BOM Matching and Semantic Validation

Hirad Asadi, Vahid Mojtahed, Martin Eklöf
Swedish Defence Research Agency,
Stockholm, Sweden
{hirad.asadi, vahid.mojtahed, martin.eklof}@foi.se

Keywords:
DCMF, BOM, Ontology, Knowledge Use, BOM composition, model matching, semantic validation.

ABSTRACT: *The Defence Conceptual Modeling Framework (DCMF) is the Swedish Defence Research Agency's (FOI) proposal for conceptual modeling in the military domain. The purpose of DCMF is to enable conceptualization, composition, visualization, and reuse of knowledge for modeling and simulation. To achieve this, DCMF requires that its final products - conceptual models represented in the Base Object Model (BOM) format - are enriched with semantics. In this paper we focus on the composition part of the framework and show how composition of BOMs can be achieved using ontologies to account for the semantics of information. We demonstrate this through a prototype which incorporates BOM matching techniques along with our proposal for semantic enrichment. We discuss how the current BOM representation could be extended so that it becomes more suitable for tasks that involve reasoning over modeled knowledge.*

1. Background

The development of a knowledge base, prior to the development of a simulation model, is a time-consuming and costly process. Moreover, the knowledge that a model represents is often not properly saved for future reuse. Even if the model is accurately stored it is difficult to reuse it in other contexts. This is primarily because knowledge of how a model was created is not documented to the extent necessary. In other words, for potential reuse of a model, relevant facts about knowledge acquisition are missing and hence so is the traceability to the knowledge source.

The Defence Conceptual Modeling Framework (DCMF) is a framework developed by the Swedish Defence Research Agency (FOI) for creating and representing conceptual models [13]. Conceptual models in this context are formalized descriptions of real world processes, entities, associated relationships, and interactions that constitute military missions, operations, or tasks. They are the final artifacts of the DCMF process, intended as means of representing requirements for simulation model development.

To enable a satisfactory analysis of domain knowledge and its shared understanding among people and software systems, DCMF requires that conceptual models are formalized and explicitly specifies the semantics of information. In this paper we discuss one such formalization developed by utilizing the Base Object Model (BOM). BOM (see next section) was created by the Simulation Interoperability Standards Organization (SISO) to enable composability and reuse of concepts intended for design of simulation models, interacting in High Level Architecture (HLA) federations [3]. The IEEE-standard Federation Development and Execution Process (FEDEP) defines the process for developing and supporting federations conforming to HLA [4]. FEDEP proposes seven major steps: Define Federation Objectives, Develop Federation Conceptual Model, Design Federation, Develop Federation, Integrate and Test Federation, Execute Federation, and Analyze and Evaluate Results.

When using BOM to conceptualize the artifacts in DCMF, the framework can be seen as a well-defined approach supporting Step 2 in FEDEP, i.e., for guiding development of conceptual models for federations. However, when considering the BOM for conceptual model representation in the DCMF process, a need arose to find a way to enrich the BOM specification with formal semantics. This became especially important when considering the composition and reuse of BOMs stored in a repository. An example of such an enrichment of BOM has already been proposed, BOM++ [10].

Giving this background, the remainder of the paper is organized as follows. Section 2 gives an overview of basic concepts underlying our work. In Section 3, we describe the method chosen for putting into practice the concepts described in Section 2 and how our prototype supports this. In Section 4, we show a case study that demonstrates our prototype. Section 5 concludes the study and describes further research.

2. Theory

In the following sections an overview of two basic concepts underlying our work, BOM and DMCF, are presented.

2.1. BOM

The BOM proposal was created by SISO to encourage and support reuse, interoperability, composability, and to help enable rapid development of HLA simulations. BOM was standardized by SISO in 2006 [6].

At a high level, BOMs are reusable packages of information representing independent patterns of simulation interplay and are intended to be used as building blocks in the development and extension of simulations. These components can also be composed to form more extensive models e.g., BOM Assemblies. Additionally, interplay within a simulation or federation can be captured and characterized in the form of reusable patterns. These are sequences of events between simulation entities. Implementation of these patterns using the HLA Object Model Template (HLA OMT) is also captured in the BOM [9].

Structured in five major parts, a BOM is an XML document that encapsulates the information needed to describe a simulation component. From the perspective of DCMF, the second part of the BOM, the Conceptual Model, is the most interesting. The Conceptual Model 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, and a listing of conceptual entities and events, which, when taken together, describe the flow and dependencies of events and their exceptions.

The BOM Conceptual Model is further divided into four different sub-models [7]:

- Pattern of Interplay,
- State Machine,
- Entity types and
- Event types.

2.1.1. Pattern of Interplay

The Pattern of Interplay (POI) describes the recurring behavior used to accomplish a common objective, capability, or purpose, as carried out by a real world entity, phenomenon, process or system. It contains actions, entity types, and event types that take part during one interaction between two components. It also contains variations and exceptions that could happen as alternatives to the normal flow of events. These actions form a flow that is sorted by the Sequence Number associated to each of them. Along with the Sequence, a Name, a Sender, and a Receiver are also specified.

2.1.2. State Machine

The State Machine model is a mechanism for modeling how entity types move from one state to another via actions. State Machines are made of states and conditions that must occur to hop to the next state, and are related to a specific conceptual entity.

2.1.3. Entity Types

Entity types represent real-world objects; they are used in POIs in order to define Senders and Receivers. Entity types also describe the conceptual entities that are used within State Machines, and have Attributes associated with them.

2.1.4. Event Types

Event types are used by conceptual entities to make transitions from one state to another and are employed within the POI model. There are two kinds of Events: Triggers and Messages. Triggers are sent by some entities without a specific receiver, while Messages do have a specific recipient.

2.2. DCMF

The DCMF [13] includes a process whose main objective is to address how knowledge can be acquired for a particular purpose, as well as how it can be structured, modeled and formatted according to predefined criteria. It also addresses how knowledge can be used, or reused, given diverse applications. The process consists of four main phases: Knowledge Acquisition (KA), Knowledge Representation (KR), Knowledge Modelling (KM) and Knowledge Use (KU).

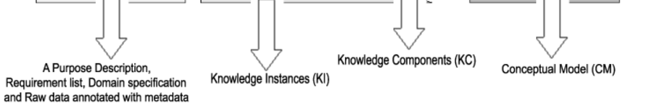
KA
KR
KM
KU
A Purpose Description, Requirement list, Domain specification and Raw data annotated with metadata
Knowledge Instances (KI)
Knowledge Components (KC)
Conceptual Model (CM)

Figure 1. Four Phases of DCMF [11].

Information is first gathered within the Knowledge Acquisition phase. The Producer role processes unstructured knowledge and transforms it into structured knowledge. To accomplish this, a parsing method must be used. During the Knowledge Representation phase, smaller sections of this data are structured as Knowledge Instances (KI) and validated for storage in the repository by the Controller role. KIs are useful for some purposes, but they are not reusable since they are specific to a particular scenario. To get reusable knowledge, KIs are abstracted to the type level, modeled as Knowledge Components (KC) and then validated in the third phase, called Knowledge Modeling. These components are, upon Consumer requests, composed to form Conceptual Models (CM) in the fourth and final phase, Knowledge Use. All artifacts produced are stored in a repository for use and reuse.

In this paper we focus on the KU phase and present a prototype which incorporates the composition algorithm discussed in [2]. We also go further and enhance the prototype with explicit semantics of information, enabling "semantic validation". We extend the composability definition in [2] with the concept of semantic validation, which states that two models are "composable" if they match "statically", "dynamically" and are "semantically valid".

3. Method

The main purpose of the prototype is to investigate how knowledge components (KC), represented as BOM-models [8], can be matched in order to create a larger composition model. Figure 2 shows a basic overview of the prototype. The approach is discussed in [2] where statemachine-matching is a fundamental issue.

The matching module performs the BOM matching and sends the result to the semantic module for semantic validation. The result is visualized by the graph module. The matching result shown by the visualized graph is either semantically valid or invalid.

Using the prototype, we can quickly see which models that fit together. If a person would perform the same task manually it would require much more time. Note that a match between two models simply means that the models can be combined into a larger model in a straightforward manner. A composition model gives a more complex description of a e.g. behaviour.

Given a repository where knowledge components are stored, the prototype searches through BOMs that potentially may be assembled together. The prototype goes through each model and tries to determine if the model can be linked to other models. The composition (matching) is done using a composition algorithm [2] where a set of rules must be fulfilled. If the prototype is able to find matches between different models, the user is notified.

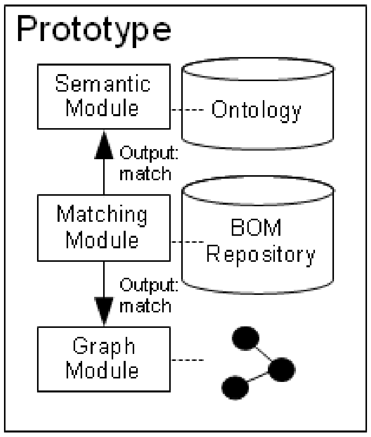

Figure 2. Overview of the prototype.

3.1. Prototype Architecture

The prototype consists of three modules:

- Matching module
- Graph module
- Semantic module

The matching module has an implementation of the composition algorithm which consists of four different steps. When matching is being performed, potential BOMs that can be relevant for composition are identified (this is verified at the end of the process). The four key steps in the matching module are:

1. Parsing
2. Static matching
3. Transformation
4. Dynamic matching

In the first step the BOMs' state machines are simply read into memory along with entities, events and patterns of interplay. The second step performs a so-called "static matching". The static matching ensures that three specific rules are met:

- The first rule says that for each state in the state machines (loaded in step 1) there should be an exit criterion. The exit criterion ensures that the state machine doesn't get trapped in an individual state.
- The second rule says that for every sender of a message (action in patterns of interplay), there must be a receiver.

- The third rule says that there should exist at least one end-state among the state machines that are being matched. This is needed in order to avoid infinite loops. When all rules are met, the static matching is completed and the program can continue to the third step.

The third step performs a transformation, where the BOM information (states and messages) are transformed into State Chart XML (SCXML)[1], a special XML-based[2] format which is used for execution of state machines.

The fourth and final step performs the dynamic matching. Here the BOMs are executed in parallel, in separate threads, in order to test model interaction in real-time. The interaction is basically messages that are sent back and forth between models. This execution is needed in order to identify potential deadlocks in run-time and also to check if end-states can be reached. Figure 3 shows an overview of the matching module.

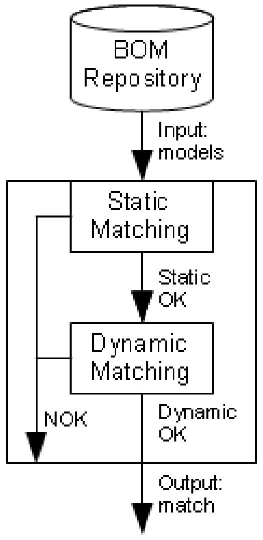

Figure 3. Overview of the matching module.

The graph module receives the matching result which consists of a data structure (a graph structure). The graph structure shows the models that can be assembled together. The graph module is also responsible of visualizing the graph structure, see Figure 4. In this figure we see that the models "Engine", "Wheel" and "Car" are matched in a simple configuration.

The semantic module which is our main contribution has a special purpose in the prototype. So far we have checked if the BOMs can be composed statically and dynamically. The semantic module is the next step in checking whether the BOMs can be composed. This is done using ontologies

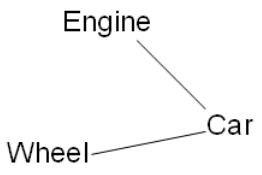

Figure 4. Visualized graph structure.

represented in OWL[3] and rules which are employed using a reasoner engine [1]. The semantic module checks if certain rules for specific entities, represented as "individuals" in an ontology, are met. We call this "semantic validation". By using semantic validation we can put external restrictions that must be fulfilled in order for the BOMs to be composable, e.g. a Car requires at least four Wheels, but only one Engine (not more or less). The reason why we use ontologies is because we want to be able to reason about facts that aren't perhaps explicitly specified in our knowledge base. Ontologies are also important from a machine learning perspective, that is, computers can easily understand standard ontologies.

3.2. Semantic Validation

BOMs have a field called "conceptual model" which contains different conceptual elements. All of those elements have a property field called "semantics". We use this property field for the semantic enhancement. We propose a specific format for the semantics property field using an URI[4] and validation sets.

The format we propose for the semantics field is:
[$URI\ s_1 \ldots s_n$], where URI points to an ontology and s_i is a validation set. Figure 5 gives an example of how a semantics field could look like in a BOM file.

```
<semantics>
 http://some-location/some-ontology.owl
Happy
</semantics>
```

Figure 5. Example of a semantics field in a BOM.

In Figure 5 the semantics field states that the element (e.g. an entity) which has the $< semantics >$ property must exist in the validation set $Happy$, in order to be semantically valid. What this means is that we can construct rules in our ontology that specify different conditions that must be fulfilled in order for the element to be "semantically valid".

[1]SCXML: http://en.wikipedia.org/wiki/SCXML
[2]XML: http://en.wikipedia.org/wiki/XML
[3]OWL: http://en.wikipedia.org/wiki/Web_Ontology_Language
[4]URI: http://en.wikipedia.org/wiki/Uniform_Resource_Identifier

A rule in our ontology can look something like this: $A(x) \wedge B(x) \Rightarrow C(x)$. Here we say that if an individual is in set A and in set B, then the individual is also in set C. For example, if x = "John", and x is "Playing" and "Winning", then x is also happy. Ontology rules are of course not limited to these simple constructions. In this case we could have a rule that specifies different conditions that lead to $Happy$, e.g. $Playing(x) \wedge Winning(x) \Rightarrow Happy(x)$.

Let us assume the BOM element is an entity which represents a person. In our ontology we must also have the same representation, so-called "individual", see Figure 6. Reasoning is performed on individuals. E.g., if "John" is an entity in the BOM (which is represented as an individual in the ontology) and after reasoning we see him in the $Happy$ set, then we say that the element is semantically valid. Note that the element must exist in all of its validation sets, e.g. if entity "John" has $Happy$ and $Married$ as its validation sets, then he must exist in those two sets after reasoning in order to be semantically valid. If all elements in a BOM are semantically valid, then we say that the BOM is semantically valid.

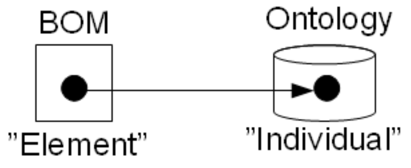

Figure 6. BOM and ontology mapping.

3.3. Algorithm for Semantic Validation

We propose a simple algorithm for semantic validation, see Algorithm 1 [12]. In this algorithm we go through all BOMs in a repository (Φ) and for each BOM (β) we go through all elements that contain semantic information. For each element (ϵ) we extract information about which ontology (ω) the element uses and also its validation sets (Ω). After this we call the reasoner with the following input: the ontology, the element's validation sets and also an identifier to locate the corresponding individual in the ontology.

The reasoner returns a result containing the inferred set (λ). For each validation set (υ) we check if it is a subset of the inferred set, if it is not we mark the element as $false$ and break. If one element in a BOM is marked as $false$, we mark the whole BOM as $false$, meaning it is not semantically valid.

Algorithm 1 Semantic Validation

```
1: $\Phi$ is the BOM repository
2: $\forall\beta \in \Phi$, where $\beta$ is a BOM
3:         $\forall\epsilon \in \beta$, where $\epsilon$ is an element (with sem. info)
4:             mark $\epsilon$ $true$
5:             $\omega \leftarrow \epsilon$.getOntology()
6:             $\{\Omega\} \leftarrow \epsilon$.getValidationSets()
7:             $\{\lambda\} \leftarrow$ reasoner($\omega$, $\Omega$, $\epsilon$)
8:             $\forall\upsilon \in \Omega$, where $\upsilon$ is a validation set
9:                 IF $\upsilon \notin \lambda$
10:                    mark $\epsilon$ $false$
11:                    break
12:            IF $\epsilon$ == $false$
13:                mark $\beta$ $false$
14:                break
```

4. Case Study

The test was based on a scenario that occured in Afghanistan in late 2009 [12]. A Swedish patrol unit was sent out to patrol an area outside of the city of Mazar-i-Sharif. At a certain point the patrol unit hit an IED which resulted in casulties. After alerting headquarters a medical unit was sent and the casualties could be dealt with.

We tested our prototype on a repository containing military BOMs based on this scenario. The BOMs represented different parts of the military organisation within the scenario. Our objective was to test if the prototype could find any matches and if the matches could be semantically validated, that is, if the models could be composable. We created a simple ontology using Protege [5]. The ontology consisted of made-up facts about individuals and rules which were needed for validation. One example of a rule was: "a helicopter is a valid rescue helicopter if it can fly within a 100 km radius from its base and has medical equipment onboard". We used the format shown in Figure 5 to point to the individuals in the ontology from the entities in the BOMs. We also tested how mis-matches could be produced, that is, how we could break different checks (static matching, dynamic matching and semantic validation).

Searching in the repository looking for potential BOMs fulfilling the reqirements from the given scenario, we found three BOMs that matched: **CommandUnit**, **PatrolUnit** and **MedicalUnit**. The models matched because they included senders and receivers for key actions and the actions could be executed without deadlock issues. The connecting sequence of actions were:

1. **PatrolUnit** sends *EmergencyHumanCallAction* to **CommandUnit**.

2. **CommandUnit** sends *OrderMedAction* to **MedicalUnit** or *MedDeniedAction* to **PatrolUnit**.

3. If *OrderMedAction* occurs then **MedicalUnit** sends *ConfirmMedicalAction* back to **PatrolUnit**.

4. When rescue mission is over **MedicalUnit** sends *MedNotifyAction* back to **CommandUnit**.

Figure 7, 8 and 9 show each individual sequence of actions. Each sequence diagram corresponds to the patterns of interplay in one BOM, e.g. the sequence diagram in Figure 8 corresponds to the behaviour of the **MedicalUnit** BOM. The BOMs were also semantically valid. Each BOM contained an entity with a semantic tag and the tags pointed to our ontology which held the rules. We had one validation rule for each entity. When the rules where executed each individual (entity) occured in its validation set, hence the BOMs became valid.

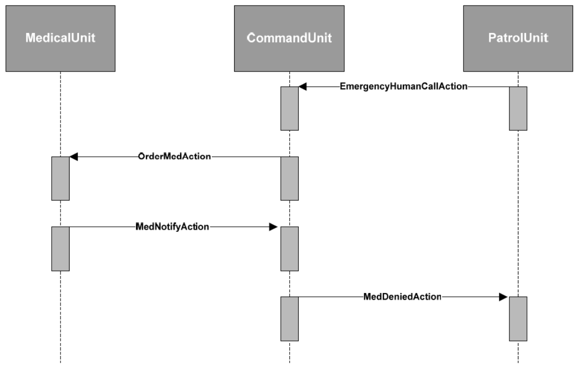

Figure 7. Sequence diagram for CommandUnit.

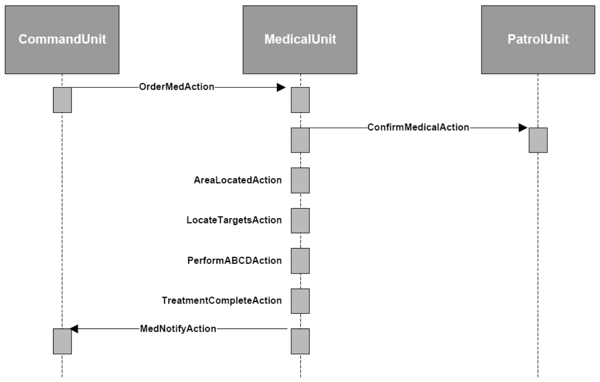

Figure 8. Sequence diagram for MedicalUnit.

After we found this semantically valid match combination, we tried to break the matching conditions in order to see how the prototype would react. For example, we removed *EmergencyHumanCallAction* from the **PatrolUnit** BOM. The prototype displayed a condition violation at the static matching level. We also reordered the sequence of the actions, e.g. we changed the order of *ConfirmMedicalAction*

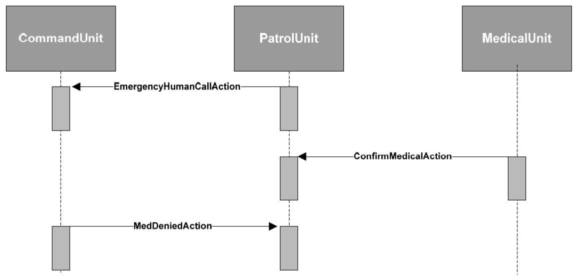

Figure 9. Sequence diagram for PatrolUnit.

to occur earlier. This resulted in a condition violation at the dynamic matching level.

We manipulated the facts in our ontology, e.g. we changed a prerequisite property ("medical equipment") for a **MedicalUnit**. The prerequisite property was needed for a **MedicalUnit** in order to perform its services. When we removed this property, the rule: "if a medical unit has medical equipment, then it is a valid medical unit" became false (individual did not occur in inferred set). The result of this was that the entity and the BOM became semantically invalid.

5. Conclusions & Future Work

The case study shows how BOMs are matched and under which conditions a set of BOMs are composable according to our definition in Section 2. The example highlights when static and dynamic matching fails, and shows that by changing property values for the individuals in the ontology we can get different inferred results, hence indicating if an element in the BOM is semantically valid or not.

The algorithms used for BOM matching and semantic validation are at some level straightforward, this creates complexity issues. The more BOMs that exist in a repository, the more time consuming the matching process becomes. For future work it would be interesting if matching and validation could be done using faster algorithms (perhaps through heuristics). Complexity issues have to be resolved either by introducing new algorithms or reducing the size of data using different filtering techniques. Otherwise the technology is not applicable in a real world application.

The relationship between a BOM and an ontology raises an important technical question. Is the current BOM format good enough for representing the semantics of information? If we look at the current format there is only a simple text field for expressing this. It is therefore not unreasonable to think that the BOM format could be expressed entirely in OWL. This way we would not only store the old BOM related information, but also the semantic aspect of information in an explicit manner. This type of combination creates a much richer BOM for future use in e.g. simulations.

BOM++ [10] is an example of such a proposal that has already been suggested by FOI.

References

[1] H. Asadi, M. Garcia-Lozano, A. Horndahl, E. Tjörnhammar, K. Rasch. Introduction to Technologies and Methods for Semantic Information Management. Technical Report FOI-R–2858-SE, Swedish Defence Research Agency, 2009. ISSN 1650-1942.

[2] I. Mahmood, R. Ayani, V. Vlassov, F. Moradi. Statemachine Matching in BOM based model Composition. 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, 2009.

[3] IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP), 2003. 1516.3.

[4] IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) Framework and Rules, 2010. 1516.

[5] Protege. Homepage of Protege.
`http://protege.stanford.edu/`.
Last accessed 2011-03-10.

[6] Simulation Interoperability Standards Organization (SISO). Base Object Model (BOM) Template Specification, 2006. SISO-STD-003-2006.

[7] Simulation Interoperability Standards Organization (SISO). Guide for Base Object Model (BOM) Use and Implementation, 2006. SISO-STD-003.1-2006.

[8] Simulation Interoperability Standards Organization (SISO). SISO Base Object Model (BOM) Template Specification, 2006. SISO-STD-003-2006.

[9] Simventions. Homepage of Base Object Model.
`http://www.boms.info`.
Last accessed 2011-03-10.

[10] V. Mojtahed, B. Andersson, V. Kabilan, J. Zdravkovic. BOM++, a semantically enriched BOM. Spring Simulation Interoperability Workshop, 2008.

[11] V. Mojtahed, E. Tjörnhammar, J. Zdravkovic, A. Khan. The Knowledge Use in DCMF, Repository, Processes and Products. Technical Report FOI-R–2606-SE, Swedish Defence Research Agency, 2008. ISSN 1650-1942.

[12] V. Mojtahed, M. Eklöf, H. Asadi, J. Zdravkovic, E. Svee. Slutrapport för projektet DCMF. Technical Report FOI-R–3059-SE, Swedish Defence Research Agency, 2010. ISSN 1650-1942.

[13] V. Mojtahed, M. Garcia-Lozano, P. Svan, B. Andersson. DCMF Defence Conceptual Modeling Framework. Methodology Report. Technical Report FOI-R–1754-SE, Swedish Defence Research Agency, 2005. ISSN 1650-1942.