IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 28, 2018, accepted March 30, 2018, date of publication April 3, 2018, date of current version April 23, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2822661

Domain Specific MetaModeling for Deep Semantic Composability

ZHI ZHU[1,2], YONGLIN LEI[1,2], ABDURRAHMAN ALSHAREEF[2], HESSAM SARJOUGHIAN[2], AND YIFAN ZHU[1]

[1]Department of Military Modeling and Simulation, School of System Engineering, National University of Defense Technology, Changsha 410073, China
[2]Arizona Center for Integrative Modeling and Simulation, School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ 85281, USA

Corresponding author: Zhi Zhu (zhuzhi@nudt.edu.cn)

This work was supported by the Natural Science Foundation of China under Grant 61273198.

ABSTRACT Current simulation models are not only represented in the form of traditional data or formula for pure theory analysis but expanded to be simulation modeling assets that are featured with complicated structure, diverse behaviors, and abundant semantics. Semantic composability, therefore, receives a constantly growing attention in recent years. So far, one of the popular solutions to enhance semantic expressiveness is domain specific modeling based on general metamodeling (GMM) facilities. But for some particular domains, researchers identified the need of deeper semantic expressiveness therefore proposed domain specific metamodeling (DSMM). Hence, this paper aims to explore the underlying methodologies of DSMM for supporting deep semantic composability. Compared with several usual alternatives based on GMM, this paper applies the multi-level metamodeling architecture to create a set of metamodeling primitives using an example named SEvent. In fact, SEvent is a novel formalism that slightly extends Petri net to support continuous states transition and continuous event triggering. As a proof of concept, we concentrate on developing the textual syntax of SEvent and using it to represent torpedo's behaviors.

INDEX TERMS DSL, metamodeling, semantics, composability.

I. INTRODUCTION

Different simulation modelers interpret and represent a similar model in different ways, leading to the variety of models in terms of syntax, structure, and semantics. Consequently, models are syntactically independent and semantically difficult to reach a consensus, which could result in model reuse difficulties and a low model composability level.

Model composability [1] has different levels. In general, it contains deep semantic composability (also called full composability), semantic composability, syntactical composability, and no composability. Furthermore, syntactical composability has several levels of difficulties with respect to the type of model heterogeneity [2]. Also, semantic composability has different levels regarding the depth of semantic mapping and matching between model components [3].

The goal of model composability is model reuse [4]. On the one hand, it is necessary to describe information about model interfaces by standard model specifications. On the other hand, it requires the standardized expression of domain concepts and relationships, and needs to support the consistency between model concepts and model implementations, since it is beneficial for simulation modelers to early judge whether given models can be composed or not. In fact, the task of model reuse is more to compose models with others than purely to use models repeatedly, thus to realize the fast development of models in a composable way. Therefore, both model reuse and model composability blend into one integrity with two sides to some degree, which means that model reuse foundationally requires the syntactic composability of different models, and what's more, it should ensure the semantic validity after models are composed completely. Specifically, semantic composability is concerned with the abstract or semantic relations that naturally exist among different model components which have the desire to work cooperatively. Therefore, it is possible to satisfy the individual and diverse requirements of simulation modelers to develop simulation applications by organizing these available model components to form a dynamic alliance.

Unfortunately, there is no general rule or guidance that can be provided to define a good solution. However, semantic composability can be evaluated according to some basic

principles derived from abundant practice [5], [6]. In this paper, we summarize some basic criteria that need to be satisfied for a successful approach to support semantic composability. For this purpose, we pay attention to three aspects listed as follows and these criteria will be used to evaluate the case example in Section IV.

1. Syntactical compatibility. Important though semantic composability is to integrate models into a meaningful whole, the premise is that these models should be compatible syntactically at least. That is, the aim is to explore an effective solution to reduce or eliminate the technical gap exists between model components.

2. Model abstraction. Since it is preferable for modelers to specify domain knowledge as intuitive as possible, the goal is to raise the level of abstraction of models to be closer to the problem domain and to be away from the implementation details. As such, modelers can concentrate on how to directly introduce domain concepts and relations into models so as to enhance their semantic expressiveness.

3. Model evolvability. Although model evolvability has a lot of connotations, the requirements derived from M&S community give more focus on evolving model representations in a convenient way. In this regard, the aim is to describe a model without considering on a specific technology, tool or platform, so as to allow modelers to update models in a more understandable and modifiable way according to the new vendor demands.

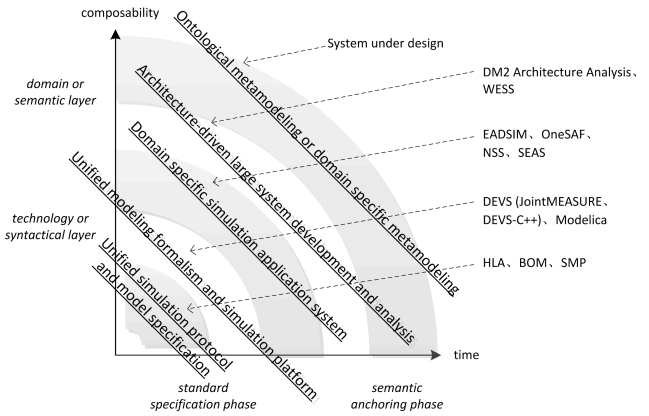

FIGURE 1. The trend of system modeling and simulation paradigms.

So far, studies on addressing model composability could mostly summarize into two phases: standard specifications and semantics anchoring. The ways formed at the first phase mainly focus on the syntactical heterogeneity from the technological perspective, while the second phase attempts to address the semantic difference from the domain's point of view, as shown in Fig. 1. Specifically, standard specifications aim to build a commonly accepted specification or formalism to represent models in a unified form, like the unified simulation protocols (e.g. HLA [7]), model specifications (e.g. BOM [8], SMP [9]), modeling formalisms (e.g. DEVS [10], Modelica [11]), and simulation platform (e.g. JointMEASURE [12], DEVS-C++ [13]). The semantics anchoring, whereas, intends to define good mapping relationships between model elements and domain concepts for a particular domain, like domain specific simulation systems (e.g. EADSIM [14], SEAS [15], etc.), and architecture-driven large system development and analysis (e.g. WESS [16], DM2 [17]), as well as ontological metamodeling [18] or domain specific metamodeling which is studied in this paper.

The trend in Fig. 1 shows that it is not sufficient to adopt a generic simulation platform that realizes one or a few standard model specifications or formalisms, neither is it sufficient to create a dedicated simulation platform for a special type of system. Rather, the key is to explore a well-defined mechanism to represent domain knowledge in a commonly understandable way that could be closer to the problem domain not the implementation details.

Domain specific modeling (DSM) as such a mechanism has effectively applied to capture domain concepts and structure in a deeper semantic way than the general-purpose programming language (GPL) [19]. However, several researchers have identified the shortage of the use of domain specific languages (DSL) because they are normally defined through the general-purpose metamodeling (GMM) facilities. Hence, domain specific metamodeling (DSMM), at a higher abstract meta-level, is proposed to customize the metamodeling primitives aiming at the definition of modeling languages for a specific domain.

The remainder of this paper is structured as follows. Section 2 introduces the metamodeling architecture based on GMM and presents two representative facilities, i.e. UML profile mechanism and EMF. With more flexible expressiveness, Section 3 focuses on designing DSMM facilities from scratch based on the multi-level metamodeling architecture. In Section 4, a slight extension of Petri net, namely SEvent, is defined as a demonstrative example and the development of its textual concrete syntax is shown, then this DSL is used to describe torpedo's physical behaviors. Section 5 summarizes this paper and gives a brief discussion on the assessment of the proposed approach.

II. DSL DESIGN WITH GMM FACILITIES

As mentioned before, DSLs are normally defined through the GMM facilities like MOF (Meta-Object Facility) and Ecore. Generally, these DSLs comprise two meta-levels [20], i.e. the definition of a DSL and its immediate use. This section introduces three alternatives within the architecture of GMM facilities: UML metamodel refinement, UML metamodel extension, and new metamodel definition, and lastly makes a simple comparison of these alternatives in regard to their properties and capabilities for specifying a certain system.

A. THE METAMODELING ARCHITECTURE BASED ON GMM FACILITIES

In metamodeling, the term *meta-level* is used to indicate the level of which a language element possesses, so as to

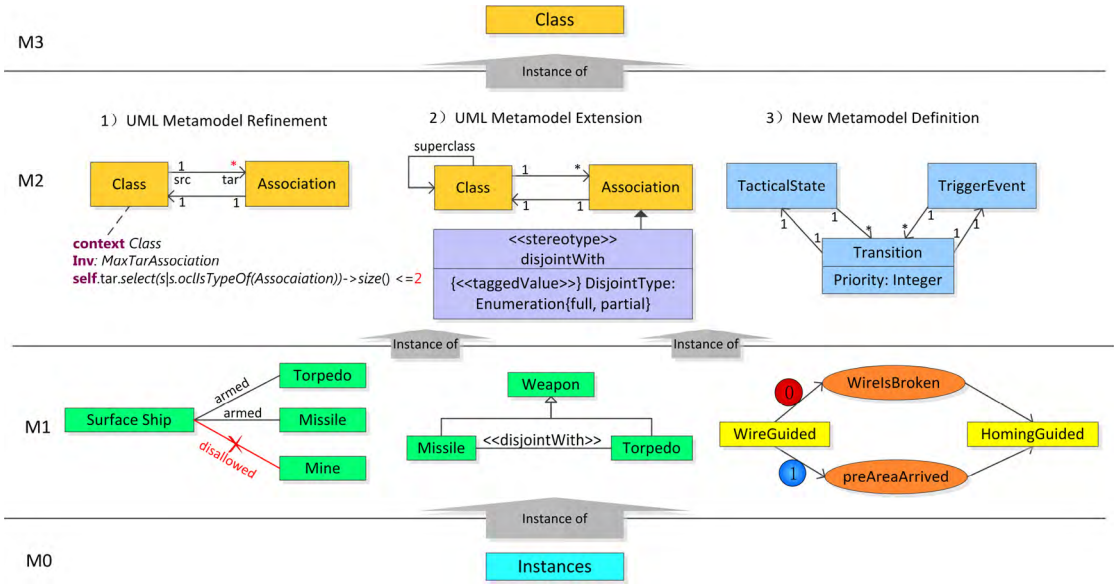

FIGURE 2. Three metamodeling alternatives using GMM facilities.

represent the level structure of a metamodel and the model that the metamodel defines. Typically, MOF is a strict four meta-levels metamodeling architecture that is widely used as a principle in the language engineering literature, in which every model element on every meta-level is strictly in correspondence with a model element of the meta-level above [21]. The four meta-levels conceptual architecture creates an infrastructure for customizing a new modeling language or making future language extensions.

Usually, there are three metamodeling alternatives based on MOF [22]: UML metamodel refinement, UML metamodel extension, and new metamodel definition, as shown in Fig. 2. Since the previous two do not change the UML metamodel, both of which are known as UML Profile [23]. The last one defines a metamodel from scratch without the conformance to the UML metamodel, like EMF (Eclipse Modeling Framework). An important point needs to be mentioned here is that either UML Profile or EMF is based on a GMM facility.

1) Metamodel refinement does not permit the modification of existing metamodels. Therefore, the semantics and the structure of UML metamodel cannot be modified, and the introduction of new language elements into the metamodel are not permitted. Moreover, additional domain specific constraints can be added, but changes to existing constraints in the UML metamodel are not allowed. So this method is more suitable for those extensions are highly specific to a particular domain. In a combat effectiveness simulation system (CESS) [24], for example, assume a particular prototype of surface ship only can be equipped with two kinds of weapons. We use OCL (Object Constraint Language) to describe such a constraint, as shown in Fig. 2 (left), because a Class in the UML metamodel is allowed to be associated with multiple classes. As a result, in the M1 meta-level, the surface ship to be equipped with a mine is not allowed when it has already been armed with both a torpedo and a missile.

2) Metamodel extension makes an extension of an existing modeling language by supplementing it with fresh domain specific constructs. It is considered to be a more flexible approach since new concepts may be represented in the metamodel. However, it may lose some support from existing tools due to the introduction of new language elements. For instance, it makes sense to say that both missiles and torpedoes are children of the *Weapon* class, and the subset *Missile* is disjoint with the subset *Torpedo*. This is to say that, in the real world, there exists not a weapon that is able to have both the torpedo and the missile properties simultaneously. So we create a stereotype named $\ll$*disjointWith*$\gg$ to be applied on the *Association* class of the UML metamodel. In addition, this stereotype has a tag named "*DisjointType*," indicating two different kinds of disjoint relationships: full and partial. The full disjoint relationship occurs between two complete disjoint relationships, i.e. given three sets A, B, C, where $B \subset A, C \subset A$, and $B \cap C = \emptyset \wedge B \cup C = A$, but if $B \cap C = \emptyset \wedge B \cup C \subseteq A$, saying B and C are partially disjoint with each other.

3) New metamodel definition is a process of metamodeling from scratch. Obviously, it enjoys the highest flexibility to customize a metamodel directly using the domain specific concepts. For example, based on EMF/Ecore, we define a *TacticalState* and a *TriggerEvent* as well as a *Transition* relationship that has a tag named "*priority*." All of these elements will be instantiated at the next meta-level with a complete individual and distinctive style. However, this method may suffer from some drawbacks because developing tools for the new defined metamodel is difficult and

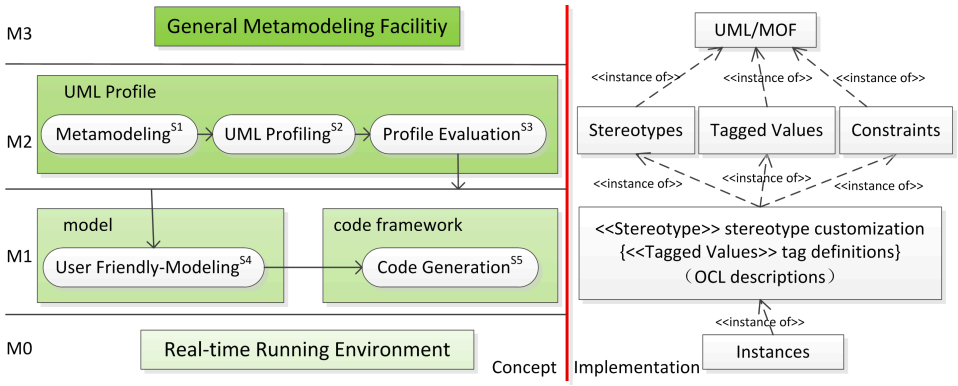

FIGURE 3. DSL design framework based on UML profile.

expensive, which could be worse in the case of metamodels with sophisticated and deep semantics.

Above mentioned methods are widely used by researchers and engineers in the metamodeling field [25], [26]. Some are from the technical perspective to address general syntactical problems, whereas others are more concerned with domain oriented issues. In either case, a wide lack of semantic composability exists. We compare the properties and capabilities of these methods, in terms of their effects on change of UML metamodel, semantic expressiveness, use of existing resources, and training expenditure, each of which is denoted with three levels, as shown in Table 1.

TABLE 1. A simple comparison of the GMM alternatives.

MTHODES	UML CHANGE	SEMANTIC EXPRESSIVENESS	EXISTING USE	TRAINING EXPENDITURE
Refinement	little	shallow	good	small
Extension	medium	medium	medium	medium
Definition	none	deep	bad	large

In addition, it is known that a single modeling language is not enough to cover all the various concerns involved in a specific domain. However, currently there have been little guidelines applied to define a DSL due to the diverse of disciplines. Yet there is not a definite answer for the discussion about the suitability of UML Profile extensions or metamodeling from scratch when facing a particular domain. But, it is important to conclude experience to avoid pitfalls in practice. Next we will discuss these metamodeling methods using the UML profile and EMF mechanisms.

B. UML PROFILE AND EMF MECHANISMS

Nowadays, many UML profiles may be either invalid because they are conflict with the standard UML principles, or incomprehensive because they do not adequately capture necessary semantic information. Therefore, overcoming these problems is a key work that an effective framework needs to do for defining well-formed and flexible profiles. It requires not only the domain expertise but also proficiency in UML modeling [27]. Fig. 3 shows a framework of DSL design based on UML Profile.

This framework conforms to the MOF four meta-levels metamodeling architecture and mainly contains five key stages which are labeled by the letter "S" plus a number: S1 metamodeling, S2 UML Profiling, S3 profile evaluation, S4 user friendly-modeling, and S5 code generation. Note the former three stages belong to the range of language definition, while the latter two stages concentrate on the use of languages.

1) Metamodeling (S1). This stage is concerned with the process of explicit modeling to capture the abstract syntax of a modeling language. In general, the abstract syntax model should include the following elements, i.e. a set of fundamental language constructs, a set of valid relationships, a set of constraints, the concrete syntax and the semantics.

In practice, it is not easy to define the above key elements because they usually become woven together, so it is of importance to identify some experience to get a valid profile of good quality. Firstly, one should specify the domain model without any consideration of UML metamodels. Secondly, one should adjust the domain model even with some loss of expressiveness if conflict occurs. Thirdly, one should check the domain model structure to reduce the complexity of constraints.

2) UML Profiling (S2). This stage is an activity of selecting the most suitable UML base language elements adapting to the domain concepts. Initially, the UML base classes should be selected semantically similar to the domain concepts. Furthermore, not all the stereotypes are from the UML base classes but may be other forms such as inheritance. At last, the selected UML base classes are not always well aligned with the domain concepts even being contradictory, thus proper constraints should be constructed to resolve these conflicts.

3) Profile evaluation (S3). This stage is to evaluate the correctness of a profile usually by constructing its concrete implementations. Firstly, create an object diagram to ensure the correctness of the abstract syntax model. Secondly, define

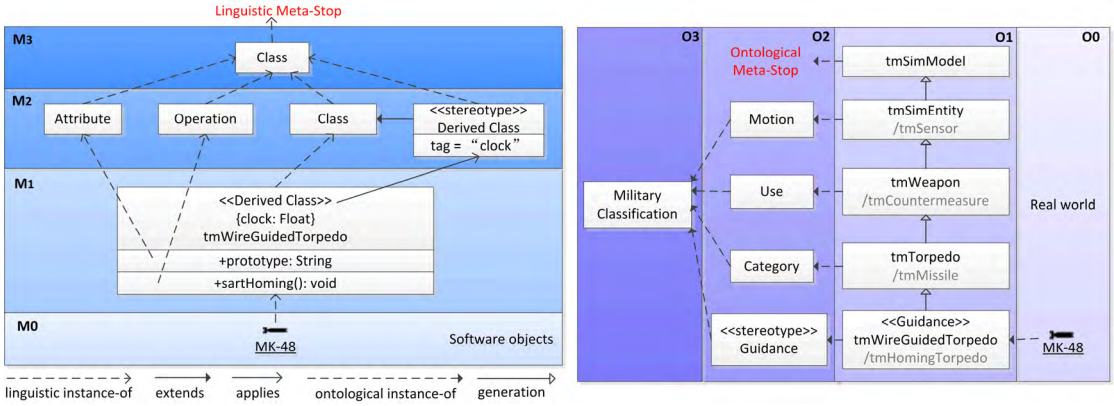

FIGURE 4. Typing from the linguistic and ontological perspectives.

rules to transform the profile to another language that has precise and well-defined semantics. Thirdly, develop a domain specific tool for the profile to implement the concrete syntax.

The latter two stages (i.e. user friendly-modeling and code generation) use the validated profile to build some models for a specific domain. In practice, we often find some practical modeling issues when using the DSL, which is inevitable because some issues are always discovered in the long run of use, even though the domain experts are very confident for their professional knowledge. Thankfully, with such a general framework accompanying with much practical experiences, one can easily tackle with the model evolvability and clearly renew languages based on previous ones.

EMF is another mean to design DSLs, which usually has close relationships with a set of OMG standards, like UML, MOF, XMI, and MDA and so on. Firstly, UML is widely used to capture various concerns of a certain system by an object-oriented method, emphasizing multi-view to describe the structure, behavior, function, and deployment, etc. While, EMF as a way of defining metamodels is only concerned with one aspect of a system, i.e. class structure. Secondly, EMF/Ecore focuses on the tool sets not the metadata warehouse management, thus avoiding some of the complex issues such as data structure, package relationships, and associations compared to the MOF. Thirdly, XMI is a widely accepted serializing standard which is not only used as the format for serializing EMF models, but also suitable for serializing the metamodel, i.e. Ecore itself.

This method is very different with the UML profiling mechanism because it defines metamodels from scratch without considering the UML rules. Hence, it has the potential for the most direct and succinct expression of domain concepts. Furthermore, it has a collection of supporting tools (e.g. GEF and GMF) thanks to the Eclipse open source architecture.

Once a DSL is completed, it does not mean that one can process to the subsequent work without any change of the DSL. In many cases, the DSL may show its shortcoming along with its wide application since the GMM facilities cannot always be qualified to specify domain concepts and relations comprehensively. Even some of modeling elements are contradictory with the domain specific characteristics. When this happens, special care must be taken to raise the level of abstraction as a result. For instance, it is time to abandon the GMM facilities and explore a more flexible approach having deeper semantics to express complex domain concepts and relations.

III. DOMAIN SPECIFIC METAMODELING

Unlike DSLs design using the GMM facilities where users are given the full power of a general-purpose metamodeling language, the DSMM approach provides a set of more suitable metamodeling primitives that tailored to a particular metamodeling task or application. Hence, the resulting DSMM language contains primitives of a specific domain with deep semantics, thus is closer to the problem domain than using the GMM facilities. This section adds another ontological dimension to the usual linguistic definition, then introduces the general DSMM architecture, and provides a motivating example for DSMM languages.

A. TYPING FROM TWO PERSPECTIVES

Typing an element is an important method to improve the meta-level of domain concepts, thus realizing the maximum reuse of model information at the level above. Base on the identified two separate orthogonal dimensions of metamodeling [28], the method has two distinct forms of typing, i.e. ontological typing and linguistic typing, as shown in Fig. 4. Ontological typing locates a model element from the perspective of domain definition hence uses ontological typing, e.g. it makes sense to say that a wire guided torpedo is a torpedo, weapon, and so on. Linguistic typing is concerned with languages definition hence uses linguistic typing, e.g. the linguistic type of *tmWireGuidedTorpedo* is the *Class*, while the *Attribute* can be instantiated to the field "*prototype*," and the instance of *Operation* is *startHoming()*. Both forms work simultaneously to precisely provide the location for a specific element within the domain space.

Traditionally, researchers have long emphasized the linguistic typing to address the syntactical comosability, e.g. UML/MOF infrastructure as shown in Fig. 4 (left), while encouraging the ontological typing as a subservient mechanism to enhance the semantic composability, e.g. UML Profile. However, UML Profile as a kind of lightweight extending mechanisms provides a known limited expressiveness using stereotypes, tagged values, and domain specific constraints as well. For example, it is possible to express that the timed *tmWireGuidedTorpedo* is an instance of the *Class* by applying the stereotype ≪*DerivedClass*≫ which is tagged by the "*clock*" property, but not all the domain concepts and relationships can be available suitably, e.g. states, events, associations, or generation relationships. Ideally, ontological typing should play an equal role to linguistic typing and vice versa. Neither should be attached to the other.

Fig. 4 (right) shows the process of the ontological typing by a wire guided torpedo example within the military effectiveness simulation space. Metaconcepts such as *Guidance*, *Category*, and so on allow new created classes to be added to the military classification system. Note that these metaconcepts can be viewed as a way of classification. For example, the military system has wire guided torpedo and homing torpedo if it is classified by a guidance way. While, *Torpedo*, *Missile*, and so on are classified if a category way is adopted. Also, a use way derives weapons, countermeasures, platforms, etc., and a motion way gives birth to entities and sensors.

In addition to the ontological and linguistic typing methods, we introduce a meta-stop principle that is significant to terminate the unlimited meta-level of domain knowledge abstraction. For a specific meta-level, if only exists one single element or exclusive elements, the meta-level structure should stop intuitively at this level since there is not the need to set a higher meta-level to abstract its commonalities, which is called the meta-stop principle. For example, the UML/MOF architecture stops at the M3 level accordingly because the elements at this level are always disjoint without overlapping parts.

According to the different perspectives of typing an element, the meta-stop principle also includes ontological meta-stop principle and linguistic meta-stop principle. On the one hand, ontological meta-stop principle requires that the domain definition terminates at a ontological meta-level whose domain elements are exclusive with each other, including the special case of only one domain element existing at this meta-level. On the other hand, the linguistic meta-stop principle declares that the language definition should terminate if no similar language elements exist at that meta-level.

B. GENERAL ARCHITECTURE OF DSMM

To support the dual linguistic/ontological typing of model elements, this study, similar to the DSMM architecture with three meta-levels [29], applies the multi-level approach [30] which can define indirect properties at several meta-levels below. In the multi-level framework, an element has a type facet that is able to be instantiated at the next meta-level and an instance facet in which instances are typed at the meta-level above. Hence, the term *Clabject* is used to represent an element that encompasses both facets, i.e. the union meaning of class and object, as shown in Fig. 5.

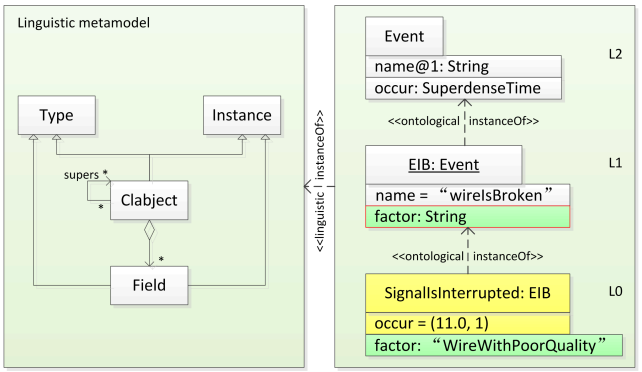

FIGURE 5. General architecture of DSMM.

Moreover, there is another important point with respect to the instantiation relationship of two dimensions: linguistic and ontological *instance of*. On the one hand, the model element *signalIsInterrupted* (L0) is an ontological instance of *EIB* (L1), hence resides a lower ontological level than *EIB*. Furthermore, *EIB* is an ontological instance of *Event* (L2). On the other hand, one can interpret all of the modeling elements in the right column as being conformant to the linguistic metamodel in the left column. For instance, *Event*, *EIB*, and *SignalIsInterrupted* are all the linguistic instances of *Clabject*.

To make the instantiation across multiple meta-levels, the multi-level framework uses a notation "*@X*" that will be attached to models, clabjects, fields, and associations, representing the concept of the level. It is a natural number (including zero) that indicates which meta-level the attaching element will be instantiated at. For example, the field *name@1* is assigned a value *wireIsBroken* at the next meta-level, and the field *occur* is assigned a value (11.0, 1) at the next two meta-level because this field receives the level of its container *Event* if this notation ("*@X*") is not explicitly given. In addition, a linguistic extension mechanism is used to create a new element that has only linguistic type, but has no ontological types. Mostly, this mechanism is useful to express those elements which are specific to the particular application at a specific meta-level. For example, the field *factor* that has not an ontological type is contained by the clabject *EIB* and is assigned a value *WireWithPoorQuality* at the L0 meta-level. Moreover, it is sometimes use more sophisticated metamodeling facilities to assist for the trimness and neatness of models. For example, an abstract clabject that cannot be instantiated may be used to define commonalities for elements, e.g. *ST* is an abstract state that is used to derive other states, thus getting a model structure with a good look, which will be seen later.

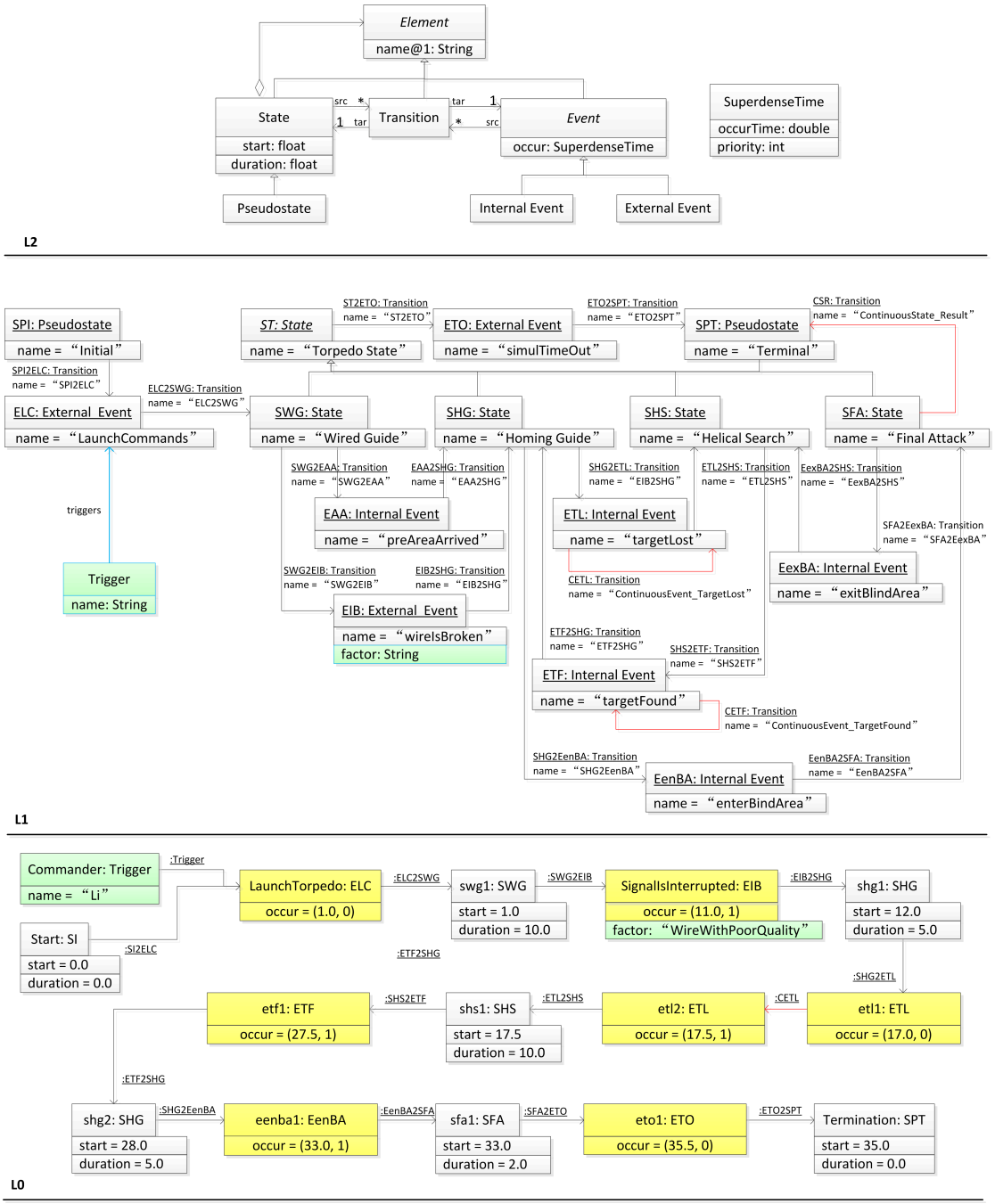

FIGURE 6. DSMM for torpedo physical behaviors representation.

C. DEEP SEMANTIC METAMODELING: A SIMPLE DEMONSTRATION

Assume we need to describe the physical behaviors for particular domains, like torpedo, fighter, and so on. These behavioral models should be able to represent the states, events, and their transitional relationships, etc. Before, we had discussed GMM facilities, either UML Profile [31] or EMF, to build such models, but results showed undesirable. Therefore, our aim is to define a metamodeling language facilitating the construction of behavioral models for specialized domains such as torpedo. Fig. 6 shows such an example as a simple demonstration of DSMM with deep semantics. The DSMM approach consists of three meta-levels, which are respectively labeled by L2, L1, and L0. At the L0 meta-level defines the metamodeling facilities constituting a DSMM language. At L1 meta-level uses the primitives with which the DSMM language provides to define a DSL. This DSL will be used to build the torpedo physical behavioral models at L0 meta-level.

At the L2 meta-level, DSMM language defines three core elements, *State*, *Event*, and *Transition*, each of which is generated from *Element*. *Element* is abstract and has a field labeled

by *name* which will be instantiated at next meta-level, thus *name* has an attached notation "*@1*." In addition, *State* has fields *start* and *duration* representing a system starts a state at a time point and will stay in the state for a time interval. Also, *Event* has a field *occur* typed by a data type *SuperdenseTime*. To specify some events occurring at the same simulation time but in different sequence, *SuperdenseTime* is introduced to represent casually related weakly simultaneous events [32]. Its value is a pair (t, n), called a time stamp, where t is the simulation time and n is a microstep (also called an index). In general, two time stamps (t1, n1) and (t2, n2) are weakly simultaneous if t1 = t2, and strongly simultaneous if n1 = n2 as well.

At the L1 meta-level, the defined DSMM language is used to define DSLs for other domains, like torpedo physical behavioral modeling. Additionally, consider some specific needs of torpedo domain, this meta-level may make a linguistic extension that should not be defined at the meta-level above. This DSL makes the torpedo physical behavioral modeling more natural than the GMM paradigms, like Petri net [33], Finite State Machine, and UML Activity Diagram, as it includes the following specialized sets of model elements.

1) A set of states including elements *SPI (name=Initial)*, *SWG (name=Wire Guide)*, *SHG (name=Homing Guide)*, *SHS (name=Helix Search)*, *SFA (name=Final Attack)*, *SPT (name=Terminal)*, and *ST (name=Torpedo State)*, representing the lifecycle of a torpedo once launched.

2) A set of events including elements *ELC (name= LaunchCommands)*, *EAA (name=preAreaArrived)*, *EIB (name=wireIsBroken)*, *ETL (name=targetLost)*, *ETF (name=targetFound)*, *EexBA (name=exitBlindArea)*, *EenBA (name= enterBlindArea)*, and *ETO (name= simulTimeOut)*, representing the overall possible events that may be triggered during the lifecycle of a torpedo.

3) A set of transitions including elements *SPI2ELC*, *ELC2SWG*, *SWG2EAA*, *EAA2SHG*, *SHG2ETL*, *ETL2SHS*, *SFA2EexBA*, *EexBA2SHS*, *CSR (name=ContinuousState_ Result)*, *ETO2SPT*, *ST2ETO*, *SWG2EIB*, *EIB2SHG*, *CETL (name=ContinuousEvent_TargetLost)*, *ETF2SHG*, *SHS2ETF*, *CETF (name=ContinuousEvent_TargetFound)*, *SHG2EenBA*, and *EenBA2SFA*, whose names that are similar to their linguistic types are omitted for brevity. Note that *CSR* represents the continuous transition between states, and *CETL* and *CETF* represent the continuous transition between events.

4) A set of linguistic extensions including the clabject *Trigger (name: String)*, relationship *triggers*, and field *factor: String*, representing the entities who triggers the events, triggering relations, and factors causing the events triggering, respectively. These concepts are all the specialized elements of torpedo physical behavioral domain so that there are not corresponding ontological types at the meta-level above. Another important point is that the field *factor* that has not an ontological type is embedded in an event *EIB* which has yet an ontological type.

Using DSMM languages for defining varieties of DSLs, we may need to conclude some commonalities underlying these DSLs, and improve the meta-levels of these commonalities as being the basic model elements of DSMM. Afterwards, DSL designers can reuse maximally the DSMM facilities to define DSLs for the domains that they are familiar with. Usually, DSL designers can be qualified for the role of DSMM designers, but DSLs are generally expected to be used by particular domain experts, like the torpedo experts. At the L0 meta-level, torpedo experts will use the defined DSL to build concrete torpedo physical behavioral models. Models at this meta-level include four sets of model instances.

1) A set of states including instances *Start (start=0.0, duration=0.0)*, *swg1 (start=1.0, duration=10.0)*, *shg1 (start=12.0, duration=5.0)*, *shs1 (start=17.5, duration= 10.0)*, *shg2 (start=28.0, duration=5.0)*, *sfa1 (start=33.0, duration=2.0)*, and *Termination (start=35.0, duration=0.0)*, representing each state of a concrete torpedo, and the concrete start time and time interval of each state are given.

2) A set of events including instances *LaunchTorpedo (occur=<1.0, 0>)*, *SignlIsInterrupted (occur=<11.0,1>)*, *etl1 (occur=<17.0, 0>)*, *etl2 (occur=<17.5, 1>)*, *etf1 (occur=<27.5, 1>)*, *eenba1 (occur=<33.0, 1>)*, and *eto1 (occur=<35.5, 0>)*, representing each event of this torpedo instance. Each event is labeled by the concrete triggered time that is denoted by a pair *<a, b>*, to set the triggering order of simultaneous events. Note that the variable *a* indicates the triggered time point of an event, while *b* refers to the priority of this event to be triggered. Besides, we regulates that an event (*b=0*) possess of the highest priority thus should be triggered firstly, and the greater b is, the lower the priority is possessed of.

3) A set of transitions including instances whose names are omitted but linguistic types are reserved *:Trigger*, *:SI2EIC*, *:ELC2SWG*, *:SWG2EIB*, *:EIB2SHG*, *:SHG2ETL*, *:CETL*, *:ETL2SHS*, *:SHS2*, *:ETF2SHG*,*:SHG2EenBA*, *:EenBA2SFA*, *:SFA2ETO*, and *:ETO2SPT*, respectively representing the transition instances between states and events. Note that *:CETL* is a continuous event transition instance from event instance *etl1* to event instance *etl2*.

4) A set of linguistic extensions including instances *Commander (name=John)* and *factor=WireWithPoorQuality*. In the real word, this represents that a commander whose name is John gives the order to launch a torpedo, and the detection is deterred due to the poor quality of the torpedo cable.

IV. CASE STUDY: SEVENT

As stated before, we attempted to use the GMM facilities to define DSLs for building torpedo physical behavioral models but the result shows to be failed or unsuitable. So we have to seek the DSMM approach despite it may lack some supports from the existing tools. As a proof of concept, this section gives the formal SEvent definition, shows its concete syntax development based on Xtext, and uses a more suitable multi-level approach to design the textual syntax.

A. SEVENT DEFINITION

The descriptive power of DSLs is an important factor in much of the success of MDE projects. One of the current issues raise in the adoption and application of DSLs is the lack of a precise description of the semantics of a DSL [34]. As such, it is desirable to give a formal definition for a under designed DSL before specifying its syntax and semantics by a typical metamodel.

Similar to the core concepts of transitions (i.e. events that may occur, represented by bars) and places (i.e. conditions, represented by circles) in Petri net, SEvent also consists of states and events. Differently, SEvent has more powerful expressiveness in that it is able to support continuous states transition and continuous events triggering, whereas Petri net conforms to the alternative use of places and transitions. In addition, each element of the SEvent metamodel uses the executive semantics of Petri net's, for instance, it is possible to provide an execution method to the element which represents the notion of transition in order to specify what happens when a transition is fired. As a result, it is possible to execute any model which conforms to the SEvent metamodel.

Definition 1 (SEvent): A 5-tuple $SEvent = (S, T, F, F_s, F_t)$ is defined as:

- S is a finite set of states
- T is a finite set of events
- F is the set of the ordered pairs $\langle s, t\rangle$, where $s \in S, t \in T$
- F_s is the set of ordered pairs $\langle s, s\rangle$, where $s \in S$
- F_t is the set of ordered pairs $\langle t, t\rangle$, where $t \in T$

where

1) $S \cup T \neq \emptyset \wedge S \cap T = \emptyset$, i.e. S and T are disjoint that no object can be both a state and an event.
2) $F \subset S \times T \cup T \times S$, i.e. F is set of ordered pairs whose left projection belongs to S and the right projection belongs to T.
3) $F_s \subset S \times S, \langle s_i, s_j\rangle_{i\neq j, i,j\subset Ns}$, Ns is the size of S, i.e. F_s is the set of ordered pairs whose both of projections belong to S, and the left projection must not be equal to the right projection because the description of transitions between a same state is meaningless.
4) $F_t \subset T \times T, \langle t_i, t_j\rangle_{i\neq j, i,j\subset Nt}$, Nt is the size of T, i.e. F_t is the set of ordered pairs whose both of projection belong to T, and the left projection must not be equal to the right projection because the description of self-triggering of an event is meaningless.
5) $dom(F) \cup cod(F) = S \cup T, dom(F_s) \cup cod(F_s) = S, dom(F_t) \cup cod(F_t) = T, dom(F)$ is the set of left projections of F, while $cod(F)$ is the set of right projections of F.

Note that Petri net is viewed as a special case of SEvent, i.e., given $SEvent = (S, T, F, F_s, F_t)$, if $F_s = \emptyset \wedge F_t = \emptyset$, thus SEvnet is equal to Petri net.

B. XTEXT BASED TEXTUAL SYNTAX DEVELOPMENT

Nowadays, the DSL textual syntax development generally falls into two categories. One is to use existing standard tools for defining language textual syntaxes, like Xtext [35], TCS [36], and ANTLR [37]. This category is very useful for the definition of the concrete syntax of the DSMM/DSM language then use at the immediate meta-level below. The other is a multi-level approach to support the DSMM. Using the multi-level approach, one has to define syntaxes for the DSMM language as well as for the language built with it, i.e. the designer need to provide both the syntax at meta-level L1 and the syntax at meta-level L0. Moreover, it should be also possible to support the definition of syntaxes for the linguistic extension.

Fig. 7 shows the syntax definition of DSMM language for SEvent as well as its use at the meta-level below. At the left contains four packages of the main project, SDK, Test, and UI. At the right is an editor for writing the rules of the syntax definition, mainly including *SEventDataType*, *SEventNameType*, *Connection*, and *Node*.

1) Node (lines 19-21).

It includes two kinds of rules, *State* or *Event*, which are labeled by "*State | Event*." The *State* rule (lines 32-39) defines a keyword "*state*," the *Feature* rule (lines 56-58), the *Transition* rule (lines 52-54), and a composite state encapsulating other states and events (line 36). The *Event* rule (lines 41-46) defines a keyword "*event*," the *Feature* rule, and the *Transition* rule.

Using above SEvent rules at the next meta-level, the DSL for torpedo physical behavioral domain contains eight nodes, i.e. *Trigger* (linguistic extension), *SPI*, *SWG*, *SHG*, *SHS*, *SFA*, *SPT*, and *ST*, and eight events, i.e. *ELC*, *EAA*, *EIB*, *ETL*, *EenBA*, *ETF*, *EexBA*, and *ETO*, where the field *factor* that embedded into its container *EIB* is a linguistic extension.

The *Feature* rule (lines 56-58) defines the name and the type of a field. A field type can be the *SEventDataType* (lines 23-26) or the *SeventNameType* (lines 28-30). In addition, the *Transition* rule (lines 52-54) defines a transition from a *Connection* cross reference to a *Node* cross reference.

2) Data Type (lines 23-26).

It includes the name of a data type, i.e. "*name=ID*," and supports the definition of a composite data type. In addition, it contains multiple optional elements, i.e. "*features += Feature*," and note the *Feature* here is a rule not a cross reference. In SEvent, data type uses a keyword named "*sevent_datatype*" and has four data types, i.e. *String*, *Float32*, *Int32*, as well as the composite data type *SuperdenseTime* which is composed of *occurTime* and *priority*.

3) Name type (lines 28-30).

It only has the name, i.e. "*name = ID*," and uses a keyword "*sevent_nametype*." The DSL based on SEvent has seven name types for *State*, i.e. *Torpedo State*, *Initial*, *Wire Guide*, *Homing Guide*, *Helical Search*, *Final Attack*, and *Terminal*, eight name types for *Event*, i.e. *LaunchCommands*, *preAreaArrived*, *wireIsBroken*, *targeLost*, *targetFound*, *enterBlingArea*, *exitBlindArea*, and *simulTimeOut*.

4) Connection (lines 48-50).

It includes the name and the code of a connection, i.e. "*name = ID code = ID*." The use of SEvent at the

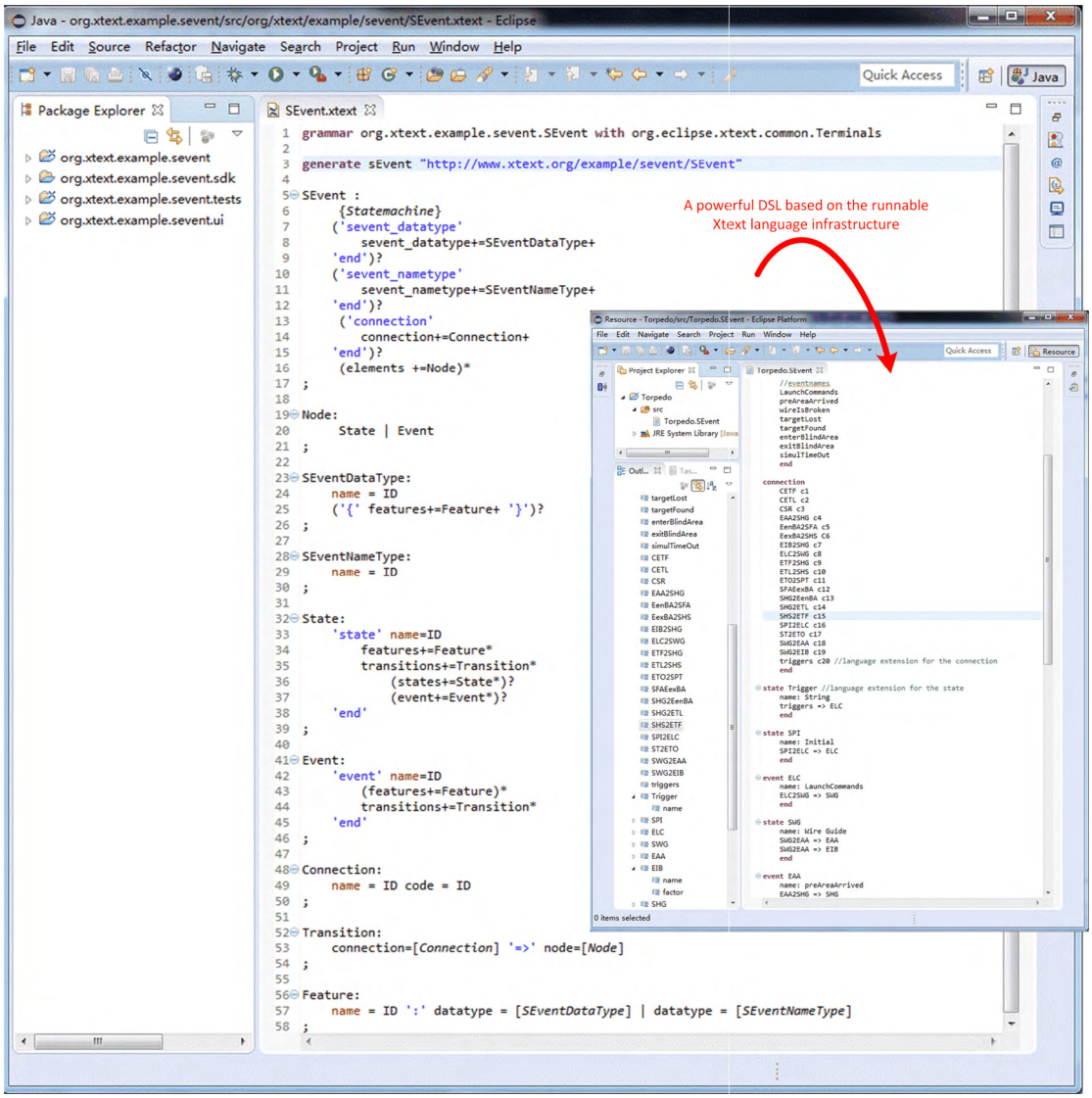

FIGURE 7. Xtext based textual syntax definition of SEvent.

next meta-level defines 20 connections, i.e. *CETF c1*, *CETL c2*, *CSR c3*, *EAA2SHG c4*, *EenBA2SFA c5*, *EexBA2SHS C6*, *EIB2SHG c7*, *ELC2SWG c8*, *ETF2SHG c9*, *ETL2SHS c10*, *ETO2SPT c11*, *SFAEexBA c12*, *SHG2EenBA c13*, *SHG2ETL c14*, *SHS2ETF c15*, *SPI2ELC c16*, *ST2ETO c17*, *SWG2EAA c18*, *SWG2EIB c19*, and *triggers c20*, where *c20* is a linguistic extension.

C. MULTI-LEVEL TEXTUAL SYNTAX DESIGN BASED ON METADEPTH

As stated above, Xtext based textual syntax development benefits from the existing standard tools and can be integrated properly with other released tools in the Eclipse community. However, it suffers from the two-level metamodeling so that we have to redefine the syntax structure when going on below meta-levels for describing implementation details. In practice, it is preferable to explore a more flexible approach to describe information across multiple meta-levels so as to avoid unnecessary iterations.

MetaDepth [38], [39] is an IDE for building a template language that supports the multi-level textual modeling and implements the deep semantics through a notation mechanism "*@X*." It is a self-contained system and provides two mechanisms to control the way in which the designed DSMM languages will be used and extended, as not any extension may be appropriate for a certain language. On the one hand, MetaDepth uses modifiers (e.g. *strict*) to indicate the non-extendable language elements. Hence, once an element is labeled by *strict*, its instances cannot be extended with new attributes, references, or constraints. For example, if we define the *External Event* as *strict* at the L2 meta-level, then its instance *EIB* at the L1 meta-level with a linguistic extension "*factor: String*" is illegal. For another example, if the SEvent is strict, then it is illegal to add a new language

element "*Trigger*" at the L1 meta-level. On the other, it may be possible to define constraints to ensure a certain extensibility degree for the non-strict language elements. For example, it is possible to ask an element (e.g. the *ETL* at the L2 meta-level) to declare some fields acting as the identifier, which will be instantiated at the L1 meta-level. In theory, we could define this identifier at the L2 meta-level with a notation "*@2*," but some identifiers (e.g. the NO. of an *ETL* instance) for some reasons should be decided by the DSL designer at the L1 meta-level and instantiated at the L0 meta- level. For this purpose, we need to construct some constraints to improve the DSMM language expressiveness.

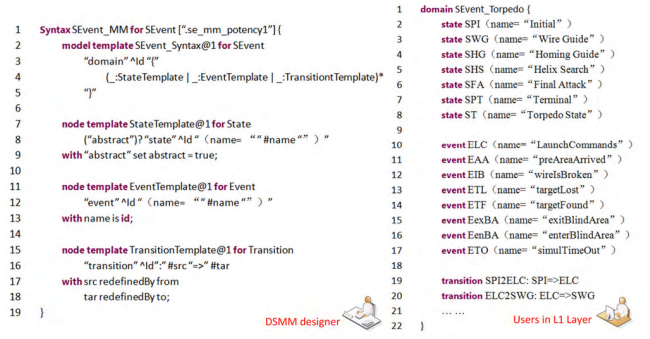

FIGURE 8. Definition of SEvent textual syntax and its use at the L1 layer.

1) Fig. 8 (left) shows a definition of the SEvent textual syntax and at the right shows its use at the L1 meta-level. At the left of this figure, line 1 declares the file extension for the DSMM language ("*se_mm*"). Lines 2-5 define the model template acting as an entry point, which can be associated to other templates, e.g. zero or multiple state templates, event templates, or transition templates. In particular, all of templates are tagged "*@1*," indicating that these templates will be used at the next meta-level. Lines 7-9 define the syntax for state, in which the keyword "^*ID*" stands for the identifier of an element (line 8), the prefix "*#*" returns the value of a field (line 8), whereas the *with* keyword introduces semantic actions and syntactic predicates (lines 9, 13, 17). Line 8 is a semantic action that sets the final property to true when the "*final*" token is recognized. Line 13 is a syntactic predicate that declares the *name* field as the identifier. Line 19 will trigger two semantic actions that *src* and *tar* will be respectively refined by *from* and *to* at the next meta-level.

2) DSL users also need to be provided with a concrete syntax to describe the models at the L0 meta-level, as shown in Fig. 9. At this time, those syntaxes defined at the L2 meta-level will be reused at the L0 meta-level. Similar to Fig. 8 that consists of two parts, Fig. 9 (left) shows the definition, whereas the right shows its use at a certain meta-level. Initially, the file extension ("se") is declared and all of the templates in this file are tagged "@2." In practice, however, the DSMM designers do not know the model type for which the syntax is defined, but only know this model type is an indirect instance of SEvent (line 2). Hence, MetaDepth uses the keyword "^*Typename*" to access name of the concrete

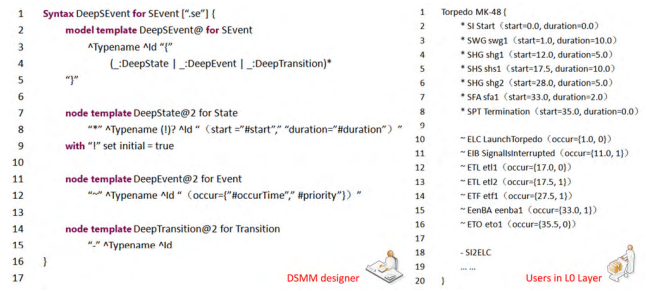

FIGURE 9. Definition of SEvent textual syntax and its use at the L0 layer.

type (line 3). Fig. 9 (right) shows the use of the DSL to build a concrete model which is prototyped by "MK-48." This model contains several instances of states, events, and transitions.

3) As discussed before, MetaDepth limits the extendability of the DSMM language through either tagging strict language elements or constructing OCL constraints. In practice, these should be considered when defining the concrete syntax of a DSMM language in order to avoid the illegal linguistic extensions.

MetaDepth allows to access to the linguistic layer of models by four keywords with respect to models, i.e. *Extends*, *Imports*, *LingElements*, and *Constraints*, as well as nine for model elements, i.e. *Extends*, *Id*, *Type*, *Typename*, *Fields*, *Constraints*, *Supers*, *FieldVaulues*, and *Instances*. Some of these keywords allow linguistic extensions by notations such as "^*Field*" decalring new atrributes, "^*LingElements*" creating new clabjects, "^*Constraints*" constructing constraints, "^*Supers*" allowing inheritance relationships, "^*FieldValue*" for field instances, and "^*Instances*" for clabject instances. In addition, it is necessary to use some semantic actions and syntactical predicates. For example, it is possible to extend the syntactical definition as shown in Fig. 8 to define new language elements (without ontological types) such as linguistic types like "*Trigger*," relationships like "*triggers*," fields like "*factor*," and constraints. Fig. 10 shows these linguistic extensions.

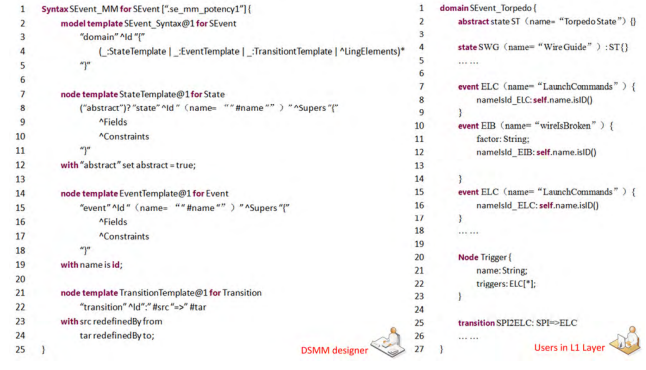

FIGURE 10. Definition of SEvent linguistic extensible textual syntax and its use at the L1 layer.

Initially, line 4 at Fig. 10 (left) uses "^*LingElement*" to define new language elements without ontological types, thus lines 20-23 at the right uses the syntax to make a language extension "*Trigger*." Then, line 8 at the left figure uses "^*Supers*" to define inheritance relationships, thus the state *SWG* in line 4 at the right are generated from the abstract state *ST*. It is similar to the states *SHG*, *SHS*, etc. which are omitted for brevity. After that, at the left figure both lines 9-10 and lines 16-17 uses "^*Field*" and "^*Constraints*" to define new constraints, which are used to create a new field "*factor*" (line 11) at the right as well as construct new constraints to ensure the names of events are identifiers (lines 8, 12, 16). Finally, line 12 at the left figure uses a semantic action to set the *abstract* property, thus at the right the state *ST* is abstract when the keyword "*abstract*" is interpreted by the parser (line 2).

D. SEMANTIC COMPOSABILITY IN SEVENT

The case example showed that deep semantic composability is obtained when the DSMM approach is applied successfully. Consider the criteria that are already presented in Section I, we detail how the case example satisfy them.

First of all, SEvent is a slight extension of the Petri net formalism and has a formal syntactical base. This provides SEvent with accurate and unambiguous semantics. Furthermore, due to the fact that the metamodeling facilities live on a relative high level of abstraction, the stability of foundational facilities is implicitly guaranteed. The benefit is that model heterogeneity reduces a lot thus models based on the similar metamodeling facilities can be easily composed.

Secondly, the case example is based on the multi-level metamodeling architecture and satisfies the model abstraction requirement. This architecture customizes the metamodeling facilities for a particular application, and then uses these primitives to define a DSL. In this way, the language is of deep semantics to be able to represent those domain characteristics that GMM cannot be qualified.

Lastly, the model evolvability requirement is partially supported since we made small scale examples and more experiments are needed for a better evaluation. In addition, thanks to the fact that DSMM provides a set of domain specific metamodeling primitives of SEvent, more user friendly modeling capabilities are guaranteed to represent other domains without considering a particular platform.

V. CONCLUSION

Improving the level of abstraction has long been recognized as a useful way to free system modelers from thousands of concrete details. Such an improvement makes people pay more attention on how to interpret and understand models than its underlying technological implementations. In this context, DSLs acquire enormous successes in terms of their friendly semantic representations for many MDE projects. Typically, UML Profile is illustrated as one successful case to extend the basic UML metamodel with additional semantics for a specific domain. Another representative example is EMF, which needs not conform to the UML metamodel and therefore is able to describe more semantic information.

However, these DSLs are normally defined through the GMM facilities, i.e. MOF and Ecore. Despite they are supported by a collection of existing modeling resources, such as professionals, tools, tutorials and so on, they still shows pale or defective for certain domains, such as the SEvent described in this study. Hence, this study takes SEvent as a proof of concept and attempts to describe this example by applying the DSMM method. For this purpose, a general DSMM architecture with two orthogonal dimensions (i.e. ontological and linguistic) is introduced to simplify the definition and use of DSMM languages. Within the architecture, SEvent as a light extension of Petri net is shown to be able to support the representation of continuous states and events, demonstrating the overall process of three meta-levels deep semantic metamodeling. Finally, these considerations are implemented by developing the textual concrete syntax of DSMM languages, using Xtext and MetaDepth. The semantic composability of DSMM adoption is obtained according to three evaluation criteria.

Currently, there are little guidelines to instruct the DSMM design and implementation. Thus, a benefit of this study is that it can be viewed as a referenced experience to guide other domains that have the needs of DSMM. Moreover, it reviews several widely used domain specific modeling methods within the model driven engineering literature. However, as a drawback some effort for further usage and evaluation details is required.

REFERENCES

[1] H. S. Sarjoughian, "Model composability," in *Proc. 38th Winter Simulation Conf.*, Dec. 2006, pp. 149–158.
[2] C. Hardebolle and F. Boulanger, "Exploring multi-paradigm modeling techniques," *Simulation*, vol. 85, nos. 11–12, pp. 688–708, 2009.
[3] C. Szabo and Y. M. Teo, "An analysis of the cost of validating semantic composability," *J. Simulation*, vol. 6, no. 3, pp. 152–163, 2012.
[4] C. Szabo and Y. M. Teo, "On syntactic composability and model reuse," in *Proc. 1st Asia Int. Conf. Modelling Simulation*, Mar. 2007, pp. 230–237.
[5] M. Challenger, G. Kardas, and B. Tekinerdogan, "A systematic approach to evaluating domain-specific modeling language environments for multi-agent systems," *Softw. Quality J.*, vol. 24, no. 3, pp. 755–795, 2016.
[6] M. Estañol, M.-R. Sancho, and E. Teniente, "Ensuring the semantic correctness of a BAUML artifact-centric BPM," *Inf. Softw. Technol.*, vol. 93, pp. 147–162, Jan. 2018.
[7] *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Framework and Rules*, Standard 1516-2010, IEEE Computer Society, 2010. [Online]. Available: http://www.ieee.org/HLA
[8] SISO Base Object Model Product Development Group. (2005). *Base Object Model (BOM) Template Specification*. [Online]. Available: http://www.sisostds.org
[9] European Space Agency (ESA). (2005). *SMP 2.0 Handbook (Issue 1 Revision 2) EGOS-SIM-GEN-TN-0099*. [Online]. Available: http://www.eurosim.nl/support/manuals/manual_4_2/pdf/SMP_2.0_Metamodel-1.2.pdf
[10] B. P. Zeigler, H. Praehofer, and T. G. Kim. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, 2nd ed. San Diego, CA, USA: Academic, 2000.
[11] Modelica Association. (2013). *Modelica-A Unified Object-Oriented Language for Systems Modeling Language Specification Version 3.3*. [Online]. Available: http://www.modelica.org/
[12] S. B. Hall, B. P. Zeigler, and H. S. Sarjoughian, "Joint measure TM: Distributed simulation issues in a mission effectiveness analytic simulator," in *Proc. Simulator Interoperability Workshop*, 1999, pp. 1–7.

[13] K.-M. Seo, C. Choi, T. G. Kim, and J. H. Kim, "DEVS-based combat modeling for engagement-level simulation," *Simulation*, vol. 90, no. 7, pp. 759–781, 2014.
[14] M. C. Azar, "Assessing the treatment of airborne tactical high energy lasers in combat simulations," Ph.D. dissertation, Air Force Inst. Technol., Dayton, OH, USA, 2003.
[15] J. O. Miller, L. Jason, and B. Honabarger, "Modeling and measuring network centric warfare (NCW) with the system effectiveness analysis simulation (SEAS)," in *Proc. 11th ICCRTS*, 2006, pp. 1–22.
[16] Z. Zhu, Y. Lei, Y. Zhu, and H. S. Sarjoughian, "A WESS-based method for anti-submarine simulation through planning waypoints of helicopter (WIP)," in *Proc. Summer Comput. Simulation Conf.*, 2016, Art. no. 15.
[17] IDEAS Group. (Oct. 2016). *IDEAS*. [Online]. Available: http://www.ideasgroup.org/dm2/
[18] D. ÃŘjuric, D. Gaševic, and V. Devedžic, "A MDA-based approach to the ontology definition metamodel," in *Proc. 4th Inter. Workshop Inf. Tech.*, 2003, pp. 51–54.
[19] S. Kelly and J.-P. Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. Hoboken, NJ, USA: Wiley, 2008.
[20] M. Strembeck and U. Zdun, "An approach for the systematic development of domain-specific languages," *Softw.-Pract. Exper.*, vol. 39, no. 15, pp. 1253–1292, 2010.
[21] G. Nordstrom, J. Sztipanovits, G. Karsai, and A. Ledeczi, "Metamodeling-rapid design and evolution of domain-specific modeling environments," in *Proc. IEEE ECB Conf.*, Apr. 1999, pp. 68–74.
[22] B. Selic, "A systematic approach to domain-specific language design using UML," in *Proc. 10th IEEE Inter. Symp. Object Compon.-Oriented Real-Time Distrib. Comput.*, May 2007, pp. 2–9.
[23] M. S. Abdulah, "A UML profile for conceptual modeling of knowledge-based systems," Ph.D. dissertation, Dept. Comput. Sci., Univ. York, York, England, 2006.
[24] Y. L. Lei, Z. Zhu, Q. Li, F. Yang, and Y. Zhu, "WESS: A generic combat effectiveness simulation system," in *Proc. 17th Asia Simulation Conf.*, 2017, pp. 272–283.
[25] T. Clark, P. Sammut and J. Willans, *Applied Metamodeling: A Foundation for Language Driven Development*, 3rd ed. New York, NY, USA: Ceteva, 2015.
[26] D. Çetinkaya, "model driven development of simulation models: Defining and transforming conceptual models into simulation models by using metamodels and model transformations," Ph.D. dissertation, Dept. Multi Actor Syst., Delft Univ. Technol., Delft, The Netherlands, 2013.
[27] Z. Zhu, Y. L. Lei, Y. F. Zhu, A. Alshareef, and H. S. Sarjoughian, "A unifying framework for uml profile-based cognitive modeling: Development and experience," in *Proc. 10th EAI Int. Conf. Simulation Tools Tech.*, Sep. 2017, pp. 1–10.
[28] C. Atkinson and T. Kuhne, "Model-driven development: A metamodeling foundation," *IEEE Softw.*, vol. 20, no. 5, pp. 36–41, Sep. 2003.
[29] J. D. Lara, E. Guerra, and J. S. Cuadrado, "Model-driven engineering with domain-specific meta-modelling languages," *Softw. Syst. Model.*, vol. 14, no. 1, pp. 429–459, 2013.
[30] C. Atkinson and T. Kühne, "The essence of multilevel metamodeling," in *Proc. Int. Conf. Unified Modeling Lang.*, 2001, pp. 19–33.
[31] Z. Zhu, Y. Lei, Y. Zhu, and H. Sarjoughian, "Cognitive behaviors modeling using UML profile: Design and experience," *IEEE Access*, vol. 5, pp. 21694–21708, 2017.
[32] C. Ptolemaeus, *System Design, Modeling, and Simulation using Ptolemy Ii*. Berkeley, CA, USA: Univ. California, 2014.
[33] C. A. Petri and W. G. Reisig, "Petri net," *Scholarpedia*, vol. 3, no. 4, p. 6477, 2008.
[34] B. R. Bryant, J. Gray, M. Mernik, P. J. R. B. Clarke France, and G. Karsal, "Challenges and directions in formalizing the semantics of modeling languages," *Comput. Sci. Inf. Syst.*, vol. 8, no. 2, pp. 225–253, 2011.
[35] Eclipse. (Feb. 2017). *Xtext*. [Online]. Available: http://www.eclipse.org/Xtext/
[36] F. Jouault, J. Bézivin, and I. Kurtev, "TCS:: A DSL for the specification of textual concrete syntaxes in model engineering," in *Proc. 5th Int. Conf. Generative Program. Compon. Eng.*, 2006, pp. 249–254.
[37] Eclipse. (Mar. 2017). *ANother Tool for Language Recgnition (ANTLR)*. [Online]. Available: http://www.antlr.org/
[38] J. de Lara and E. Guerra, "Deep meta-modelling with MetaDepth," in *Proc. Int. Conf. Modelling Tech. Tools Comput. Perform. Eval.*, 2010, pp. 1–20.
[39] J. de Lara and E. Guerra, "Domain-specific textual meta-modelling languages for model driven engineering," in *Proc. Eur. Conf. Modelling Found. Appl.*, Jul. 2012, pp. 259–274.

ZHI ZHU was born in Anshun, Guizhou, China, in 1989. He received the B.S. degree in information management and information system from Sichuan University, Chengdu, Sichuan, China, in 2011, and the M.S. degree in control science and engineering from the National University of Defense Technology, Changsha, Hunan, China, in 2013. He is currently pursuing the Ph.D. degree in computer simulation with the National University of Defense Technology.

From 2016 to 2017, he was a Visiting Ph.D. Student with the Arizona Center for Integrative Modeling and Simulation, School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ, USA. He has co-/authored over 20 articles and over five industry projects. His research interests include model-driven engineering, simulation-based system design and demonstration, and domain-specific modeling. He has attended and presented at international conferences, such as the 2016 Summer Simulation, Montreal, Canada, and the 2017 Simulation Tools, Hong Kong.

Mr. Zhu was a recipient of the National University of Defense Technology Project for Excellent Ph.D. Candidate in 2015 ($5000). Email: zhuzhi@nudt.edu.cn

YONGLIN LEI received the B.S. degree in education economy from Xi'an Jiaotong University, Xi'an, Shanxi, China, in 2000, and the M.S. and Ph.D. degrees in management science and engineering from the National University of Defense Technology, Changsha, Hunan, China, in 2002 and 2006, respectively. He is a currently an Associate Professor with the National University of Defense Technology.

From 2014 to 2015, he was a Visiting Scholar with the Arizona Center for Integrative Modeling and Simulation, School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ, USA. He is the first author of over two books, over 50 articles indexed in SCI and EI, and over five industry projects (over $400 000). His research interests include complex system simulation, model composability, model-driven architecture, domain-specific modeling, and their applications in defense simulations.

Dr. Lei's major awards and honors include the National University of Defense Technology for Excellent Ph.D. Thesis in 2006 and the National University of Defense Technology for Excellent Technical Personnel in 2012.

ABDURRAHMAN ALSHAREEF received the B.S. degree in information systems from the College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia, in 2005, the master's degree in information technology with a focus on software architecture from the Queensland University of Technology, Brisbane, Australia, in 2010.

He is currently pursuing the Ph.D. degree in computer science program with the School of Computing, Informatics and Decision Systems Engineering, Arizona State University, Tempe, AZ, USA. He is currently a Lecturer with the College of Computer and Information Sciences, King Saud University. His research interests lie in the discrete event system models development and simulation, model-driven engineering, software architecture, and metamodeling.

Mr. Alshareef was a recipient of the King Saud University Scholarship for master's and Ph.D. degrees. He was also a recipient of travel grants from the Saudi Arabia Cultural Mission in USA, the School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, and from the Society for Modeling and Simulation International to show his research in different conferences.

HESSAM SARJOUGHIAN received the B.S. degree from Mississippi State University in 1984 and the M.S. and Ph.D. degrees from the University of Arizona in 1988 and 1995, respectively, all in electrical and computer engineering. He is a currently an Associate Professor and the Director of the Arizona Center for Integrative Modeling and Simulation, School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ, USA.

His research has been supported by NSF, Intel, DARPA, and Boeing among others. His industry experience has been with IBM and Honeywell. He has been the architect and the lead for the DEVS-Suite simulator, which is being used at universities and research institutes in many countries across America, Europe, Asia, and Africa. His research interests include agent-based modeling, multiformalism modeling, simulation-based design, and software architecture. Since 2004, he has been the Area Editor of the SIMULATION: TRANSACTIONS OF THE SOCIETY FOR MODELING AND SIMULATION.

Dr. Sarjoughian is a Founding Member of the Certified Modeling & Simulation Profession, a Certified Modeling and Simulation Professional. He was a recipient of the SCS Distinguished Service Award.

YIFAN ZHU received the B.S. degree in solid mechanics from Peking University, Beijing, China, in 1983, the M.S. degree in solid mechanics and the Ph.D. degree in systems engineering from the National University of Defense Technology, Changsha, Hunan, China, in 1989 and 2003, respectively. He is currently a Professor and the Director of the Simulation Engineering Institute, School of Information System and Management, National University of Defense Technology.

From 2007 to 2008, he was a Visiting Scholar with the Virginia Polytechnic Institute and State University, Blacksburg, VA, USA. Since 2000, he has been the Director of the China Simulation Federation, the Assistant Sectary General with the China Military Science Society, and an Editorial Board Member with the *Journal of System Simulation*. He is the first author of over five books, over 80 articles indexed in SCI and EI, and over 10 industry projects (over $600 000). His research interests include simulation-based system design and demonstration, and agent-based modeling and simulation.

Dr. Zhu's major awards and honors include the National Ministry Technological Process for one first prize, three second prizes, and three third prizes.