# Conceptual Model and Perdurantist Modeling with Reasoning

M. Fatih Hocaoğlu

Istanbul Medeniyet University,
Faculty of Engineering & Architecture,
Department of Industrial Engineering, Istanbul, mfatih.hocaoglu@medeniyet.edu.tr

**Abstract.** In this study, the perdurantist modeling approach that entities have four dimensions (spatial and temporal) and only briefly exist during the different stages of their lifespan is extended by a reasoning mechanism. The extension allows a modeler to manage behaviors depending on reasoning results and also provides well-designed support for time-delayed systems. Language support is provided for this purpose, starting with an ontological commitment and covering design and coding phases, including conceptual model description. This paper discusses how an agent-driven simulation language supports extending a perdurantist modeling to ontology-based modeling by high-level action descriptions, higher-order world envisionment, dynamic relation management and a knowledge base for reasoning purposed as the necessities of ontology based modeling. One of the study's main aims is to match the predicate logic ontological commitment with the ontological commitment presented here and bring them into a comment framework to handle behavioral management in simulation.

## Introduction

Agent-driven Simulation Framework (AdSiF) [1], [2] is a declarative simulation language and a development environment for simulation and agent programming. It basically provides a state-oriented interpreter and a simulation layer to manage simulation execution algorithms for both discrete-event and continuous-event systems. Compared to current agent programming systems (such as Jason [3]), AdSiF gives a different perspective by fusing the agent-based programming paradigm, object-oriented paradigm, aspect-oriented paradigm [4] and logic programming paradigm [5] into a single paradigm, referred to here as a state-oriented paradigm [1], [6]. State-oriented programming (SOP) was originally introduced in 1987 by D. Harel [7] as a visual formalism to model complex reactive systems. SOP was adopted by OMG in 1997 as a part of UML 2.0 specification and is based on state charts [8]. AdSiF enhances state-oriented programming using multi-programming paradigms and defines it as a programming language. As a language, AdSiF provides programming by states instead of the programming states as performed in the state charts (as is done in Generic State Chart Libraries). It interprets the extended state charts and does not require coding the chart itself. State-oriented programming handles the state transition process, which is declared in the form of the State Charts of AdSiF. In each state, the simulation model spends a certain amount of time to pass through the entire state (or, at least, the model attempts to pass through the whole state) in an orderly fashion, and the simulation models are capable of executing many behaviors in parallel, at any time. The execution of a state-oriented program has a timeline due to the duration that the simulation model spends in each state. The power of this paradigm stems from its ontological commitment, which extends from describing what exists to include the modeling of mental abilities through the use of a reasoning mechanism, thereby driving behaviors.

From the ontology-based modeling point of view, AdSiF provides a strong programming background to satisfy the fundamental ontological notions, which are identity, unity, rigidity and dependency. Ontologically, AdSiF constructs a world composed of entities capable of managing their behaviors reactively and proactively.

The behavioral aspect of an entity consists of a set of behaviors, which is defined as a sequence of states, a set of events triggering behaviors and behavioral declaration of the relations it has. A behavior is created by putting states, each of which represents a specific atomic action, in order according to a reasonable sequence, so that they represent the behaviour captured in state charts. In this respect, behaviors undertake an important role in sharing domain semantic. In addition, because of the logic programming paradigm, capabilities of agent-hood modeling characteristics, continuous/discrete simulation support and symbolic time management, AdSiF provides a perdurantist modeling environment. Perdurantist approaches assume that objects have four dimensions (spatial and temporal) and only briefly exist during the different stages of their life span [9]. That is entities only exist for a period of time and continually change over such periods. Such entities are unfolding themselves over time in successive temporal parts [10]. Therefore, objects are viewed from the past, present and future. According to this paradigm, entities are usually referred to as “space-time worms” or a slice of such a worm [11] given that they are identified based on space and time dimensions.

The paper focuses on ontology-based modeling enriched with logic programming and reasoning. It aims to give an answer and a means of connecting logic and ontology with each other. The role of relations, behavioral aspects and reasoning mechanism are also emphasized in ontology-based modeling and the perdurantist modeling approach is extended by a reasoning mechanism. The extension allows modelers to manage behaviors depending on reasoning results and also gives well-designed support for time-delayed systems. One of the main aims of the study is to match the predicate logic ontological commitment with the ontological commitment presented here and bring them into a comment framework to handle behavioral management in simulation. A language support is provided for this purpose, beginning with an ontological commitment and covering design and coding phases, including conceptual model description.

The paper is organized as follows. The first section introduces brief information about ontology-based modeling. The second section (2) focuses on logic programming and how it supports entity description and a relation concept between entities. In the fourth section, AdSiF ontology support is presented and then, in the following section, it is exemplified with an example from an air defense simulation.

## 1 Ontological Commitment of AdSiF

An ontological commitment refers to a relation between a language and certain objects postulated to be extant by that language. The overall philosophical project of ontology is categorized into at least two parts: the first part is about what there is, what exists and what the thing is is made from and the second part concerns what are the most general features and relations of these things.

Concepts, relations of the phenomenon and the objects surrounded by this phenomenon are strongly related to the conceptual model and the conceptual model paradigm that is most effective for ontology modeling. From this point of view, not only does AdSiF provide multi-modeling paradigm support, but also it combines the paradigms into a single paradigm termed state-oriented paradigm. This gives a rich expressiveness that is one of the quality metrics of an ontology [12]. The paradigms, which are supported by AdSiF and combined in state-oriented programming, are logic programming, aspect-oriented programming, agent-based programming and object- oriented programming.

A system has a time base, inputs (events), states, behaviors, a reasoning mechanism and a mechanism that manages the dynamic characteristics of the system. In AdSiF, the dynamic characteristic of the system is represented by the behavior descriptions (in the specialized state charts). As a framework, AdSiF promotes a set of design rules and presumes a design skeleton, which is based on its programming paradigm, called the SOP paradigm, and its ontological view. AdSiF provides an ontological view, which is defined as follows in terms of the paradigms, on which AdSiF are based.

AdSiF’s ontological commitment covers time, space and provides an answer as to what exists, posits a type of relation definition between existances and it is given below.

*Entities live in a certain environment and have their own properties that distinguish them from each other and an atomic action that manages their properties (OOP perspective). The atomic action creates the interactions that change the environment where the entities reside and share the interactions with other* *entities as a communication element. The entities sequentially implement their atomic actions in a reasonable semantics called behavior, and the behaviors are executed in parallel and/or sequentially. Each atomic action is wrapped by a state that constructs the behaviors (SOP perspective). The entities interact with one another using event transactions and constitute relations among one another. An interaction has autonomy, reactivity, and a goal — (AgOP perspective). From a taxonomic view, the behavior categories of an entity represent different behavioral aspects of the entity —(AOP perspective). An entity has beliefs and facts about the environment and about the other entities with which they share the environment. These beliefs and facts constitute a fact dual world envision that contains the entity; the envision may have a set of goals to succeed and a reasoning mechanism with a set of decision-making algorithms — Logic Programming (LP perspective).*

AdSiF's ontology covers the common properties of entities following an inheritancy path. Common properties are defined as public or protected and inherited from base models, as in OOP. Agenthood perspective also gives another property of the ontology, such as being in an environment, reacting to events happening around and trying to achieve certain goals by behaving proactively. Also, an entity changes its behavioral aspects depending on the conditions which it is in .

## 2 Logical Envisonment and Behavior Management by Reasoning

As presented in the ontological commitment, simulation models and agents have facts (beliefs) about the environment in which they exist and truth-preserved predicates to infer new facts, relations and identities. The time-stamped facts about simulation or agent environment not only constitute a fact dual world representation as an inner representation of the environment in which the models are, but also the dual world retains knowledge of the time axis with past values. Dual world representation is defined as an inner representation of the environment created by the entity. Each entity has its own representation of the environment in which it is in and this is constituted via sensed and inferred information. This allows modelers to associate truth level and define temporally valid (for a certain duration) knowledge for both truths about simulation models sharing the same environments and the relations (dependencies) between them. This can be seen as a modeling characteristic supporting the *rigidity* notion [12] of ontology-based modeling. The capabilities are enriched by a logic paradigm which turns a perdurandist model into a model which has cognitive capability and can manage relations dynamically.

Semantic representation of an agent or a simulation model (namely, an entity) is represented by behavioral descriptions. Any interaction, in other words any event transition between simulation entities and agents, causes a set of behavioral reactions, either reactive or proactive. Each behavior taken consists of a series of actions connected with each other logically. In addition to the behavior that is activated by an event, activated or cancelled by a condition, generally, managed behavior after a series of reasoning processes is a good example of shared semantic. An example for activating a behavior as a result of reasoning is given in Section 5.

Meta-knowledge, higher-order rules are vital for both agents and ontology-based modelling. To enable the development of high-level easy-to-configure agent behavior, it is important to provide agents with the means to reason about their surrounding environment using a generic reasoning mechanism and knowledge base. The agents must be able to analyze unexpected situations to dynamically adapt their behavior to achieve their personal goals [13]. This means agents must be capable of evaluating situations using abstract, higher-order rules. One of the higher-order sources comes from the answers to such ontology questions as "what is it?" or "what can be said to exist", because the answers given consist of a set of relations from a specific instance to more abstract ones. In other words, an answer becomes more and more generalized up to the infinitesimal. The answers constitute a fact set for rules binding to the behaviors. This provides a means to be able to make a decision about any new instance inserted into the knowledge base generalizing it. In AdSif, a rule consists of a head and a body, similar to Prolog syntax. The head and body are connected by a symbol :-, which is made up of a colon : and a hyphen –[14]. The ":-" is pronounced *if*. Return parameters of a rule are used to bind to Boolean logical expressions (Figure 1) and rule truth value, which shows whether it has succeeded or not, is used as bool value. Parameter-binding pseudo code is shown in Figure 1. In the figure, the parameter numbered with *no* of the rule named *rulename* is compared with a value *val*. Boolexpreation *name0* can be used as a trigger (activation) condition, cancel condition, suspend condition or a resume condition for a behavior, a guard constraint (whether the state is activated or not) for a state, temporal relation, or a sending condition for an event, etc.

```
rulename(param0,param1...paramN):-rule0(paramset),
rule1(paramset),....
    Boolexpname=<"name0">
    Type="<comparison/logical>"
operator=<"EQ,GT,LT,GE,LE/and,or">
    <leftvalue                    type="predicate"
predicatename="rulename" OutputFieldNo="no">
    <rightvalue       type=<"constant/function/etc..">
value=val
```

Figure 1: Rule Representation

An agent can behave in different ways depending on the situation it is in based on its dual world representation (inner model retained in the knowledge base). In this sense, behaviors can be categorized according to well-defined world views and each category that describes how the model behaves under certain conditions. The condition is defined as a decision model and is used to activate or deactivate the related category or categories [15]. It is a very usefulproperty to be able to change a model world view in both design time and run time. It can be seen as a dynamic description. The parameters and truth value of the rule are also used to shift from one aspect to another. It is shown in Figure 2 as pseudo code. The main point is to make a decision based on an ontological description of entities and manage behaviors based on the decision.

## 3 Ontology-based Modeling

Ontology is a term that originated in philosophy and refers to the systematic explanation and study of the nature of existence, or being [16]. Ontologies are composed of concepts or entities, relations between these concepts (or entities) and axioms to limit the interpretation of concepts for a real world phenomenon. Whereas in computer science, ontologies are recognized as a useful means for achieving semantic interoperability between different systems and are key enablers for sharing precise and machine-understandable semantics among different applications and parties [10]. In the simulation world, it is aimed to use ontology to give meaning to entities at different abstraction levels by binding rules (axioms) and defining behaviors. Similar to the ontology definition used in information systems, in the simulation world a common language is also developed to be used for an entity at each level. In modelling and simulation, the use, benefits and the development requirements of Web-accessible ontologies for discrete-event simulation are investigated [17].

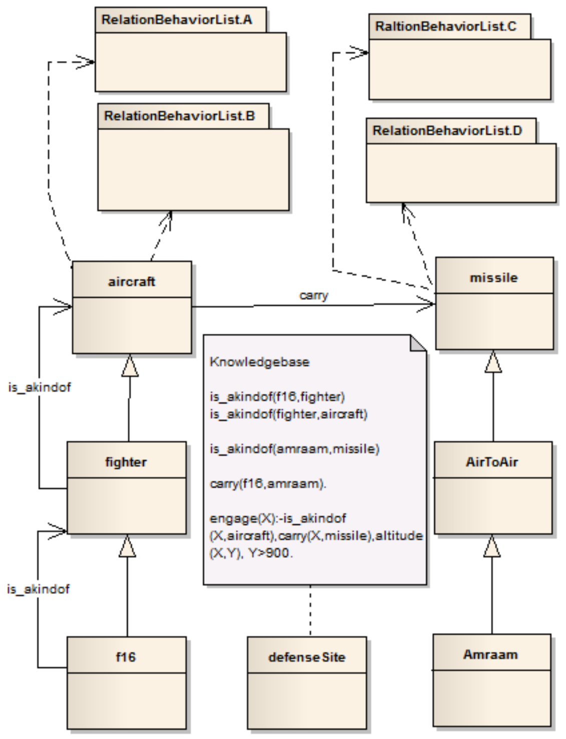
```
<BehaviorList ListA>
    <behavior A>
    <behavior B>
    <behavior C>
    ..
<driveCond>
    <activation cond="<gettingLower>"
    <cancel Cond="<>">
</driveCond
</ BehaviorList >
```

Figure 2: Shifting Aspects by Reasoning

Ontology studies are related to questions such as "*what is it?*" or "*what exists*" and "*what relations has it?*". Looking for an answer for the first question is strictly related to generalizing a particular situation and/or an entity. This allows us to create more general rules and management capabilities for behaviors using higher-order reasoning. The second question is related to the relation concept. An entity (both an agent and a simulation model) may have relations with other objects. In AdSiF representation, the relations are categorized under three main headings: 1) Predicate (Logical), 2) Functional (Behavioral) and 3) Structural. A predicate relation consists of facts (e.g. position information of a missile time-by-time), higher-order descriptions (e.g. missile types and their behavioral patterns, such as being in boost phase, etc.) and rules (decisions about what an object is or how to behave).

Answers to "*what is it?*" are given at different abstraction levels each time it is asked. The answer given on each occasion makes the entity definition one more level abstracted. For example, the answer "*it is a fighter*" given for a question asked about an F16. The F16 is a multi-role fighter aircraft designed in the 70s and still being produced and actively used by the air-forces of various nations. After having the answer, a new question arises, such as about "*what a fighter is*". The answer carries the definition one more level up, such as *it is a plane*, and so on. This continues until arriving at the most primitive, minutest definition, such as *it is an object*. All these abstracted declarations are linked to each other with a predicate such as *is_a_kindof(f16 ,fighter), is_a_kindof(fighter, plane)* and continues until arriving at an answer referring to the most basic well known entity or concept. Quality of the answer is measured by whether a behavior set and a set of properties are defined at each level, determined by the answer given or not.

As mentioned earlier, the relation concept is considered to be functional, predicate or structural. Functional relations that are established between two parties force a definite behavior set for both sides. In this sense, the relations define behavioral templates at different abstract levels. The relations create a behavior set which is activated for the entities on both sides of the relation. The behavior set being activated differs depending on the model type, entity abstraction level and environment or entity state vector. The entity drives a function in regard to any other entity; for example, "*f16 carries missiles*" shows a functional relation and the relation triggers functions or behaviors on both of the side objects, namely *f16* and *missiles*, at the phases in which the relation is established and detached. In Figure 3, the behavior sets are shown to be executed in the "*carry*" relation establishment phase and detachment phase for both F16 and missiles. F16 executes "*RelationBehaviorList.A*" and "*RelationBehaviorList.B*" any time relation is established and detached, respectively. Similarly, missiles does the same thing in the same phases for "*RelationBehaviorList.C*" and "*RelationBehaviorList.D*". The representation presented in Figure 3 is executable and is interpreted by AdSiF core engine.



```
Knowledgebase

is_akindof(f16,fighter)
is_akindof(fighter,aircraft)

is_akindof(amraam,missile)

carry(f16,amraam).

engage(X):-is_akindof
(X,aircraft),carry(X,missile),altitude
(X,Y), Y>900.
```

Figure 3: Functional Relation Declaration

A predicate relation defines declarations between two objects or among more than two objects constituted on stative descriptions of the objects. The declarations are taken into consideration as rules that are used to manage a behavior or a set of behaviors. As an example, let us take the premises "*F16 is equipped with missiles*" and "*F16 is over 1000 ft at time t0* " into consideration. They can be symbolized as *equipedWith(f16,missile)* and *altitudeOver(1000, t0).* In the first, relation is representation and we have two objects, F16 and a missile or a set of missiles, and the second is related to the object itself. Whereas the functional relation is categorized as a relation that forces objects on both sides of the relation, a specific behavior set, and the predicate relation is also related with third object decision. The object that knows or infers the relation between two objects uses the information for decision-making and the result of the decision triggers a behavior or a set of behavior. An entity possibly uses relations that it has to handle its own behavior. In other words, an entity can manage its own behaviors using its own predicate relations. For example, F16 (or a missile) that has predicate *equippedWith(f16, missile)* can trigger a behavior set by using the predicate. Similarly, any other model, such as an air defense site, can detect the relation between F16 and missiles and uses this knowledge as a rule to trigger engagement behavior. The inference made by the air defense site site is translated into natural language, such as "*I (the* air defense site*) have detected an f16 with missiles*". The inference is evaluated using a rule such as "*if an aircraft is detected with missiles and its altitude is over 900 ft, then start an engagement with it*" (Figure 3). It is clear that this is triggering a behavior. In the rule given here, it does not say directly anything about F16 specifically, but it was previously known, after a set of "*what it is*" questions, that an F16 is an aircraft. In this sense, predicate relation is strongly related to propositional logic. The predicates detected or inferred about entities in the environment are kept as time-labeled facts in a knowledge base (such as position information set time-by-time ). The knowledge base constituted by the facts is defined as a dual world representation of the environment and also consists of information other than relation predicates.

It is important that ontologies are of a good quality, in order that they serve their intended purposes and be shared as well as reused by different applications [12]. A good quality depends on how clear the answers are to the "*what it is*" question, constituted of depth of model abstraction sequence (such as F16 is a kind of plane, plane is a kind of aircraft, etc.), relations defined between entities and also the rules and behaviors attached to both the relations and abstract model definitions. Relations follow an inheritance path. A functional relation constituted for any specific entity is valid for the entity derived from it, in other words, its child. This is valid for predicate relations, but exceptions can be made. In this respect, we can say the *carry* relation (F16 carries missiles) is valid for a fighter derived from the F16 model. The generalized form of that relation is "*An aircraft carries missiles*". Similarly, the relation is also valid for any type of missiles that are derived from the missile.

The relation mentioned here is constituted in run time. In design time, composition and aggregation relations are defined. In simulation execution, the entity manages simulation components that it aggregates and/or composes. Management consists of event handling and time management. Composition and aggregation are defined very similar to that between classes [18]. Composition and aggregation are a way to combine simple objects and data types, in this case, entities, into more complex ones. If the more complex one is destroyed, the simple object is also destroyed. Because, it is a non-detachable part of the owner (more complex) entity. But in aggregation cases, they can still survive without the owner entity. One can imagine the simple entity as an attribute of the complex one. Another difference between composition and aggregation is seen in task sharing in the model design of AdSiF. The entity undertakes the event handling and time management tasks of its components (both composed and aggregated). In Figure 4, dashed and solid arcs show aggregation and composition relations, respectively. EntityS, EnitityA and EntityB coordinate with each other and handle their own event handling and time management. But Time requirement of EntitiyB depends on EntityZ requirement and EntityA time requirement depends on the entity to which it is connected. EntityZ time requirement also depends on the entities to which it is connected. The only interface of aggregated and composed entities is their owner entity, and, as a result, they distribute and collect their event messages across owner entities. The behaviors are still separated and the owner entity never intervenes in the behavior of its components. As seen in Figure 5, F16 undertakes behavior and time management of external fuel tanks and a control panel. This means an entity is capable of managing the simulation loop of sub-entities. In Figure 5, F16 has two types of behavior list. The behavior list named "*f16 behavior list*" consists of the F16's or, more generally a fighter's, capabilities. The behavior list named "*Composed & Aggregated model management behavior list*" is inherited from the base model and the behaviors allow it to create time and event managements of sub-components.

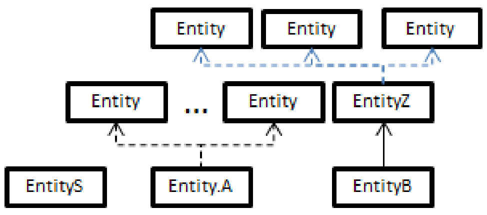


Figure 4: Sub-Simulation Loops

From an ontology point of view, an entity may have several atomic sub-components, each of which is an individual entity. They maintain their own behaviors and data structures, such as attributes, custom data, etc. Time requirement and delivering events of sub-components are managed by the owner entity (in the example, it is F16) and management behaviors are inherited from the base model that is an extended AdSiF entity class. This gives a new dimension to AdSiF's ontology. An entity (or an existence) may be formed by a collection of single entities. Nevertheless, the owner entity has interfaces with the environment.

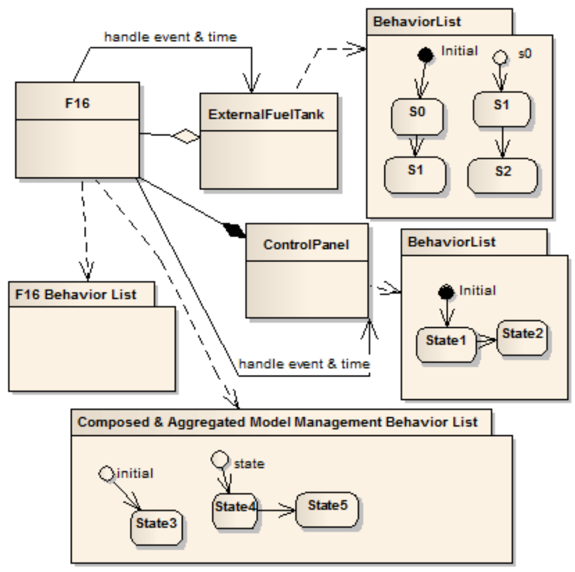


Figure 5: Composition and Aggregation

## 4 Ontology-based Modeling Language Support

Fundamental ontological notions are summarized by [19] as identity, a unity that is related to the problem of distinguishing the parts of an instance from the rest of the world by means of a unifying relation that binds them together (not involving anything else); rigidity, which is a property it necessarily holds for all its instances; and dependence, which is the property of an entity that is dependent of the existance of other entities. The ontological commitment of AdSiF embraces the identity notion by giving a unique id to each object; the unity notion by relations, the rigidity notion by time-framed and consistent facts about objects, and the dependence notion by composition and aggregation depending on design choices; it also extends notions by logical commitments, behavioral semantics and conditional aspects notions.

In the literature, it is possible to find several ontology-based modeling tools, such as Anemone [20], Protégé [21], OilEd [22], Apollo [23], OntoLingua [24], OntoEdit [25], WebODE [26], Kaon [27], DOE (Differential Ontology Editor) [28], WebOnto [29] and K-Infinity [30]. Anemone provides a methodology which differs from previous methodologies in the way that it defines concrete development steps, to facilitate use by both novice and expert ontology developers. This methodology is also supported by ontology design patterns and a prototypical ontology development tool [20]. System Entity Structure and Model Base (SES/MB) is developed for modeling and simulation domain [31].

The distinguished and prominent supports provided by AdSiF are brought by the logic programming paradigm and relation concept. Beyond allowing modelers to define environments, objects and relations between objects at meta-level, it also affords the possibilities of developing a reasoning mechanism simultaneously operating on past and present time-stamped knowledge (such as *position(f16, x0,y0,z0, time0), position(f16, x1, y1 ,z1, time1), position(f16, x2 ,y2 ,z2, time2)*, ..etc.) and driving behaviors depending on inferences (such as *gettingLower(X):- position(X,_,_,Z,T), position(X,_,_,Z2,T2), T2>T, Z2>Z an object X getting lower*) which constitute future information sets. Driving a behavior is defined as activating, cancelling, suspending and resuming it. This is seen as distilled knowledge inferred from a set of meta-level knowledge and declarations that are placed at the kernel of ontology-based modeling.

Both relation predicates and facts about the world maintained in the knowledge base are used for decision. Decision-making results in a truth value and a set of output parameters. Truth value is directly bound as a drive condition (in Figure 6, drive conditions are shown in pseudo code form) to behaviors. The output parameters are used as input parameter for boolean expressions, such as logical expressions or comparisons. The boolean expressions are also used to manage behaviors, such as to activate, cancel, suspend and reactivate and as guard constraints in any place required.

First order predicate logic (FOL) and propositional logic are the fundamental reasoning techniques used in predicate relation and fact-based reasoning. Both logics are perfectly matched with AdSiF's ontology. FOL consists of variables for individual objects, quantifiers, symbols for functions and symbols for relations [32].

```
<behavior A>
      <state Set>
<driveCond>
    <activation                    type=”predicate”
cond=”<gettingLower>”
    <cancel type=”boolExp” Cond=”name0”>
    <suspend Cond=”<>”>
    <reactivate Cond=”<>”>
</driveCond
</behavior>
```

Figure 6: Drive Conditions

## 5 Ontology Example

A simple example from a defense modeling and simulation is chosen to show how the concepts are implemented. In Figure 7, the relations between objects are depicted. The relation between different objects with the same name activates different behavior and action sets. Each relation states a meaning depending on the behavior space of models that are the relation constituted between them. This is also valid for the abstraction level of models. The behavioral description of the relations is shown in Figure 8 for the relation *use* between “*F16*” and “*Missile*”. The relation rule is applicable to all types of F16 and missiles and any type of models derived from these models. In the plane domain of Figure 7, it can be seen what behaviors are executed in both activations, which constitute the relation and passivation phase, which means breaking it. All ontology commitment is not given here; a good example of missile phase and dynamic modeling can be seen in [33].

In Figure 9, an ontological description is given from the more abstract class level up to instance level. The second part of the figure shows a set of predicates inferring what the target is and what kind of missile is to be fired using the facts given in the third part and populated by sensory data. The sensors providing detection information send their detections to commanders, which drive F16, using the relation “informs”. The decision triggers the behavior seen in Figure 10. The behavior selects a missile as a result of an inference using target information received from event-named detection and a set of predicates, facts and ontological descriptions given in the knowledge base shown in Figure 9.

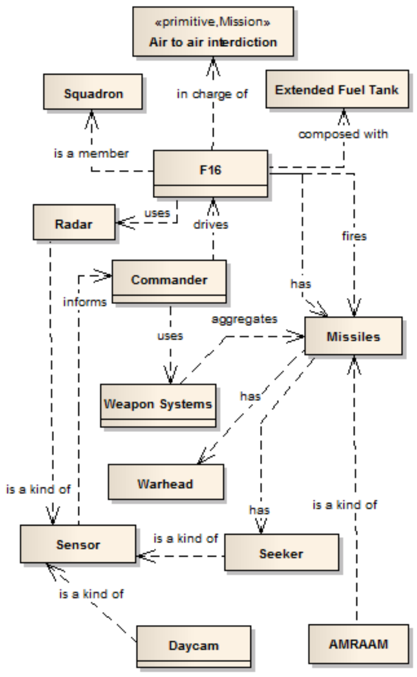


Figure 7: Relations Between Objects

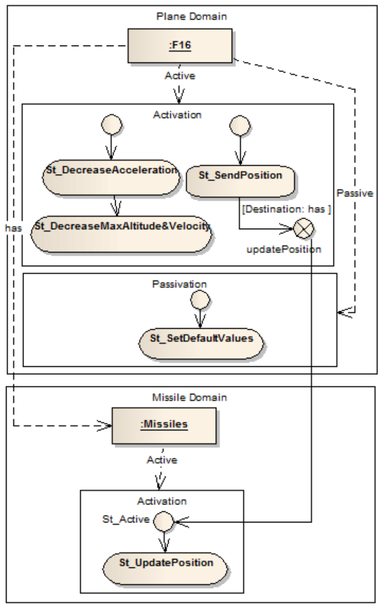
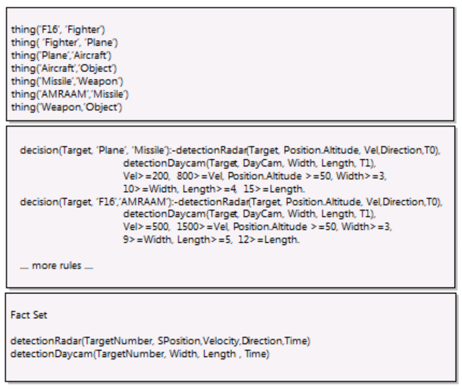


Figure 8: Behavioral Descriptions



```
thing('F16', 'Fighter')
thing( 'Fighter', 'Plane')
thing('Plane','Aircraft')
thing('Aircraft','Object')
thing('Missile','Weapon')
thing('AMRAAM','Missile')
thing('Weapon,'Object')

decision(Target, 'Plane', 'Missile'):-detectionRadar(Target, Position.Altitude, Vel,Direction,T0),
                          detectionDaycam(Target, DayCam, Width, Length, T1),
                          Vel>=200,  800>=Vel, Position.Altitude >=50, Width>=3,
                          10>=Width, Length>=4,  15>=Length.
decision(Target, 'F16','AMRAAM'):-detectionRadar(Target, Position.Altitude, Vel,Direction,T0),
                          detectionDaycam(Target, DayCam, Width, Length, T1),
                          Vel>=500,  1500>=Vel, Position.Altitude >=50, Width>=3,
                          9>=Width, Length>=5,  12>=Length.

.... more rules ....

Fact Set

detectionRadar(TargetNumber, SPosition,Velocity,Direction,Time)
detectionDaycam(TargetNumber, Width, Length , Time)
```

Figure 9: A Part of the Knowledge Base

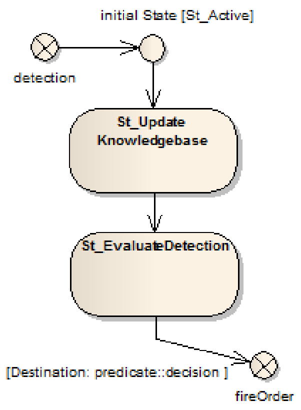


Figure 10: Driving Behavior by Inferences

## 6 Discussions

AdSiF programming approach and state-oriented paradigm, which is the main programming paradigm, expands a perdurantist modeling approach to a reasoning-capable model by a logic programming paradigm and ontology-based modeling world view.

The ontological commitment that AdSiF is based on matches the predicate logic ontological commitment and bring them into a comment framework to handle behavioral management in simulation.

Logic programming and modeling paradigms enrich ontologies by giving inference capability, keeping the information as time-framed and allowing it to expire, defining relations between objects and conditions in them. In addition to inference, logic programming combined with state-oriented programming allows modelers to model domain information at meta-level. Each meta-level system model has a model family rather than a specific implementation model.

This study points out how to use concepts and axioms defined as logical premises and how the relations between higher-order entity descriptions are used to combine in a behavior management structure. Logical premises define model structures at different abstraction levels of domain information. Relations provide indirect interaction based on entity description, not direct interaction with objects, and also straightforward behavioral descriptions.

The proposed solution also provides a logic based solution for time-delayed systems [34], allowing simulation and agent models to use time-stamped facts stored in the knowledge base of the model. A time-delayed system needs an earlier value of the decision variables and the time-labeled facts in the knowledge base of entities provide a good solution to the problem.

## References

[1] M. F. Hocaoglu, "AdSiF : Agent Driven Simulation Framework," Hunstv. Simul. Conf. -HSC2005, 2005.
[2] M. F. Hocaoğlu, "ETSİS : Etmen Tabanlı Simülasyon Sistemi AdSiF : Agent driven Simulation Framework Abstract," 2011.
[3] M. W. R. H. Bordini and J. F. Hübner, Programming Multi-agent Systems in AgentSpeak Using Jason. Wiley Series in Agent Technology, 2007.
[4] K. R., "Aspect-Oriented Programming," Computerworld, vol. 37, no. 40, 2003.
[5] F. Pfenning, "Logic Programming," 2007.
[6] M. F. Hocaoğlu, "AdSiF: Developer Guide," Istanbul, 2013.
[7] D. Harel, "State Charts: A visual formalism for Complex Systems," Sci. Comput. Program., vol. 8, pp. 231–274, 1987.
[8] Y. Shoham, "Agent-Oriented Programming." 1990.
[9] T. A. Hales and S.D., Johnson, "Endurantism, Perdurantism and Special Relativity," Philos. Q., vol. 53, no. 213, pp. 524–539, 2003.
[10] M. M. Al-Debei, M. M. Al Asswad, S. de Cesar and M. Lycett, "Conceptual Modelling and the Quality of Ontologies : Endurantism vs, Perdurantism," Int. J. Database Manag. Syst., vol. 4, no. 3, pp. 1–19, Jun. 2012.
[11] T. Sider, Four Dimensionalism: An Ontology of Persistence and Time. Oxford University Press, 2001.
[12] N. Guarino, D. Oberle and S. Staab, "What is an Ontology ?," pp. 1–17, 2009.
[13] F. Béhé, S. Galland, N. Gaud, C. Nicolle and A.

Koukam, “An ontology-based metamodel for multiagent-based simulations,” Simul. Model. Pract. Theory, vol. 40, pp. 64–85, Jan. 2014.
[14] C. S. M. W.F. Clocksin, Programming in Prolog, Second. Cambridge, England: Springer-Verlag Berlin Heidelberg New York Tokyo, 1984.
[15] M. F. Hocaoğlu, “Aspect Oriented Programming Perspective in Agent Programming,” in USMOS 2013- National Defense Application and Modeling & Simulation Conference -(in Turkish), 2013.
[16] A. G. Bruzzone, R. Mosca, R. Revetria, E. Bocca and E. Briano, “Agent Directed HLA Simulation for Complex Supply Chain Modeling,” Simulation, vol. 81, no. 9, pp. 647–655, 2005.
[17] J.A. Miller, ., G.T. Baramidze, PA.P. Sheth and A.P. Fishwick, “Investigating Ontologies for Simulation Modeling,” in ANSS ’04 Proceedings of the 37th annual symposium on Simulation, 2004, p. 55.
[18] B. Stroustrup, The C++ Programming Language, 4th Edition. 2013.
[19] N. Guarino and C. Welty, “Towards a methodology for ontology based model engineering.”
[20] T. Özacar, Ö. Öztürk and M. O. Ünalır, “ANEMONE: An environment for modular ontology development,” Data Knowl. Eng., vol. 70, no. 6, pp. 504–526, Jun. 2011.
[21] M. Horridge, H. Knublauch, A. Rector, R. Stevens, C. Wroe, S. Jupp, G. Moulton, N. Drummond and S. Brandt, “A Practical Guide To Building OWL Ontologies Using Protege 4 and CO-ODE Tools,” 2011.
[22] S. Bechhofer, I. Horrocks, C. Goble and R. Stevens, “OilEd: A Reasonable Ontology Editor for the Semantic Web,” Lect. Notes Comput. Sci., vol. 2174, pp. 396–408, 2001.
[23] M. Kamil Matoušek and Luboš Král, “Apollo CH Manual,” 2004.
[24] J. Fikes, R., Farquhar and A., Rice, “Tools for assembling modular ontologies in Ontolingua,” in AAAI/IAAI, 1997.
[25] S. Sure, Y. Angele and J. Staab, “OntoEdit: Multifaceted inferencing for ontology engineering.,” Data Semant., vol. 1, pp. 128–152, 2003.
[26] G.- Arpirez, J. Corcho, O. Fernandez-Lopez and M. and A. Perez, “WebODE: A scalable workbench for ontological engineering,” in First International Conference on Knowledge Capture (KCAP’01), 2001, pp. 6–13.
[27] B. Volz, R., Oberle, D., Staab and S.Motik, “KAON SERVER: A semantic Web management system,” in Alternate Track Proceedings of the Twelfth International World Wide Web Conference, WWW2003, 2001, pp. 20–24.
[28] V. Isaac, A. Troncy and R. Malais, “Using XSLT for interoperability: DOE and the travelling domain experiment,” in Proceedings of the Second International Work- shop on Evaluation of Ontology-based Tools (EON 2003), Sanibel Island, FL., 2003.
[29] O. Domingue, J. Motta and E. Corcho Garcia, “Knowledge modelling in WebOnto and OCML: A user guide,” 1999.
[30] S. Youn and D. Mcleod, “Ontology Development Tools for Ontology- Based Knowledge Management Ontology Development Tools for.” CREATE Research Archive, 2006.
[31] P. E. H. B. P. Zeigler, Modeling & Simulation-Based Data Engineering: Introducing Pragmatics into Ontologies for Net-Centric Information Exchange. Elsevier, 2007.
[32] P. N. S. Russel, Artificial Intelligence: A Modern Approach. Prentice Hall, 1995.
[33] U. Durak, H. Oğuztüzün and S. K. İder, “Ontology-Based Domain Engineering for Trajectory Simulation Reuse,” Int. J. Softw. Eng. Knowl. Eng., vol. 19, no. 08, pp. 1109–1129, Dec. 2009.
[34] B. P. Zeigler, Theory of Modelling and Simulation. Robert E. Krieger Publishing Company, Malabar, Florida, 1976.