OSM-Logic: A Fact-Oriented, Time-Dependent Formalization of Object-oriented Systems Modeling

Stephen W. Clyde[1], David W. Embley[2], Stephen W. Liddle[3], and Scott N. Woodfield[2]

[1] Computer Science Department
Utah State University, Logan, Utah 84322, USA
[2] Department of Computer Science,
[3] Information Systems Department,
Brigham Young University, Provo, Utah 84602, USA

Abstract. The lack of fact-oriented, time-dependent formalizations of conceptual models leads to difficulties in inspecting and reasoning about system properties and predicting future behavior from past behavior. We can better serve these needs by formalized conceptualizations that more closely match the demands of such applications. We therefore set forth in this chapter a fact-oriented, time-dependent formalism, called *OSM-Logic*, for object existence, object interrelationships, object behavior, and object interaction. OSM-Logic is formally grounded in predicate calculus, and is thus mathematically sound and well defined.

1 Introduction

Recent initiatives by government agencies (e.g., IARPA [IAR]) and by academic think-tank groups (e.g., ACM-L [ACM]) require conceptualizations with the ability to track behavior, model what has already happened and is currently happening, and analyze past and present behavior. The objectives of tracking, modeling, and analyzing include being able to predict future behavior, play out "what-if" scenarios, and warn of possible impending disasters.

Conceptual models can provide the formal foundation for storing the necessary information to support these initiatives. The conceptual models that meet these requirements, however, must be powerful: they must be able to conceptualize objects, relationships among objects, object behavior, and object interaction, and the conceptualizations must be fact-oriented and time-dependent. Without, being able to formalize and store time-dependent facts about objects—their interrelationships, their individual behavior, and their interaction with other objects—analysis of, and predictions based on, current, past, and proposed happenings cannot be carried out. We thus seek for, and propose here, a formalization of fact-oriented, time-dependent conceptualizations of objects—their existence, their interrelationships, their behavior, and their interactions with other objects.

To define precisely what we mean by *fact-oriented, time-dependent conceptualizations*, we note that conceptual modelers have observed that fact-oriented modeling focuses of facts of interests that can be expressed as first-order-logic predicates [ORM]. Further, for every fact, in addition to knowing if it is true, we should know when it is true. Since facts hold at points in time or over a period of time, we obtain time-dependent facts by adding to every logic predicate, an argument for a time variable for facts that hold for a point in time and two time-variable arguments for facts that hold over a period of time. Thus, as the basis for our formalism, we seek for first-order-logic predicates augmented with either one time variable for point-in-time facts or two time variables for time-period facts. Further, all conceptualizations—including objects, relationships among objects, object behavior, and object interactions—should be formalized with fact-oriented, time-dependent first-order-logic predicates.

A number of conceptual-model formalizations have been developed as evidenced by hundreds of articles and books ranging from the earliest abstract formulations from more than 50 years ago [YK58] through books that encapsulate much of the 50-year history of conceptual modeling (e.g., [Oli07]) to a recent handbook of conceptual modeling taking formal foundations for granted and as being expected [ET11]. Although plentiful and useful for their intended applications, none of the formalizations fully have the characteristics required for fact-oriented, time-dependent conceptualizations of object and relationship existence, object behavior, and object interaction for the applications we target in this chapter. Only a few conceptual models even span the space from object existence through object behavior, and of those that do, even fewer have formal definitions. Those that span the space and have worked-out or mostly worked-out formalisms include the Unified Modeling Language (UML) [UML], Object Process Modeling (OPM) [Dor09], Object Role Modeling (ORM) [HM08], the software production environment for MDA based on the OO-Method and OASIS [PM07], the Higher-order Entity Relationship Model (HERM) [Tha00], and Object-oriented Systems Modeling (OSM) [EKW92]. Even though these conceptual models have formalizations that span the space from object existence to object behavior, most of the behavior formalizations are neither fact-oriented nor time-dependent, but are, instead, based on ideas from state charts, finite state machines, and Petri nets. ORM and OSM formalisms are based on predicate calculus, which does not inherently deal with time dependencies. However, with some work, predicate calculus can be extended to a two-sorted first-order logic that captures notions of time, including events, time intervals, and time dependences. Because OSM is modeled completely in terms of predicate calculus, with some work it can be made to be time dependent. Indeed, this is the contribution of this chapter, where we formally define fact-oriented, time-dependent semantics for OSM by showing how to convert any OSM model instance to formulas in *OSM-Logic* [Cly93] and how to interpret these formulas.

Briefly and succinctly, OSM-Logic is a two-sorted, first-order logic language with temporal semantics specifically developed for defining the meaning of OSM model instances. Since OSM captures static and dynamic properties of real-world

systems, OSM-Logic must be able to express relationships among objects, object behavior, and interactions with respect to time. In real-world systems, time involves continuous intervals and individual points. Most changes in a system of objects occur during time intervals and are not instantaneous. Even simple changes like an object entering a new state or becoming a member of an object set occur over a time interval. In fact, for any interpretation of a system, a change that appears instantaneous may actually be occurring during a time interval that is simply smaller than smallest unit of time in that interpretation. By using a finer unit of time, what once appeared instantaneous can appear as a time interval. On the other hand, an event is a single time point that represents a true instantaneous occurrence. An event typically corresponds to the beginning or end of a particular time interval during which something interesting occurred.

In the remainder of the chapter, we give the details of OSM-Logic. Section 2 describes OSM-Logic itself, Section 3 describes how we attach semantics to a set of formulas, Section 4 summarizes the OSM-to-OSM-Logic conversion algorithm, and we conclude in Section 5.

2 OSM-Logic Language Definition

OSM-Logic is a multi-sorted language with two basic sorts, $S = \{s_t, s_o\}$; s_t is for time points and s_o is for objects. OSM-Logic consists of an infinite set of symbols arranged as prescribed in [End72]. *Logic symbols* include *logical connectors*, $\vee$, $\Rightarrow$, $\neg$; *time-variable symbols*, V_t (e.g., t_1, t_2, ...); *object-variable symbols*, V_o; *equality symbols*, $=_t$ for s_t and $=_o$ for s_o; and *auxiliary symbols*, parentheses and comma. *Parameters* include *quantifiers*, $\{\exists_t, \forall_t\}$ for s_t and $\{\exists_o, \forall_o\}$ for s_o;[1] *time-constant symbols*, C_t, *object-constant symbols*, C_o; *predicate symbols* of sort $\langle s_1, ..., s_n \rangle$, where $s_j \in S$ for $1 \leq j \leq n$[2]; and n-place *function symbols* of sort $\langle s_1, ..., s_n, s_o \rangle$, where $s_j \in S$ for $1 \leq j \leq n$. Note that we have restricted the results of functions to be objects.

Terms with sort s_t include time-variable and time-constant symbols ($V_t \cup C_t$). Terms with sort s_o include object-variable and object-constant symbols ($V_o \cup C_o$) along with function terms, which we construct from function symbols by filling each place of the symbol with a term of the designated basic sort. For example, if + is a 2-place function symbol of sort $\langle s_o, s_o, s_o \rangle$ and x and y are object-variable symbols in V_o, we construct a function term + written either $+(x, y)$ (prefix notation) or $x + y$ (infix notation).

An *atomic formula* is a sequence $P(z_1, ..., z_n)$, where P is an n-place predicate symbol or an equality symbol (in which case $n = 2$) of sort $\langle s_1, ..., s_n \rangle$ and z_1, ..., z_n are terms of sort s_1, ..., s_n, respectively. For example, let $Pizza(\_, \_, \_)$ be a 3-place predicate symbol of sort $\langle s_o, s_t, s_t \rangle$ that represents the membership of the *Pizza* object set in Figure 1. Also, let x be an object-variable symbol and t_1 and t_2 be time-variable symbols. We can construct the atomic formula

[1] When the sort is clear from the context, we may drop the t or o subscript from an equality symbol or quantifier.

[2] However, no predicate symbol has more than two places of sort s_t.

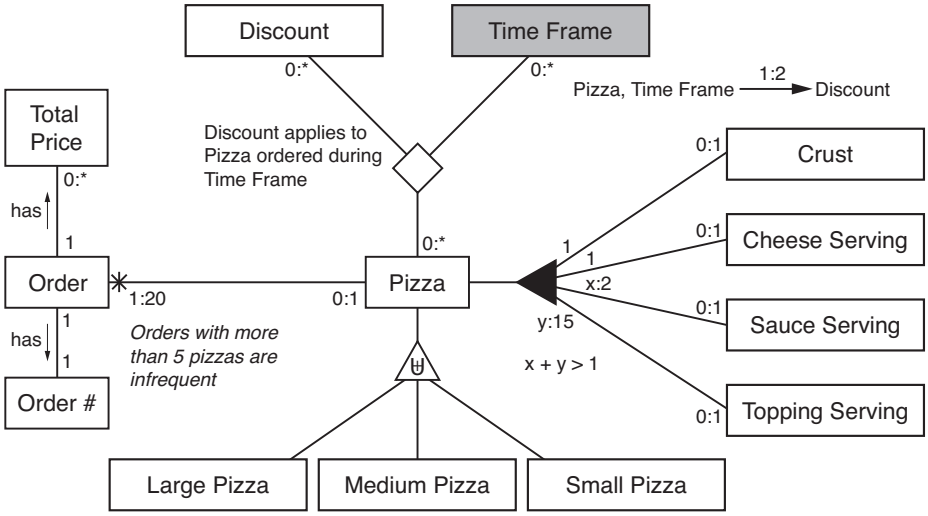

Fig. 1. Sample ORM for Pizza Ordering System.

$$\forall x \forall y \forall t_1 \forall t_2 (\mathit{Crust}(x) \textit{ is subpart of } \mathit{Pizza}(y)(t_1, t_2) \Rightarrow \mathit{Crust}(x, t_1, t_2))$$
$$\forall x \forall y \forall t_1 \forall t_2 (\mathit{Crust}(x) \textit{ is subpart of } \mathit{Pizza}(y)(t_1, t_2) \Rightarrow \mathit{Pizza}(y, t_1, t_2))$$

Fig. 2. Sample OSM-Logic Formulas.

$\mathit{Pizza}(x, t_1, t_2)$. We often write atomic formulas using an infix notation, as with the general constraint in Figure 1 written $x + y > 1$ rather than $> (x + y, 1)$.

A *well-formed formula* (wff) is constructed from atomic formulas, logical connectors, and quantifiers in the traditional way. Figure 2 shows two wff's constructed from atomic formulas, the $\Rightarrow$ logical connector, and universal quantifiers. In the first formula, $\mathit{Crust}(\_)$ *is subpart of* $\mathit{Pizza}(\_)(\_, \_)$ is a predicate symbol of sort $\langle s_o, s_o, s_t, s_t \rangle$ that represents the membership of the *Crust is subpart of Pizza* relationship set in Figure 1. To aid readability, we write the object terms for this predicate symbol in-line, using an infix notation. The second predicate symbol in the first formula, $\mathit{Crust}(\_, \_, \_)$, represents the memberships of the *Crust* object set in Figure 1. Informally, the first formula guarantees that an object is a member of the *Crust* object set whenever it relates to a pizza in the *Crust is subpart of Pizza* relationship set. Similarly, the second formula guarantees that an object is a member of the *Pizza* object set whenever it relates to a crust in the *Crust is subpart of Pizza* relationship set.

3 Interpretations

We establish the meaning of a formula or a set of formulas through an *interpretation*.

An interpretation maps the language's parameter symbols to a mathematical structure, consisting of a time structure, a universe of objects, a set of functions,

and a set of relations. As a result, an interpretation gives meaning to the symbols of the language. Without an interpretation, a formula is just a sequence of symbols and nothing more. For example, the formulas shown in Figure 2 are by themselves just sequences of symbols. An interpretation gives them meaning by mapping *Crust*(_) *is subpart of Pizza*(_)(_, _), and *Pizza*(_, _) to relations, x and y to objects, and t_1 and t_2 to time points.

Formally, we define an *interpretation* to be an 8-tuple $\langle T, U, F, R, g_T, g_U, g_F, g_R\rangle$ where T, U, F, and R form the mathematical structure and g_T, g_U, g_F, and g_R map parameter symbols to elements of T, U, F, and R, respectively:

- T is a time structure such that it includes (1) a (possibly infinite) set of time points TP, (2) an ordering $<$ on TP, and (3) a time-interval magnitude function $f_{\|} : TP \times TP \rightarrow U$, such that $f_{\|}(\tau_1, \tau_1) = 0$, $f_{\|}(\tau_1, \tau_2) = f_{\|}(\tau_2, \tau_1)$, and $f_{\|}(\tau_1, \tau_2) < f_{\|}(\tau_1, \tau_3)$ for $\tau_1 < \tau_2 < \tau_3$.
- U is a non-empty universe of objects.
- F is a set of functions such that it includes the time-interval magnitude function. Each function in F of arity n has a sort $\langle s_1, ..., s_n, s_o\rangle$ where $s_j \in S$ for $1 \leq j \leq n$. The time-interval magnitude function has arity 2 and sort $\langle s_t, s_t, s_o\rangle$.
- R is a set of relations such that it includes the $<$ ordering relation. Each relation of arity n has a sort $\langle s_1, ..., s_n\rangle$, where $s_j \in S$ for $1 \leq j \leq n$. The $<$ relation has arity 2 and sort $\langle s_t, s_t\rangle$.
- g_T is a mapping of time constant symbols to TP.
- g_U is a mapping of object constant symbols to U.
- g_F is a mapping of function symbols to F such that it maps an n-place function symbol to an n-ary function of the same sort.
- g_R is a mapping of predicate symbols to R such that it maps an n-place predicate symbol to an n-ary relation of the same sort.

To give OSM-Logic temporal semantics, we add three restrictions to the definition of an interpretation. First, relations in R include exactly zero, one, or two arguments of the time sort. We call these respectively *time-invariant*, *event*, and *temporal relations*. The two time points in a tuple from a temporal relation identify a *time interval* over which the other objects in the tuple are related. Let τ_1 and τ_2 be time points. If $\tau_1 < \tau_2$, then the time interval $[\tau_1, \tau_2)$ is the set of time points $t \in TP$ such that $\tau_1 \leq t < \tau_2$. If $\tau_2 < \tau_1$ then the time interval $[\tau_1, \tau_2)$ is the set of time points t, such that $\tau_2 \leq t < \tau_1$. If $\tau_1 = \tau_2$, then the time interval $[\tau_1, \tau_2)$ is the empty set. By definition, the time interval $[\tau_1, \tau_2)$ is the same as $[\tau_2, \tau_1)$. The notation "[...)" reminds us that the interval includes the starting point but not the ending point.

Second, we restrict temporal relations so if a temporal relation includes a tuple that relates a set of objects for some time interval, then it also includes tuples that relate the same set of objects for all non-empty sub-intervals of that time interval. Let R be an n-ary temporal relation R. If $(x_1, ..., x_n, t_1, t_2) \in R$ and there exists $t_3 \in TP$ such that $t_1 < t_3 < t_2$, then $(x_1, ..., x_n, t_1, t_3) \in R$ and $(x_1, ..., x_n, t_3, t_2) \in R$. The temporal relations r_a, r_b, and r_c in Figure 3 satisfy this restriction.

$I = \langle T, U, F, R, g_T, g_U, g_F, g_R \rangle$, where

$T = \{$ $TP = \{1, 2, 3, 4, 5\}$, the $<$ ordering relation, and the time-interval magnitude function $f_{\|}$, where $f_{\|}(\tau_1, \tau_2)$ is the absolute value of $\tau_2 - \tau_1$ $\}$

$U = \{p_1, p_2, p_3, c_1, c_2, 0, 1, 2, 3, 4\}$

$F = \{f_{\|}\}$

$R = \{$

r_a

c_1	p_1	3	4
c_2	p_2	3	5
c_2	p_2	3	4
c_2	p_2	4	5

,

r_b

c_1	2	4
c_1	2	3
c_1	3	4
c_2	3	5
c_2	3	4
c_2	4	5
c_3	4	5

,

r_c

p_1	3	4
p_2	3	5
p_2	3	4
p_2	4	5

,

$<$

1	2
1	3
1	4
1	5
2	3
2	4
2	5
3	4
3	5
4	5

$\}$

$g_T = \{\}$

$g_U = \{\}$

$g_F = \{\langle \|, f_{\|} \rangle\}$

$g_R = \{$ $\langle$*Crust*(_) *is subpart of* *Pizza*(_)(_, _), $r_a\rangle$,
$\langle$*Crust*(_, _, _), $r_b\rangle$,
$\langle$*Pizza*, $r_b\rangle$
$\}$

Fig. 3. An Interpretation for the Formulas in Figure 2.

Third, we restrict temporal relations so if a temporal relation includes two tuples with the same set objects and the tuples have adjacent or overlapping time intervals, then the relation must also contain a third tuple with the same objects and a time interval that spans both of the others. Two time intervals, $[t_1, t_2)$ and $[t_3, t_4)$, are *adjacent* if $t_2 = t_3$ or $t_1 = t_4$ and *overlapping* if $t_1 < t_3 < t_2 < t_4$ or $t_3 < t_1 < t_4 < t_2$. We formally define this restriction as follows. Let R be an n-ary temporal relation R. If (x_1, ..., x_n, t_1, t_2), (x_1, ..., x_n, t_3, t_4) $\in R$ and the time intervals $[t_1, t_2)$ and $[t_3, t_4)$ are either adjacent or overlapping, then (x_1, ..., x_n, t_5, t_6) $\in R$ where $t_5 = t_1$ and $t_6 = t_4$ if $t_1 < t_3$, otherwise $t_5 = t_3$ and $t_6 = t_2$. The temporal relations r_a, r_b, and r_c Figure 2 also satisfy this restriction.

Given an interpretation, a formula is either true or false. For example, consider the interpretation I in Figure 3. Note that I maps the *Crust*(_) *is subpart of* *Pizza*(_)(_, _) predicate symbol to r_a, the *Crust*(_, _, _) predicate symbol to r_b, and the *Pizza*(_, _, _) predicate symbol to r_c. Using I, the first formula in Figure 2 is true, because every object $x \in U$, time $t_1 \in TP$, and time $t_2 \in TP$ that is associated in the first, third, and fourth places of r_a, respectively, is also associated in r_b. The second formula in Figure 2 is also true, because every object $x \in U$, time $t_1 \in TP$, and time $t_2 \in TP$ that are associated in the second, third, and fourth places of r_a, respectively, are also associated in r_c.

If an interpretation I makes a formula α true, then I is called a *valid interpretation*[3] for α. If I makes each formula in a set of formulas true, then I is a valid interpretation for the set of formulas. The interpretation in Figure 3 is a valid interpretation for the set of formulas in Figure 2.

The semantics of a set of formulas Γ is defined by all the possible valid interpretations for Γ. Intuitively, we think of a set of formulas in a logic language as a declaration about the characteristic properties of a system. Valid interpretations formalize this notion by defining all the possible situations that reflect these characteristic properties. For example, consider a system described by the formulas in Figure 2. The characteristic properties of this system are that whenever an object is in the first place of r_a, it must also be in the first place of r_b, and whenever an object is the second place of r_a, it must also be in the first place of r_b. In other words, pizzas can relate only to crust objects and crust objects can relate only to pizza objects in *Crust is subpart of Pizza*. The set of all valid interpretations for these formulas represents precisely all situations that reflect these characteristic properties.

4 OSM-to-OSM-Logic Conversion Algorithm

Using OSM-Logic we can now formally define the semantics of OSM by algorithmically converting any OSM model instance to a set of OSM-Logic formulas. Using an interpretation, we can then give meaning to the set of formulas by mapping their symbols to a mathematical structure. The mathematical structures of the valid interpretations for these formulas represent all possible situations that exhibit the characteristic properties described by the model instance.

The OSM-to-OSM-Logic conversion algorithm consists of a set of preliminary transformations and a set of independent conversion procedures. The preliminary transformations simply convert an OSM model instance to a standard form (e.g., standardizing names and giving identifiers to unnamed elements). Each conversion procedure generates OSM-Logic formulas for a particular type of modeling component or modeling construct that links components together in a specific way. In Sections 4.1 and 4.2 we summarize these procedures for OBMs and OIMs respectively using the pizza example (Figure 1). Because the details are extensive, we only summarize here; the full conversion algorithm is provided in [CEW92].

We outline the conversion procedures in Figures 4, 5, and 6. We organize the conversion procedures into sub-procedures for converting static and dynamic properties in each of the sub-model types.

For example, the ORM conversion procedure (Figure 4) includes nine sub-procedures for mapping static properties of ORMs into OSM-Logic, and two for for mapping dynamic properties. The first static-property procedure generates formulas that ensure the referential integrity of relationship sets. If an object o is involved in the i^{th} position of a tuple in some relationship set R, then o must be a

[3] "Model" is the usual logic term, but since "model" is heavily overloaded in the context of computer science, we use the term "valid interpretation" instead.

Static Properties

1. Ensure referential integrity of relationship sets
2. Represent generalization/specialization relationship sets
3. Ensure generalization/specialization constraints
4. Represent aggregation relationship sets
5. Represent participation constraints
6. Represent co-occurrence constraints
7. Map variables used in constraints
8. Represent notes
9. Represent general constraints

Dynamic Properties

1. Represent "becoming" and "ceasing-to-be" for object sets
2. Represent "becoming" and "ceasing-to-be" for relationship sets

Fig. 4. Summary of ORM-to-OSM-Logic Conversion Procedures for Pizza Example.

member of the object set associated with the i^{th} connection of R. The formulas in Figure 2 illustrate this: the first guarantees that if $\langle x, y, t_1, t_2\rangle$ is a tuple in *Crust is subpart of Pizza*, then x must be a member of *Crust*, and the second similarly ensures the referential integrity of *Pizza* for the same relationship set. These formulas are true for all time intervals. Apart from the addition of temporal semantics, the formulas in Figure 2 are straightforward and correspond to what one would expect for structural properties. We omit discussion of procedures 2–9 for ORM static properties because they are similarly straightforward.

However, converting dynamic properties with their full temporal formalisms is a more interesting and less common problem that requires more explanation. Formulas representing dynamic properties have time-dependent membership conditions and are not true for all time intervals. In real-world systems, classification and relationships change over time, so objects and relationships become and cease to be members of object and relationship sets over time as well. Thus, we must write formulas that capture the notions of "becoming" and "ceasing-to-be" formally over time intervals. For each object and relationship set we generate additional predicates (e.g., *Becoming Pizza*(_, _, _) of sort $\langle s_o, s_t, s_t\rangle$ and *Ceasing to be Crust*(_) *is subpart of Pizza*(_)(_, _) of sort $\langle s_o, s_o, s_t, s_t\rangle$) to represent these properties.

The conversion routine generates formulas like the following to capture the semantics of these dynamic properties:

$$\forall x \forall t_1 \forall t_2 (\mathrm{STI}(\mathit{Pizza}(x, t_1, t_2), t_1, t_2) \Rightarrow \exists t_3 (t_1 < t_3 \wedge \mathit{Becoming\ Pizza}(x, t_1, t_3))) \quad (1)$$

where $\mathrm{STI}(F, t_1, t_2)$ is shorthand notation for a starting-time interval formula defined as $(t_1 < t_2 \wedge F \wedge \forall t_3 (t_3 < t_1 \Rightarrow \neg F^{t_1}_{t_3}) \wedge \exists t_3 (t_3 < t_1))$. Formula 1 ensures that the time intervals are aligned such that when an object starts being a member of the *Pizza* object set, it was in the process of becoming a pizza. Other generated formulas ensure that all the required conditions relating the

Static Properties

1. Ensure referential integrity of states
2. Ensure referential integrity of transitions
3. Ensure referential integrity of real-time markers
4. Represent state conjunctions
5. Represent prior-state conjunction reset action
6. Represent transition firing phase
7. Ensure mutual exclusion of transition states
8. Ensure that objects are in at least one phase for each transition
9. Ensure that objects do not commit conflicting transitions
10. Represent real-time markers
11. Represent real-time constraint semantics

Dynamic Properties

1. Represent transition ready phase conditions
2. Represent transition committed phase conditions
3. Represent transition execution phase conditions
4. Represent transition finishing phase conditions
5. Represent transition inactive phase conditions
6. Specify when objects can enter each state
7. Specify when objects can exit each state

Fig. 5. Summary of OBM-to-OSM-Logic Conversion Procedures for Pizza Example.

Static Properties

1. Ensure referential integrity of temporal relations
2. Ensure that a single interaction only occurs once
3. Represent source/destination parameter exchange

Dynamic Properties

1. Guarantee alignment of interaction with object time intervals

Fig. 6. Summary of OIM-to-OSM-Logic Conversion Procedures for Pizza Example.

alignment of various dynamic time intervals hold true. There are eight such conditions [CEW92]. The conversion procedure generates similar formulas for relationship sets as well.[4]

4.1 Converting OBM Components

The procedures that convert the OBM components to OSM-Logic define the static and dynamic behavioral properties for members of object sets. Static behavioral properties are object-set membership conditions involving object behavior represented by states, transitions, and real-time markers. Like the static properties represented in an ORM instance, static behavioral properties for an object set are conditions that must be satisfied for all time intervals. Dynamic

[4] We also define shorthand notations for ending- and maximum-time intervals. $\mathrm{ETI}(F, t_1, t_2)$ represents a formula that is true iff t_1 and t_2 represent an ending-time interval for F, regardless of the interpretation. Similarly, $\mathrm{MTI}(F, t_1, t_2)$ is true iff t_1 and t_2 represent a maximum-time interval.

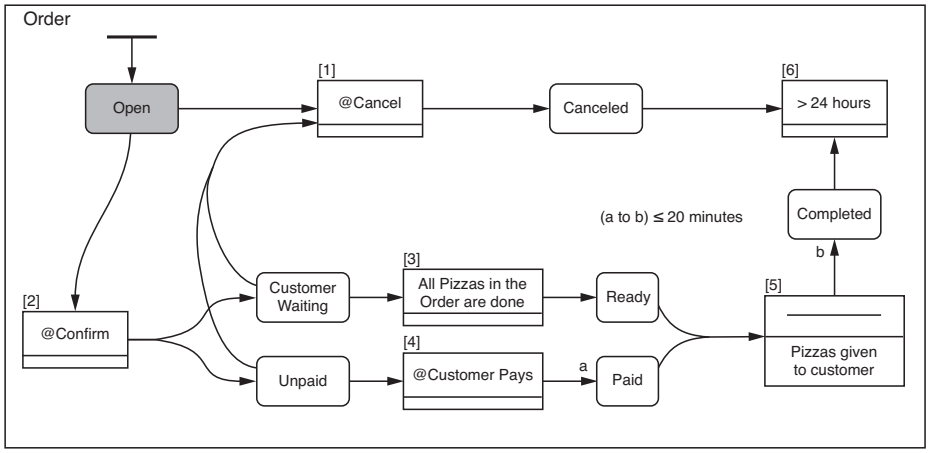

Fig. 7. State Net for the *Order* Object Set.

behavioral properties are object-set membership conditions that relate various time intervals represented by OBM components.

Since an OBM instance is a collection of state nets and each state net describes object behavior for a single object set, we can execute the OBM conversion procedures independently for each state net. Therefore, without the loss of generality, we simplify our definition of the OBM conversion procedures by describing them for a single state net. Using the state net shown in Figure 7 as an example, we summarize the OBM conversion procedures in the next two subsections.

Static Properties. To represent the static behavioral properties described in a state net, we construct predicates for the events and temporal relations represented by various components in the state net, including: states, state conjunctions, transitions, triggers, actions, and real-time markers. We use component names and identifiers, together with an object-set name for the associated object set, to construct these predicates using a set of templates as Figure 8 shows. The template parameters are defined as follows: ⟨ObjectSet⟩ is an object-set name (e.g., *Order*). ⟨SC⟩ is a state name or a conjunction of state names (e.g., *Paid* or *Ready & Paid*). ⟨TID⟩ is a transition identifier (e.g., *[1]*) and ⟨RTM⟩ is a real-time marker (e.g., *a* or *b*). ⟨PSC⟩ and ⟨SSC⟩ are prior-state conjunctions (e.g., *Ready & Paid*) and subsequent-state conjunctions (e.g., *Customer Waiting & Unpaid*) respectively. ⟨phase⟩ is one of *inactive*, *ready*, *committed*, *executing*, and *finishing*. When a transition is committed, executing, or finishing, we say that it is *firing*, and so we also include a sixth *firing* phase that is equivalent to the disjunction of the other three.

The templates in Figure 8 each include one symbol of sort s_o and one or two symbols of sort s_t because all the predicates relate objects in an object set

Predicate	Sort
⟨ObjectSet⟩(_) *in state* ⟨SC⟩(_, _)	$\langle s_o, s_t, s_t \rangle$
⟨ObjectSet⟩(_) *transition* ⟨TID⟩ ⟨phase⟩(_, _)	$\langle s_o, s_t, s_t \rangle$
⟨ObjectSet⟩(_) *transition* ⟨TID⟩ *trigger true*(_, _)	$\langle s_o, s_t, s_t \rangle$
⟨ObjectSet⟩(_) *transition* ⟨TID⟩ *committed using* ⟨PSC⟩(_)	$\langle s_o, s_t \rangle$
⟨ObjectSet⟩(_) ⟨PSC⟩ *reset*(_, _)	$\langle s_o, s_t, s_t \rangle$
⟨ObjectSet⟩(_) *transition* ⟨TID⟩ *action done*(_)	$\langle s_o, s_t \rangle$
⟨ObjectSet⟩(_) *transition* ⟨TID⟩ *finished using* ⟨SSC⟩(_)	$\langle s_o, s_t \rangle$
⟨ObjectSet⟩(_) *passed* ⟨RTM⟩ *at time*(_)	$\langle s_o, s_t \rangle$

Fig. 8. Predicate Templates for OBM Components.

to events (e.g., finishing a transition) or time intervals (e.g., being in a state) associated with an OBM model instance.

Using the generated predicate symbols, we construct formulas for the static behavioral properties. In all, there are 23 conversion procedures that generate formulas. However, only 11 of them produce formulas for our sample model instance. The first three generate referential-integrity formulas, which we do not show here. They guarantee, for example, that when an object is in the *Open* state, it is a member of the *Order* object set, etc.

A fourth procedure generates formulas like Formula 2 to guarantee equivalence between a state conjunction and the conjunction of its states.

$$\forall x \forall t_1 \forall t_2 ((Order(x) \textit{ in state } Ready(t_1, t_2) \wedge \textit{ } Order(x) \textit{ in state } Paid(t_1, t_2)) \\ \Leftrightarrow Order(x) \textit{ in state } Ready \textit{ \& state } Paid(t_1, t_2)) \quad (2)$$

A fifth procedure generates formulas like Formula 3 that specify what it means for a prior-state conjunction to be reset (i.e., an object uses the prior-state conjunction to commit a transition).

$$\forall x \forall t_1 \forall t_2 ((\text{NTI}(Order(x) \textit{ in state } Ready(t_1, t_2), _t 1_t 2) \wedge \\ \text{NTI}(Order(x) \textit{ in state } Paid(t_1, t_2), t_1, t_2)) \Leftrightarrow \\ Order(x) \textit{ Ready \& } Paid \textit{ reset}(t_1, t_2)) \quad (3)$$

Here, NTI($Order(x)$ *in state* $Ready(t_1, t_2), t_1, t_2$) is a shorthand for the following formula that is true if and only if x is not in the *Ready* state for all time points in $[t_1, t_2)$ time interval.

$$t_1 < t_2 \wedge \forall t_3 \forall t_4 ((t_1 \leq t_3 \wedge t_3 \leq t_2 \wedge t_1 \leq t_4 \wedge t_4 \leq t_2) \Rightarrow \\ \neg Order(x) \textit{ in state } Ready(t_3, t_4)) \quad (4)$$

Similarly, NTI($Order(x)$ *in state* $Paid(t_1, t_2), t_1, t_2$) represents a formula that is true if and only if x is not in the *Paid* state for all time points in time interval $[t_1, t_2)$. Like the STI notation, an NTI shorthand can be substituted with the formula it represents without changing the meaning of the original formula.

A sixth procedure generates formulas that equate the firing phase of each transition with the concatenation of its committed, executing, and finishing

phases. A seventh procedure generates formulas that guarantee the elementary phases of a transition are mutually exclusive. An eighth procedure generates formulas to guarantee that objects in the associated object set are in at least one phase of each transition in a state net. We do not give examples here.

A ninth procedure generates formulas like Formula 5 to guarantee that objects never commit conflicting transitions with common prior states at the same time. Two transitions are conflicting if they share prior states (e.g., transitions *[1]* and *[3]* conflict w.r.t. *Customer Waiting*).

$$\begin{aligned}\forall x \forall t(&Order(x)\ transition\ [1]\ committed\ using\\ &Customer\ Waiting\ \&\ Unpaid(t) \Rightarrow\\ \neg&Order(x)\ transition\ [3]\ committed\ using\ Customer\ Waiting(t))\end{aligned} \tag{5}$$

A tenth procedure generates formulas like Formula 6 that restrict temporal relations represented by the predicate symbols for real-time markers so that they only relate objects to time points for which the objects passed to the real-time markers.

$$\begin{aligned}\forall x \forall t(&Order(x)\ passed\ a\ at\ time(t) \Leftrightarrow\\ &Order(x)\ transition\ [4]\ finished\ using\ Paid(t))\end{aligned} \tag{6}$$

Formula 6 specifies that if an object passes real-time marker a at time t, that object finishes transition *[4]* and enters the *Paid* state at time t and the converse. Thus the temporal relation represented by the predicate symbol *Order*(_) *passed a at time*(_) is equivalent to the temporal relationship represented by the predicate symbol *Order*(_) *transition [1] finished using Paid*(_).

Finally, an eleventh procedure generates formulas like Formula 7 to express real-time constraints. The real-time constraint, *(a to b)* $\leq$ *20*, in Figure 7 is an a OSM-Logic Formula, where *to* and $\leq$ are predicate symbols and a and b are variables representing values for real-time markers.

$$\begin{aligned}\forall x \forall t_1 \forall t_2((&Order(x, t_1, t_2) \wedge t_1 < t_2 \wedge Order(x)\ passed\ a\ at\ time(t_1) \wedge\\ &Order(x)\ passed\ b\ at\ time(t_2) \wedge \forall t_3((t_1 < t_3 \wedge t_3 < t_1) \Rightarrow\\ &(\neg Order(x)\ passed\ a\ at\ time(t_3) \wedge \neg Order(x)\ passed\ b\ at\ time(t_3)))) \Rightarrow\\ &(t_1\ to\ t_2) \leq \text{“20”})\end{aligned} \tag{7}$$

The last part of Formula 7 is derived from the real-time constraint. Here, as in Figure 7, “20” represents 20 minutes. An interpretation of this formula would map this constant symbol to an appropriate object in the range of the time-interval magnitude function. As a result, *minutes* is normalized to the time unit represented in the interpretation. The *to* symbol in the (t_1 *to* t_2) formula represents the time-interval magnitude function.

Dynamic Properties. The dynamic behavioral properties are conditions that relate various time intervals represented by OBM components. We use the predicate symbols described in the prior section to express dynamic behavioral properties in OSM-Logic. There are 12 conversion procedures that generate formulas for these properties, 7 of which generate formulas for our sample model instance. We present these 7 procedures here. One procedure generates formulas that relate certain time intervals associated with an object and the ready phase of a transition to other time intervals that occur just before, during, and just after the ready phase. For example, the procedure generates the following formulas for transition *[1]*.

$$\begin{aligned}
&\forall x \forall t_1 \forall t_2 (\mathrm{STI}(\mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{ready}(t_1, t_2), t_1, t_2) \Rightarrow \\
&\qquad \exists t_3 (t_3 < t_1 \wedge (\mathrm{NTI}(\mathit{Order}(x, t_3, t_1), t_3, t_1) \vee \\
&\qquad\qquad \mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{inactive}(t_3, t_1)))) \qquad (8) \\
&\forall x \forall t_1 \forall t_2 (\mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{ready}(t_1, t_2) \Rightarrow \\
&\qquad (\mathit{Order}(x)\ \mathit{in\ state}\ \mathit{Open}(t_1, t_2) \vee \\
&\qquad\ \mathit{Order}(x)\ \mathit{in\ state}\ \mathit{Customer\ Waiting}\ \&\ \mathit{state}\ \mathit{Unpaid}(t_1, t_2))) \qquad (9) \\
&\forall x \forall t_1 \forall t_2 (\mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{ready}(t_1, t_2) \Rightarrow \\
&\qquad \mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{trigger}\ \mathit{true}(t_1, t_2)) \qquad (10) \\
&\forall x \forall t_1 \forall t_2 (\mathrm{ETI}(\mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{ready}(t_1, t_2), t_1, t_2) \Rightarrow \\
&\qquad \exists t_3 (t_2 < t_3 \wedge (\mathrm{NTI}(\mathit{Order}(x, t_2, t_3), t_2, t_3) \vee \\
&\qquad\qquad \mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{committed}(t_2, t_3) \vee \qquad (11) \\
&\qquad\qquad \mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{inactive}(t_2, t_3))))
\end{aligned}$$

Formula 8 defines what happens before an object enters the ready phase of transition *[1]*. The first part of this formula is true when $[t_1, t_2)$ is a starting-time interval for an object x being in the ready phase of transition *[1]*. In other words, t_1 is the time point when x enters the ready phase of transition *[1]*. The second part is conditional upon the first. It specifies that there is a time point t_3 before t_1 such that either x is not a member of *Order* or x is in the inactive phase of transition *[1]* during the $[t_3, t_1)$ time interval. Formulas 9 and 10 define what must be true while an object is in the ready phase of transition *[1]*. Formula 9 specifies that at least one of the transition's prior-state conjunctions must be true, and Formula 10 specifies that the transition's trigger must be true. Formula 11 defines what happens after an object is the ready phase of transition *[1]*. The first part of this formula is true when $[t_1, t_2)$ is an ending-time interval for an object x being in the ready phase of transition *[1]*. In other words, t_2 is the time point when x exits the ready phase. The second part, which is conditional upon the first, specifies that there exists a time t_3 after t_2 such that either x is not a member of *Order*, x is in the committed phase of transition *[1]*, or x is in the inactive phase of transition *[1]* during the $[t_2, t_3)$ time interval.

Another procedure generates formulas like Formulas 12–16 that define what happens before and after an object is in the committed phase of a transition.

$$\forall x \forall t_1 \forall t_2 (\text{STI}(\mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{committed}(t_1, t_2), t_1, t_2) \Rightarrow \exists t_3 (t_3 < t_1 \wedge \mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{ready}(t_3, t_1))) \tag{12}$$

$$\forall x \forall t_1 (\mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{committed\ using}\ \mathit{Open}(t_1) \Rightarrow \exists t_3 \exists t_4 (t_3 < t_1 \wedge t_1 < t_4 \wedge \mathit{Order}(x)\ \mathit{in\ state}\ \mathit{Open}(t_3, t_1) \wedge \mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{ready}(t_3, t_1) \wedge \mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{committed}(t_1, t_4) \wedge \mathit{Order}(x)\ \mathit{Open}\ \mathit{reset}(t_1, t_4))) \tag{13}$$

$$\forall x \forall t_1 (\mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{committed\ using}\ \mathit{Open}(t_1) \Rightarrow \neg \mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{committed\ using}\ \mathit{Customer\ Waiting}\ \&\ \mathit{state\ Unpaid}(t_1)) \tag{14}$$

$$\forall x \forall t_1 (\text{STI}(\mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{committed}(t_1, t_2), t_1, t_2) \Rightarrow (\mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{committed\ using}\ \mathit{Open}(t_1) \vee \mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{committed\ using}\ \mathit{Customer\ Waiting}\ \&\ \mathit{state\ Unpaid}(t_1))) \tag{15}$$

$$\forall x \forall t_1 \forall t_2 (\text{ETI}(\mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{committed}(t_1, t_2), t_1, t_2) \Rightarrow \exists t_3 (t_2 < t_3 \wedge \mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{executing}(t_2, t_3))) \tag{16}$$

Formulas 12 through 16 specify what must be true just prior to or at the time an object enters the committed phase of transition *[1]*. Formula 12 guarantees that an object is in the ready phase of transition *[1]* just prior to starting the committed phase. Formula 13 says that if an object commits transition *[1]* using the *Open* state at time point t_1, then the object is in the ready phase of transition *[1]* and in the *Open* state just before t_1. Further, it says that the object is in the committed phase at t_1 and that the *Open* state is reset at t_1. Formula 14 guarantees that an object can commit transition *[1]* with only one of its prior-state conjunctions at a time. Formula 15 says that if $[t_1, t_2)$ is a starting-time interval for an object being in the committed phase of transition *[1]*, then that object committed the transition at time t_1 using one of its prior-state conjunctions. Formula 16 guarantees that an object is in the executing phase of transition *[1]* just after it exits the committed phase.

A third procedure generates formulas like Formulas 17–18 to require that an object is in the committed phase of transition *[1]* prior to being in the executing phase, and is in the finishing phase after being in the executing phase.

$$\forall x \forall t_1 \forall t_2 (\text{STI}(\mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{executing}(t_1, t_2), t_1, t_2) \Rightarrow \exists t_3 (t_3 < t_1 \wedge \mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{committed}(t_3, t_1))) \tag{17}$$

$$\forall x \forall t_1 \forall t_2 (\text{ETI}(\mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{executing}(t_1, t_2), t_1, t_2) \Rightarrow \exists t_3 (t_2 < t_3 \wedge \mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{finishing}(t_2, t_3))) \tag{18}$$

A fourth procedure generates formulas like Formulas 19–22 to express the dynamic properties of the finishing phase, such as how the completion of actions and the entering of subsequent-state conjunctions relate to the finishing of transitions.

$$\begin{aligned}
&\forall x \forall t_1 \forall t_2 (\mathrm{STI}(\mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{finishing}(t_1, t_2), t_1, t_2) \Rightarrow \\
&\qquad \exists t_3 (t_3 < t_1 \wedge \mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{executing}(t_3, t_1) \wedge \\
&\qquad \mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{action\ done}(t_1))) \qquad (19) \\
&\quad \forall x \forall t_1 (\mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{finished\ using\ Canceled}(t_1) \Rightarrow \\
&\qquad \exists t_3 \exists t_4 (t_3 < t_1 \wedge t_1 < t_4 \wedge \\
&\qquad \mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{finishing}(t_3, t_1) \wedge \qquad (20) \\
&\qquad \mathit{Order}(x)\ \mathit{in\ state\ Canceled}(t_1, t_4))) \\
&\forall x \forall t_1 \forall t_2 (\mathrm{ETI}(\mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{finishing}(t_1, t_2), t_1, t_2) \Rightarrow \\
&\qquad \mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{finished\ using\ Canceled}(t_2)) \qquad (21) \\
&\forall x \forall t_1 \forall t_2 (\mathrm{ETI}(\mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{finishing}(t_1, t_2), t_1, t_2) \Rightarrow \\
&\qquad \exists t_3 (\mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{inactive}(t_2, t_3))) \qquad (22)
\end{aligned}$$

Formula 19 guarantees that an object must be in executing phase of transition *[1]* prior to be being in the finishing phase. In addition, it specifies that the time point when an object enters the finishing phase corresponds to the time point it completed the action of transition *[1]*. Formula 20 says that if an object finishes transition *[1]* and enters the *Canceled* state at time t_1, then that object is in the finishing phase of transition *[1]* prior to t_1, and it is in the *Canceled* state at t_1. Formula 21 guarantees that if $[t_1, t_2)$ is an ending-time interval for an object in the finishing phase of transition *[1]*, then that object finished transition *[1]* using the *Canceled* state at time t_2. Formula 22 guarantees that an object is in the inactive phase of transition *[1]* after it is in the finishing phase.

A fifth procedure generates formulas like Formulas 23–24 to define dynamic properties of the inactive phase of a transition.

$$\begin{aligned}
&\forall x \forall t_1 \forall t_2 (\mathrm{STI}(\mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{inactive}(t_1, t_2), t_1, t_2) \Rightarrow \\
&\qquad \exists t_3 (t_3 < t_1 \wedge (\mathrm{NTI}(\mathit{Order}(x, t_3, t_1), t_2, t_3) \vee \qquad (23) \\
&\qquad \mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{finishing}(t_3, t_1)))) \\
&\forall x \forall t_1 \forall t_2 (\mathrm{ETI}(\mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{inactive}(t_1, t_2), t_1, t_2) \Rightarrow \\
&\qquad \exists t_3 (t_2 < t_3 \wedge (\mathrm{NTI}(\mathit{Order}(x, t_2, t_3), t_2, t_3) \vee \qquad (24) \\
&\qquad \mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{ready}(t_2, t_3))))
\end{aligned}$$

Formula 23 guarantees that an object is either not a member of the *Order* object set or in the finishing phase of transition *[1]* prior to being in the inactive phase of transition *[1]*. Formula 23 specifies that an object is either not a member of the *Order* object set or in the ready phase of transition *[1]* after being in the inactive phase of transition *[1]*.

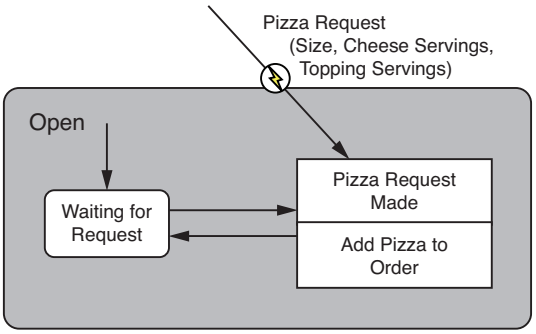

Fig. 9. Details of the Open State for the *Order* Object Set.

The last two procedures generate formulas like Formulas 25–26 that specify when objects can enter and exit a state. An object can enter (exit) a state only when the object uses it to finish (commit) a transition.

$$\forall x \forall t_1 \forall t_2 (\text{STI}(\mathit{Order}(x)\ \mathit{in\ state}\ \mathit{Canceled}(t_1, t_2), t_1, t_2) \Rightarrow \\ \mathit{Order}(x)\ \mathit{transition}\ [1]\ \mathit{finished\ using}\ \mathit{Canceled}(t_1)) \quad (25)$$

$$\forall x \forall t_1 \forall t_2 (\text{ETI}(\mathit{Order}(x)\ \mathit{in\ state}\ \mathit{Canceled}(t_1, t_2), t_1, t_2) \Rightarrow \\ \mathit{Order}(x)\ \mathit{transition}\ [6]\ \mathit{committed\ using}\ \mathit{Canceled}(t_2)) \quad (26)$$

Formula 25 guarantees that the time point when an object starts being in the *Canceled* state corresponds to when it finishes transition *[1]* using the *Canceled* state. Similarly, Formula 26 specifies that the time point when an object stops being in the *Canceled* state corresponds to when it commits transition *[6]*.

4.2 Converting OIM Components

Static Properties. To express static interaction properties, we first construct predicate symbols for an OBM. For example, we construct the predicate symbol *Interaction Pizza Request*(_): *to Order*(_) *transition [7] firing with Size*(_) *Cheese Servings*(_) *Topping Servings*(_)(_,_) for the interaction described in Figure 9. (A preliminary transformation supplied *[7]* as the identifier for the transition in Figure 9.) The predicate symbol construction varies for each interaction component C, depending on whether C has an activity description; whether C has an origin, a destination, or both; if C has an origin, whether it is an object set, state, or transition; if C has a destination, whether it is an object set, state, or transition; and whether C has object descriptions.

All interaction predicate symbols start with *Interaction* and a place that represents interaction objects. If an interaction component has an activity description, we use that description to label the first place. For the interaction in Figure 9, we embed its activity description, *Pizza Request*, in the predicate symbol just before the first place. Next, we include places for the interaction's origin and destination. Since our sample interaction does not have an origin, we only

include a place for its destination. We prefix the destination place with *to Order* to help identify it. Also, since the destination of the interaction is transition *[7]*, we embed transition *[7]* firing after the destination place to indicate that only objects firing transition *[7]* can receive this type of interaction. Next, we add a place for each object description associated with the interaction component. An object description identifies a parameter object that origin and destination objects exchange during the interaction. For our example, *Size*, *Cheese Servings*, and *Topping Servings* are the object descriptions. Finally, we add on two places of sort s_t to represent the time interval during which the interactions occur.

Using the predicate symbols generated for interaction components, objects sets, states, and transitions, we generate the formulas for static interaction properties. There are 10 conversion procedures that generate interaction formulas; however, only 3 of them pertain to our sample model instance.

One procedure generates formulas that guarantee the referential integrity of temporal relations represented by interaction predicate symbols. For example, any origin object for an interaction must be a member of the object set associated with the interaction component's origin. A second procedure generates formulas like Formulas 27–28 to ensure that a single interaction occurs only once.

$$\begin{aligned}
&\forall x_0 \forall x_1 \forall x_2 \forall x_3 \forall x_4 \forall t_1 \forall t_2 \forall t_3 \forall t_4 ((\mathrm{STI}(\textit{Interaction Pizza Request}(x_0) : \\
&\quad \textit{to Order}(x_1)\ \textit{transition}\ [7]\ \textit{firing with Size}(x_2)\ \textit{Cheese Serving}(x_3) \\
&\qquad \textit{Topping Serving}(x_4)(t_1, t_2), t_1, t_2) \wedge \\
&\quad \textit{Interaction Pizza Request}(x_0) :\ \textit{to Order}(x_1) \\
&\qquad \textit{transition}\ [7]\ \textit{firing with Size}(x_2)\ \textit{Cheese Serving}(x_3) \\
&\qquad \textit{Topping Serving}(x_4)(t_3, t_4)) \Rightarrow (t_1 \leq t_3 \wedge t_1 \leq t_4)) \qquad (27) \\
&\forall x_0 \forall x_1 \forall x_2 \forall x_3 \forall x_4 \forall t_1 \forall t_2 \forall t_3 \forall t_4 ((\mathrm{ETI}(\textit{Interaction Pizza Request}(x_0) : \\
&\quad \textit{to Order}(x_1)\ \textit{transition}\ [7]\ \textit{firing with Size}(x_2)\ \textit{Cheese Serving}(x_3) \\
&\qquad \textit{Topping Serving}(x_4)(t_1, t_2), t_1, t_2) \wedge \\
&\quad \textit{Interaction Pizza Request}(x_0) :\ \textit{to Order}(x_1) \\
&\qquad \textit{transition}\ [7]\ \textit{firing with Size}(x_2)\ \textit{Cheese Serving}(x_3) \\
&\qquad \textit{Topping Serving}(x_4)(t_3, t_4)) \Rightarrow (t_3 \leq t_2 \wedge t_4 \leq t_2)) \qquad (28)
\end{aligned}$$

Formula 27 specifies that if an interaction, x_0, starts at time t_1 and it also occurs during the interval $[t_3, t_4)$, then the time points, t_3 and t_4, are no earlier than t_1. Similarly, Formula 28 says that if an interaction, x_0, ends at time t_2 and it also occurs during the interval $[t_3, t_4)$, then the time points, t_3 and t_4, are no later than t_2.

A third procedure generates formulas like Formula 29 to guarantee that source and destination objects exchange only one set of parameter objects during a single interaction.

$$\forall x_0 \forall x_1 \forall t_1 \forall t_2 \exists^1_{x_2} \exists^1_{x_3} \exists^1_{x_4} (\textit{Interaction Pizza Request}(x_0) : \\ \textit{to Order}(x_1)\ \textit{transition}\ [7]\ \textit{firing with Size}(x_2) \\ \textit{Cheese Serving}(x_3)\ \textit{Topping Serving}(x_4)(t_1, t_2)) \tag{29}$$

Dynamic Properties. To express dynamic interaction properties we generate one or two additional predicate symbols for each interaction that represent the potential for interaction to occur. One predicate symbol, which we generate only when the interaction has an origin, relates interactions and objects to time intervals during which the objects are potential origins for the interactions. The other predicate symbol, which we generate only when the interaction has a destination, relates interactions and objects to time intervals during which the objects are potential destinations for the interactions. For example, we generate the predicate symbol *Receive Interaction Pizza Request*(_): *to Order*(_) *transition [7] firing with Size*(_) *Cheese Servings*(_) *Topping Servings*(_)(_, _), for the interaction in Figure 9.

Our algorithm includes only one procedure that generates formulas for dynamic interaction properties. The formulas guarantee that the maximum-time interval for an interaction (1) starts after a time interval corresponding to its origin object's potential to be an origin, and (2) ends before a time interval corresponding to its destination object's potential to be an destination. For example, the procedure generates the following formula for our sample model instance.

$$\begin{array}{l} \forall x_0 \forall x_1 \forall x_2 \forall x_3 \forall x_4 \forall t_1 \forall t_2 (\\ \quad (\text{MTI}(\textit{Interaction Pizza Request}(x_0) : \\ \qquad \textit{to Order}(x_1)\ \textit{transition}\ [7]\ \textit{firing with Size}(x_2)\ \textit{Cheese Serving}(x_3) \\ \qquad\quad \textit{Topping Serving}(x_4)(t_1, t_2), t_1, t_2) \Rightarrow \\ \quad \exists t_3 \exists t_4 (t_1 \leq t_3 \wedge t_3 < t_2 \wedge \\ \qquad \textit{Receive Interaction Pizza Request}(x_0) :\ \textit{to Order}(x_1) \\ \qquad\quad \textit{transition}\ [7]\ \textit{firing with Size}(x_2)\ \textit{Cheese Servings}(x_3) \\ \qquad\quad \textit{Topping Servings}(x_4)(t_3, t_4) \wedge \\ \qquad \forall t_5 ((t_3 \leq t_5 \wedge \\ \qquad\quad \text{ETI}(\textit{Receive Interaction Pizza Request}(x_0) :\ \textit{to Order}(x_1) \\ \qquad\qquad \textit{transition}\ [7]\ \textit{firing with Size}(x_2)\ \textit{Cheese Servings}(x_2) \\ \qquad\qquad \textit{Topping Servings}(x_3)(t_3, t_5), t_3, t_5)) \Rightarrow t_2 \leq t_5)))) \end{array} \tag{30}$$

5 Concluding Remarks

We have shown how to formalize conceptualizations for applications that need a time-dependent facts. The conceptualizations span the space of object existence,

the existence of relationships among objects, individual object behavior, and interactions between among groups of objects. The formalization defines the semantics of OSM by showing how to convert any populated OSM model instance to formulas in OSM-Logic and how to interpret these formulas.

References

[ACM] ACM-L-2010 workshop. http://www.cs.uta.fi/conferences/acm-l-2010/.

[CEW92] S.W. Clyde, D.W. Embley, and S.N. Woodfield. The complete formal definition for the syntax and semantics of osa. Technical Report BYU-CS-92-2, Department of Computer Science, Brigham Young University, February 1992.

[Cly93] S.W. Clyde. *An Initial Theoretical Foundation for Object-Oriented Systems Analysis and Design*. PhD thesis, Brigham Young University, 1993.

[Dor09] D. Dori. *Object-Process Methodology: A Holistic Systems Paradigm*. Springer, Berlin, Germany, 2009.

[EKW92] D.W. Embley, B.D. Kurtz, and S.N. Woodfield. *Object-oriented Systems Analysis: A Model-Driven Approach*. Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[End72] H.B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, Inc., Boston, Massachusetts, 1972.

[ET11] D.W. Embley and B. Thalheim, editors. *Handbook of Conceptual Modeling: Theory, Practice, and Research Challenges*. Springer, Heidelberg, Germany, 2011.

[HM08] T.A. Halpin and T. Morgan. *Information Modeling and Relational Databases*. Morgan Kaufmann, Burlington, Massachusetts, 2nd edition, 2008.

[IAR] Knowledge discovery and dissemination program. http://www.iarpa.gov/-solicitations_kdd.html/.

[Oli07] A. Olivè. *Conceptual Modeling of Information Systems*. Springer, Berlin, Germany, 2007.

[ORM] The ORM Foundation. http://www.ormfoundation.org.

[PM07] O. Pastor and J.C. Molina. *Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling*. Springer, New York, New York, 2007.

[Tha00] B. Thalheim. *Entity-Relationship Modeling: Foundations of Database Technology*. Springer, Berlin, Germany, 2000.

[UML] Omg: Documents associated with UML version 2.3. http://www.omg.org/-spec/UML/2.3/.

[YK58] J.W. Young and H.K. Kent. Abstract formulation of data processing problems. *The Journal of Industrial Engineering*, 9(6):471–479, 1958.