A Rule-based Approach to Syntactic and Semantic Composition of BOMs

Farshad Moradi  
Swedish Defence Research Agency, FOI  
farshad.moradi@foi.se

Rassul Ayani, Shahab Mokarizadeh, Gholam Hossein Akbari Shahmirzadi  
KTH, Sweden

Gary Tan  
National University of Singapore, NUS

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 process. However, in order to successfully compose models one has to solve the issues of syntactic and semantic composability of components. HLA is the most widely used architecture for distributed simulations today. It provides a simulation environment and standards for specifying simulation parts and interactions between simulation parts. But it provides little support for semantic composability. The Base Object Model (BOM) standard is an attempt to ease reusability and composition of simulation models. However, BOMs do not contain sufficient information for defining concepts and terms in order to avoid ambiguity, and provide no methods for matching conceptual models (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 semantic enhancement includes ontologies for entities, event and interactions in each component. We also present an OWL-S description for each component including the state-machines. The three-layer model consists of syntactic matching, static semantic matching and dynamic semantic matching utilising a set of rules for reasoning about the compositions. We also describe our discovery and matching rules, which have been implemented in the Jess inference engine. In order to test our approach we have defined some simulation scenarios and implemented BOMs as building blocks for development of those scenarios, one of which has been presented in this paper. Our result shows that the three-layer model is promising and can improve and simplify composition of BOM-based components.

1. Introduction

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 process. This approach has been successfully deployed in manufacturing industry and software engineering. However, in order 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 [4], [5].” 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 [3], [6]. 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 [4], [5].

HLA is the most widely used architecture for distributed simulations today [12]. 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 formalize 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 [9]. In our earlier papers we have investigated how BOMs can be used to
develop simulation models in a component-based fashion and suggested a process for component-based simulation development using BOMs [1]. We argued that even though the BOM standard looks promising and exhibits better capabilities for reuse and composability, through e.g. its conceptual model, it lacks the required semantic information for semantic matching and composition. There is little support for defining concepts and terms in order to avoid ambiguity, and there is no method for matching conceptual models (state machines).

We have also discussed utilization of Semantic Web and Web Service (WS) [14, 15] technologies for further refinement of the process and improving the semantic composition of BOMs [2].

In this paper, we present our approach for enhancement of the semantic contents of BOMs and our process for discovery and matching of BOMs based on that information, which is done based on our three layer model for syntactic and semantic matching of BOMs.

2. BOM based model components

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 modeling building-blocks or components. The interplay within a simulation or federation can be captured and characterized 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 [9] 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.

The Conceptual Model, which is our main concern here, contains information that describes the patterns of interplay of the component. This part includes what 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.

![BOM Structure](image)

Figure 1 BOM structure

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 [16] from SimVentions.

3. Model composition approach

In order 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 meta-data and contributes to simplified use of a component by others [17].

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 [8]. 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 [8].

The current BOM standard lacks the required semantic information to avoid ambiguity. Furthermore,
there is no method for matching of state machines in the conceptual models of different BOMs. In order to address the above issues, we suggest extending the BOM description with a semantic attachment through utilization of Web Service technology and OWL-S language (Web Ontology Language for Services) [11] (explained in section 4). The semantic attachment provides the meta-data required for discovering and 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 5.2, utilising a set of rules for reasoning about the compositions.

3.1. Composition process

In this section we describe our process for component-based model development using BOMs, which is a continuation of our previous work where the process was explained in detail [1].

The process comprises four phases: a) SRML Parsing b) BOM Discovery c) BOM Matching and Composition, and d) BOM Assembly Building. These four phases altogether make seven steps:

The first phase starts with a description of the target simulation written in SRML [13]. 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 [1]. The “Item Class” tag is utilized 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.

```xml
<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.

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 5.1. In our example the result of the query could be a number of Queue BOMs. However, not all of them do 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 5.2. For instance a Customer BOM that is designed for Fastfood restaurant scenarios is probably not suitable for an Italian restaurant scenario where she/he is expected to be waited, since it does not have 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 2 shows these seven steps in a flow chart format, where each step is represented by a rectangle.

In order to realize 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 utilizes 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 Semantic BOM Attachment is based on OWL-S and is explained in more detail in section 4.

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

3.2. Modelling and composition assumptions

The following assumptions have been considered in our work. First of all simulation models are seen as combinations 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 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.

![Figure 2 The simulation development process](image-url)
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 [7]. Two operations/actions can be combined Horizontally if they model a supply-chain like combination [7]. 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. In order 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 mode or Out mode. The mode can not be changed dynamically and it is assigned to the action during Semantic BOM attachment 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.

3.3. Matching methodology

As explained in the previous section we consider simulation models as combinations of components and events. Therefore, in order 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 utilization 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.

4. Semantic BOM Attachment

In order to facilitate semantic matching and composition of BOM based components we suggest an enhancement of the current BOM description through utilization of the Semantic Web Services concepts and methods for composition, here referred to as Semantic BOM Attachment. For this purpose we have mapped the BOM descriptions into the OWL-S [11] upper ontology. The main reason for this mapping is to use OWL (Web Ontology Language) [10] as the underlying language for describing BOMs and also utilize the language features of OWL-S to improve the semantic expressiveness of BOMs and hence, facilitate semantic discovery and composition of them.

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 hierarchy 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. The BOM conceptual model and its items are shown I figure 1. Figure 3 depicts the items of BOM meta-data converted into semantic BOM attachment plus the supportive ontology.

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

149
4.1. Mapping pattern of interplay

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 [9].

A one-to-one of mapping of the events and their contents to respective ontology classes and Service Model of OWL-S results in the ontology presented in figure 4.

![Figure 3 Semantic BOM Attachment](image)

**Figure 3 Semantic BOM Attachment**

**4.2. 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.” [9]

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 [18] 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 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.

```plaintext
(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)
)
```

4.3. 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

![Figure 4 Action ontology](image)

**Figure 4 Action ontology**

Each action should clarify its mode, whether it is the result of sending a message (Out mode) or of receiving a message (In mode). In addition an action in the ontology 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 pair of pre-condition and post-condition, helps the composer to find a suitable component which its action(s) matches the current environment facts. These pre- and post-conditions can partly be retrieved from the information provided by the BOM Model Identification, such as “Use Limitation” and “Security Restrictions”. It can also be added by the user or an intelligent application in case it is not covered by the BOM model.

When mapping actions to the OWL-S upper ontology, each action is associated to an atomic process since both by definition simulate an atomic action. More correctly it is the message which is associated to the atomic process of the OWL-S service model. Thus, we will have the following mapping between messages and the service model of OWL-S (Table 1). It should be noted that there is nothing to be mapped to the output of the service model. The pattern of Interplay in BOM provides a mechanism for defining a sequence of Pattern Actions. Consequently, the pattern of interplay can be assumed as a composite process in OWL-S or even a service model, as was suggested in [2].

<table>
<thead>
<tr>
<th>Service Model (OWL-S)</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Parameter</td>
<td>Source Characteristics, Target Characteristics, Content Characteristics</td>
</tr>
<tr>
<td>Output Parameter</td>
<td>N.A</td>
</tr>
<tr>
<td>Pre/Post Condition</td>
<td>Should be extracted from information provided by BOM ModelIdentification or introduced by a user or an agent</td>
</tr>
</tbody>
</table>

**Table 1 Mapping BOM Message to Service Model of OWL-S**
the role of conceptual entities identified within a state machine”[9].

A hierarchy of entity types can be defined in the ontology, 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.

5. Overall architecture and implementation

After describing our assumption, composition methodology and the Semantic BOM Attachment, 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 5 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.

5.1. 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 2 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. This is a very naive discovery algorithm. However, it can be improved if some semantic filtering is also applied during the discovery phase.

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. 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 Semantic BOM Attachment. 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 Semantic BOM Attachment. The accuracy of the discovery process is dependent on the amount of meta-data 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.

5.2. Matching phase

The idea of composability stack is inspired by the work done by Mojtahed and Bouguettaya [7]. In [7] 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.”

<table>
<thead>
<tr>
<th>Table 2 Discovery rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discovery Layer</td>
</tr>
<tr>
<td>Syntactic</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Semantic</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Even though we use the same terminology as in [7] 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.

<table>
<thead>
<tr>
<th>Table 3 Composability stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composition Layer</td>
</tr>
<tr>
<td>Syntactic</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Static Semantic</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Dynamic Semantic</td>
</tr>
</tbody>
</table>

Our composability stack is presented in Table 3. 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. The weight value indicates the significance of the corresponding rule from the composer’s point of view [7]. 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.

5.2.1. 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.

5.2.2. Static semantic layer. Static Semantic layer compares semantics of the events by using the semantic information provided by the Semantic BOM Attachment. 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 pre-defined 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”[7]. The unit of a parameter in both sides either should be the same or can be convertible without loss of information...
For example, converting Swedish Krona to Singaporean Dollar.  

- **Data Type (of Parameters)**: "It gives the range of parameters that may be assigned to the parameter"[7]. The type of parameters of the events in both sides, event initiator and event receiver, should be compatible (Rule-5).

### 5.2.3. Dynamic semantic layer

In [7], the authors point out that plugin-pre form is particularly useful to check 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: PreCondi ∨ PostCondi → PreCondj. Here PreCondi and PostCondi refer to pre condition and post condition of operation OPi, and PreCondj 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 (PreCondi ⟨send⟩ ∧ PostCondi ⟨send⟩ → PreCondj ⟨receive⟩) should be held between the actions of a send event and the corresponding receive event (Rule-6).

### 5.2.4. 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 two issues still remain unchecked:

1. 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. As we are considering Horizontal composition in which the Out action of the entity initiating the interaction is combined with respective In action of the entity accepting that interaction, semantic of the Out action (the post-condition of the Out action) should be matched with that of the respective In action (the pre-condition of the In action). For example, let’s 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 should be discovered.

2. 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.

In order to discover both types of mismatch mentioned above, 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.

### 6. Case study

In order to evaluate our composition scheme, a simple Restaurant scenario test application was implemented in Java, which utilizes the Jess rule engine. The scenario can be seen at figure 6 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 Semantic BOM Attachment, 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 [1] 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 2 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 3 and described at section 5.2 (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 5.2).

After matching the actions and assigning composability degree, the process ends up with the state machine composition as stated in section 5.2. Figure 7 and figure 8 show partial state machine of Customer and Waiter entities.

Figure 9 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.

7. Summary and future work

In this paper we have presented our approach to enhance the semantic expressiveness of BOMs, using a three-layer model for composing BOM based components and our process for component based development of simulation models.

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. This semantic enhancement is called Semantic BOM Attachment.

![Diagram](https://example.com/diagram.png)

**Figure 9 Partial sequence of Event Execution resulted form State Machine Composition**

The three-layer model contains syntactic matching, static semantic matching and dynamic semantic matching based on the information provided by the Semantic BOM Attachment. We have also described our discovery and matching rules, which have been implemented in the Jess inference engine. In order to test our composition scheme we have defined simulation scenarios and implemented BOMs as building blocks for development of those scenarios. In this paper we have presented one of our case studies. 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.

8. References


