured into four major parts (SISO, 2005) as can be seen in Figure 1, Model Identification, Conceptual Model, Model Mapping and HLA Object Model. The Model Identification contains metadata about the component. This part includes Point Of Contact (POC) information, as well as general information about the component itself, such as Type, Security Classification, Purpose, Application Domain, Use Limitations, and Keywords.

Figure 1 BOM structure



The Conceptual Model, which is our main concern here, contains information that describes the patterns of interplay of the component. This part includes the types of actions and events that take place in the component, and is described by a pattern description, a state-machine, a listing of conceptual entities and events, which correspond to how real-world objects and phenomena are modelled in the simulation. The pattern description describes the flow and dependencies of events and their exceptions. There are two additional parts in the BOMs, namely Notes and Definitions. These two parts contain semantic information about events and entities as well as actions that are specified in the Conceptual Model, and are used to provide a human readable understanding of the patterns described in the BOM.

As BOMs are very new, there is a limited toolset available. One of the most comprehensive tools available for BOM creation and modelling is BOM works (http://www.simventions.com/bomworks/) from SimVentions.

The current BOM standard lacks the required semantic information to avoid ambiguity. Furthermore, there is no method for matching state-machines in the conceptual models of different BOMs. To address the above issues, we suggest extending the BOM description with a semantic attachment through utilisation of WS Technology and OWL-S language (Web Ontology Language for Services) (http://www.w3.org/Submission/OWL-S/) (explained in Section 3). The semantic attachment provides the metadata required for discovering the composition of BOMs. The matching is performed based on a three-layer model containing the syntactic layer, static semantic layer and dynamic-semantic layer as explained in Section 4.2, utilising a set of rules for reasoning about the compositions.

2.2 Composition process

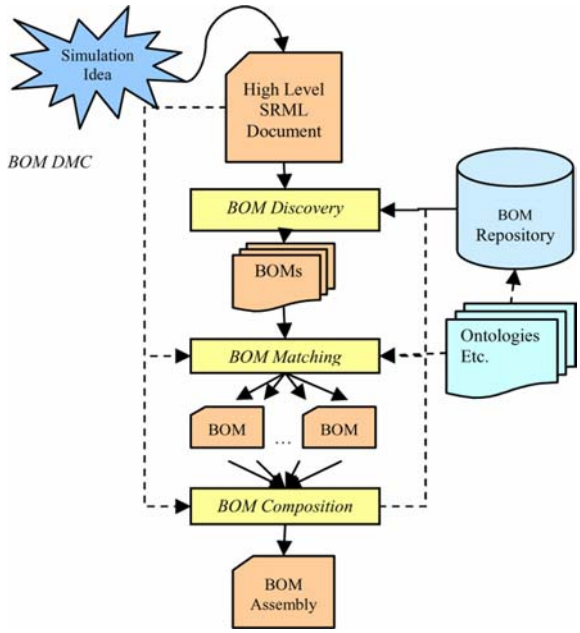
In this section, we describe our process for component-based model development using BOMs. We assume that a simulation developer describes the target scenario (simulation to be developed) in a formal manner using SRML-Light. The composition process is made up of parsing the SRML document to identify the necessary components, and then a three-phase process consisting of Discovery, Matching and Composition (DMC) of the components, as shown in Figure 2.

The BOM Discovery fetches the BOMs from a repository. This activity identifies BOMs only at a very high level. BOMs that roughly fit the intent of the simulation or match the components specified in the simulation document are simply fetched from the repository.

The BOM Matching compares the fetched set of BOMs and decides which BOMs might be suitable for the simulation. This is a more complex activity that needs to take into account the simulation intent (as described in the SRML document). One has to handle issues such as, what components fit together semantically and practically and how it is done. To compare BOMs, other means such as ontologies and reference documents will also be used in this activity. In the third phase, the selected components are

assembled into a BOM assembly. The DMC procedure is shown in Figure 2, where each of the three phases is shown in square boxes, and is further discussed in the subsequent sections.

Figure 2 BOM discovery, matching and composition (see online version for colours)



2.2.1 *An example*

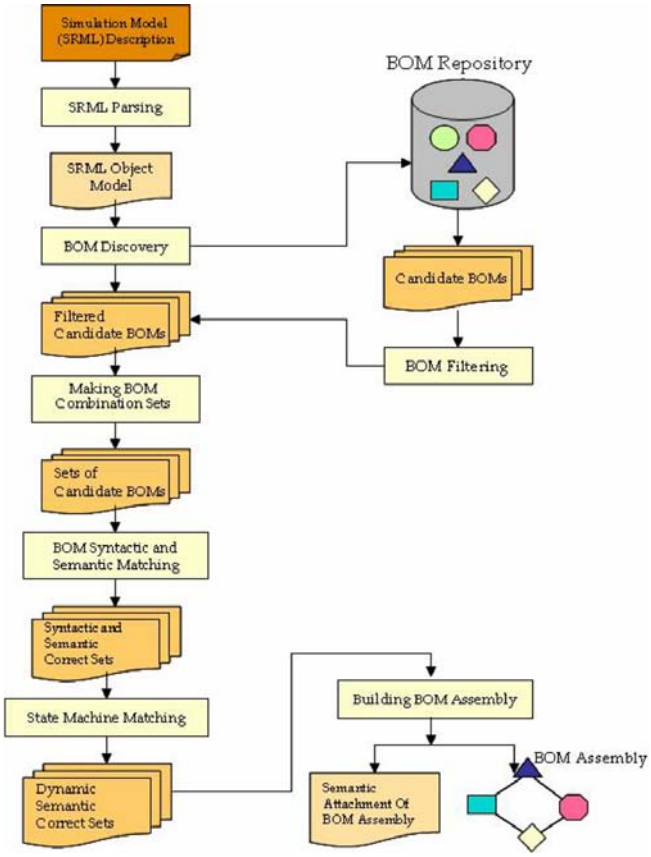
The idea of the DMC phases could be explained via a simple scenario. Assume for example that we wish to construct a model of a car that can move forward, turn left, turn right and brake. In SRML, we would specify that the car is made up out of a car body, an engine, steering, four tires and a suspension. Furthermore, there is a repository available containing many different types of tires, engines, suspensions, etc. In the Discovery phase, we identify the components needed in our model, i.e., tires, car bodies, engines and fetch them from a BOM repository. In the Matching phase, we try out these tires, car bodies and engines to see if they can be assembled together. For example, it might be that we found bus engines that will not be suitable in a small car body. In the last phase, composability, we check if the selected parts, that could be seen to fit together, can be assembled into a car that have the specified functionalities, i.e., can move forward, turn right, turn left and brake.

The composition process described above therefore comprises four phases:

- SRML Parsing
- BOM Discovery
- BOM Matching and Composition
- BOM Assembly Building.

These four phases can be further broken down into seven steps, as shown by yellow boxes in Figure 3. Here, we give a more detailed description of these four phases (seven steps).

Figure 3 The simulation development process (see online version for colours)



The first phase starts with a description of the target simulation written in SRML (http://www.w3.org/TR/2002/NOTE-SRML-2002121). The simulation model contains simulation components, and events, as connectors of those components. The SRML item classes are seen as representation of BOM candidates while events (script-tag of SRML) represent actions between components. The SRML parsing phase comprises one step, where the simulation scenario is parsed and information about candidate components is extracted. Here, we assume that a simplified version of the SRML, called SRML-Light standard is used to describe the simulation scenario (Moradi et al., 2006). The 'Item Class' tag is utilised as a heuristic to identify and extract type of the candidate BOMs. As an example, the following program shows how an Item Class Queue is written in SRML.

```
<ItemClasses Name="AllItemClasses">
            <ItemClass Name="Customer" Source="the place of CustomerBOM" />
            <ItemClass Name="Table" Source="the place of TableBOM" />
            <ItemClass Name="Waiter" Source="the place of WaiterBOM" />
            <ItemClass Name="Queue" Source="the place of QueueBOM" />
            <ItemClass Name="Kitchen" Source="the place of KitchenBOM" />
</ItemClasses>
<Item ItemClass="Queue" ID="q00" Source="the place of QueueBOM">
            <EventSink Name="QueueNeedsCustomerToJoin" EventClasses="Join" />
            <Script Name="QueueScript1"
            type="text/javascript">PostEvent(Customer,TakeSit,CustomerID,TableID);
            </Script>
            <Script Name="QueueScript2"
            type="text/javascript">PostEvent(Customer,JoinAck,CustomerID,QueueID);
            </Script>
            <EventSink Name="QueueNeedsTableToFree" EventClasses="TableIsFree" />
</Item>
```

The output of the parsing step is a collection of entity names and their corresponding send/receive events. We call this collection *SRML Object Model*. In our above example, the Queue is an entity in the *SRML Object Model* with four events, two send events, *TakeSit* and *JoinAck*, and two receive events *QueueNeedsCustomerToJoin* and *QueueNeedsTableToFree*. *CustomerID, TbaleID* and *QueueID* are parameters.

During the BOM discovery phase, a query is built based on the SRML Object Model and is sent to the BOM repository. The repository returns a set of potential candidates corresponding to the query. Afterwards, the candidate BOMs are matched syntactically (number of parameters and event name) and semantically (parameter data type and entity type) against the SRML object model and the irrelevant BOMs are filtered, see 4.1. In our example, the result of the query could be a number of *Queue* BOMs. However, not all of them will necessarily match the syntactic and semantic requirements described in the SRML object model.

The BOM matching and composition phase is more comprehensive and is about finding the right combination of components that satisfy the target simulation description (the scenario). Since the discovery step can result in variations of components, there can be more than one combination of components that may build the desired simulation. Hence, this phase starts by making different combinations of candidate BOMs. There are different methods and algorithms for generating these combinations. However, they are not covered in this paper. Next, the composer adjusts the combinations based on received feedback from syntactic and semantic BOM matching. The following steps are done for each combination.

During the next step of this phase for each interaction, stated in the simulation scenario, the syntax of messages and actions between involving BOMs in the interaction is verified. If the BOMs have consensus on syntactic composition, the semantic of BOMs is compared against each other based on the interactions, as explained in Section 4.2. For instance, a *Customer* BOM that is designed for Fastfood restaurant scenarios is probably not suitable for an Italian restaurant scenario where she or he is expected to be waited on, since it does not have the right type of interactions.

If the syntax and semantic of all BOMs in the set are correct, the state-machine composition starts. The state-machines of all components in combination are run according to the simulation scenario interactions. In a successful run, events passed between components will result in successful state transitions. After a successful run, the order of action executions will be obtained.

Finally, having in hand a right set of components, their interactions and the order of those, we enter the BOM assembly building phase during which a BOM assembly can be created from the current set of BOMs. Figure 3 shows these seven steps in a flow chart format, where each step is represented by a rectangle.

To realise the proposed process, we have defined a set of rules for checking the composability between BOMs. These rules are divided into discovery and composition rules. An inference engine utilises these rules together with BOM descriptions and related ontology to reason about composability of components. This requires that BOMs are described in a logical language and have proper structure, which facilitate the reasoning process. For this purpose, we suggest an extension to the current BOM description called *Semantic BOM Attachment*. The SBA is based on OWL-S and is explained in more detail in Section 3.

Before proceeding with the description of our matching methodology and composition rules we will explain our modelling and composition assumptions in the next section.

2.3 Modelling and composition assumptions

For implementation of the composition process, we have made the following assumptions. First of all, each simulation model is a combination of *components* and *events*. A composed model consists of a number of communicating components. These components are event-driven, which means they act upon occurrence of events. After occurrence of an event a corresponding *action* will be executed, which may cause the generation of another event. Components in a composition communicate through sending messages. A component C_i can interact with another component C_j by sending event message E_{ij} to it.

An event is something that happens in time such as receiving information about the position of a target by an airplane. But by action we mean the service, operation or action that is executed after an event has happened, e.g., the computation done by the airplane and its change of state can be considered as the corresponding action to the above event. Basically, we define an action in terms of its effects on the environment and itself. Candidates to describe such properties are pre- and post-conditions.

We also assume that an event can be initiated (a message can be sent) whenever the precondition of the corresponding action becomes true. Similarly, a message can be received (a receive event can happen) whenever the precondition of the action of respective receive event becomes true. Occurrence of an event might result in state change in the initiator and the receiver components while execution of the corresponding action results in effects stated in post-condition on itself or its environment (other components).

Besides the above concepts, there are some assumptions regarding the way the composition of actions is done. First, composition check is done through matching of the SBAs, which will be described in detail in Section 3.

To avoid any confusion, we assume that each BOM with corresponding SBA represents only one entity type. In other words, we assume a one-to-one mapping between BOMs and entity types. Hence, the terms component and entity are used interchangeably in this paper. There is also

a one-to-one mapping between each event and the corresponding action.

Finally, the *Horizontal* combination of actions is assumed. In our approach, we do not consider other models such as *Vertical* combination. Horizontal and Vertical combinations are fully described in Medjahed and Bouguettaya (2006). Two operations/actions can be combined *Horizontally* if they model a supply-chain-like combination. We have adopted the original definition to the concepts and conditions in our work. We use same definition for Horizontal composition but with an extra condition. To describe our adopted version of Horizontal composition, we need to explain the *In* and *Out* mode of actions. The *mode* of an action indicates whether the action initiates an interaction, via sending a message, or it is invoked as a result of receiving a message. If initiating a send event results in execution of an action, this action is in *out* mode. Similarly, capturing a message leads to immediate execution of corresponding action which by our definition is in *In* mode. So an action is either in *In* mode or *Out* mode. The mode cannot be changed dynamically and it is assigned to the action during SBA development.

The Horizontal composition should be a combination of an *Out* action belonging to message sender entity with the respective *In* action of the entity (ies) receiving that message. Since getting a message cannot happen before posting/sending that message, the first action in the Horizontal combination should be the *Out* action of a send event, and the second one should be the *In* action of a receive event.

2.4 Matching methodology

As explained in the previous section, we consider simulation models as combinations of components and events. Therefore, to have a successful composition of components these two items should match both syntactically and semantically. Our approach for composition is based on 'event matching'. Components are checked against each other both syntactically and semantically according to the interactions they have in the composition. The interactions in discrete event simulations are through messages that are sent and received. Our aim is to make sure that both sender and receiver components have a common understanding (syntactic and semantic) of the transmitted message, i.e., the message transmitted by a send event to the designated target will be captured and handled correctly by the target component via respective receive event.

Furthermore, in a semantically correct composition, each component should send and receive events designated to it. The challenge here is to make sure that the components are capable of 'following' the interactions stated in the simulation model, i.e., they are capable of executing the interactions stated in the simulation model in the right order. This problem can be solved through utilisation of state-machines. State-machine of each BOM represents the events that a component can send or receive in each state. One can start from the initial state and run each state-machine based on the events it can send (stated in the simulation model) or receive (again stated in the simulation model). However, the practical solution is a bit more complicated since the state-machines of all the components involved in the composition need to be run and compared against each other while the events (sent or received) are handled by the components. In a successful run, components must be in the 'correct' state when events are executed.

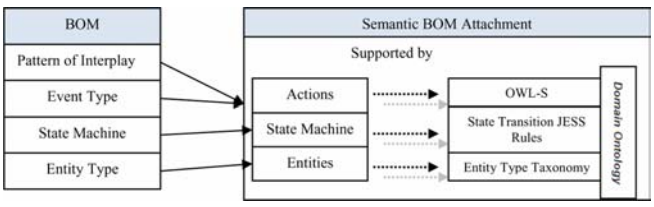
3 Semantic BOM attachment

To facilitate semantic matching and composition of BOM-based components, we suggest an enhancement of the current BOM description through utilisation of the Semantic WSs concepts and methods for composition, here referred to as SBA. For this purpose, we have mapped the BOM descriptions into the OWL-S (http://www.w3.org/Submission/OWL-S/) upper ontology. The main reason for this mapping is to use OWL (Web Ontology Language) (http://www.w3.org/2004/OWL/) as the underlying language for describing BOMs and also utilise the language features of OWL-S to improve the semantic expressiveness of BOMs and hence, facilitate their semantic discovery and composition.

As mentioned earlier, a BOM consists of three main parts: Model Identification, Conceptual Model and Model Mapping. Model Identification and Model Mapping sections of BOMs provide information about the BOM, entities, their attributes, interactions and their parameters. However, the metadata of a BOM is mainly embedded in its Conceptual Model. Semantic information is added to each item of the conceptual model such as data type hierarchy and unit. Entity type taxonomy is also introduced for each item. Here, we might use information provided by the Model Identification, such as 'Purpose' and 'Application Domain'. The product is a semantic attachment for each BOM. Figure 4 depicts the items of BOM metadata converted into SBA plus the supportive ontology.

In the following sections, we explain our approach for adding semantic information to each item of the conceptual model.

Figure 4 Semantic BOM attachment (see online version for colours)



3.1 Mapping pattern of interplay

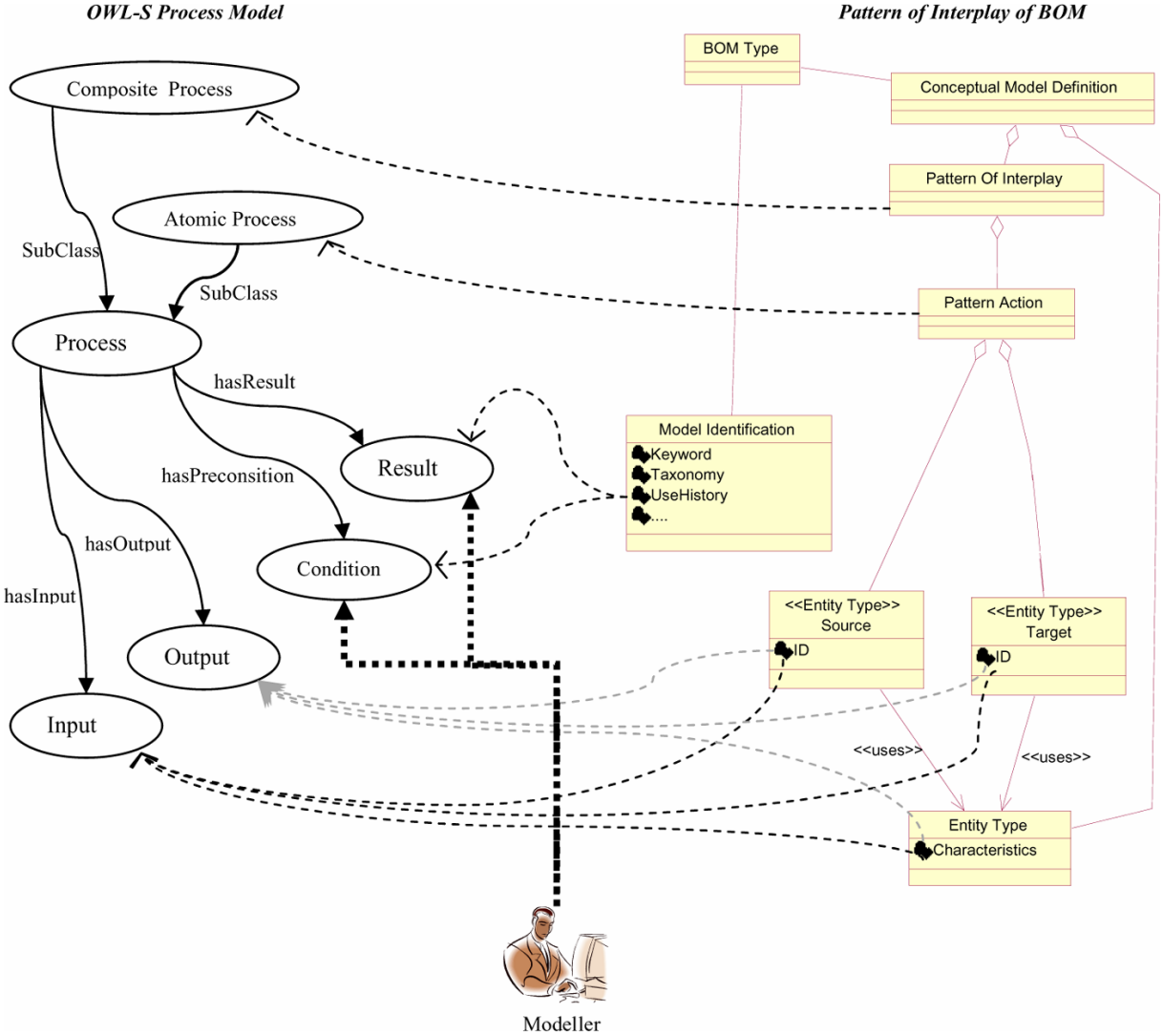
The pattern of interplay in BOM provides a mechanism for defining a sequence of actions.

Each event type in BOM is associated with an *action*, which contains information about the sender entity, receiver entity and content of the message being sent. A one-to-one mapping pattern of interplay constitutes (action and entity)

to Process Model of OWL-S upper ontology is presented in Figure 5.

In this figure, the BOM pattern of interplay is presented by UML class diagram on the right side of the figure, and the OWL-S process model on the left side of the figure. The dashed lines reveal the mapping direction of constitutes of the BOM pattern of interplay to corresponding ones in OWL-S process model.

Figure 5 Mapping pattern of interplay to OWL-S process model (see online version for colours)



When mapping actions to the OWL-S upper ontology, each action is mapped to an atomic process since both by definition represent a single step task. Actions are resulted from occurrence of events and there are two categories of events in BOM: directed events (the intended receivers are specified) or undirected events (the intended target entities are not specified). In BOM, the former is known as Message, and the latter as Trigger. In this work, only the action of a BOM Message is discussed (and mapped). The mode of an action indicates whether the action is the result of sending a message (Out mode) or of receiving a message (In mode). In case of Out actions, the input of atomic process will be empty whereas the output of atomic process will be empty for In actions. This issue is shown in Figure 5 by grey vs. black dashed lines coming into input and output of process model, respectively. What is mapped to input or output of process model is the BOM event content consisting of identifiers of event initiator and event target entities and related attributes of the involved entities in simulation scenario.

In addition, an action in the OWL-S process model will be annotated by a pre- and post-condition pair. The reason is to show the effect which the execution of an action has on its environment. Such effects represented by pairs of pre-condition and post-condition, helps the composer to find a suitable component to which its action(s) matches the current environment facts. BOM messages are not decorated with any type of condition. Hence the pre-/post-conditions should be either automatically extracted from other parts of BOM like Model Identification by an intelligent agent or to be defined manually by the component developer (shown as Modeller in Figure 5).

Consequently, the pattern of interplay can be assumed as a composite process in OWL-S or even a service model, as was suggested in Moradi (2007).

3.2 State-machine mapping

> "The state machine template component provides a mechanism for identifying the behaviour states expected to be exhibited by one or more conceptual entities." (SISO, 2005)

The state-machine of an entity can basically be seen as a transition from one state to the next upon occurrence of an event. When composing BOM components, the state-machines of the components are matched against each other. To verify matching of the state-machines, the Jess rule engine (http://herzberg.ca.sandia.gov/jess/) has been used. Hence, we need to transform the state transitions rules of the BOMs into the Jess rule format. Jess is a rule engine and scripting environment, which is used for building one type of intelligent software called Expert Systems. The BOM state transitions can be described as Jess rules such that the head of the rule states the 'current state' condition while the body states the 'next state' assertion. The conversion from BOM state-machine format into Jess rule format can be done automatically, since a Jess rule has a static template. The template is filled by component name, current and next state instances. The resulting Jess rule can be stored together with other OWL items in an OWL file. The following program illustrates transition of a Customer entity state-machine from 'Ready' state to 'Waiting' state written in the Jess rule format.

(defrule Rule-Customer-Send-Join

(object (is-a Customer) (OBJECT ?objCustomer)

(:NAME "Customer_Inst") (hasCurrentState ?state&:(eq (isInstnceName ?state "*Customer_Ready***") TRUE)))***

=>

(slot-set ?objCustomer hasCurrentState Customer_Waiting))

3.3 Entity type mapping

> "The entity type template component provides a mechanism for describing the types of conceptual entities used to represent senders and receivers identified within a pattern of interplay and carry out the role of conceptual entities identified within a state machine." (SISO, 2005)

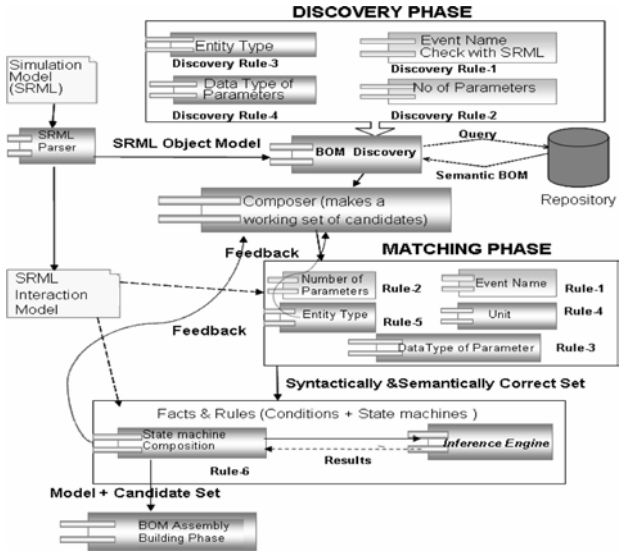
Taxonomy of entity types can be defined using, for instance, information provided by the component's Model Identification part. The attributes of an entity are annotated with semantic concepts, for example, by defining unit for the attributes supported by a hierarchy of units in the ontology. As already mentioned, since pre- and post-conditions are defined over these attributes, more enriched and semantically expressive attributes are developed.

4 Overall architecture

After describing our assumption, composition methodology and the SBA, we now explain our implementation of the proposed composition process and the rules that we have defined to check the composability of BOM-based components. Figure 6 depicts the proposed architecture for implementing the component-based simulation model development process. The two main phases, namely discovery and matching, are highlighted and the applicable rules (if any) at each phase are shown. Obviously, the syntax checking rules are applied before semantic counterparts.

In the following sections, we will describe these two phases in more detail and explain the rules that are being defined and applied at each phase.

Figure 6 Proposed Architecture for implementing the component-based model development process



4.1 Discovery phase

Although the primary focus of this work is not implementing the discovery phase, we briefly describe how BOMs can be extracted from a repository. In Table 1, we present the discovery rules and the items, which are being examined. As described earlier, the SRML object model contains the names of BOMs and their interactions. This is mainly syntactical information. The discovery module will look for any BOM containing entity (ies) with name (s) stated in the SRML object model. Consequently, a set of BOMs will be retrieved. The discovery rules are later employed to filter out irrelevant BOMs by comparing their signature (event name, number of parameters, entity type and message data type) with those stated in the simulation model (SRML Object Model). *Discovery Rule-1* compares for each BOM the name of events initiated by or targeted to that BOM (an entity in the BOM) and filters out BOMs lacking those interactions. Next, the number of parameters for a message carried by each event is checked by *Discovery Rule-2*, and mismatching ones will be removed.

Table 1 Discovery rules

Discovery layer	*Feature to be checked*	*What is checked*	*Rule*
Syntactic	*Action*	*Event Name*	*D-Rule-1*
	Message	*Number of parameters*	*D-Rule-2*
Semantic	*Entity*	*Entity Type*	*D-Rule-3*
	Message	*Data Type*	*D-Rule-4*

This is a very naive discovery algorithm. However, it can be improved if some semantic filtering is also applied during the discovery phase. A hierarchy of entity types can be defined in the Ontology as part of the SBA so that, one can also retrieve all the BOMs containing entities which are sub-class or super-class of the queried entity type (Discovery Rule-3). More BOMs would be filtered out if we check the data type of message parameters (Discovery Rule-4). In that case, we need to state the data type of message parameters in the simulation model. The comparison is done by having taxonomy of data types, defined in the SBA. The accuracy of the discovery process is dependent on the amount of metadata and semantic information that each component contains and exposes, the search criteria, and the structure of the component repository. The latter is out of the scope of this paper.

4.2 A three-layer matching method

The idea of composability stack is inspired by the work done by Medjahed and Bouguettaya (2006). In Medjahed and Bouguettaya (2006), the authors define Syntactic, Static-Semantic and Dynamic-Semantic attributes as the following:

> "Syntactic attributes represent the structure of a service operation. An example of syntactic attribute is the list of input and output parameters that define the operation's messages. The semantic attributes are divided into two groups:"
>
> "Static Semantic attributes describe features that are not related to the execution of the operation. Dynamic-semantic attributes refer to the way and constraints under which the operation is executed. An example of dynamic attributes is the business logic of the operation, i.e., the results returned by the operation given certain parameters and conditions."

Even though we use the same terminology as in Medjahed and Bouguettaya (2006), our definitions of the layers are slightly different. Syntax layer rules verify whether two actions can be combined syntactically or not. The static semantic rules compare the semantic values of the actions, messages and entities via their supportive ontology. Since this ontology information is unchanged (static) during the action execution, the operating rules are called Static Semantic rules. Dynamic-semantic rules verify the BOM matching during action execution. Since the rules deal with pre-/post-condition and state-machine of BOMs (the conditions and state-machine are supported by the ontology), it is called Dynamic-Semantic layer.

Our composability stack is presented in Table 2. There are some composition rules in each layer of the stack verifying the composability of different items (data type, unit, component, state-machine, etc.). The weight column is used to set the *composability degree* of components, as explained in Section 4.2.5. The weight value indicates the significance of the corresponding rule from the composer's point of view (Medjahed and Bouguettaya, 2006)). The composability degree of components is based on the composability degree of events, i.e., a send event in one component with the corresponding receive event in the peer component. The composability degree for an event is computed after finding the degree of similarity at each feature and level. If the degree is greater than or equal to some threshold, then the component or entity is a potential candidate.

Table 2 Composability stack

Composition layer	*Feature to be checked*	*What is checked*	*Rule*	*Weight*
Syntactic	Message	Message name	Rule-1	W1
	Action	Mode	In/Out	
		Binding	Horizontal	
	Parameters	Number of parameters	Rule-2	W2
Static semantic	Entity	Entity type (Source and target of message	Rule-3	W3
	Parameters	Unit	Rule-4	W4
		Data Type	Rule-5	W5
Dynamic semantic	State machine of composition (State machines + Pre/Post conditions on events)	Action (Pre-cond and post cond)	Rule-6	
		State machine		

4.2.1 Syntactic layer

In the syntactic layer, the syntax of composition is verified by checking syntactic aspects of actions and messages. The bindings and mode attributes present the syntactic attributes of an action. The bindings clarify how two actions can be combined. The '*mode*' of an action indicates whether the action initiates the interaction or it is invoked as a result of the interaction. The number of message parameters and event name should be the same for both *In* action and respective *Out* action. Syntactic matching is done by the following two rules:

- *(Rule-1)–Message Name*. The name of the message in the simulation scenario with the one in the candidate component should be exactly matched. Also, we could use a dictionary to resolve synonym names.
- *(Rule-2)–Number of Parameters*. The quantity of parameters of an event in the model with those of the candidate should be equal.

4.2.2 Static semantic layer

Static Semantic layer compares semantics of the events by using the semantic information provided by the SBA. This semantic information consists of data type and unit of each parameter (for instance, Centimetre or Inch) as well as the ontology of the event initiator and receiver entities.

- *Entity Type*. It gives the type of the entity initiating/receiving the event. We assume a predefined taxonomy for entities in the ontology. The entity type for event initiator and the one which receives the event in both components, BOMs, should be either exactly the same or be in the same hierarchy (Rule-3). For example, consider Customer, RestaurantCustomer and SwedishRestaurantCustomer taxonomy.
- *Unit*. *"It refers to the measurement unit in which parameter's content is provided"* (Medjahed and Bouguettaya, 2006). The unit of a parameter in both sides either should be the same or can be convertible without loss of information (Rule-4). For example, converting Swedish Krona to Singaporean Dollar.
- *Data Type (of Parameters)*. *"It gives the range of values that may be assigned to the parameter"* (Medjahed and Bouguettaya, 2006). The type of parameters of the events in both event initiator and event receiver should be compatible (Rule-5).

4.2.3 Dynamic-semantic layer

Pre/post-conditions can be used to indicate the context within which a component operates, hence defining different types of constraints. By comparing these pre/post-conditions one can identify potential constraint violations. In Medjahed and Bouguettaya (2006), the authors point out that **plugin-pre** form is particularly useful to check pre/post condition in horizontal composability. It is stated that plugin-pre relation exists between each two operations OPi and OPj, if the execution of OPi can be followed by the execution of OPj and the following implication is true: PreCond**i** Λ PostCond**i** $\rightarrow$ PreCond**j**. Here PreCond**i** and PostCond**i** refer to pre condition and post condition of operation OPi, and PreCond**j** refers to pre condition of OPj.

By adjusting the original definition of *plugin-pre* definition to the concepts in our work, the definition can be seen as the relation between post-condition of an *Out* action with the pre-condition of a matching *In* action. Thus, the *plugin-pre* implication (PreCond**send** Λ PostCond**send** $\rightarrow$ PreCond**receive**) should be held between the actions of a send event and the corresponding receive event *(Rule-6)*.

For example, let us consider a Join interaction in which Customer entity wants to join the queue of a Queue entity by firing a Join event targeted at the Queue. If Customer assumes 'LIFO' policy for the list it wants to be queued, and Queue considers 'FIFO' policy for the waiting list; obviously these two entities will not have a successful Join interaction. This kind of semantic mismatch is discovered in this layer.

4.2.4 State-machine composition

Syntactic and Static Semantic matching are done to make sure that the components involved in an interaction, have consensus on the syntax and semantic of the message being transmitted and the event causing the interaction. But this matching is not sufficient since behaviour of participants (entities) in the interaction still remains unchecked: Entities expect events to be fired or to be received at right state (time). Entities will not accept messages at wrong states. So the order of interactions among entities should match the current states of the involved entities.

To illustrate the need for checking the state-machines, consider the following simple examples with entities, their states and events that are sent between them. The entities are denoted as C_i, where i is the number of the entity. The states are denoted as S_{ij} where i is the number of the entity that has the state and j is the number of the state. And finally the events, which are synchronous, are denoted as $E_{k,l,m}$ where k is the number of the event, l is the number of the sender entity and m is the number of the receiver entity. In the first example (Figure 7), entity C_1 has the states S_{11}, S_{12}, and S_{13} (S_{11} is the initial state), and entitiy C_2 has the states S_{21}, S_{22}, and S_{23} (S_{21} is the initial state). Three events $E_{1,2,1}$, $E_{2,1,2}$, and $E_{3,2,1}$ are sent between C_1 and C_2 in this order. We should also mention that the entities agree on the syntax and the semantics of the events. However, the two entities are not composable since when event $E_{2,1,2}$ is fired C_2 is at the wrong state (S_{22} instead of S_{23}) and hence can not accept the event and change its state. While in Figure 8, entities C_1 and C_3 are composable since both entities are at right states when the events are sent. These examples show that an agreement on syntax and semantics of events between components is not sufficient to ensure that they are composable.

Figure 7 Example illustrating state-machine mismatch between entities C_1 and C_2

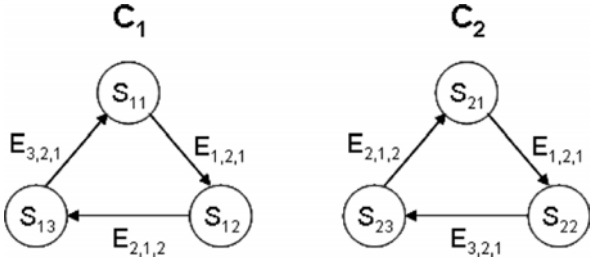
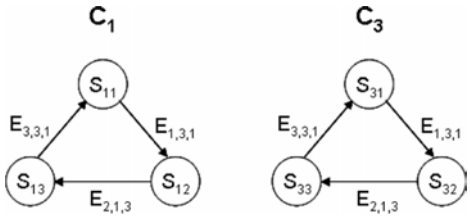


Figure 8 Example illustrating state-machine match between entities C_1 and C_3



Since post-conditions are directly resulted from state-machine transitions, both kind of matching, state-machine and dynamic-semantic matching, are done at the same time. To discover both types of mismatches, state-machine of all involved entities will be 'executed' based on given interactions in the simulation model. The state-machine execution follows two major goals:

- All events, stated in the simulation scenario, can be applied to the state-machine of the involved entities such that each entity accepts or initiates the designated events and possibly changes its state.
- There is no conflict in understanding the semantic of interactions (*Out* action and the corresponding *In* action) among the entities.

In a successful state-machine execution, there are no unapplied rules and each send event is proceeded by corresponding receive event.

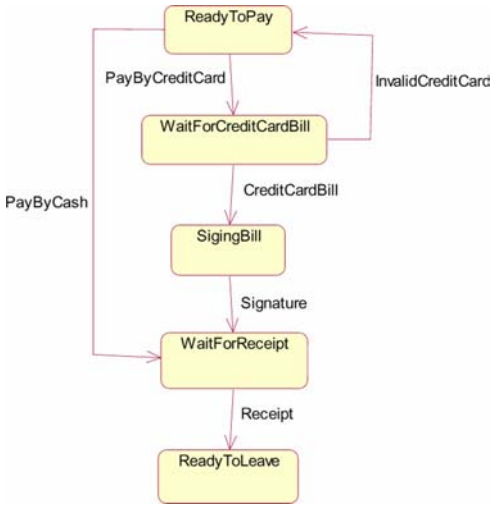
4.2.5 *Deterministic vs. non-deterministic state-machines*

Determinism, in the context of state-machines, means that for every input stream (set of events indicated in simulation scenario in our case) there is exactly one execution path (e.g., state-machines presented in Figures 7 and 8). In the case of non-deterministic state-machines, there could be more than one possible execution for some of the input streams. Consequently, the matching algorithm becomes more complex.

In this paper, we address an automatic state-machine matching, as a necessary step for automatic BOM composition, provided that the behaviour of the components is deterministic. However, since this is a simplification of the reality, we shortly discuss how non-deterministic behaviour in state-machines affects our work. There are two types of non-deterministic behaviour:

- *Non-deterministic events*. Non-deterministic events occur when there is more than one applicable event to the current state and each event has a probability (of taking place) greater than zero. In the BOM specification (SISO, 2005), two groups of such non-deterministic events are identified, Variations and Exceptions. Variation means that different events result in successful execution of a task. For instance, in Figure 9, both 'PayByCach' and 'PayByCredit' events are applicable to 'ReadyToPay' state and (eventually) lead the Customer entity to the 'WaitForReceipt' state (i.e., payment is done!).
  The second group, Exceptions, refer to undesired but potential behaviour of an entity. They usually refer to error events, e.g., 'InvalidCreditCard' event in Figure 9. At the moment our work does not cover this type of non-determinism automatically, but it can be extended easily to support it.
- *Non-deterministic states*. Non-deterministic states mean that one event causes two or more transitions to different states. This might, for instance, occur because of modelling randomness exposing non-deterministic properties of a component (Henzinger, 2001).
  Neither of our matching algorithm nor the rule-based library (Jess), leveraged in the implementation, can handle this type of non-determinism.

Figure 9 Non-deterministic events in customer state-machine (see online version for colours)



The effect of non-deterministic behaviour (state-machine) on WS composition is widely studied in WSs area (for instance Berardi et al. (2005)), but their approaches and results are not easily applicable to modelling and simulation domain and further investigation is required.

4.2.6 *Composability degree*

In many practical situations, it may be difficult to compose a model that satisfies all SRML specifications. In such cases, one would be interested to find out how good the composed model is. For this purpose, we define a composability degree.

In Table 3, each level i is assigned a weight W_i, and each rule R_{ij} belonging to level i is assigned a weight W_{ij}. The function $Satisfied(Rule_x)$ is defined for each pair of events and returns 1 if the $Rule_x$ is satisfied between the two events and zero otherwise. The following formula is used to calculate the Composability Degree.

ComposabilityDegree(*Action*X)

$$= \sum_{i=1}^{L} W_i \times \left(\sum_{j=1}^{R_i} W_{ij} \times satisfied(Rule_{ij}) \right)$$

where

L: Number of composability layers

R_i: Number of rules in layer i

W: Weight assigned to a composability layer or a rule.

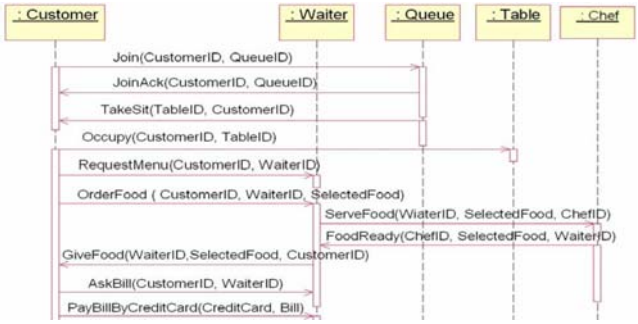
The composition framework described here, has been implemented and evaluated. We have implemented the application in Java, and utilised Jena inference engine and the Jess rule engine for reasoning about matching and composition of BOMs. The BOMs are supported by SBAs including related ontology. SBAs provide an important constitute for reasoning about the composability of BOMs. We believe that development of SBAs by employing techniques and methods from Semantic Web and WSs is feasible, and the three-layered scheme is promising and can improve composition of BOM-based components. However, the methods used in this approach can be refined, especially the algorithm for matching conceptual models, which can be further developed to handle more complex state-machines.

5 Case study

To evaluate our composition scheme, a simple Restaurant scenario test application was implemented in Java, which utilises the Jess rule engine. The scenario can be seen in Figure 10 in sequence diagram format. There are five entities in the scenario: Customer, Waiter, Queue, Table and Chef. Each BOM is supported by an ontology as part of the SBA, as stated in Section 4. For example, one can consider a hierarchy of Customers and attributes of a Customer, for example, FavoriteFood, defined in the ontology.

The simulation model is described in SRML Light and in the format which has been explained in Moradi et al. (2006) and it is given to the test application. The application first discovers and extracts the potential BOMs from a repository. Then BOMs which do not pass the discovery rules in Table 1 are filtered out. Discovery is not the main goal of this work; therefore, we are not going into detail in this example. Thereafter, the discovered BOMs are matched based on the interactions designated to them in the simulation model. For example, in the above sequence diagram, Customer asks Waiter for Bill via 'AskBill' message, which consists of two subsequent events: event of sending the 'AskBill' message by Customer BOM and event of receiving the message by Waiter BOM. So, the action of sending event should be matched with the action of the corresponding receive event via the composability rules stated in Table 2 and described at Section 4.2 (Matching Phase). The semantic comparison is done by querying Ontology to find out which of the three relations (≡, ⊆, disjoint) exists between each two items in the In and respective Out actions .The matched actions receive a composability degree (explained in Section 4.2).

Figure 10 Simulation scenario in form of sequence diagram (see online version for colours)



After matching the actions and assigning composability degree, the process ends up with the state-machine composition as stated in Section 4.2. Figures 11 and 12 show the partial state-machine of Customer and Waiter entities, respectively.

Figure 11 Partial state-machine of customer entity (see online version for colours)

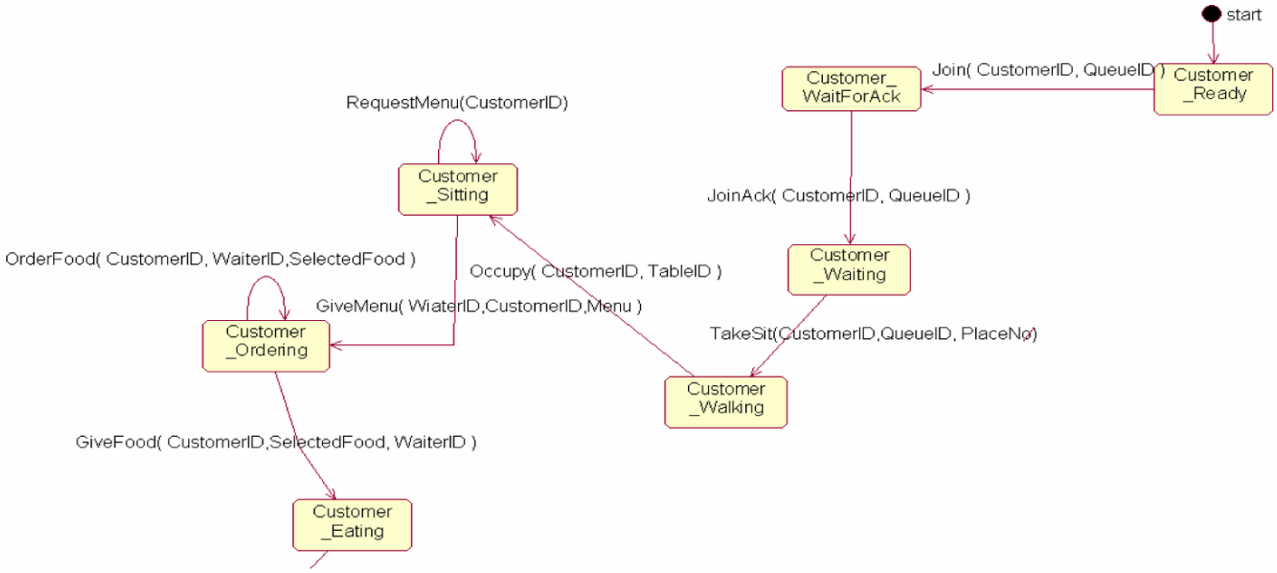


Figure 12 Partial state-machine of waiter entity (see online version for colours)

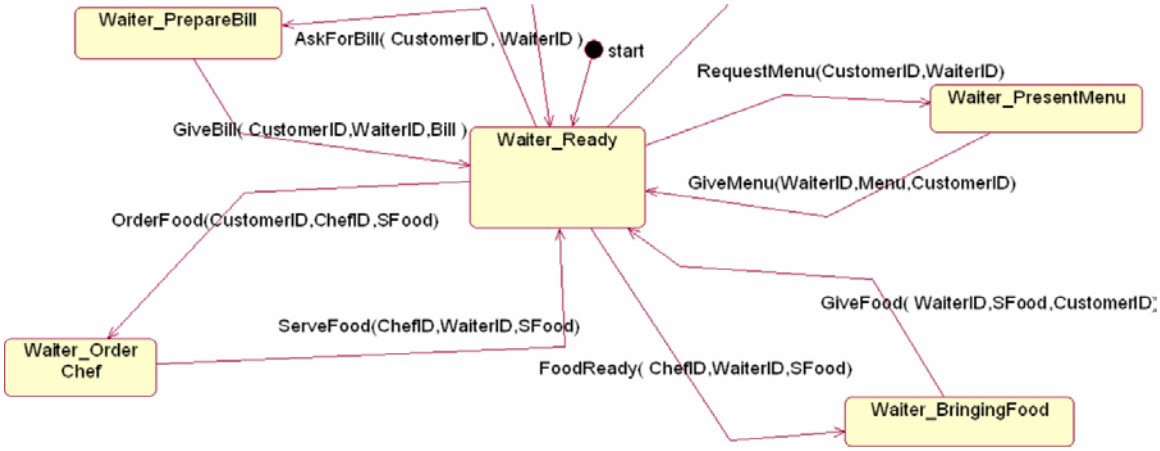
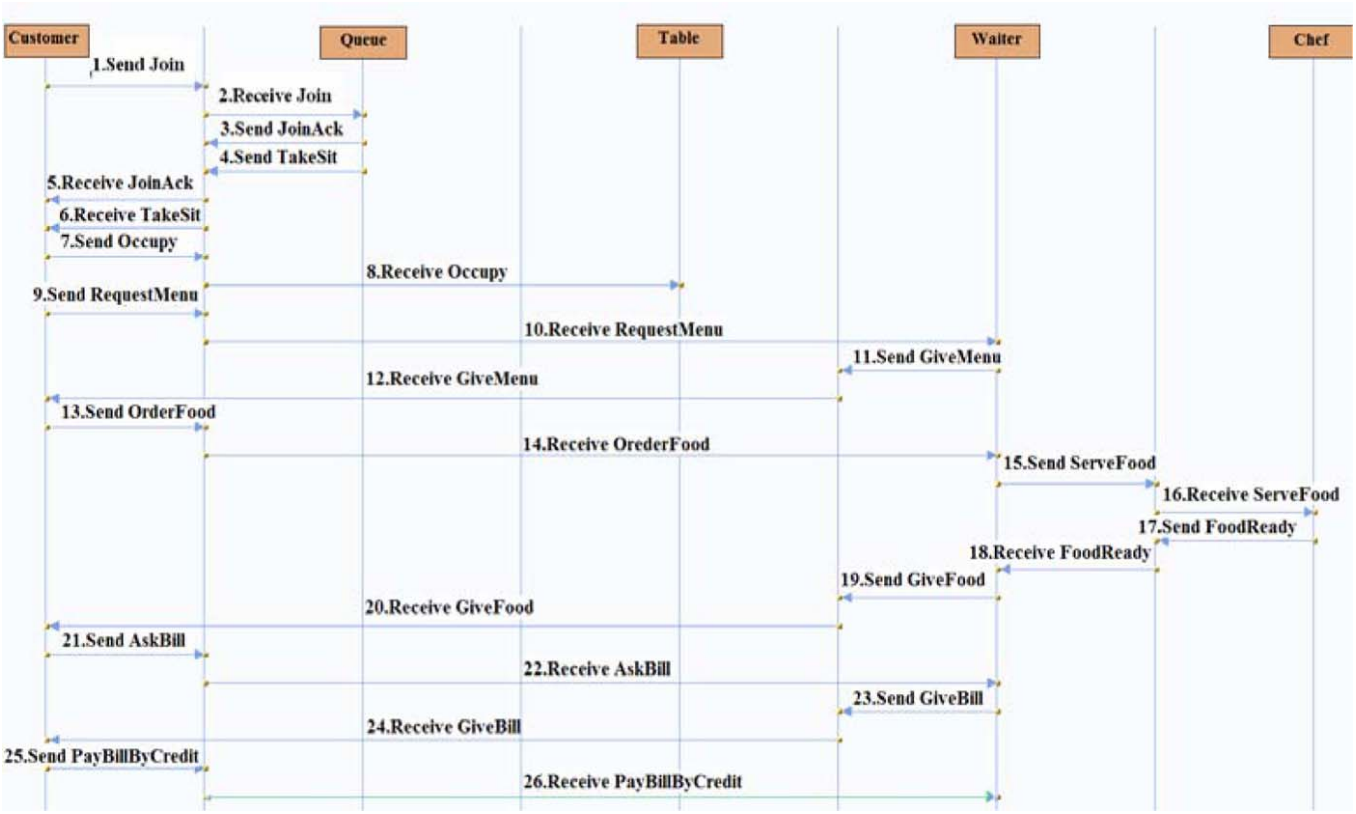


Figure 13 shows one possible sequence of event passing among the entities resulting from the state-machine composition. This sequence of event passing can be identified using our definition of a successful state-machine composition (Section 5.2) or alternatively by using the sequence diagram of the scenario. In this example, the composition is shown to be successful. As explained earlier by a successful composition, we mean that all the events are sent when the sender entities are in the 'right' state and are received by the receiving entities when they are in the state that allows them to accept those events. In our example, we verify the matching of the state-machines of the *Customer* and *Waiter* entities. For instance, the *Customer* entity sends an event with the action *RequestMenu* when it is in the *Customer_Sitting* state and stays in the same state. The *Waiter* entity receives that event when it is in the *Waiter_Ready* state. The *Waiter* changes its state to *Waiter_PresentMenu* and sends an event with the *Give_Menu* action back to the customer, who receives the event and changes its state from *Customer_Sitting* to *Customer_Ordering*. Here, the pre-/post-conditions are also checked and the *In* and *Out* actions are matched both syntactically and semantically. The Components in our example pass all the matching criteria.

Figure 13 Sequence of event execution resulted form state-machine composition (see online version for colours)



6 Conclusions

In this paper, we proposed an SBA to enhance the semantic expressiveness of BOMs and a three-layer model for composing BOM-based components. The semantic enhancement is mainly done through inclusion of ontology for entities, event and interactions in each component, and presentation of an OWL-S description for each component including the state-machines. The three-layer model contains syntactic matching, static semantic matching and dynamic-semantic matching based on the information provided by the SBA. We also described our discovery and matching rules, which have been implemented in the Jess inference engine. To test our composition scheme, we defined simulation scenarios and implemented BOMs as building blocks for development of those scenarios. A composability degree was defined to quantify closeness of the composed model to the model specification. We also presented a case study to show how SBA and the three-layer model can be deployed. Our preliminary results show that the three-layered scheme is promising and can improve composition of BOM-based components. Future work consists of further development of our test environment, as well as more experiments and comprehensive test of our method using a larger number of components.

Acknowledgement

This is a substantially modified version of a paper published in DS-RT 2007

References

Berardi, D., De Giacomo, G. and Mecella, M. (2005) 'Automatic composition of web services with nondeterministic behavior', *Proceedings of the 31st International Conference on Very Large Databases*, Norway, pp.613–624.

Gruber, T.R. (1993) 'A translation approach to portable ontology specifications', *Knowledge Acquisition*, Vol. 5, No. 2, pp.199–220.

Henzinger, T. (2001) *Lecture Notes on State Machines*, University of California at Berkeley, February, http://ptolemy.eecs.berkeley.edu/eecs20/lectures

Hu, Y., Tan, G. and Moradi, F. (2003) 'Automatic SOM compatibility check and FOM development', *Proceedings of 7th IEEE Distributed Simulation and Real-time Applications*, October, Delft, The Netherlands, pp.60–67.

Medjahed, B. and Bouguettaya, A. (2006) 'A multilevel composability model for semantic web services', *Journal of IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, No. 7, July, pp.954–968.

Moradi, F. (2007) 'Component-based simulation model development using BOMs and web services', *Proceedings of the first Asia Modelling Symposium, AMS 2007*, March, Thailand, pp.238–246.

Moradi, F., Ayani, R. and Nordvaller, P. (2006) 'Simulation model composition using BOMs', *Proceedings of the 10-th International Symposium on Distributed Simulation and Real Time Applications, DS-RT '06*, October, Spain, pp.242–252.

Morse, K., Petty, M., Reynolds, P., Waite, W. and Zimmerman, P. (2004) 'Findings and recommendations from the 2003 composable mission space environments workshop', *Proceedings of the Spring 2004 Simulation Interoperability Workshop*, April, Arlington, VA, USA, pp.313–323.

Petty, M.D., Weisel, E.W. and Mielke, R.R. (2004) 'Overview of a theory of composability', *Proceedings of the Huntsville Simulation Conference 2004*, October, Huntsville, AL, USA, pp.363–368.

Simulation Interoperability Standards Organization (SISO) (2005) *Guide for Base Object Model (BOM) Use and Implementation*, SISO-STD-003.0-DRAFT-V0.11, SISO Inc., Orlando, USA, pp.21–64.

Szabo, C. and Teo, Y.M. (2007) 'On syntactic composability and model reuse', *Proceedings of the First Asia Modelling Symposium, AMS 2007*, March, Thailand, pp.230–237.

Weisel, E.W., Petty, M.D. and Mielke, R.R. (2003) 'Validity of models and classes of models in semantic composability', *Proceedings of the Fall 2003 SIW*, 14–19 September, Orlando, FL, USA, pp.526–536.

Websites

BOMworks, http://www.simventions.com/bomworks/

HLA at *DMSO*, https://www.dmso.mil

IBM Web services tutorial, Online: http://www-106.ibm.com/developerworks/webservices/

Jess web site, http://herzberg.ca.sandia.gov/jess/

OWL-S: *Semantic Markup for Web Services*, http://www.w3.org/Submission/OWL-S/

Semantic Web, http://www.w3.org/2001/sw/

World Wide Web Consortium, *Simulation Reference Markup Language*, http://www.w3.org/TR/2002/NOTE-SRML- 2002121

World Wide Web Consortium, *Web Ontology Language*, http://www.w3.org/2004/OWL/