T_Chimera: A Temporal Object-Oriented Data Model

Elisa Bertino, Elena Ferrari
Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Via Comelico 39/41 - 20135 Milano, Italy. E-mail: bertino/ferrarie@dsi.unimi.it

Giovanna Guerrini
Dipartimento di Informatica e Scienze dell'Informazione, Università degli Studi di Genova, Via Dodecaneso, 35 - 16146 Genova, Italy. E-mail: guerrini@disi.unige.it

Temporal data models are an active research area. However, although many temporal extensions have been proposed for the relational data model, there is no comparable amount of work in the context of object-oriented data models. This paper presents *T*_Chimera, a temporal extension of the Chimera object-oriented data model. The main contribution of this work is to define a formal temporal object-oriented data model and to address on a formal basis several issues deriving from the introduction of time in a semantically rich model like the object-oriented one.

1. Introduction

Most conventional database systems do not offer the possibility of dealing with time-varying data. The content of a database represents a *snapshot* of the reality in that only the current data are recorded, without the possibility of maintaining the complete history of data over time. If such a need arises, data histories must be managed at application program level, thus making the management of data very difficult, if at all possible. To overcome this drawback, in the past years there has been a growing interest in extending data models to deal with temporal aspects [27,28]. Most proposals are temporal extensions of the relational data model [12,14,24,26]. Extensions of the relational model can be classified into two main categories. The *tuple timestamping* approach uses normalized (1NF) relations in which special time attributes are added to tuples [24]. The *attribute timestamping* approach uses non-normalized (N1NF) relations and attaches time to attribute values [14].

By contrast, research on temporal object-oriented databases is still in its early stage. Although various object-oriented temporal models have been proposed [23], there is no amount of theoretical work comparable to the work reported for the relational model. In particular, none of the temporal object-oriented data models proposed so far has a formal semantics. For example, Wuu and Dayal temporal extension of the OODAPLEX model [29] addresses the introduction of time in an object-oriented context but not on a formal basis. By contrast, a great amount of theoretical research is being carried on in the area of non-temporal object-oriented programming languages and databases, and theoretical foundations have been established [1,3,19]. In particular, issues concerning type systems and type checking have been widely investigated in a non-temporal object-oriented framework [2,8,17]. These issues, however, have never been addressed in the framework of temporal object-oriented data models. Thus, the extension of established foundations to temporal object-oriented data models still needs deep investigations.

In this paper we present *T*_Chimera, a temporal extension of the Chimera data model. Chimera is an object-oriented, deductive, active data model, developed as part of the ESPRIT Project Idea P6333 [16]. It provides all concepts commonly ascribed to object-oriented data models, such as object identity, complex objects, user-defined methods, classes and inheritance. Moreover, it provides capabilities for defining deductive rules, that can be used to define views and integrity constraints, to formulate queries, to specify methods, and to compute derived information. Finally, it supports a powerful language for defining triggers. A formal model has been defined, providing a foundation for the various features of the Chimera data model [16]. In Chimera, as in many other object-oriented data models, *objects* are abstraction of real-world entities (such as a person or an employee). Each object has a unique system defined object identifier (*oid*). The *oid* is assigned automatically by the system upon the object creation and remains immutable for the whole life of the object. Properties of objects are described by means of *attributes*. Attribute values can change over time, without changing the object identity. Objects can be manipulated by means of *methods*, which can be built-in or user-defined. Objects having similar properties and behavior are grouped into *classes* organized in inheritance hierarchies. Chimera provides a uniform notion of object, in that classes are themselves objects.

At the current stage, T_Chimera extends the Chimera object-oriented data model with the time dimension, but it does not cover other features of the Chimera model, such as rules. The main contribution of this paper is thus to address on a formal basis several issues deriving from adding temporal capabilities to an object-oriented model. First, the notion of temporal type is introduced. Temporal types handle in a uniform way temporal and non-temporal domains. Then, we provide a formal definition of classes and objects, as well as of schema and database. The notions of inheritance and method dispatching are revisited in a temporal context. Finally, some constraints to be satisfied by any legal database, such as referential integrity and object consistency, are discussed. Among the novel aspects of the proposed model, we would like to mention the support for immutable, static and temporal attributes, the possibility of defining *snapshot conditions*, to selectively keep track of the history of attribute changes, and the notion of substitutability through coercion from temporal to non-temporal domains. This paper extends [4] with the possibility of expressing both snapshot conditions and time-dependent behavior. The influence of these notions on object consistency and inheritance is also investigated.

The remainder of the paper is organized as follows. Next section reviews related work. Section 3 introduces the symbols most frequently used in this paper. Sections 4 and 5 formally introduce T_Chimera types and values. Sections 6 and 7 deal with classes and objects, respectively. Inheritance is considered in Section 8, whereas Section 9 introduces the notions of schema and database. Section 10 deals with method dispatching in a temporal context. Finally, Section 11 concludes the paper and outlines future work. Appendix A reports the language for defining snapshot conditions, whereas formal proofs are reported in Appendix B.

2. Related work

In this section we compare some of the temporal object-oriented data models proposed so far. Some of the approaches considered here are compared under a quite different perspective by Snodgrass [23]. In [23] the emphasis is on temporal object-oriented query languages, while we consider data model characteristics. Moreover, in [23] only the temporal characteristics are compared, disregarding the object-oriented ones, whereas we consider both. Dimensions considered here can be classified into three broad categories: *(i)* object-oriented characteristics; *(ii)* temporal characteristics (that is, notions of time supported); *(iii)* how the above characteristics are integrated, that is, how time is associated with objects. The upper part of Table 1 considers the first two categories, whereas in the lower part of the table the last category is taken into account.

Consider first the upper part of Table 1. Concerning the temporal aspects, the third column identifies the time structure supported by each of the considered data models. Note that most models assume time to be linear, that is, they assume that time advances from the past to the future in a totally ordered fashion, and discrete (i.e., isomorphic to the natural numbers). By contrast, only few models support a user-defined hierarchy of time types. The fourth column indicates the time dimensions supported by the models. Two time dimensions are of general interest in temporal databases: *valid* time (the time a fact was true in reality) and *transaction* time (the time the fact was stored in the database). Note that most models support only valid time. For the object-oriented characteristics, the fifth column identifies whether values (literals) are distinguished from objects [3,19]. "Objects" indicates that values and objects are undistinguished, whereas "both" indicates that values are distinguished from objects. Finally, the last column identifies whether class features are supported. As a general remark, we point out that most proposed approaches focus on the temporal characteristics of the model and consider rather simple object-oriented models.

Let us now consider how temporal and object-oriented characteristics are combined (lower part of Table 1). The second column indicates at which granularity data are timestamped. Note that some approaches associate a timestamp with the entire object, whereas others associate a timestamp separately with each object attribute. "Arbitrary" indicates that the user decides what granularity of data should be timestamped. The third column identifies how the value of a temporal attribute is modeled. Note that, among the approaches associating timestamps with single attribute values, the majority regards the value of a temporal attribute as a partial function from a temporal domain to the set of legal values for the attribute. The fourth column identifies whether temporal, immutable and static attributes are supported. We define an attribute to be *temporal* (or historical) if its value may change over time, and if the values of the attribute at different times are recorded in the database. An *immutable* attribute is an attribute whose value cannot be modified during the object lifespan, whereas a *static* (or non-temporal) attribute is an attribute whose value can change over time, but whose past values are not meaningful for the application at hand, and thus are not stored in the database. We think that, for a model like ours that distinguishes between objects and values, the notion of static attribute is more relevant in

TABLE 1. Comparison among temporal object-oriented data models

	o-o data model	time structure	time dimension	values & objects	class features
[11]	OODAPLEX	linear, discrete	valid	objects	NO[b]
[12]	generic	linear, discrete	valid	objects	NO
[15]	TIGUKAT	user-defined	valid	objects	NO
[18]	MAD	linear, discrete	valid + transaction	objects	NO
[20]	3DIS	linear, discrete	valid	objects	NO
[22]	generic	linear, discrete	valid + transaction	objects	NO
[25]	OSAM*	linear, discrete	valid	objects	NO
[29]	OODAPLEX	user-defined	arbitrary[a]	objects	NO[b]
Our model	Chimera	linear, discrete	valid	both	YES

	what is timestamped	temp. attr. values	kinds of attributes	histories obj. types	temporal methods	snapshot conditions
[11]	attributes	functions[c]	temporal + immutable	NO	NO	NO
[12]	attributes	functions[c]	temporal + immutable	YES	NO	NO
[15]	arbitrary	sets of pairs	temporal + immutable	YES	YES	NO
[18]	objects	atomic valued[d]	temporal + immutable	NO	NO	NO
[20]	attributes	sets of triples[e]	temporal	NO	NO	YES
[22]	attributes	sequences of tuples	temporal + immutable	NO	NO	NO
[25]	objects	atomic valued[d]	temporal + immutable	NO[f]	NO	NO
[29]	arbitrary	functions[c]	temporal + immutable	YES	NO	NO
Our model	attributes	functions[c]	temporal + immutable + static	YES	YES	YES

[a] One single time dimension is considered, but it can be interpreted either as transaction or as valid time.
[b] OODAPLEX supports metadata, but neither [11] nor [29] consider them.
[c] With the term *functions* we denote partial functions from a temporal domain.
[d] Time is associated with the entire object state.
[e] The triple elements are (oid, attribute name, attribute value); each element of the triple has a time interval and a version number.
[f] The information is not associated with objects; however it can be derived from the histories of objects.

practice than the notion of immutable one. The fifth column identifies whether the considered models keep track of the history of associations between an object and its class. An important dynamic aspect of object-oriented databases is that an object can dynamically

TABLE 2. Notation and terminology

Symbol	Meaning
$\mathcal{OI}$, $\mathcal{CI}$	a set of object identifiers/class identifiers (i.e., class names)
$\mathcal{AN}$,$\mathcal{MN}$	a set of attribute/method names
$\mathcal{OT}$, $\mathcal{VT}$, $\mathcal{CT}$	the set of Chimera object types/Chimera value types/Chimera types
$\mathcal{TT}$, $\mathcal{ST}$, $\mathcal{T}$	the set of T_Chimera temporal types/T_Chimera static types/T_Chimera types
$\mathcal{V}$	the set of T_Chimera legal values
$\mathcal{SC}$	the set of T_Chimera snapshot conditions
$c_1 <_{ISA} c_2$, $c_1 \leq_{ISA} c_2$	c_1 is a proper subclass/subclass of c_2
$T_1 <_T T_2$,$T_1 \leq_T T_2$	T_1 is a proper subtype/subtype of T_2
$T_1 \preceq_T T_2$	type T_1 is a refinement of type T_2
$sc_1 \preceq_s sc_2$	snapshot condition sc_1 is a refinement of snapshot condition sc_2
$[\![T]\!]_t$	the extension of type T at time t
$\pi(c,t)$	the extent of class c at time t
$\gamma(c)$	the class denoted by the class identifier c
$type(c)$,$h\_type(c)$,$s\_type(c)$	the structural/historical/static type of class c
$snapshot(i,t)$	the state of the object identified by i at time t
$ISA(c)$	the set of directed superclasses of class c
$ref(i,t)$	the set of oids to which the object identified by i refers at time t
$h\_state(i,t)$	the historical value at time t of the object identified by i
$s\_state(i)$	the static value of the object identified by i
$\Pi_{\{a_1,\ldots,a_m\}}(rt)$	the record type obtained by projecting the record type rt onto attributes $\{a_1,\ldots,a_m\}$
τ_m^c	the set of intervals for which an implementation for method m is specified in class c
$inh\_from(c,m,t)$	the direct superclass of class c from which c inherits method m at time t
$IM(m,c)$	the set of implementations for method m specified in class c

change class, by specializing or generalizing its current class. Thus, we distinguish the approaches keeping track of the dynamic links between an object and its most specific class from those not considering this aspect. The sixth column indicates whether the models support temporal methods, that is, whether the models allow to record the different implementations of a method over time, taking into account the time of the invocation during dispatching. Finally, the last column identifies whether *snapshot conditions* can be associated with attributes. A snapshot condition associated with an attribute denotes a set of events upon which the value of the attribute must be recorded in the database. Object migrations or creations are examples of such events. Note that most models do not provide this possibility. In these models it is possible to keep track either of the whole history of changes of an attribute or of its current value only. There is no possibility of selectively keeping track of the changes to an attribute value over time.

As a final remark let us mention that, as pointed out in [23], in contrast to temporal relational data models, the specification of temporal object-oriented data models is in most cases informal. In particular, none of the considered proposals addresses the issues resulting from the introduction of time in an object-oriented data model on a formal basis. Only Wuu and Dayal in [29] discuss the consistency of an object with respect to the classes it belongs to, and only few proposals [10,15,29] consider the impact of inheritance in a temporal framework. However, none of them addresses issues concerning domain refinement and substitutability.

3. Notations used in the paper

Table 2 illustrates the symbols more frequently used in this paper. For each symbol, the table reports a brief explanation of its meaning.

4. Types

In this section we introduce the set of T_Chimera types. First we briefly review the Chimera types, then we introduce T_Chimera types.

In Chimera the existence of a finite set $\mathcal{BVT} = \{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$ of predefined basic value types is postulated, containing the types *integer*, *real*, *bool*, *character* and *string*. Moreover, Chimera allows class names to be used in the definition of types. The following definition states that each class name is a (object) type.

Definition 1. (Object Types) *[16] The set of Chimera object types $\mathcal{OT}$ is defined as the set of class identifiers $\mathcal{CI}$.* □

Chimera supports *structured types* such as sets, lists and records. A set type is denoted *set-of*(T), where T is a type. Instances of *set-of*(T) are sets of instances of type T. A list type is denoted *list-of*(T), where T is a type. Instances of *list-of*(T) are lists of instances of type T. A record type is denoted *record-of*$(a_1 : T_1, \ldots, a_n : T_n)$, where $T_1, \ldots, T_n$ are types and $a_1, \ldots, a_n \in \mathcal{AN}$ are distinct names. Instances of this type are records with n components, whose *i-th* component is an instance of type T_i, $i = 1, \ldots, n$. Chimera allows the use of class names, i.e. object types, in the definition of structured types.

The following definition introduces Chimera value types.

Definition 2. (Value Types) *[16] The set of Chimera value types $\mathcal{VT}$ is recursively defined as follows:*

- the predefined basic value types are value types ($\mathcal{BVT} \subseteq \mathcal{VT}$);
- if T is a value type or an object type then *list-of*(T) and *set-of*(T) are structured value types, respectively indicated as list type and set type;
- if $T_1, \ldots, T_n$ are value types or object types and $a_1, \ldots, a_n$ are distinct elements of $\mathcal{AN}$, then *record-of*$(a_1 : T_1, \ldots, a_n : T_n)$ is a structured value type, indicated as record type.

□

The set of Chimera types $\mathcal{CT}$ is defined as the union of Chimera value types $\mathcal{VT}$ and Chimera object types $\mathcal{OT}$, as illustrated in Figure 1.

T_Chimera extends the set of Chimera types with a collection of *temporal types*. Temporal types are introduced to type in a uniform way historical (i.e., temporal) variables, and static variables, that is, variables for which the history of changes over time is recorded and variables for which only the current value is kept. For each Chimera type T, a corresponding temporal type, denoted *temporal*(T), is defined. Moreover, the set $\mathcal{BVT}$ of Chimera predefined basic value types is extended to contain also the type *time*. Type *time* is used in the definition of temporal types. Intuitively, instances of type *temporal*(T) are partial functions from instances of type *time* to instances of type T.

Definition 3. (T_Chimera Temporal Types) *The set of T_Chimera temporal types $\mathcal{TT}$ is defined as the set of types temporal(T), for each $T \in \mathcal{CT}$.* □

In T_Chimera temporal types can be used in the definition of structured types in that the definition of sets or lists of instances of temporal types is allowed in T_Chimera. Moreover, attributes in a record may have temporal types as domains. The following definition formalizes these concepts.

Definition 4. (T_Chimera Types) *The set of T_Chimera types $\mathcal{T}$ is defined as follows:*

- *time* is a T_Chimera type (*time* $\in \mathcal{T}$);
- the Chimera types are T_Chimera types ($\mathcal{CT} \subseteq \mathcal{T}$);
- the T_Chimera temporal types are T_Chimera types ($\mathcal{TT} \subseteq \mathcal{T}$);
- if T is a T_Chimera type then *list-of*(T) and *set-of*(T) are T_Chimera structured types;
- if $T_1, \ldots, T_n$ are T_Chimera types and $a_1, \ldots, a_n$ are distinct elements of $\mathcal{AN}$, then *record-of*$(a_1 : T_1, \ldots, a_n : T_n)$ is a T_Chimera structured type.

□

In the following we refer to the elements in the set $\mathcal{ST} = \mathcal{T} \setminus \mathcal{TT}$ as *static types*. Figure 2 gives a graphic representation of T_Chimera types.

Example 1. Let project be a class name, belonging to $\mathcal{CI}$. The following are T_Chimera types:

time
temporal$(integer)$
list-of$(boolean)$
temporal$(set\text{-}of(project))$
record-of$(task : temporal(project), startbudget : real, endbudget : real)$.

By contrast, temporal$(temporal(real))$ is not a T_Chimera type since temporal$(real) \notin \mathcal{CT}$. □

Note that, according to Definition 4 a static type may contain as a component a temporal type. For instance *set-of*$(temporal(integer))$ is a legal T_Chimera type according to Definition 4.

In T_Chimera inheritance relationships among classes are described by an ISA hierarchy specified by

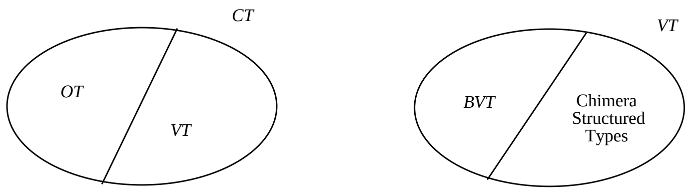

FIG. 1. Chimera types

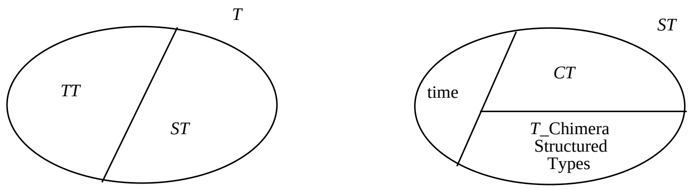

FIG. 2. $\mathcal{T}$_Chimera types

the user. The ISA hierarchy represents which classes are subclasses of (inherit from) other classes. We assume this information to be expressed as a partial order $<_{ISA}$ on the set of class identifiers $\mathcal{CI}$. Starting from the ISA hierarchy, an ordering relation $<_{\mathcal{T}}$ can be imposed on the set of T_Chimera types $\mathcal{T}$.

Definition 5. (Subtypes) *Given $T_1, T_2 \in \mathcal{T}$, T_2 is a proper subtype of T_1 (denoted as $T_2 <_{\mathcal{T}} T_1$) iff one of the following conditions holds:*

- $T_2, T_1 \in \mathcal{OT}$ and $T_2 <_{ISA} T_1$;
- $T_2 = set\text{-}of(T_2')$, $T_1 = set\text{-}of(T_1')$ and $T_2' <_{\mathcal{T}} T_1'$;
- $T_2 = list\text{-}of(T_2')$, $T_1 = list\text{-}of(T_1')$ and $T_2' <_{\mathcal{T}} T_1'$;
- $T_1 = record\text{-}of(a_1 : T_1', \ldots, a_n : T_n')$, $T_2 = record\text{-}of(a_1 : T_1'', \ldots, a_n : T_n'')$ and $\forall i \in [1, n]$: $T_i' <_{\mathcal{T}} T_i''$;
- $T_2 = temporal(T_2')$, $T_1 = temporal(T_1')$ and $T_2' <_{\mathcal{T}} T_1'$.

□

Example 2. *Let project, research_project, person and employee be elements of $\mathcal{CI}$, such that $research\_project <_{ISA} project$ and $employee <_{ISA} person$. Then, the following subtyping relationships hold:*

$research\_project <_{\mathcal{T}} project$
$employee <_{\mathcal{T}} person$
$temporal(set\text{-}of(research\_project)) <_{\mathcal{T}} temporal(set\text{-}of(project))$
$list\text{-}of(employee) <_{\mathcal{T}} list\text{-}of(person)$
$record\text{-}of(task : temporal(research\_project), startbudget : real, endbudget : real) <_{\mathcal{T}} record\text{-}of(task : temporal(project), startbudget : real, endbudget : real).$

□

The set $\mathcal{T}$ of T_Chimera types with the order $<_{\mathcal{T}}$ is a partially ordered set. Indeed, $<_{\mathcal{T}}$ is an antisymmetric and transitive relation. In what follows, $\leq_{\mathcal{T}}$ denotes the reflexive closure of the $<_{\mathcal{T}}$ order, i.e., given $T_1, T_2 \in \mathcal{T}$, T_2 is said a subtype of T_1 (denoted as $T_2 \leq_{\mathcal{T}} T_1$) iff $T_1 = T_2$ or $T_1 <_{\mathcal{T}} T_2$.

5. Values

In this section we introduce the set of T_Chimera legal values $\mathcal{V}$. We start by introducing, for each T_Chimera type, the corresponding set of legal values.

For each predefined basic value type $\mathcal{B} \in \mathcal{BVT}$ we postulate the existence of a non-empty set of values, denoted as $dom(\mathcal{B})$. For instance, the domain of the basic value type *integer* is the set $\mathcal{Z}$ of integer numbers, whereas the domain of the basic value type *bool* is the set $\{\texttt{true}, \texttt{false}\}$. Moreover, we assume as the domain of type *time* the set $\mathcal{TIME} = \{\texttt{0}, \texttt{1}, \ldots, \texttt{now}\}$. Symbol '0' denotes the relative beginning, whereas now is a special constant denoting the current time. Thus, we assume time to be discrete. An interval, denoted as $I = [\texttt{t}_1, \texttt{t}_2]$ is a set of consecutive time instants. It includes all time instants between $\texttt{t}_1$ and $\texttt{t}_2$, $\texttt{t}_1$ and $\texttt{t}_2$ included. A single time instant $\texttt{t}$ can be represented as the time interval $[\texttt{t}, \texttt{t}]$, while $[,]$ denotes the null interval, that is, the interval not containing any time instant. The operations of union $(I_1 \cup I_2)$, intersection $(I_1 \cap I_2)$, and inclusion $(I_1 \subseteq I_2)$ have the usual semantics of set operations. Moreover $t \in I$ is true if t is one of the time instants represented by interval I. In the following we use a set of disjoint intervals $I = \{[\texttt{t}_i, \texttt{t}_j], \ldots, [\texttt{t}_r, \texttt{t}_s]\}$ as a compact notation for the set of time instants included in these intervals.

Note that in T_Chimera oids in $\mathcal{OI}$ are handled as values. Thus, an object identifier i is a value of an object type in $\mathcal{OT}$. According to the usual terminology, an object is an instance of a class c, if c is the most specific class, in the inheritance hierarchy, to which the object belongs. If an object is an instance of a class c, it is also a member of the superclasses of c. Therefore, we consider as legal values for an object type c the oids of objects that are either instances or members of c. The set of objects members or instances of a class changes dynamically over time. Thus, to define the extension, that is, the set of legal values for each T_Chimera type, we introduce a function π: $\mathcal{CI} \times \mathcal{TIME} \rightarrow 2^{\mathcal{OI}}$, assigning an extent to each class, for each instant t. For each $c \in \mathcal{CI}$, for each $t \in \mathcal{TIME}$, $\pi(c,t)$ is the set of the identifiers of objects that, at time t, belong to c either as instances or as members. To emphasize the fact that the interpretation of a type can only be given by fixing a time instant t, we denote the set of legal values for type T at time t as $[\![T]\!]_t$.

Definition 6. (Type Legal Values) *Let t be a time instant. The legal extension at time t for each T_Chimera type T, denoted $[\![T]\!]_t$, is defined as follows:*

- $\forall T \in \mathcal{T}$: $null \in [\![T]\!]_t$;
- $\forall \mathcal{B} \in \mathcal{BVT}$: $[\![\mathcal{B}]\!]_t = dom(\mathcal{B})$;
- $[\![time]\!]_t = \mathcal{TIME}$;
- $\forall c \in \mathcal{CI}$: $[\![c]\!]_t = \pi(c,t)$;
- $\forall T \in \mathcal{T}$: $[\![set\text{-}of(T)]\!]_t = 2^{[\![T]\!]_t}$;
- $\forall T \in \mathcal{T}$: $[\![list\text{-}of(T)]\!]_t = \bigcup_{n \in \mathbb{N}^+} \{[v_1,\ldots,v_n] \mid v_i \in [\![T]\!]_t, i=1,\ldots,n\}$;
- $\forall \{T_1,\ldots,T_n\} \in 2^{\mathcal{T}}$, $\forall n \in \mathbb{N}^+$: $[\![record\text{-}of(a_1 : T_1,\ldots,a_n : T_n)]\!]_t = \{(a_1 : v_1,\ldots,a_n : v_n) \mid a_i \in \mathcal{AN}, v_i \in [\![T_i]\!]_t, i=1,\ldots,n\}$;
- $\forall T \in \mathcal{CT}$: $[\![temporal(T)]\!]_t = \{f : \mathcal{TIME} \rightarrow \bigcup_{t' \in \mathcal{TIME}} [\![T]\!]_{t'} \mid f$ is a partial function such that $\forall t' \in \mathcal{TIME}$: if $f(t')$ is defined, then $f(t') \in [\![T]\!]_{t'}\}$.

□

Intuitively, given a time instant t, the extensions of predefined basic value types at time t are the elements of their corresponding domains, the extensions of classes are their explicit extents at time t, while the set of legal values for structured types is defined recursively in term of the legal values for their component types. The extension of a temporal type $temporal(T)$ is the set of partial functions from $\mathcal{TIME}$ (i.e, the set of legal values for type $time$) to the union of the sets of legal values for type T, for each instant t' in $\mathcal{TIME}$. The value of a variable of type $temporal(T)$ can therefore be represented as a set of pairs $(t, f(t))$, where f is a partial function, t is an element of $\mathcal{TIME}$, and $f(t)$ is the value of function f at time t. Usually, the value of a variable of type $temporal(T)$ does not change at each instant. Therefore, its value can be represented in a more compact way as a set of pairs $\{\langle \tau_1, v_1 \rangle, \ldots, \langle \tau_n, v_n \rangle\}$, where $v_1,\ldots,v_n$ are legal values for type T, and $\tau_1,\ldots,\tau_n$ are time intervals such that the variable assumes the value v_i for each instant in τ_i, $i = 1,\ldots,n$. We adopt this representation throughout the paper.

The set of T_Chimera legal values $\mathcal{V}$ is defined as the union of the set of legal values of each T_Chimera type, that is, $\mathcal{V} = \bigcup_{T \in \mathcal{T}, t \in \mathcal{TIME}} [\![T]\!]_t$.

Note that T_Chimera supports a polymorphic *null* value; *null* is a legal value for any T_Chimera type.

Example 3. Let t be a time instant, $\mathtt{i}_1,\mathtt{i}_2 \in \mathcal{OI}$ *such that* $\mathtt{i}_1$, $\mathtt{i}_2 \in \pi(person,t)$ *and* $\mathtt{i}_2 \in \pi(employee,t)$*. The following are T_Chimera legal values:*

- `10,100` $\in [\![integer]\!]_t$, since $[\![integer]\!]_t = \mathcal{Z}$;
- $\mathtt{i}_2 \in [\![employee]\!]_t$, since $[\![employee]\!]_t = \pi(employee,t)$ and $\mathtt{i}_2 \in \pi(employee,t)$;
- $\{\mathtt{i}_1,\mathtt{i}_2\} \in [\![set\text{-}of(person)]\!]_t$, since $[\![set\text{-}of(person)]\!]_t = 2^{[\![person]\!]_t}$, $[\![person]\!]_t = \pi(person,t)$ and $\mathtt{i}_1$, $\mathtt{i}_2 \in \pi(person,t)$;
- $\{\langle$`[5,10]`$,$`12`$\rangle, \langle$`[11,30]`$,$`5`$\rangle\} \in [\![temporal(integer)]\!]_t$, since $\{\langle[5,10],12\rangle, \langle$`[11,30]`$,$`5`$\rangle\}$ is a partial function from $\mathcal{TIME}$ to $\bigcup_{t' \in \mathcal{TIME}} [\![integer]\!]_{t'} = \mathcal{Z}$, and `12`, `5` $\in \mathcal{Z}$;
- $(name$: 'Bob', $score$:$\{\langle$`[1,100]`$,$`40`$\rangle, \langle$`[101,200]`$,$`70`$\rangle\})$ $\in [\![record\text{-}of(name : string, score : temporal(integer))]\!]_t$, since 'Bob' $\in [\![string]\!]_t$ and $\{\langle$`[1,100]`$,$`40`$\rangle, \langle$`[101,200]`$,$`70`$\rangle\}) \in [\![temporal(integer)]\!]_t$.

□

Definition 6 has formally introduced the set of legal values for each T_Chimera type. Figure 3 shows the T_Chimera typing rules for value. The meaning of these inference rules is the following: if the conditions in the rule premises (the upper part of the rule) are satisfied, then the rule consequence (the lower part of the rule) can be inferred. Notation $v :^t T$ means that value v has type T at time t.

In the above rules, $\sqcup$ denotes the least upper bound of a set of types with respect to the subtyping order introduced in Section 4. Note that the notion of least upper bound is well defined since the set $\mathcal{T}$ of T_Chimera types with the $<_T$ relation is a partially ordered set.

The above typing rules are used to check whether a database state is structurally consistent, that is, to verify whether the values of the attributes of each object are legal with respect to their definition. Such typing rules are also the basis for type checking the expressions of T_Chimera language. The following theorems state the soundness and completeness of our type system.

Theorem 7. (Soundness of typing rules for values) *Let T be the type deduced for a value v at a time t according to rules in Figure 3, then $v \in [\![T]\!]_t$.*

Theorem 8. (Completeness of typing rules for values) *Let v be a legal value for type T at time t, that is, $v \in [\![T]\!]_t$, then, according to rules in Figure 3we deduce type T for v at time t.*

(R 1) $\dfrac{}{null :^t T}$ $T \in \mathcal{T}, t \in \mathcal{TIME}$

(R 2) $\dfrac{v \in dom(\mathcal{B})}{v :^t \mathcal{B}}$ $\mathcal{B} \in \mathcal{BVT}, t \in \mathcal{TIME}$

(R 3) $\dfrac{v \in \mathcal{TIME}}{v :^t time}$ $t \in \mathcal{TIME}$

(R 4) $\dfrac{i \in \pi(c,t)}{i :^t c}$ $i \in \mathcal{OI}, c \in \mathcal{CI}, t \in \mathcal{TIME}$

(R 5) $\dfrac{\forall i \in [1,n] : v_i :^t T_i}{\{v_1, \ldots, v_n\} :^t set{-}of(T)}$ $T = \bigsqcup_{i=1}^n T_i, t \in \mathcal{TIME}, n \in \mathbb{N}^+$

(R 6) $\dfrac{\forall i \in [1,n] : v_i :^t T_i}{[v_1, \ldots, v_n] :^t list{-}of(T)}$ $T = \bigsqcup_{i=1}^n T_i, t \in \mathcal{TIME}, n \in \mathbb{N}^+$

(R 7) $\dfrac{\forall i \in [1,n] : v_i :^t T_i}{(a_1 : v_1, \ldots, a_n : v_n) :^t record{-}of(a_1 : T_1, \ldots, a_n : T_n)}$ $a_1, \ldots, a_n \in \mathcal{AN}, t \in \mathcal{TIME}, n \in \mathbb{N}^+$

(R 8) $\dfrac{\forall i \in [1,n] : v_i :^{t_i} T, t_i :^t time}{\{(t_1, v_1), \ldots, (t_n, v_n)\} :^t temporal(T)}$ $t_i \neq t_j, 1 \leq i,j \leq n, t \in \mathcal{TIME}, n \in \mathbb{N}^+$

FIG. 3. T_Chimera typing rules for values

6. Classes

A class in T_Chimera consists of three components: the *signature*, the *implementation*, and the *extent*. The signature of a class contains all the information for the use of the class and its instances, whereas the implementation provides an implementation for the signature. Finally, the extent keeps track of the set of objects instances of the class over time.

6.1. Class signature

The signature of a class contains information about the attributes and methods of its instances. Each attribute is characterized by its name, its type and its domain. T_Chimera supports three different types of attribute: temporal, immutable and static. A temporal attribute is an attribute whose value may change over time, and whose values at different times are recorded in the database. An immutable attribute is an attribute whose value cannot be modified during the object lifespan, whereas a static attribute is an attribute whose value can change over time, but whose past values are not recorded in the database. Note that the support for temporal, immutable and static attributes represents an innovative feature with respect to all the temporal object-oriented data models proposed in the literature.

Moreover, a *snapshot condition* is associated with each temporal attribute. A snapshot condition is a mean to selectively record the values of a temporal attribute. A snapshot condition denotes an event, or a set of events. Each time the events denoted by the snapshot condition occur, the value of the corresponding attribute at the time of the event occurrence is stored in the database. For instance, a snapshot condition allows to state that the value of an attribute must be recorded each time the object migrates from a class to another, or upon the object deletion. Moreover, it is possible to state that the salary of an employee must be recorded for all the time instants during which the salary of his manager is above a given threshold, or for all the times the invocation of a given method produces a predefined result. Also explicit time instants can be specified by means of snapshot conditions. For example, it is possible to specify that the value of an attribute must be recorded at a given set of instants. A `null` snapshot condition implies that the entire history of the attribute values must be kept. T_Chimera provides a language to express snapshot conditions. A detailed syntax of the language is given in Appendix A. In the following, $\mathcal{SC}$ denotes the set of the snapshot conditions expressible in our language.

Each method is characterized by its name, and the type of the input and output parameters. Moreover, since the objects belonging to a class vary over time, each T_Chimera class also maintains the history of all the objects instances or members of the class over time. Each class has a *lifespan*, representing the time interval during which the class has existed. We use the notation `mm/dd/yy:hh:mm:ss` to represent the starting and the ending time of the class lifespan. For instance, the notation `1/1/96:08:00:02` represents the second instant of `8a.m` on `1/1/96`. When `1/1/96` is used as the starting time of an interval, it denotes the first instant of the first day of `January 1996`, while, when it is used as the ending point of an interval, it denotes the last instant of `1/1/96`. We make the assumption, common to most of the other models, that the lifespan of a class is contiguous, that is, it consists of a set of consecutive time instants.

The signature of a class is formally defined as follows.

Definition 9. (Class Signature) *A class signature is a 4-tuple* $(c, lifespan, attr, meth)$, *where*

$c \in \mathcal{CI}$ is the class identifier;

$lifespan \in (\mathcal{TIME} \times \mathcal{TIME})$ is the lifespan of the class;

$attr$ is a set containing an item for each attribute of the class instances. This item is a 4-tuple $(a\_name, a\_type, a\_domain, a\_snapshot)$, where

- $a\_name \in \mathcal{AN}$ is the attribute name;
- $a\_type \in \{\texttt{temporal}, \texttt{static}, \texttt{immutable}\}$ is the attribute type;
- $a\_domain \in \mathcal{ST}$ is the attribute domain;
- $a\_snapshot \in \mathcal{SC}$ is a snapshot condition.[1]

$meth$ is a set containing an item for each method of the class instances. Each item is a pair (m_name, m_sign) where

- $m\_name \in \mathcal{MN}$ is the method name;
- m_sign is the signature of the method, expressed as: $T_1 \times \ldots \times T_n \rightarrow T$, where $T_1, \ldots T_n$ and $T \in \mathcal{T}$, denote types of input parameters and of the output parameter of the method, respectively.

□

Note that, in order to avoid redundancy, we allow only the use of static types in the definition of attribute domains, that is, we specify a temporal attribute a of type $temporal(T)$ by means of the 4-tuple $(a, \texttt{temporal}, T, sc)$ instead of by $(a, \texttt{temporal}, temporal(T), sc)$. Moreover, we consider immutable attributes as a particular kind of temporal ones, that is, as temporal attributes whose values never change during the object lifespan. Thus, the value of an immutable attribute is modeled by means of a constant function from the temporal domain to the set of legal values for the attribute. Therefore, we can distinguish among temporal, immutable and static attributes. A temporal attribute is an attribute with a temporal type and whose value is a partial function from a temporal domain; an immutable attribute is an attribute with a temporal type but whose value is a *constant* function from the temporal domain; finally, a static attribute is an attribute with a static type. The following example clarifies these concepts.

Example 4. Consider a class project, created on `1/1/95`, *for the management of a number of projects. A project can be organized in several subprojects and is characterized by its name, the objective and the workplan, which consists of several tasks. Moreover, each project has a set of participants, a budget and a leader. Suppose that the variations over time of the objective and the workplan of each project are not relevant for the application at hand. By contrast, we are interested in keeping track of the complete history of changes for both the project participants and budget. Suppose moreover that: we are interested in recording the subprojects for all the instants for which the project participants are at least four or the project balance is greater than 100,000; and we would like to maintain information about who is the leader of the project at the time the corresponding object migrates from a class to another. The corresponding T_Chimera class signature is the following:*[2]

$$
\begin{aligned}
c &= project\\
lifespan &= [\texttt{1/1/95}, \texttt{now}]\\
attr = \{&(name, \texttt{immutable}, string),\\
&(objective, \texttt{static}, string),\\
&(workplan, \texttt{static}, set\text{-}of(task)),\\
&(subprojects, \texttt{temporal}, set\text{-}of(project),\\
&\qquad \texttt{count(participants)} \geq 4 \text{ OR}\\
&\qquad \texttt{balance > 100,000}),\\
&(budget, \texttt{temporal}, integer),\\
&(participants, \texttt{temporal}, set\text{-}of(person)),\\
&(leader, \texttt{temporal}, person, \texttt{specialize} \text{ OR}\\
&\qquad \texttt{generalize})\}\\
meth = \{&(add\_participant, person \rightarrow project),\\
&(best\_paid\_participant, \rightarrow person),\\
&(balance, \rightarrow integer)\}
\end{aligned}
$$

Note that the above signature implies that attribute name is of type temporal(string), since we regard immutable attributes as a particular kind of temporal ones. Moreover, attribute subprojects is of type temporal(set-of(project)), budget is of type temporal(integer), participants is of type temporal(set-of(person)), and leader is of type temporal(person). □

We now discuss the relationships between a class and its associated types. The identifier of a class C denotes the object type corresponding to C, that is, the type of the identifiers of the objects instances of C. Suppose that class C has as $attr$ component the set $\{attr_1, \ldots, attr_n\}$, where $attr_i = (a_i, type_i, T_i, sc_i)$, $i = 1, \ldots n$. The following types are associated with C.

- *Structural type.* It represents the type of the attributes of instances of C. It is defined by the record type: $record\text{-}of(a_1 : T'_1, \ldots, a_n : T'_n)$, where $\forall i \in [1,n]$: $T'_i = T_i$, if $type_i$=`static`; $T'_i = temporal(T_i)$, if $type_i \in$ {`immutable`,`temporal`}.
- *Historical type.* It represents the type of the temporal and immutable attributes of instances of C. It is defined by the record type: $record\text{-}of(a_k : T_k, \ldots, a_m : T_m)$, where $1 \leq k \leq m \leq n$ and $\{a_k, \ldots, a_m\} = \{a_i \mid a_i \in \{a_1, \ldots, a_n\} \wedge type_i \in \{\texttt{immutable}, \texttt{temporal}\}\}$.
- *Static type.* It represents the type of the static attributes of instances of C. It is defined by the record type: $record\text{-}of(a_j : T_j, \ldots, a_l : T_l)$, where

$1 \leq j \leq l \leq n$ and $\{a_j, \ldots, a_l\} = \{a_i \mid a_i \in \{a_1, \ldots, a_n\} \wedge type_i = \texttt{static}\}$.

Therefore, we define three functions *type*, *h_type*, *s_type*: $\mathcal{CI} \rightarrow \mathcal{T}$ which take as argument a class identifier c, and return the structural, the historical and the static type of the class identified by c, respectively. Note that function *h_type* returns a null value when its argument is the identifier of a class whose instances contain only static attributes, whereas function *s_type* returns a null value when its input is a class whose instances have only temporal attributes.

Example 5. Referring to the class of Example 4:

$$\begin{aligned}
type(project) = record\text{-}of(&name : temporal(string),\\
&objective : string,\\
&workplan : set\text{-}of(task),\\
&subprojects : temporal(set\text{-}of(project)),\\
&budget : temporal(integer),\\
&participants : temporal(set\text{-}of(person)),\\
&leader : temporal(person))\\
h_type(project) = record\text{-}of(&name : string,\\
&subprojects : set\text{-}of(project),\\
&budget : integer,\\
&participants : set\text{-}of(person),\\
&leader : person)\\
s_type(project) = record\text{-}of(&objective : string,\\
&workplan : set\text{-}of(task)).
\end{aligned}$$

□

The distinction between the structural, historical and static type of a class will be used in Section 9 when dealing with object consistency.

6.2. Class implementation

In addition to a signature, classes have an implementation. The implementation of a class must specify an implementation for all the methods that are specified in the signature. Method implementations are expressed by means of update operations constrained by declarative formulae.[3]

T_Chimera extends the set-oriented, rule-based method definition language of Chimera with the possibility of expressing time-dependent behaviors. Each method has a set of implementations, each referring to a different time interval. The time intervals of different implementations must be disjoint. The reason for supporting different method implementations, associated with different time intervals, is that not only stored data vary over time, but also rules to compute derived data from stored data. If we did not keep track of the evolution of those rules, we would be able to retrieve the values of stored attributes at past time instants, but we would not be able to reconstruct the value of derived information at these instants.

A T_Chimera formula is the conjunction of possibly negated atomic formulae. T_Chimera atomic formulae are in turn composed by applying a predicate symbol to a list of parameter terms;[4] three kinds of atomic formulae can be distinguished: comparison formulae, membership formulae and class formulae. Class formulae are used for typing purposes, and they cannot be negated; each formula must contain exactly one class formula for every variable appearing in it. Additional details on (T_)Chimera terms and formulae can be found in [16].

Example 6. Referring to class project of Example 4:

- $project(X)$ and $task(Y)$ are class formulae;
- $X.name =$'`IDEA`' and $X.budget >$ `1M` are comparison formulae;
- $X.leader\ in\ X.participants$ and $Y\ in\ X.tasks$ are membership formulae;
- $project(X) \wedge X.name =$'`IDEA`'$\wedge X.budget >$ `1M` is a formula.

□

Definition 10. (Method Implementation) *The implementation of a method m in a class c is a set*

$\{\langle \tau_1, impl_1\rangle, \ldots, \langle \tau_n, impl_n\rangle\}$ *where*

$\tau_1, \ldots \tau_n$ are time intervals such that $\tau_i \cap \tau_j = \emptyset$, $i \neq j$;

$impl_i$, $i \in [1, n]$, is an expression of the form
$meth\text{-}name : \{cond_1 \rightarrow code_1, \ldots, cond_k \rightarrow code_k\}$, $k \geq 0$ where

$meth\text{-}name$ is the method name m applied to a list of parameters;

$cond_j$, $j \in [1, k]$, is a T_Chimera formula, specifying a declarative control upon method execution;

$code_j$, $j \in [1, k]$, is a sequence of method invocations and update primitives, including object creation and deletion, object migration from one class to another and state changes.

□

Example 7. Referring to class project of Example 4, consider the following implementation for method balance:

$\{\ \langle[\texttt{1/1/95,12/31/95}], \texttt{balance(B)} : \{\texttt{B} = (\texttt{self.budget} - \texttt{self.participants} * 1,000) * 0.20 \rightarrow\}\rangle,$
$\langle[\texttt{1/1/96,now}], \texttt{balance(B)} : \{\texttt{B} = (\texttt{self.budget} - \texttt{self.participants} * 1,500) * 0.18 \rightarrow\}\rangle\ \}$

Note that no action is associated with method balance*; balance is indeed a side-effect method, in that it only computes derived information, without modifying the database state. If we did not keep the first implementation of balance, but only the current one, we would have not been able to reconstruct the balance of* *a project at a time t before* 1/1/96, *even if the value of budget and participants were available at that time.* □

Since T_Chimera supports time-dependent behaviors, the invocation of a method is performed with respect to a time instant t, and method dispatching takes care of this additional argument. We deal with that aspect in Section 10.

A class implementation is a collection of method implementations, as stated by the following definition.

Definition 11. (Class Implementation) *Let S be a class signature such that $S.meth = \{meth_1, \ldots, meth_n\}$, $meth_i = (m_i, sign_i)$, $i \in [1,n]$. An implementation for S is a set $\{m_1\_impl, \ldots, m_n\_impl\}$, such that $\forall i \in [1,n]$: m_i_impl is an implementation (according to Definition 9) of m_i.* □

6.3. Class Extent

In addition to a signature and an implementation, classes have an extent, that is, a set of instances, which is a time-varying component.

Definition 12. (Class) *A class C is a 3-tuple*

$(sign, impl, history)$ *where*

- *sign* is the class signature, as specified by Definition 8;
- *impl* is the class implementation, as specified by Definition 10;
- *history* $\in \mathcal{V}$ is a record value of the form: $(ext : E, proper\text{-}ext : PE)$, where E and PE are temporal values representing the set of objects members and instances of C over time.

□

In the following, we make use of function γ that, given a class identifier c' returns the class denoted by c', that is, $\forall c' \in \mathcal{CI} : \gamma(c') = C$, where C is the (unique) class such that $C.sign.c = c'$.

The type of *ext* and *proper-ext* is $temporal(set\text{-}of(c))$, where c is the class identifier of class C, since *ext* and *proper-ext* denote respectively the set of oids of the objects members of the class, and the set of oids of the objects instances of the class, for each instant t of the class lifespan. Obviously $\forall t \in C.sign.lifespan$: $PE(t) \subseteq E(t)$, as all objects instances of a class at a given instant are also members of the class at the same instant. Function π (cfr. Table 2), is such that, for each class name c, and for each $t \in \gamma(c).sign.lifespan$: $\pi(c,t) = \gamma(c).history.ext(t)$.

For each class C, the *ext* component of its *history* can be derived from the *proper-ext* components of its superclasses as follows. $\forall t \in \mathcal{TIME}$: $\mathrm{C}.history.ext(t) = \mathrm{C}.history.proper\text{-}ext(t)$, if $\not\exists \bar{c} : \bar{c} <_{ISA} C.sign.c$; $\mathrm{C}.history.ext(t) = \bigcup_{\bar{c} <_{ISA} C.sign.c} \bar{c}.history.ext(t)$, otherwise.

Example 8. A possible history for the class project of Example 4 is the following:

$($ *ext* : $\{\langle$[1/1/95,now]$,\{$i$_1, \ldots,$ i$_4\}\rangle\}$, *proper-ext* : $\{\langle$[1/1/95,1/8/95]$,\{$i$_1\}\rangle$, $\langle$[1/9/95,now], $\{$i$_1$,i$_2\}\rangle\})$

□

7. Objects

T_Chimera supports both *historical* and *static* objects. An object is historical if it contains at least an attribute with a temporal type or if at least a temporal method can be invoked on it; it is static otherwise. Each object, either historical or static, has a lifespan, representing the time interval during which the object exists. As for classes we assume the lifespan of an object to be contiguous. Therefore, we do not consider a *reincarnate* operation like the one proposed in [11]. Note that the contiguity assumption does not lessen the expressive power of our model. Objects can be instances of different classes during their lifespan, but we simply assume that for each instant in their lifespan, there exists at least a class to which they belong. This class is the most general class (in the inheritance hierarchy) the object has ever belonged to. For example, even if the lifespan contiguity is assumed, we can still model the fact that an employee is fired and rehired, as we can assume he remains instance of the generic class *person*, superclass of the class *employee*, till the end of its lifespan.

Moreover, for each historical object the history of the most specific classes to which it belongs during its lifespan is recorded. By contrast, for each static object, only the class identifier of the most specific class to which it currently belongs is maintained. The following definition formally introduces the notion of object.

Definition 13. (Object) *An object o is a 4-tuple*

$(i, lifespan, v, class\text{-}history)$ *where*

- $i \in \mathcal{OI}$ is the oid of o;
- $lifespan \in (\mathcal{TIME} \times \mathcal{TIME})$ is the lifespan of object o;
- $v \in \mathcal{V}$ is a record value, containing the values of each attribute of o. It is of the form: $(a_1 : v_1, \ldots, a_n : v_n)$, where $a_1, \ldots, a_n \in \mathcal{AN}$ are the names of the attributes of o, and $v_1, \ldots, v_n \in \mathcal{V}$ are their values;
- *class-history* stores information about the most specific classes to which o belongs over time. It is a set: $\{\langle\tau_1, c_1\rangle, \ldots, \langle\tau_n, c_n\rangle\}$, where $\tau_1, \ldots, \tau_n$ are time intervals and $c_1, \ldots, c_n \in \mathcal{CI}$ are class identifiers, such that c_i is the class identifier of the most specific class to which o belongs in the interval τ_i, $i = 1, \ldots, n$.

□

If o is static, *class-history* contains only one pair $\langle \tau, c\rangle$, where τ = [now,now] and c is the most specific class to which o belongs at the current time. Note that the information about the most specific classes to which an object belongs, that is, the information contained in the *class-history* component, could have also been derived by accessing the class of which the object is an instance. Indeed, each class contains the history of the objects instances of the class over time. We maintain this information in both the object and the class for efficiency reason. Indeed, information stored in the class are useful during query execution, whereas recording the information about the most specific class in the object makes easier consistency check and method dispatching.

Example 9. *Suppose* $\mathtt{i}_2, \ldots, \mathtt{i}_7 \in \mathcal{OI}$. *The following is an instance of class project of Example 4:*

$i = \mathtt{i}_1$
$lifespan = $ [3/5/95,now]
$v = \{(name : \{\langle$[3/5/95,now], 'IDEA'$\rangle\})$,
$(objective$: 'Implementation'$)$,
$(workplan : \{\mathtt{i}_7\})$,
$(subprojects : \{\langle$[1/2/96,now]$,\{\mathtt{i}_5,\mathtt{i}_6\}\rangle\})$,
$(budget : \{\langle$[3/5/95,1/1/96],0.3M$\rangle$,
$\langle$[1/2/96,now],1M$\rangle\})$,
$(participants : \{\langle$[3/5/95,now]$,\{\mathtt{i}_2,\mathtt{i}_3,\mathtt{i}_4\}\rangle\})$,
$(leader : null)\}$
$class\text{-}history = \{\langle$[3/5/95,now]$, project\rangle\}$

This object contains attribute name which is immutable, attributes objective and workplan, which are static, and attributes subprojects, budget, participants, and leader which are temporal. Note that for attributes budget and participants the whole history of changes is recorded, as they have a null snapshot condition. By contrast, only the value of subprojects in the interval [1/2/96,now] *is recorded as only in this interval the corresponding snapshot condition, i.e.,* count(participants) $\geq$ 4 *OR* balance > 100,000*, is satisfied. Moreover, no value for attribute leader is recorded, since the corresponding snapshot condition states that its value must be recorded only upon object migration, and object* $\mathtt{i}_1$ *has never migrated. Note that information about the object migration is recorded in the class_history component of* $\mathtt{i}_1$. *Indeed, since* $\mathtt{i}_1$ *is an historical object, the class-history component keeps track of the history of the most specific classes to which the object belongs over time. In this example the object is an instance of class project for all its lifespan. This is why a null value for attribute leader is recorded.* □

Chimera, like most object-oriented systems, supports two notions of equality: *equality by identity* (meaning that two objects are the same object), and *equality by value* (meaning that two objects have the same attribute values). Equality by value can be further refined into: *shallow value equality*, which considers the equality of all the direct attributes of an object, and *deep value equality*, which considers in addition to the equality of the attributes, the equality of the attributes of objects which are recursively reached by means of oid references. A formalization of these concepts for the Chimera model can be found in [16]. Here, we consider only shallow value equality.

In a temporal context, we still have the classical notion of object identity, being the oid of an object a time invariant property.

Definition 14. (Equality by Identity) *Two objects* o_1 *and* o_2 *are equal by identity iff they have the same object identifier, that is,* $o_1.i = o_2.i$. □

Obviously, if two objects have the same oid, also all their other components are equal. Note that the notion of object identity uniformly applies to both historical and static objects. Value equality is defined as follows.

Definition 15. (Value Equality) *Two objects* o_1 *and* o_2 *are equal by value iff* $o_1.v = o_2.v$. □

The above definition implies both the equality of the attribute values and the attribute names and applies to both static and historical objects. For static objects, it simply reduces to the conventional notion of value equality, whereas for historical objects the equality of the whole history of temporal attributes is required.

In a temporal context, two further notions of value equality can be devised: *instantaneous-value equality* and *weak-value equality*. Two objects are instantaneously value equal if there exists an instant t in which their attributes have the same values. Two objects are weakly value equal if their attributes have ever had the same values, also in different instants. These notions of equality obviously make sense for historical objects consisting only of temporal attributes. Moreover, also historical objects containing static attributes can be compared under these types of equalities, but only at the current time, since we cannot reconstruct the value of a static attribute at an instant t lesser than the current time. Finally, static objects can be compared only at the current time, too. Consider a function $snapshot : \mathcal{OI} \times \mathcal{TIME} \rightarrow \mathcal{V}$ that given an object identifier i' and an instant t, "projects" the state of the object identified by i' at time t. Consider an object o', with oid i', such that $o'.v = (a_1 : v'_1, \ldots, a_n : v'_n)$. Thus, $\forall t \in \mathcal{TIME}$, $snapshot(i', t)$ returns a record value $(a_1 : v_1, \ldots, a_n : v_n)$ such that, $\forall j \in [1, n]$:

- if attribute a_j is static, $v_j = o.v.a_j$;
- if a_j is temporal, $v_j = o.v.a_j(t)$;

where symbol o denotes the object identified by i' $(o.i = i')$.[5] Note that, for historical objects containing static attributes and for static objects, we are able to reconstruct only the snapshot at the current time; in this case $snapshot(i,t)$ is undefined for $t \neq$`now`. For instance, referring to the object of Example 9:

$$snapshot(\mathtt{i}_1, \mathtt{now}) = (name : \text{`}\mathtt{IDEA}\text{'},\\ objective : \text{`}\mathtt{Implementation}\text{'},\\ workplan : \{\mathtt{i}_7\},\\ subprojects : \{\mathtt{i}_5, \mathtt{i}_6\},\\ budget : \mathtt{1M},\\ participants : \{\mathtt{i}_2, \mathtt{i}_3, \mathtt{i}_4\},\\ leader : null)$$

whereas $snapshot(\mathtt{i}_1, t)$, for $t \neq$`now` is undefined, since $\mathtt{i}_1$ contains two static attributes.

We are now ready to introduce the notions of instantaneous-value equality and weak-value equality.

Definition 16. (Instantaneous-value Equality) *Two objects o_1 and o_2 are instantaneously value equal if there exists an instant $t \in o_1.lifespan \cap o_2.lifespan$ such that, $snapshot(o_1.i, t) = snapshot(o_2.i, t)$.* □

Definition 17. (Weak-value Equality) *Two objects o_1 and o_2 are weakly value equal if there exist $t' \in o_1.lifespan$ and $t'' \in o_2.lifespan$, such that $snapshot(o_1.i, t') = snapshot(o_2.i, t'')$.* □

Note that the last three notions of equality do not require the two objects be instances of classes related by inheritance. Obviously, value equality implies instantaneous-value equality, and instantaneous-value equality implies weak-value equality, while equality by identity implies all the other kinds of equality, when they are applicable.

Example 10. Referring to the class project of Example 4, two projects having the same current state and the same history of modifications for subprojects, budget, participants and leader attributes, are value equal. Note that, for attributes subprojects and leader we do not require the equality of the entire history of changes, but the equality at all the time instants at which the corresponding snapshot conditions are satisfied. Two projects having the same current value for all the attributes are instantaneous (and thus, weak) value equal. □

8. Inheritance

*T*_Chimera classes are related by inheritance relationships, establishing an ISA hierarchy expressed as a partial order $<_{ISA}$ on $\mathcal{CI}$. Definition 5 states how this order induces a (partial) order $<_T$ on the set $\mathcal{T}$ of *T*_Chimera types. Therefore, the type system $\mathbf{T} = (\mathcal{T}, <_T)$ can be represented as a directed acyclic graph (DAG). In this DAG there is a node for each type in $\mathcal{T}$, and there is a path from node T_1 to node T_2 if and only if $T_1 <_T T_2$. In *T*_Chimera the existence of a common root of the entire class hierarchy is not imposed. *T*_Chimera supports multiple inheritance, that is, a class may inherit from more than one (direct) superclass. However, the constraint is imposed that for multiple inheritance a common ancestor must exist. The class hierarchy is thus partitioned into multiple strongly connected components. Each strongly connected component is characterized by a single node with no incoming edge: this node is called *root* of the strongly connected component. Therefore, we may think of the set of all class identifiers $\mathcal{CI}$ as partitioned in m distinct hierarchies $\mathcal{H}_1, \ldots, \mathcal{H}_m$, corresponding to the m strongly connected components of the class hierarchy.

Let us consider a function $ISA : \mathcal{CI} \rightarrow 2^{\mathcal{CI}}$ such that for each $c \in \mathcal{CI}$, $ISA(c)$ is the set of the (direct) superclasses of c, that is, $ISA(c) = \{c' \mid c' \in \mathcal{CI} \wedge c <_{ISA} c' \wedge \nexists c^* \in \mathcal{CI} \text{ such that } c^* <_{ISA} c' \wedge c <_{ISA} c^*\}$. The properties of the $<_{ISA}$ relation are summarized by the following rule.

Rule 1. *The partial relation $<_{ISA} \subseteq \mathcal{CI} \times \mathcal{CI}$ must satisfy the following properties:*

- it must be a partial order (that is, it must be antisymmetric and transitive);
- if $c_1 \in ISA(c) \wedge c_2 \in ISA(c)$, then $\exists c^* \in \mathcal{CI}$ such that $c_1 <_{ISA} c^* \wedge c_2 <_{ISA} c^*$.

□

In what follows, $\leq_{ISA}$ denotes the reflexive closure of the $<_{ISA}$ relation.

A first restriction we impose on the inheritance hierarchy is that a subclass can exist only when its superclasses exist, as formalized by the following rule.

Rule 2. *Given $c_1, c_2 \in \mathcal{CI}$, such that $c_2 \leq_{ISA} c_1$, then $\gamma(c_2).sign.lifespan \subseteq \gamma(c_1).sign.lifespan$ must hold.* □

Referring to the class of Example 4, the lifespan of a subclass *research_project* of class *project* must be a subinterval of `[1/1/95,now]`.

Inheritance has two important implications in *T*_Chimera. The first is *substitutability*, that is, the property that each instance of a class can be used whenever an instance of one of its superclasses is expected. The second is *extent inclusion*, that is, the property ensuring that the extent of a class is included in the extents of its superclasses.

To ensure substitutability, a set of conditions must be satisfied by two classes related by the ISA relation. Those conditions are related to the fact that each subclass must contain all attributes and methods of its superclasses. Apart from the inherited concepts, additional features can be introduced in a subclass. Inherited concepts may be redefined (overwritten) in a subclass under a number of restrictions. In T_Chimera the redefinition of the signature of an attribute is possible by specializing the domain of the attribute. The redefinition of the signature of a method must verify the *covariance rule* for result parameters and the *contravariance rule* for the input ones. Therefore, in the subclass, result parameter domains may be specialized, whereas input parameter domains may be generalized.

In what follows we first consider conditions imposed on attribute and method refinement, then we consider extent inclusion.

8.1. Attribute refinement

First of all we formally introduce the notion of type refinement.

Definition 18. (Type Refinement) *Given $T, T' \in \mathcal{T}$, T' refines T (written $T' \preceq_T T$) iff one of the following conditions holds:*

- $T' \leq_T T$;
- $T' = temporal(T'')$ and $T'' \leq_T T$.

□

The idea is that attribute domains can be replaced in subclasses by refined types. This means that, as suggested in [10], a non-temporal attribute can be refined in a temporal attribute (on the same domain or on a most specific one), but not vice-versa. However, to ensure substitutability we need to introduce a *coercion* function, since the value of a temporal attribute, which is a function from a temporal domain, cannot be substituted (from a typing viewpoint) by a value of a non-temporal attribute, which is not a function. Suppose that class c_2 refines the non-temporal domain of attribute a of its superclass c_1 in a temporal domain. Then, whenever an object o instance of c_2 must be seen as an instance of c_1, the value of attribute a is "coerced" to $o.v.a(\texttt{now})$, that is, to the value at instant `now` of the function modeling the value of a in c_2. The coercion function we use is the function *snapshot* introduced in Section 7. By considering a snapshot of the object at instant `now`, the coerced value is $snapshot(o.i, \texttt{now}).a$. Note that this coercion is semantically meaningful. Indeed, in c_1 we are not interested in the history of the values taken by attribute a. Thus, whenever we see an object instance of c_2 as an instance of c_1, we forget the history of attribute a and consider only its current value.

Another aspect to be taken into account, when stating the conditions under which a given attribute can be modified in subclasses, is the refinement of the set of snapshot conditions associated with temporal attributes. The basic idea is that each time a value should be maintained for the attribute of the superclass then a value must be stored for that attribute also in the subclass. In addition to those requested by the superclass, the subclass can maintain more values for the attribute. Thus, the snapshot condition of the subclass must be *weaker* than the snapshot condition of the superclass. The following notion of refinement formalizes this concept.

Definition 19. (Refinement of Snapshot Conditions) *Let sc_1, $sc_2 \in \mathcal{SC}$. sc_2 is a refinement of sc_1, denoted as $sc_2 \preceq_s sc_1$, if each time sc_1 is satisfied sc_2 is also satisfied.* □

Consider for example the conditions sc_1 = `count(participants)` $\geq$ `4` *OR* `balance > 100,000` and sc_2 = `count(participants)` ≥ 4 *OR* `balance > 80,000` *OR* `name = 'IDEA'`. $sc_2 \preceq_s sc_1$, since `balance` $>$ `80,000` is true each time `balance >` `100,000` is. Appendix A deals with how refinement can be decided for conditions expressed in the snapshot language we consider. The following rule summarizes the restrictions we impose on attribute refinement.

Rule 3. (Attribute Refinement) *Let $c_1, c_2 \in \mathcal{CI}$ be such that $c_2 <_{ISA} c_1$. Suppose that $\gamma(c_1).sign.attr = \{attr_1, \ldots, attr_n\}$, where $attr_i = (a_i, type_i, T_i, sc_i)$, $i = 1, \ldots, n$. Then, $\forall i \in [1, n]$: $\gamma(c_2).sign.attr$ must include an item $attr'_i = (a'_i, type'_i, T'_i, sc'_i)$ such that one of the following conditions is satisfied:*

- $type_i = \texttt{static}$, $a_i = a'_i$, and $T'_i \preceq_T T_i$;
- $type_i = \texttt{temporal}$, $type'_i = \texttt{temporal}$, $a_i = a'_i$, $T'_i \preceq_T T_i$,and $sc'_i \preceq_s sc_i$;
- $type_i = \texttt{immutable}$, $type'_i = \texttt{immutable}$, $a_i = a'_i$, and $T'_i \preceq_T T_i$.

□

Example 11. *Consider the class project of Example 4. Suppose that project is refined into a subclass research_project whose attr component is as follows:*

$attr = \{(name, \texttt{immutable}, string),$
$(objective, \texttt{temporal}, string, \texttt{budget > 10M}),$
$(workplan, \texttt{static}, set\text{-}of(task)),$
$(subprojects, \texttt{temporal}, set\text{-}of(project),$
`count(participants)` ≥ 4 OR
`balance > 100,000` OR
`name = 'IDEA'`$),$
$(budget, \texttt{temporal}, integer),$
$(participants, \texttt{temporal}, set\text{-}of(employee)),$
$(leader, \texttt{temporal}, person,$ `specialize` OR
`generalize`$)\}$

assuming that employee $<_T$ person. The static attribute objective is refined into a temporal attribute with the same domain and a non null snapshot condition. Attribute subprojects is refined by generalizing the associated snapshot condition, whereas the temporal attribute participants is refined by replacing the domain person with employee. □

8.2. Method overriding

In this subsection, we consider how methods can be refined in subclasses. First of all, we consider the redefinition of method signatures. To ensure substitutability, we keep the basic rule of contravariance for input parameters and covariance for output ones, extending it to temporal types. We therefore impose the following conditions.

Rule 4. (Refinement of Method Signatures) *Let $c_1, c_2 \in \mathcal{CI}$ be such that $c_2 <_{ISA} c_1$. Let m be a method whose signature in $\gamma(c_1).sign.meth$ is $T_1 \times \ldots \times T_k \rightarrow T$. Method m in class c_2 can have as signature $T_1' \times \ldots \times T_k' \rightarrow T'$ where:*

- $\forall i \in [1, k]$: $T_i \preceq_T T_i'$;
- $T' \preceq_T T$.

□

Example 12. Suppose employee $<_T$ person. A subclass research_project of class project of Example 4 can refine the signature of method best_paid_participant from: $\rightarrow$ person to:
$\rightarrow$ employee, $\rightarrow$ temporal(person), or
$\rightarrow$ temporal(employee). □

Consider now method implementations. As in traditional object-oriented data models, a subclass may redefine the implementation of a method it inherits. Since we support time-dependent behaviors, method implementations can have time intervals. A subclass may perform only a partial (with respect to time) overriding of the methods inherited from its superclasses. Thus, we do not impose the condition that the implementations of a method in a class are specified for each time instant of the class lifespan, provided that an implementation for that time instant is specified in a superclass. In the following, given a class identifier c, we denote with τ_m^c the set of time intervals for which an implementation for m is specified in $\gamma(c).impl$, that is, if the implementation for m in $\gamma(c).impl$ is $\{\langle\tau_1, impl_1\rangle, \ldots, \langle\tau_n, impl_n\rangle\}$, then $\tau_m^c = \bigcup_{i \in [1,n]} \tau_i$. The following condition, referred to as *covering condition*, must always be verified.

Rule 5. *Let $c \in \mathcal{CI}$. Suppose that $\gamma(c).sign.meth = \{meth_1, \ldots, meth_n\}$, where $meth_i = (m_i, sign_i)$, $i = 1, \ldots, n$. Then, $\forall i \in [1, n]$:*

$$\bigcup_{c \leq_{ISA} c'} \tau_{m_i}^{c'} = \gamma(c).sign.lifespan$$

□

Example 13. Given the class project of Example 4, and the implementation for method balance of Example 7, consider a subclass research_project with the following implementation for method balance:

$\{ \langle[\texttt{6/1/95}, \texttt{9/30/95}], \texttt{balance(B)} : \{\texttt{B} = (\texttt{self.budget} - \texttt{self.participants} * 1,000) * 0.15 \rightarrow\}\rangle \}$.

Suppose that the lifespan of research_project is `[1/1/95,now]`*. Then:*

- $\tau_{\texttt{balance}}^{research\_project} = [\texttt{6/1/95}, \texttt{9/30/95}]$;
- $\tau_{\texttt{balance}}^{project} = [\texttt{1/1/95}, \texttt{now}]$;
- $\tau_{\texttt{balance}}^{project} \cup \tau_{\texttt{balance}}^{research\_project} = [\texttt{1/1/95}, \texttt{now}]$, and thus Rule 5 is satisfied for class *research_project*.

□

If, as in the example above, only a partial overriding of a method m is performed in a class c, then in c an implementation for m is specified only for some instants of the class lifespan. Whenever m is invoked on an object of class c at a time instant t for which an implementation is not specified in c, the implementation for m in the most specific superclass of c containing an implementation for m at time t, is used.

8.3. Extent inclusion

At the extensional level, we require that in each database state the extent of a class is a subset of the extent of its superclasses, as stated by the following rule.

Rule 6. *Given $c_1, c_2 \in \mathcal{CI}$, such that $c_2 \leq_{ISA} c_1$, then $\forall t \in \gamma(c_2).sign.lifespan$: $\gamma(c_2).history.ext(t) \subseteq \gamma(c_1).history.ext(t)$.* □

As we have already mentioned, in T_Chimera the class hierarchy is a DAG, consisting of a number of connected components whose roots are the classes without superclasses. Since we consider objects which are instances of a unique class, the sets of oids in different hierarchies, that is, hierarchies with different roots, are disjoint. Consider a set of root classes $C_1, \ldots, C_m$ and a time instant $t \in \mathcal{TIME}$, then Ext_i^t, $i = 1, \ldots, m$, denotes the extension of C_i at time t (that is, $C_i.history.ext(t)$) which is the extent of the entire hierarchy rooted at C_i at this time. Since an object cannot migrate over different hierarchies, we require that the sets of objects that have ever belonged to different hierarchies are disjoint (that is, an object cannot belong to different hierarchies even at different times). This is formalized by the following rule.

Rule 7. *Let $C_1, \ldots, C_m$ be the root classes of the ISA hierarchy. For each $i, j \in [1, m]$, $i \neq j$: $\bigcup_{t \in \mathcal{TIME}} Ext_i^t \cap \bigcup_{t \in \mathcal{TIME}} Ext_j^t = \emptyset$.* □

The above rule also implies that given an instant t, the sets of objects belonging to different hierarchies at time t are disjoint.

Finally, the following result holds, relating the subtype relationship to type extensions defined in Section 5.

Theorem 20. *If $T_1 \leq_T T_2$, then $\forall t \in \mathcal{TIME}$, $[\![T_1]\!]_t \subseteq [\![T_2]\!]_t$.*

Note that the above result is important since it states that in T_Chimera the intensional aspect of inheritance (that is, subtyping and substitutability) goes along with the extensional one (that is, inclusion of the extent).

9. Schema and database

In this section we integrate the notions of values, types, objects, classes, and inheritance we have dealt with in the previous sections. T_Chimera, as most data models, distinguishes between the schema level and the instance level. A database schema is a set of classes (signatures and implementations) related by inheritance relationships, as stated by the following definition.

Definition 21. (Schema) *A T_Chimera schema is a 3-tuple $(S, <_{ISA}, I)$ where*

S is a finite set of class signatures satisfying Rules 2, 3 and 4;

$<_{ISA}$ is a partial relation on $S.c$[6] satisfying Rule 1;

I is a finite set of class implementations, such that I includes an implementation, satisfying Rule 5, for each class signature in S. is satisfied.

□

Before defining the notion of database, we identify some constraints imposed on objects and on assignments of objects to classes in a temporal context.

Objects can be defined by means of a set of component objects. An object o is said to *refer to* an object o' at an instant t, if the object identifier of o' appears in one of the attribute values of o. In traditional object-oriented systems, the *referential integrity* property must be enforced to ensure that if an object is in a set, also the objects to which it refers are in the set. Similarly, in our temporal model, we first require that if at instant t an object o refers to an object o', both o and o' exist, that is, instant t belongs to the lifespan of both o and o'. Another important property is the *oid-uniqueness*. The oid of an object represents a time invariant property of the object, shared by no other object in the system. It is analogous to the concept of *essence* introduced in [11]. These notions are formalized by the following definition.

We make use of function ref: $\mathcal{OI} \times \mathcal{T} \rightarrow 2^{\mathcal{OI}}$ that receives as input an object identifier i and a time instant t and returns the set of identifiers of the objects to which the object identified by i refers at time t (cfr. Table 2).

Definition 22. (Consistent Set of Objects) *A finite set of objects OBJ is consistent at time t if and only if the following conditions hold:*

- *oid-uniqueness*
  For each pair of objects o_1 and $o_2 \in$ OBJ: if $o_1.i = o_2.i$, then $o_1.lifespan = o_2.lifespan$, $o_1.v = o_2.v$ and $o_1.class\text{-}history = o_2.class\text{-}history$;
- *referential integrity*
  For all objects $o \in$ OBJ: each object identifier in $ref(o.i, t)$ must be contained in the set $\{i' \mid i' = o.i, o \in \text{OBJ}\}$.

□

Example 14. Referring to objects of Example 9, the set $S = \{\mathtt{i}_1, \mathtt{i}_2, \mathtt{i}_3, \mathtt{i}_4, \mathtt{i}_5, \mathtt{i}_6, \mathtt{i}_7\}$ is consistent, whereas if $\mathtt{i}_4$ were removed from S, the set would be no longer consistent, because it would contain a dangling reference from $\mathtt{i}_1$ to $\mathtt{i}_4$. □

Actually, it is not enough that referred objects exist, rather they must also be of the appropriate type. Therefore, we require each object be consistent with respect to the classes to which it has belonged during its lifespan. In a context where objects can have both static and temporal attributes, the notion of consistency assumes a slightly different semantics with respect to its classical definition. First, the history of the object migration must be taken into account when checking object consistency. The most specific class to which an object belongs can vary over time. Therefore, an object can, at a certain point of its lifespan, be inserted in a more specific subclass or moved up to a more general superclass with respect to the one from which it has been created. Moreover, an object can be an instance of the same class in different, not consecutive, time instants. The migration of an object from a class to another can cause the addition (deletion) of some attributes to (from) the object. For example, consider an employee that is promoted to manager (*manager* being a subclass of *employee* with some extra attributes, like *subordinates* and *official_car*). The promotion of the employee to the manager status has the effect of adding the attributes *subordinates* and *official_car* to the corresponding object. Similarly, the transfer of a manager back to normal employee status results in dropping the attributes *subordinates* and *official_car* from the corresponding object. If the attributes *subordinates* and *official_car* are static, they are simply deleted from the object and no track of their existence is recorded in the object when it migrates to the class *employee*. If they are temporal, some of the values they have assumed before the object migration might be maintained in the object, even if they are not part of the object anymore. These values must be considered when checking object consistency.

Moreover even if an object never migrates from a class to another, it may be characterized by different attributes in different time instants. Indeed, if we consider a time instant t lesser than the current time, we are able to reconstruct only the values of the temporal attributes for which a value at time t has been recorded. For the other temporal attributes, as well as for static attributes, checking the correctness of their value at time t is not possible. Therefore, verifying the consistency of an object in a temporal context requires two steps. First it is necessary to identify, for each instant t of the object lifespan, the set of attributes which characterize the object at time t, that is, the set of attributes for which a value at time t has been recorded. Then, the correctness of their values must be checked. Clearly, if we consider a time instant t lesser than the current time, we are able to identify only the temporal attributes that characterize the object at time t. Therefore, we start by introducing the following definition.

Definition 23. (Meaningful Temporal Attribute) *Let a be a temporal attribute of an object o. Let v be its value. Attribute a is said to be meaningful for o at time t, if t belongs to the domain of v.* □

Note that, as the value of a temporal attribute is a partial function from $\mathcal{TIME}$ to the attribute value domain, its domain is the set of time instants for which the value is defined. Therefore, the domain of the value of a temporal attribute of an object o represents the set of time instants in which the attribute is meaningful for o.

We distinguish two kinds of consistency:

- *Historical consistency.* We say that an object is historically consistent with respect to a class at a given instant t if the values of the temporal attributes meaningful for the object at time t are legal values for the corresponding temporal attributes of the class.
- *Static consistency.* We say that an object is statically consistent with respect to a class if the values of the static attributes of the object are legal values for the static attributes of the class.

Consider an object $o = (i, lifespan, v, class\text{-}history)$, such that $v = (a_1 : v_1, \ldots, a_n : v_n)$. Given an instant $t \in o.lifespan$, the following values are defined:

- *Historical value.* It is a record value representing the values of the temporal attributes meaningful for the object at time t. Let $\{a_k, \ldots, a_m\}$, $1 \leq k \leq m \leq n$, be the subset of $\{a_1, \ldots, a_n\}$, consisting of the names of the temporal attributes meaningful for o at time t. The historical value of o at time t is defined as: $(a_k : v_k(t), \ldots, a_m : v_m(t))$, where $v_i(t)$ denotes the value of a_i at time t, $i = k, \ldots, m$.
- *Static value.* It is a record value representing the values of the static attributes of the object. Its definition is analogous to that of the historical value, considering static attributes instead of temporal ones.

Thus, we can define two functions *h_state*: $\mathcal{OI} \times \mathcal{TIME} \rightarrow \mathcal{V}$ which takes as argument an object identifier and a time instant t, and returns the record representing the historical value of the object at time t, and *s_state*: $\mathcal{OI} \rightarrow \mathcal{V}$, which takes as argument an object identifier, and returns the static value of the object.

Example 15. Referring to the object of Example 9:

$s\_state(\texttt{i}_1) = (objective : \text{`}\texttt{Implementation}\text{'}, workplan : \{\texttt{i}_7\})$
$h\_state(\texttt{i}_1, \texttt{1/1/96}) = (name : \text{`}\texttt{IDEA}\text{'}, budget : \texttt{0.3M}, participants: \{\texttt{i}_2, \texttt{i}_3, \texttt{i}_4\}, leader : null)$
$h\_state(\texttt{i}_1, \texttt{3/10/96}) = (name : \text{`}\texttt{IDEA}\text{'}, subprojects : \{\texttt{i}_5, \texttt{i}_6\}, budget : \texttt{1M}, participants: \{\texttt{i}_2, \texttt{i}_3, \texttt{i}_4\}, leader : null)$.

□

We are now ready to formally introduce the notions of *historical* and *static* consistency, by making use of functions *h_type* and *s_type* introduced in Section 6. Moreover, we introduce function $\Pi_{\{a_1,\ldots,a_m\}}$: $\mathcal{T} \times 2^{\mathcal{AN}} \rightarrow \mathcal{T}$ which takes as argument a record type and a set of attribute names $\{a_1, \ldots, a_m\}$, and returns the record type obtained by dropping components not belonging to $\{a_1, \ldots, a_m\}$ from the input record type.

Definition 24. (Historical Consistency) *Let $o = (i, lifespan, v, class\text{-}history)$ be an object. Let $h\_state(o.i, t) = (a_k : v_k, \ldots, a_m : v_m)$. Object o is historically consistent with respect to a class c' at time t iff $h\_state(o.i, t)$ is a legal value for the type $\Pi_{\{a_k,\ldots,a_m\}}(h\_type(c'))$.* □

Definition 25. (Static Consistency) *An object $o = (i, lifespan, v, class\text{-}history)$ is statically consistent with respect to a class c' iff $s\_state(o.i)$ is a legal value for the type $s\_type(c')$.* □

Checking the consistency of an historical object means verifying its consistency with all the classes to which it belongs during its lifespan. For time instants lesser than the current time, only its historical consistency can be checked. For the current time also the static consistency must be checked. Note that, the consistency of an object with respect to its most specific class ensures the consistency of the object with respect to all the superclasses.

Definition 26. (Object Consistency) *An object $o = (i, lifespan, v, class\text{-}history)$ is consistent iff the following conditions hold:*

- $\forall \langle \tau, c \rangle \in o.class\text{-}history$: interval τ is contained in the lifespan of the class identified by c, that is, $\tau \subseteq \gamma(c).sign.lifespan$;
- $\forall \langle \tau, c \rangle \in o.class\text{-}history$, $\forall t \in \tau$, $t \neq \texttt{now}$: o is historically consistent with respect to class c at time t;

- let $\langle\tau, c\rangle$ be the (unique) element in *o.class-history*, such that $\texttt{now} \in \tau$. Object o is statically consistent with respect to class c.

□

The above definition states that each object, for each instant t of its lifespan, must contain a value for each attribute meaningful for the object at time t, and that this value must be of the correct type. This notion of consistency allows to uniformly handle both static and historical objects. In the case of static objects, Definition 24 reduces to the notion of consistency usually adopted in traditional object-oriented systems [19]. Note moreover that for historical objects with no temporal attributes (but only temporal methods) historical consistency for any $t \neq \texttt{now}$ is trivially satisfied since *h_type* returns the null record type and *h_state* returns the null record value.

Example 16. *The object of Example 9 is consistent iff:*

- $\texttt{i}_7 \in [\![\, task \,]\!]_{now}$;
- $\texttt{i}_5$, $\texttt{i}_6 \in [\![project]\!]_t$, for each $t \in$ `[1/2/96,now]`;
- $\texttt{i}_2$,$\texttt{i}_3$, $\texttt{i}_4 \in [\![person]\!]_t$, for each $t \in$ `[3/5/95,now]`.

□

We are now able to define the notion of database.

Definition 27. (Database) *Let S be a T_Chimera schema. A T_Chimera database over S is a set of objects OBJ such that:*

- $\forall o \in OBJ$: $\langle\tau, c\rangle \in o.class\text{-}history$ if $\forall t \in \tau\ o.i \in \gamma(c).history.proper\text{-}ext(t)$;
- $\forall o \in OBJ$, one of the following condition is satisfied:
  - o is temporal and $o.lifespan = \bigcup_{\langle\tau,c\rangle \in o.class-history} \tau$,
  - o is static and $o.class\text{-}history = \{\langle[\texttt{now},\texttt{now}], c\rangle\}$;
- each object in OBJ is consistent according to Definition 24;
- Rules 6 and 7 are satisfied.

□

10. Method dispatching

In an object system, because of overriding in inheritance hierarchies, a method name may correspond to several implementations. The choice of the actual implementation to execute (referred to as method *dispatching* or method *lookup*) depends on the types of the actual parameters of the method invocation. In presence of subtype relationships, this selection is in general performed at run-time (*late binding*). In this section we revisit the problem of method dispatching in a temporal context, and specify how methods are dispatched in T_Chimera.

Each method invocation is associated with a specific time instant, that is, methods have an additional implicit parameter, in addition to the receiver object *self*, which represents the time of the invocation. The rule for method dispatching is thus modified to take into account also that parameter. If no value is supplied for that argument, it takes the default value `now`. According to the classic late binding mechanism of object-oriented languages, for each method invocation the most specific implementation (that is, the implementation that most closely matches the invocation), must be executed. This principle must be applied at each time instant, taking into account that subclasses may perform a partial (over time) overriding of a method inherited from a superclass. Thus, given a method invocation $o.m(v_1, \ldots, v_n)$ at time t:

- first, the most specific class of object o at time t must be determined; let c be that class,
- then, the method is dispatched to the most specific superclass of c for which an implementation for m at time t is specified.

However, because of multiple inheritance, name clashes may arise, that is, there can be two different superclasses of a class c, not related by the ISA relationship, containing the definition of a method m. In such a situation, the constraint is imposed that in the definition of class c either the method is redefined, or an explicit specification (through a qualification mechanism) is given of the class from which the method is inherited. When searching for the most specific implementation for a method, the ISA hierarchy is ascended by taking into account this information, in case of multiple inheritance.

We express this information as a function $inh\_from : \mathcal{CI} \times \mathcal{MN} \times \mathcal{TIME} \rightarrow \mathcal{CI}$ that, given a class identifier c, a method m of c and a time instant t, returns the direct superclass of c from which class c inherits m at time t, if m is not defined locally to c at that time. $inh\_from(c, m, t)$ is defined as follows:

- if a unique class c' exists such that $c' \in ISA(c)$ and m is defined in c' at time t, then $inh\_from(c, m, t) = c'$;
- otherwise (if more than one class c' exists such that $c' \in ISA(c)$ and m is defined in c' at time t), the resolution of name conflicts for multiple inheritance in T_Chimera requires that in the definition of c an explicit qualification mechanism is used to state from which class m is inherited; if c^q is the qualified class, then $inh\_from(c, m, t) = c^q$.

Finally, let $inh\_from^*_{m,t} \subseteq \mathcal{CI} \times \mathcal{CI}$ denote the transitive closure of the relation $inh\_from_{m,t} = \{(c_1, c_2) \mid c_2 = inh\_from(c_1, m, t)\}$.

Let $IM(m, c)$ denote the set of implementations for method m specified in class c. Method dispatching is then specified by the following dispatching rule.

Rule 8. (Method Dispatching) *Given a method invocation $o.m(v_1,\ldots,v_n)$ at time t, the implementation $impl_i$ in class c^* is selected if*

$$c^* = min_{<_T}\{c \mid c \in \mathcal{CI}, o \in [\![c]\!]_t, \langle c_t^o, \tau\rangle \in o.class\text{-}history, t \in \tau, (c_t^o, c) \in inh\_from^*_{m,t}, \langle \tau_i, impl_i\rangle \in IM(m,c), t \in \tau_i\}$$

□

The following examples clarify the approach.

Example 17. *Consider a class person, with two subclasses student and employee. Consider moreover a method m defined in class person, and redefined in both subclasses. Suppose that in each class a single implementation for m is provided, that refers to the entire lifespan of the class. Finally, consider an object o, with* $lifespan = $ `[10/8/69,now]` *and*

$class\text{-}history = \{\langle$`[10/8/69,9/30/76]`$, person\rangle,$
$\langle$`[10/1/76,4/15/93]`$, student\rangle,$
$\langle$`[4/16/93,4/30/93]`$, student\rangle,$
$\langle$`[5/1/93,now]`$, employee\rangle\}$

Depending on the most specific class of o at the time of the invocation, a different implementation of m is chosen. For instance, at `6/11/88:16:00` *the implementation for m in class student is selected.* □

Example 18. *Given the class project of Example 4, whose implementation for method balance is specified in Example 7, consider the subclass research_project with the following implementation for method balance:*

{ ⟨[6/1/95, 9/30/95], `balance(B)`: {`B` = (`self.budget`− `self.participants` ∗ 1,000) ∗ 0.15 →}⟩ }.

Then, an invocation $o.balance()$ on an object $o \in [\![research\_project]\!]_{t'}$ for any $t' \in$ `[1/1/95,now]`*, is dispatched as follows:*

- if $t \in$ `[1/1/95,5/31/95]` the first rule of class *project* is executed;
- if $t \in$ `[6/1/95,9/30/95]` the rule of class *research_project* is executed;
- if $t \in$ `[10/1/95,12/31/95]` the first rule of class *project* is executed;
- if $t \in$ `[1/1/96,now]` the second rule of class *project* is executed.

□

According to the Rule 8 each *type correct* method invocation is unambiguously dispatched, that is, the rule selects one and only one method. We first formalize the notion of type correct method invocation as follows. Note that, the following definition refers to a static method invocation, therefore we denote with X a static object reference.

Definition 28. (Type Correct Method Invocation) *An invocation of method m on an object reference X of type T at time t is type correct if $\exists T'$, $T \leq_T T'$ such that $M(m,T') \neq \emptyset$ and $t \in \gamma(T).sign.lifespan$.* □

Note that, because of the *covering condition* (Rule 5), $M(m,T') \neq \emptyset$ ensures that for any time instant for which T' is defined, an implementation for m is available.

Therefore, the following result holds.

Theorem 29. *Given a type correct method invocation $o.m(v_1,\ldots,v_n)$, Rule 8 selects one and only one implementation for m.*

11. Conclusions and future work

In this paper we have presented T_Chimera, an extension of the Chimera data model incorporating temporal capabilities. We have introduced the notion of temporal type to handle in a uniform way temporal and non-temporal domains, and we have formally defined the notions of T_Chimera schema and database. We have discussed the notions of object consistency and equality and we have also investigated problems related to inheritance and method dispatching in a temporal framework.

We plan to extend this work in several directions. First, we are interested in investigating temporal object references and, more generally, issues related to the query language and its typing. In particular, we plan to focus on the navigational approach to query object-oriented databases and show how this approach is extended to a temporal context. A related topic of future research concerns the management of incomplete temporal information. Since we do not store all the past values of attributes, it may happen that an object access expression denotes a value which is not available in the database. Indeed, each object access expression is associated with a time instant, denoting the time for which the corresponding object snapshot value must be accessed. If no time is specified, the current instant `now` is considered as default. While for temporal attributes with null snapshot conditions any time instant can be specified, for static attributes only the current value can be accessed and for temporal attributes with non null snapshot conditions, only the value at some time instants can be retrieved. To supply a value also for the time instants for which a value is not stored, we plan to exploit the semantic assumptions proposed in [6]. An assumption is a way to derive certain implicit data from the explicit data stored in the database. Such semantic assumption allows, for example, to specify that the value of an attribute is valid till a new value for that attribute is stored.

Our current approach is that whenever an expression denotes an instant for which no corresponding attribute value is stored in the database, a `null` value is returned. We are currently investigating [5] the static restrictions that could be imposed on object accesses for limiting these situations. The idea is that for object o at time t only attributes meaningful for o at t can be accessed. It is, however, not possible to statically detect all the situations in which a non-available past attribute value is accessed. An example is object accesses in method bodies for which the time instant is not known, since it is a parameter of the invocation. Thus, whenever such a situation arises, returning a null value seems the most intuitive behavior.

Moreover, we plan to extend other features of the Chimera model like triggers and deductive rules with time. In particular, temporal triggers have not been much investigated; issues such as termination and confluence will need to be revisited when dealing with temporal triggers. We also plan to define a temporal integrity constraint language for the T_Chimera data model. In this paper we have only dealt with integrity constraints which are implicit in the data model (such as referential integrity, or attribute domains); however, a data model should also provide a general language for supporting the specification of user-defined constraints. Such a language would allow, among other things, to express constraints based on past histories of objects.

The only kind of evolution currently supported by T_Chimera is object evolution, that is, object creations, deletions, migrations and state changes. The extension of T_Chimera to support also schema updates and schema versioning [13,21] is another topic of future research.

Notes

1. A `null` snapshot condition is associated with immutable and static attributes.
2. Here and in the following we omit empty snapshot conditions.
3. Actually, in T_Chimera method implementations may be expressed in an external programming language. However, we consider here only implementations expressed in the T_Chimera language itself.
4. T_Chimera terms are either variables or values (except oids) or path expressions (built with the use of the dot notation). Moreover, a number of terms obtained using classical predefined operators for integers, reals, sets and lists are included.
5. For historical objects containing only temporal attributes, the *snapshot* function coincides with the *h_state* function (cfr. Table 2).
6. By abuse of notation, given a set of class signatures S we denote as $S.c$ the set $\{C.c \mid C \in S\}$.

References

1. Abiteboul, S., Hull, R. and Vianu, V. (1995). *Foundations of Databases.* Addison-Wesley.
2. Balsters, H. and Fokkinga, M. (1991). Subtyping can have a Simple Semantics. *Theoretical Computer Science*, 87:81–96.
3. Beeri, C. (1989). Formal Models for Object Oriented Databases. In W. Kim et al., editors, *Proc. First Int'l Conf. on Deductive and Object-Oriented Databases*, pages 370–395.
4. Bertino, E., Ferrari, E. and Guerrini, G. (1996). A Formal Temporal Object-Oriented Data Model. In P. Apers et al., editors, *Proc. Fifth Int'l Conf. on Extending Database Technology (EDBT'96)*, pages 342–356.
5. Bertino, E., Ferrari, E. and Guerrini, G. (1996). Navigational Accesses in a Temporal Object Model. Technical Report TR 169-96, Department of Computer Science, University of Milano. Submitted for publication. Available via anonymous ftp from `ftp.disi.unige.it` as `/pub/guerrini/reports/tr-dsi-169-96.ps`.
6. Bettini, C., Wang, X.S., Bertino, E. and Jajodia, S. (1995). Semantic Assumptions and Query Evaluation in Temporal Databases. In *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, pages 257–268.
7. Buchheit, M., Jeusfeld, M., Nutt, W. and Staudt, M. (1994). Subsumption Between Queries to Object-Oriented Databases. In M. Jarke, J. Bubenko, and K. Jeffery, editors, *Proc. Fourth Int'l Conf. on Extending Database Technology*, pages 15–22. Extended version in *Information Systems*, 19(1).
8. Cardelli, L. (1988). Types for Data Oriented Languages. In J.W. Schmidt, S. Ceri, and M. Missikoff, editors, *Proc. First Int'l Conf. on Extending Database Technology*, pages 1–15.
9. Chan, E. (1992). Containment and Minimization of Positive Conjunctive Queries in OODBMs. In *Proc. of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 202–211.
10. Cheng, T. and Gadia, S. (1993). An Object-Oriented Model for Temporal Databases. In *Proc. of the Int'l Workshop on an Infrastructure for Temporal Databases.*
11. Clifford, J. and Crocker, A. (1988). Objects in Time. In *Proc. Fourth IEEE Int'l Conf. on Data Engineering*, pages 11–18.
12. Clifford, J. and Crocker, A. (1993). The Historical Relational Data Model (HRDM) Revisited. In A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors, *Temporal Databases: Theory, Design, and Implementation*, pages 6–26. Benjamin/Cummings.
13. DeCastro, C., Grandi, F. and Scalas, M. R. (1995). On Schema Versioning in Temporal Databases. In *Proc. Workshop on Temporal Databases*, pages 272–291.
14. Gadia, S. and Nair, S. (1993). Temporal Databases: a prelude to Parametric Data. In A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors, *Temporal databases: Theory, Design, and Implementation*, pages 28–66. Benjamin/Cummings.
15. Goralwalla, I. and Özsu, M. (1993). Temporal Extensions to a Uniform Behavioral Object Model. In E. Elmasri, V. Kouramajian, and B. Thalheim, editors, *Proc. Twelfth Int'l Conf. on the Entity-Relationship Approach*, pages 110–121.
16. Guerrini, G., Bertino, E. and Bal, R. (1994). A Formal Definition of the Chimera Object-Oriented Data Model. Technical Report IDEA.DE.2P.011.01, ESPRIT Project 6333. Submitted for publication. Available via anonymous ftp from `ftp.disi.unige.it` as `/pub/guerrini/chimera/model.ps`.
17. Gunter, C. and Mitchell, J. (1994). *Theoretical Aspects of Object-Oriented Programming.* MIT Press.
18. Käfer, W. and Schöning, H. (1992). Realizing a Temporal Complex-Object Data Model. In M. Stonebraker, editor, *Proc. of the ACM SIGMOD Int'l Conf on Management of Data*, pages 266–275.
19. Lecluse, C., Richard, P. and Velez, F. (1990). 0_2, an Object-Oriented Data Model. In F. Bancilhon and P. Buneman, editors, *Advances in Database Programming Language*, pages 257–276. Addison-Wesley.

20. Pissinou, N. and Makki, K. (1992). A Framework for Temporal Object Databases. In *Proc. First Int'l Conf. on Information and Knowledge Management*.

21. Roddick, J. F. (1995). A Survey of Schema Versioning Issues for Database Systems. *Information and Software Technology*, 37:383–393.

22. Rose, E. and Segev, A. (1991). TOODM - A Temporal Object-Oriented Data Model with Temporal Constraints. In *Proc. Tenth Int'l Conf. on the Entity-Relationship Approach*, pages 205–229.

23. Snodgrass, R. (1995). Temporal Object-Oriented Databases: A Critical Comparison. In W. Kim, editor, *Modern Database Systems: The Object Model, Interoperability and Beyond*. Addison-Wesley/ACM Press.

24. Snodgrass, R. (1995). *The TSQL2 Temporal Query Language*. Kluwer Academic Publisher.

25. Su, S. and Chen, H. (1991). A Temporal Knowledge Representation Model OSAM*/T and its Query Language OQL/T. In G.M. Lohman, A. Sernadas, and R. Camps, editors, *Proc. 17th Int'l Conf. on Very Large Data Bases*, pages 431–441.

26. Tansel, A. (1996). Temporal Relational Data Model. *IEEE Transactions on Knowledge and Data Engineering*, to appear.

27. Tansel, A., Clifford, J., Gadia, S., Jajodia, S., Segev, A. and Snodgrass, R. (1993). *Temporal Databases: Theory, Design, and Implementation*. Database Systems and Applications Series. Benjamin/Cummings.

28. Tsotras, V.J. and Kumar, A. (1996). Temporal Database Bibliography Update. *SIGMOD-RECORD*, 25.

29. Wuu, G. and Dayal, U. (1993). A Uniform Model for Temporal and Versioned Object-Oriented Databases. In A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors, *Temporal Databases: Theory, Design, and Implementation*, pages 230–247. Benjamin/Cummings.

Appendix: A: Snapshot language

In this appendix we first give the BNF syntax of the language provided by T_Chimera to express snapshot conditions, and then give rules for snapshot conditions refinement. Non terminal symbols `value`, and `set_of_time_instants` represent elements of the domains $\mathcal{V}$ and $2^{\mathbb{N}}$, respectively, whereas `meth_inv` denotes a method invocation, that is, `method_name(list_of_parameters)`.

```
⟨snapshot_cond⟩  ::= ⟨c_snap_cond⟩ OR ⟨c_snap_cond⟩ |
                     ⟨c_snap_cond⟩

⟨c_snap_cond⟩    ::= ⟨update_expr⟩ | ⟨set_of_time_instants⟩ |
                     ⟨value_expr⟩ |
                     | ⟨c_snap_cond⟩ AND ⟨c_snap_cond⟩

⟨update_expr⟩    ::= generalize| specialize| create| delete

⟨value_expr⟩     ::= ⟨simple_expr⟩ ⟨comp_op⟩ ⟨simple_expr⟩ |
                     ⟨simple_expr⟩ ⟨comp_op⟩ ⟨value⟩

⟨simpl_expr⟩     ::= attribute_name | method_inv |
                     ⟨path_expr⟩ |
                     count(⟨simpl_expr⟩) |
                     ⟨simpl_expr⟩ ⟨op⟩ ⟨simpl_expr⟩

⟨path_expr⟩      ::= class_name.attribute_name |
                     class_name.method_inv |
                     class_name.⟨path_expr⟩

⟨comp_op⟩        ::= > | < | ≥ | ≤ | = | ≠ | ∈ | ∉

⟨op⟩             ::= + | - | * | / | ∪ | ∩ | \
```

Note that, according to the above syntax, snapshot conditions are in disjunctive normal form. This particular form is chosen for ease of refinement checking.

Snapshot conditions refinement

Let sc_1 and $sc_2 \in \mathcal{SC}$ be snapshot conditions, sc_2 is a *refinement* of sc_1 ($sc_2 \preceq_s sc_1$) if the following conditions hold:

- $sc_1 = e_1 \ OR \ldots OR \ e_n,\ n \geq 1$
- $sc_2 = e'_1 \ OR \ldots OR \ e'_m,\ m \geq 1$
- $\forall i \in [1,n], \exists j \in [1,m]$ such that $e'_j \propto e_i$

where $e \propto e'$ is a containment relationship among conjunctions of atomic expressions that can appear in a snapshot condition, expressing the fact that e is true each time e' is. In particular, $e \propto e'$ holds if:

- e, e' are `update_expr` and $e = e'$;
- e, e' are `set_of_time_instants` and $e' \subseteq e$;
- e, e' are `value_expr` and e contains e' according to the usual notion of containment (or subsumption) between formulae [7,9];
- $e = e_1 \ AND \ldots AND \ e_n,\ n \geq 1,\ e' = e'_1 \ AND \ldots AND \ e'_m,\ m \geq 1$, and $\forall i \in [1,m], \exists j \in [1,n]$ such that $e_j \propto e'_i$.

Appendix: B: Proofs

Proof of Theorem 1.

The theorem is proved by induction on the derivation tree for $v :^t T$.

Basis. We distinguish four cases, depending on the typing rule used in the tree.

- Rule R 1:
  if we deduce type T for v by R 1, then $v = null$. For the first item in Definition 6, $null \in [\![T]\!]_t$, $\forall T \in \mathcal{T}\ \forall t \in \mathcal{TIME}$
- Rule 2: $T \in \mathcal{BVT}$
  $v :^t T$ can be deduced by R 2 if $v \in dom(T)$. Then, for the second item in Definition 6, $v \in [\![T]\!]_t$, since $[\![T]\!]_t = dom(T)$
- Rule R 3: $T = time$
  $v :^t T$ can be deduced by R 3 if $v \in \mathcal{TIME}$. Then, for the third item in Definition 6, $v \in [\![T]\!]_t$, since $[\![time]\!]_t = \mathcal{TIME}$
- Rule R 4: $T \in \mathcal{CI}$
  $v :^t T$ can be deduced by R 4 if $v \in \pi(T,t)$. Then, for the fourth item in Definition 6, $v \in [\![T]\!]_t$, since $[\![T]\!]_t = \pi(T,t)$.

Inductive Step. We distinguish four cases, depending on the typing rule used in the tree.

- Rule R 5: $T = set\text{-}of(T')$, $T' = \bigsqcup_{i=1}^{n} T_i$
  $v :^t T$ can be deduced by R 5 if $v_i :^t T_i, i = 1, \ldots, n$ has been deduced. Then, by inductive hypothesis, $v_i \in [\![T_i]\!]_t$. We recall the following lemma, proved in [16]: if $T = T_1 \sqcup T_2$ then, $\forall t \in \mathcal{TIME}$, $[\![T_1]\!]_t \subseteq [\![T]\!]_t$ and $[\![T_2]\!]_t \subseteq [\![T]\!]_t$. Then, since $[\![T_i]\!]_t \subseteq [\![T']\!]_t$:, $v_i \in [\![T']\!]_t$, $i = 1, \ldots n$. Hence, $\{v_1, \ldots, v_n\} \in 2^{[\![T']\!]_t}$ and thus $\{v_1, \ldots, v_n\} \in [\![T]\!]_t$ for the fifth item in Definition 6
- Rule R 6: $T = list\text{-}of(T')$, $T' = \bigsqcup_{i=1}^{n} T_i$
  $v :^t T$ can be deduced by R 6 if $v_i :^t T_i, i = 1, \ldots, n$ has been deduced. Then, by inductive hypothesis, $v_i \in [\![T_i]\!]_t$. Since $[\![T_i]\!]_t \subseteq [\![T']\!]_t$, $v_i \in [\![T']\!]_t$, $i = 1 \ldots n$. Hence, $[v_1, \ldots, v_n] \in [\![T]\!]_t$ for the sixth item in Definition 6
- Rule R 7: $T = record\text{-}of(a_1 : T_1, \ldots, a_n : T_n)$
  $v :^t T$ can be deduced by R 7 if $v_i :^t T_i, i = 1, \ldots n$ has been deduced. Then, by inductive hypothesis, $v_i \in [\![T_i]\!]_t$. Hence, $(a_1 : v_1, \ldots, a_n : v_n) \in [\![T]\!]_t$ for the seventh item in Definition 6
- Rule R 8: $T = temporal(T')$
  $v :^t T$ can be deduced by R 8 if $t_i :^t time$ and $v_i :^{t_i} T$, $i = 1, \ldots, n$ has been deduced. Then, by inductive hypothesis, $t_i \in [\![time]\!]_t = \mathcal{TIME}$ and $v_i \in [\![T]\!]_{t_i}$; $v = \{(t_1, v_1), \ldots, (t_n, v_n)\}$ is then the punctual representation of a function $f : \mathcal{TIME} \rightarrow \bigcup_{t' \in \mathcal{TIME}} [\![T]\!]_{t'}$. Moreover, $\forall i \in [1, n] : v_i = f(t_i) \in [\![T]\!]_{t_i}$ and then $v \in [\![T]\!]_t$ for the eighth item in Definition 6.

■

Proof of Theorem 2.
The theorem is proved by induction on the structure of type T.

Basis. We distinguish four cases

- $v = null$
  Rule R 1 allows to deduce $null :^t T$, $\forall T \in \mathcal{T}$ $\forall t \in \mathcal{TIME}$
- $T \in \mathcal{BVT}$
  for the second item in Definition 6, $v \in [\![T]\!]_t$ if $v \in dom(T)$. Then, $v :^t T$ can be deduced by R 2
- $T = time$
  for the third item in Definition 6, $v \in [\![time]\!]_t$ if $v \in \mathcal{TIME}$. Then, $v :^t T$ can be deduced by R 3
- $T \in \mathcal{CI}$
  for the fourth item in Definition 6, $v \in [\![T]\!]_t$ if $v \in \pi(T, t)$. Then, $v :^t T$ can be deduced by R 4

Inductive step. We distinguish four cases, depending on the constructor on which type T is built.

- $T = set\text{-}of(T')$
  for the fifth item in Definition 6, $v \in [\![T]\!]_t$ if $v \in 2^{[\![T']\!]_t}$. Then, $v = \{v_1, \ldots, v_n\}$ and $v_i \in [\![T']\!]_t, i = 1, \ldots n$. Thus, by inductive hypothesis, $v_i :^t T', i = 1 \ldots n$. Hence, $v :^t T$ can be deduced by R 5
- $T = list\text{-}of(T')$
  for the sixth item in Definition 6, $v \in [\![T]\!]_t$ if $v = [v_1, \ldots, v_n]$ and $v_i \in [\![T']\!]_t, i = 1, \ldots n$. Then, by inductive hypothesis, $v_i :^t T', i = 1, \ldots, n$. Hence, $v :^t T$ can be deduced by R 6
- $T = record\text{-}of(a_1 : T_1, \ldots, a_n : T_n)$
  for the seventh item in Definition 6, $v \in [\![T]\!]_t$ if $v = (a_1 : v_1, \ldots, a_n : v_n)$ and $v_i \in [\![T_i]\!]_t, i = 1, \ldots, n$. Then, by inductive hypothesis, $v_i :^t T_i, i = 1, \ldots n$. Hence, $v :^t T$ can be deduced by R 7
- $T = temporal(T')$
  for the eighth item in Definition 6, $v \in [\![T]\!]_t$ if v is a partial function $f : \mathcal{TIME} \rightarrow \bigcup_{t' \in \mathcal{TIME}} [\![T]\!]_{t'}$, such that $f(t_i) \in [\![T]\!]_{t_i}$, for each t_i for which $f(t_i)$ is defined. If we see f expressed as a set of pairs $\{(t_1, v_1), \ldots, (t_n, v_n)\}$, then, by inductive hypothesis, $t_i :^t time$, $v_i :^{t_i} T$, $i = 1, \ldots, n$. Hence, $v :^t T$ can be deduced by R 8.

■

Proof of Theorem 3.
The theorem is proved by induction on the structure of types T_1 and T_2.

Basis. We distinguish two cases, corresponding to the basis of Definition 5

- $T_1 = T_2$
  if $T_1 = T_2$ then, $\forall t \in \mathcal{TIME}$ $[\![T_1]\!]_t = [\![T_2]\!]_t$ and thus, it is trivial to show $[\![T_1]\!]_t \subseteq [\![T_2]\!]_t$
- $T_1, T_2 \in \mathcal{OT}, T_1 <_{ISA} T_2$
  $[\![T_1]\!]_t \subseteq [\![T_2]\!]_t$, $\forall t \in \mathcal{TIME}$ follows from Rule 6 since, by Definition 6, $[\![T_i]\!]_t = \pi(T_i, t) = C_i.history.ext(t)$, being $C_i.sign.c = T_i$, $i = 1, 2$

Inductive step. We can distinguish four cases, corresponding to the inductive cases of Definition 5.

- $T_1 = set\text{-}of(T_1')$, $T_2 = set\text{-}of(T_2')$, $T_1' \leq_T T_2'$
  by inductive hypothesis, $[\![T_1']\!]_t \subseteq [\![T_2']\!]_t$, $\forall t \in \mathcal{TIME}$. Thus, $[\![T_1]\!]_t \subseteq [\![T_2]\!]_t$ follows since $[\![T_i]\!]_t = 2^{[\![T_i']\!]_t}$, $i = 1, 2$ (fifth item of Definition 6)
- $T_1 = list\text{-}of(T_1')$, $T_2 = list\text{-}of(T_2')$, $T_1' \leq_T T_2'$
  by inductive hypothesis, $[\![T_1']\!]_t \subseteq [\![T_2']\!]_t$, $\forall t \in \mathcal{TIME}$. Thus, $[\![T_1]\!]_t \subseteq [\![T_2]\!]_t$ follows since $[\![T_i]\!]_t = \bigcup_{n \in \mathbb{N}^+} \{[v_1, \ldots, v_n] \mid v_j \in [\![T_i]\!]_t, j = 1, \ldots, n\}$, $i = 1, 2$ (sixth item of Definition 6)
- $T_1 = record\text{-}of(a_1 : T_1', \ldots, a_n : T_n')$, $T_2 = record\text{-}of(a_1 : T_1'', \ldots, a_n : T_n'')$, and $\forall i \in [1, n]$: $T_i' \leq_T T_i''$
  by inductive hypothesis, $\forall t \in \mathcal{TIME}$ and $\forall i \in [1, n]$: $[\![T_i']\!]_t \subseteq [\![T_i'']\!]_t$. Thus, $[\![T_1]\!]_t \subseteq [\![T_2]\!]_t$ follows since, by the fifth item of Definition 6, $[\![T_1]\!]_t = \{(a_1 : v_1, \ldots, a_n : v_n) \mid a_i \in \mathcal{AN}, v_i \in [\![T_i']\!]_t, i = 1 \ldots n\}$ and $[\![T_2]\!]_t = \{(a_1 : v_1, \ldots, a_n : v_n) \mid a_i \in \mathcal{AN}, v_i \in [\![T_i'']\!]_t, i = 1 \ldots n\}$

- $T_1 = temporal(T'_1)$, $T_2 = temporal(T'_2)$, $T'_1 \leq_T T'_2$ by inductive hypothesis, $[\![T'_1]\!]_t \subseteq [\![T'_2]\!]_t$, $\forall t \in \mathcal{TIME}$. Thus, $\bigcup_{t' \in \mathcal{TIME}} [\![T'_1]\!]_{t'} \subseteq \bigcup_{t' \in \mathcal{TIME}} [\![T'_2]\!]_{t'}$, and $\{f \mid f : \mathcal{TIME} \rightarrow \bigcup_{t' \in \mathcal{TIME}} [\![T'_1]\!]_{t'}\} \subseteq \{f \mid f : \mathcal{TIME} \rightarrow \bigcup_{t' \in \mathcal{TIME}} [\![T'_2]\!]_{t'}\}$. Then, $[\![T_1]\!]_t \subseteq [\![T_2]\!]_t$ follows from the eighth item of Definition 6.

■

Proof of Theorem 4.

Consider the static method invocation $X.m(X_1, \ldots, X_n)$, where X is an object reference with type T. Since, by hypothesis, the invocation is type correct (cfr Definition 26) there exists T', $T \leq_T T'$, such that $IM(m, T') \neq \emptyset$. Consider now the dynamic invocation $o.m(v_1, \ldots v_n)$ at time t. o is a legal value for the reference X, if there exists $\langle \tau, c^o_t \rangle \in o.class\text{-}history$, such that $t \in \tau$ and $c^o_t \leq_T T$. Then, $\{c \mid c \in \mathcal{CI}, o \in [\![c]\!]_t\} \neq \emptyset$, since this set contains at least c^o_t.

Moreover, since $c^o_t \leq_T T$, the transitivity of $\leq_T$ implies that there exists T', $c^o_t \leq_T T'$ such that $IM(m, T') \neq \emptyset$. Let T^* be the most general (that is, the maximum with respect to $<_T$) superclass from which c^o_t inherits method m at time t. By Definition 6, $o \in [\![T^*]\!]_t$. Moreover, by the definition of relation $inh\_from^*_{m,t}$, $(c^o_t, T^*) \in inh\_from^*_{m,t}$. Finally, from the definition of relation $inh\_from^*_{m,t}$, follows that $\exists \langle \tau_i, rule_i \rangle \in IM(m, T^*)$ such that $t \in \tau_i$.

Thus, $T^* \in \{c \mid c \in \mathcal{CI},\ o \in [\![c]\!]_t,\ \langle c^o_t, \tau \rangle \in o.class\text{-}history,\ t \in \tau,\ (c^o_t, c) \in inh\_from^*_{m,t},\ \langle \tau_i, impl_i \rangle \in IM(m, c),\ t \in \tau_i\}$. Therefore, at least an implementation can be selected.

Let us prove uniqueness. Let $S = \{c \mid c \in \mathcal{CI},\ o \in [\![c]\!]_t,\ \langle c^o_t, \tau \rangle \in o.class\text{-}history,\ t \in \tau,\ (c^o_t, c) \in inh\_from^*_{m,t},\ \langle \tau_i, impl_i \rangle \in IM(m, c),\ t \in \tau_i\}$. The selected class is defined as the minimum wrt $\leq_T$ of the set S. We have just proved that $S \neq \emptyset$. Classes in S are totally ordered with respect to $<_{ISA}$ since they are all elements of the ascending path in the inheritance graph from c^o_t to class T^* specified above, induced by the $inh\_from_{m,t}$ relation. Thus, $min_{<_T} S$ is unique. ■