J. King Saud Univ., Vol. 11, *Comp. & Info. Sci.*, pp. 25-43 (A.H. 1419/1999)

Share-kno: Knowledge Sharing Mechanism of the Temporal Object System

Abad Shah[*], Farshad Fotouhi[**] and William Grosky[**]
[*]*Computer Science Department, King Saud University*
P.O. Box 51178, Riyadh 11543, Saudi Arabia
[**]*Computer Science Department, Wayne State University*
Detroit Michigan, 48202 U.S.A

(Received 08 March 1997; accepted for publication 03 June 1997)

Abstract. Temporal Object System (TOS) handles the structural and stature changes to an object in a uniform and temporal fashion. The TOS takes a hybrid approach of the class-based and prototype-based approaches in modeling the real-word objects and knowledge sharing among the objects. This hybrid approach makes it flexible than the two approaches, i.e., the class-based approach and the prototype-based approach. In this paper, we propose a model of knowledge sharing mechanism, *Share-kno* (**SK**) for object hierarchy of the TOS. We also formally prove that the *Share-kno* (**SK**) model encapsulates *more knowledge* than the inheritance hierarchy and delegation hierarchy models proposed by Stein.

Keywords: Object-oriented Databases; Temporal Database; Temporal Objects; Knowledge Sharing Mechanism; Inheritance; Delegation, Modeling.

1. Introduction

In object-oriented paradigm, an object is defined by the two parameters, *structure* and *state*. The *structure* (*SR*) of an object provides the structural and behavioral capabilities to the object, which is defined by a set of instance variables and methods. The *state* (*ST*) of the object assigns data values to the instance variables of the object, and its methods operate on them. In the object-oriented paradigm there are two approaches, the class-based approach and prototype-based approach, to model and share knowledge of objects of a universe of discourse [4], [11]. We define the *knowledge* of an object in the class-based approach as the structure of the object. But in general, we define the *knowledge* of

an object as the structure, the state, or a combination of both. The first approach (i.e., the class-based approach) is based on the mathematical concept of *set,* a set of objects sharing the same structure is referred to as a *class* [11]. Using this approach an object-oriented database is a collection of classes which are organized as a directed acyclic graph (DAG), and it is referred to as *class-lattice* or *class-hierarchy.* The classes in a class-lattice hold a *parent-child* relationship. The child class inherits the structure from its parent class, and this knowledge sharing mechanism through the parent-child relationship is referred to as the *inheritance.* The parent class and child class are referred to as *superclass* and *subclass,* respectively.

The prototype-based approach is not very common. This approach uses a different technique for modeling objects of a universe of discourse, for organizing its objects in a hierarchy, and to share knowledge among the objects. This approach differs from the class-based approach because it forges the structure and the state of an object into a single entity that is referred to as a *prototype* [4], [11], [19]. In this approach an object is also called a prototype. A prototype is defined by its default knowledge [4], [11]. A prototype can consist of structure, state, or both, and the prototype is a complete unit of an object's knowledge [4], [11]. New objects can be created by sharing knowledge of one or more existing prototypes and defining additional knowledge (structure, state, or both). The prototype-based approach is a classless approach, all objects (prototypes) are at the same level [4], [11]. There are two techniques which are used in the approach to create a new prototype by sharing the knowledge of existing prototypes. The first technique is the *copying* (or *cloning*) of existing prototypes into the new prototype. After *copying* knowledge from an existing prototype, the new prototype has no connection with the copied prototype. Later, changes to the copied prototype are not reflected in the new prototype and vice versa [4], [5], [11], [20]. In the second technique, a new prototype is created as an *offspring* of one or more existing prototypes. In this technique, a new prototype is defined in incremental fashion as an extension of an existing prototype (or prototypes) by defining additional knowledge in the new prototype. The additional knowledge is the difference in knowledge of the existing prototype (or prototypes) and the new prototype [5], [11], [19]. In the prototype-based approach, a group of objects (or prototypes) which share a common knowledge, can be grouped together by placing the common knowledge at some common place (as a prototype). New prototypes are created as the offsprings of the common knowledge [3], [11], [19], [20]. The structure and state of an object are considered the type of entities in this approach unlike the class-based approach.

The knowledge sharing mechanism of the prototype-based approach is very simple. When a message is received in a prototype - *self,* it first tries to answer the message from its local knowledge. If it could not answer from its local knowledge, then it forwards the message to another object (prototype) - *client.* This process of message forwarding continues until the message is replied to or an error message is generated. The knowledge sharing mechanism of this approach is achieved through the process of

message forwarding, and it is referred to as *delegation* [4], [11]. The delegation mechanism is considered more flexible and powerful than the inheritance mechanism [2], [3], [11], [20].

The class-based approach and object-oriented paradigms have become almost synonymous in the computer science community, and many people think of every object-oriented system in terms of the class-based approach. The source for this misconception is an early programming language *Simula-67* which used the class-based approach. But in fact the prototype-based approach is simpler, and more flexible and powerful than the class-based approach [2], [3], [11], [20]. A complete comparison of both approaches is available in [4].

In the existing object-oriented database systems, changes in the state of an object are maintained via *version management* [1]. Also, the structural changes are supported in most of object-oriented database systems. Such changes to a class are referred to as *schema evolution* in the literature [14]. Current object-oriented database systems keep only the current version of each class structure. *After* any change, it is necessary to reload a previous version of the database to retrieve any information from the previous version of a class structure.

In [9],[10], [16] we introduced a temporal object system (TOS) which maintains the history of changes to both the structure and the state of an object in a consolidated and elegant manner. We associated time with both structure and state of an object. Such an object is referred to as a *temporal object.* A temporal object evolves over time by changing its state, structure, or a combination of the two. A set of temporal objects which share a common knowledge (i.e., structure, state, a combination of the two) is referred to as a *family.* The TOS also facilitates the construction of a *complex family* which is an aggregation of temporal objects from various families. The objects in a complex family are referred to as *temporal complex objects* [10], [17]. A complex family increases the knowledge sharing among non-homogeneous temporal objects and their transportability. The temporal object system (TOS) is a collection of families which are defined at different time instances.

In this paper, we propose a hierarchy model of *Share-kno* (**SK**) for temporal objects of the TOS, and formally prove that the hierarchy model **SK** encapsulates *more knowledge* (this term is defined later) of an object than the inheritance hierarchy and delegation hierarchy models which are proposed by Stein in [19]. The remainder of this paper is organized as follows: In Section 2, we briefly describe the TOS, and give an example of the TOS to elaborate the basic elements of the TOS. Section 3 discusses inheritance hierarchy and delegation hierarchy models proposed by Stein in [19]. In Section 4 we give the Share-kno hierarchy model **SK** for temporal objects of the TOS. In Section 5 we formally prove that the hierarchy model **SK** is *more knowledgeable* than the Stein's inheritance hierarchy and delegation hierarchy models. Finally, in Section 6

we give our concluding remarks and future work directions. In this paper, we assume that readers are familiar with the common terminology of object-oriented paradigm, works of Lieberman and Stein [11], [19], and mathematical notations and conventions used by Stein in [19].

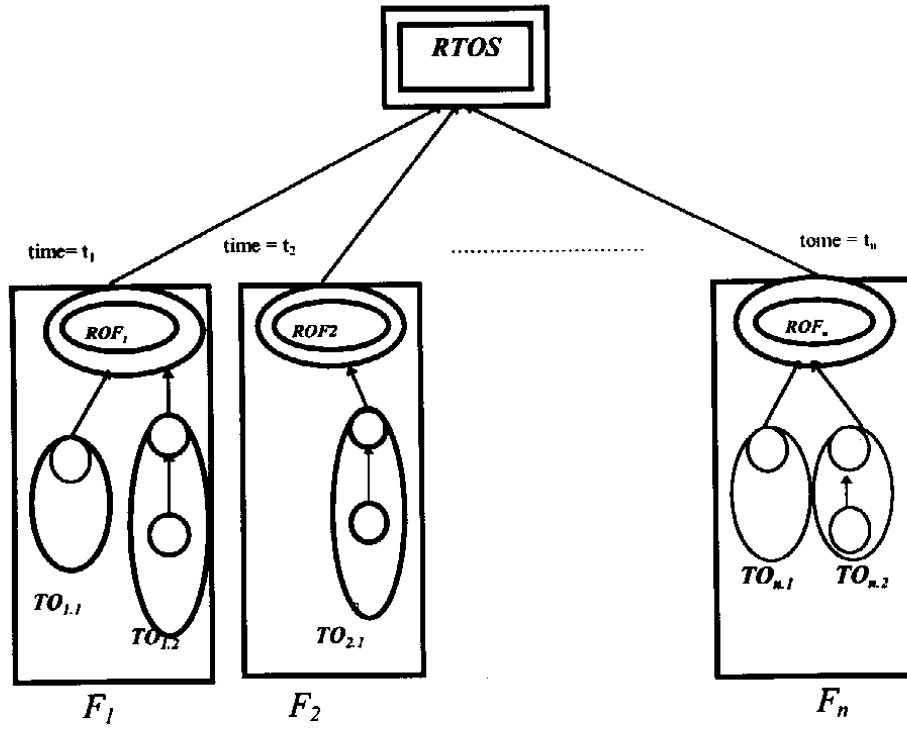


Fig. 1. A general schema of temporal object system (TOS).

2. Temporal Object System (TOS)

As we have already mentioned in the previous section that the Temporal Object System (TOS) is defined as a collection of families which are defined at different time instances. A family is a collection of temporal objects, and a temporal object is a collection of *stages* (for details see [9],[16]). Temporal object and stage are formally defined in Section 2.1. Fig. 1 shows a general schema of the TOS, where *RTOS* represents the *root node* of the system with n number of families, i.e., $F_1, F_2, \ldots, F_n$ as its children.

In all figures of this paper, double rectangle, double oval, rectangle, single oval, and circle represent Root of the TOS (*RTOS*), root-of-family (*ROF*), family, temporal object, and stage, respectively. Single arrow in a temporal object represents a structure

and /or state change to the temporal object. The single arrow also represents inheritance link between a temporal object and an *ROF*, and between a family and the *RTOS* (see Fig. 1). The terms *ROF*, family and stage are formally defined in the next section.

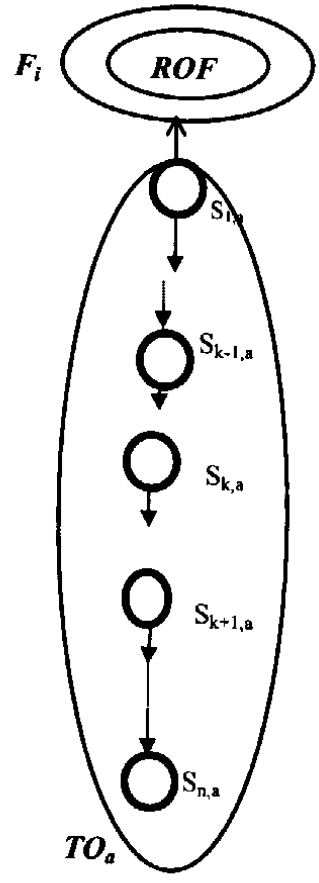


Fig. 2. A temporal object TO_a

2.1. Temporal objects and their families

An object is represented by its structure and state. With the passage of time an object may change its structure, state, or both. By associating time to a change to an object, we can keep the history of changes to that object. We define a temporal object (*TO*) to be an ordered set of objects which is constructed at different time instances. A temporal object is represented as $TO = < (SR_{t1}, ST_{t1}), (SR_{t2}, ST_{t2}), \ldots, (SR_{tn}, ST_{tn})$ where $t_i \leq t_{i+1}$ for all $1 \leq i < n$, where the ordered pair (SR_{ti}, ST_{ti}) is the i-th object of the temporal object which is constructed at the time instance t_i with structure SR_{ti} and state ST_{ti}. An i-th object of the temporal object is referred to as its i-th *stage* [9],[16].

A stage is maintained in a prototypical form, i.e., a structure, a state, or a combination of the two [4]. For example, if a temporal object suffers a structural change, then a new stage captures only the structural change to the temporal object. A temporal object may also be referred to as an ordered set of stages. For example, in Fig. 2, the temporal object TO_a of the family F_i has n number of stages. The first and last stages of a temporal object are significant because they hold the initial and current knowledge (or changes) of the temporal object. We refer to these stages as the *birth stage* (stage $S_{1,a}$ in Fig. 2) and the *current stage* (stage $S_{n,a}$ in Fig. 2), respectively. A new stage is appended to a temporal object if the structure and/or the state associated with its changes (see [9],[16] for more details).

The concept of a family is used to assemble a group of temporal objects sharing a common context. All temporal objects within a family can be handled in a similar fashion by responding uniformly to a set of messages. A set of similar structures and/or states defines *common context* of a family. The common context of a family is referred to as the *root-of-family* (*ROF*) where the *common knowledge* (available at that time) of all its temporal objects is maintained (see [9], [10],[16],[17] for more details). Temporal objects in a family can only be defined *after* the construction of the *ROF* of the family. In a family, each temporal object of the family shares the *ROF* only at the time of its birth (see Fig. 1 and Fig. 2). *After* that each temporal object is independent and a change to a temporal object does not effect the *ROF*. *ROF* of a family is *read-only*, and it does not change with the passage of time.

In the TOS, two types of families, *simple families* and *complex families*, can be defined. A *simple family* represents an independent object development environment in which temporal objects can be constructed without sharing any knowledge from temporal objects of other families. A *complex family* provides a facility for the integration of non-homogeneous temporal objects belonging to different families in order to build another temporal object of a higher level of abstraction, which is referred to as *temporal complex object* (TCO) [10], [17]. In Fig. 1 we did not specify simple or complex families.

A time dimension is associated with the creation of a stage, a temporal object, and a family in the TOS. The time is explicitly defined by the user as an instance variable. The granularity of time depends on the application domain. In the TOS, we use time point model for creation of families, temporal objects and stages [12], [13]. A time point is referred to as a *time instance*. We are using time instance as a *physical time* and *time point model* [7], [8], [12], [13].

2.2. An example of the TOS

In this section, we give an example of the TOS to understand some of the basic elements (i.e., stage, temporal objects, and families) of the TOS and their evolutions. For this purpose a vehicle design and development system is considered.

Figure 3 shows the schema of a vehicle design and development system. As mentioned earlier the double rectangle represents the root of the TOS (*RTOS*) The *RTOS* is a system object and the *root node* the system. The system has four families: *Engine, Body, Wheel* and *Vehicle*. They are designed at the time instances *1965, 1966, 1960* and *1970* (see Fig. 3), respectively. The *Engine* family, *Body* family and *Wheel* family are simple families, while the *Vehicle* family is a complex family. The *Vehicle* family is designed as an aggregation of the three simple families, i.e., *Engine, Body*, and *Wheel.* Fig. 4 shows the *ROF*'s of the three simple families with their time instances when they were defined in the system.

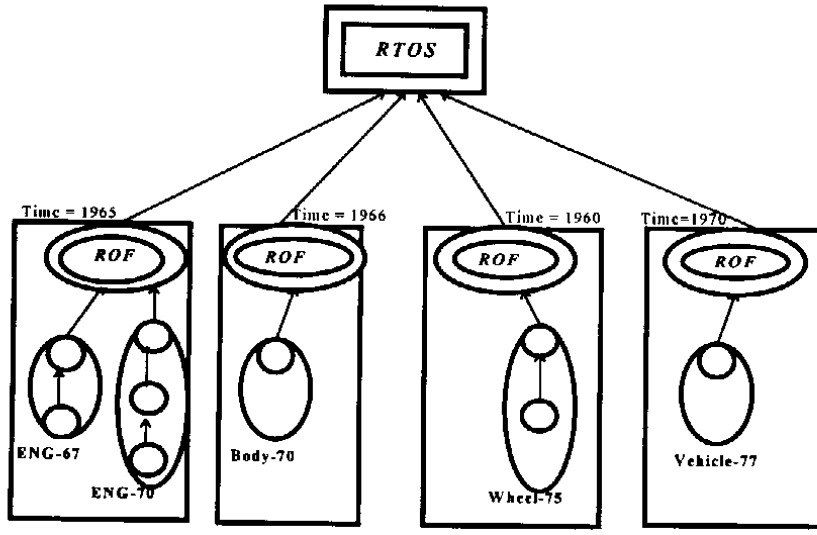
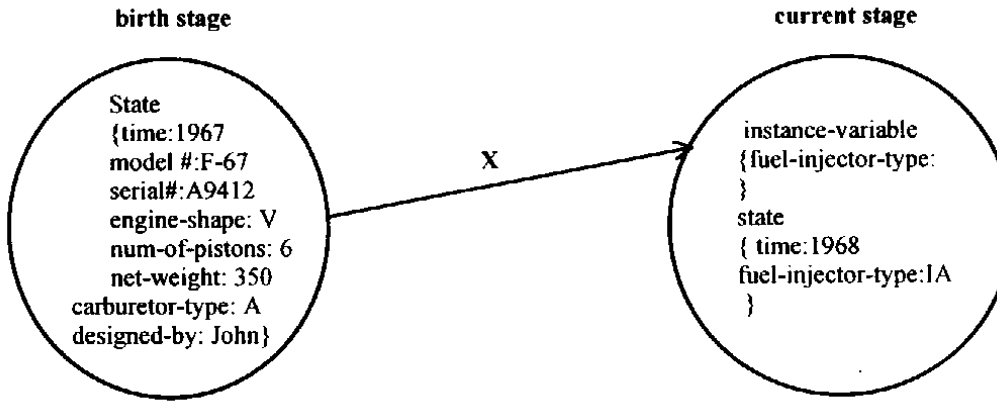


Fig. 3. Vehicle design and development system - *an example* of the TOS.

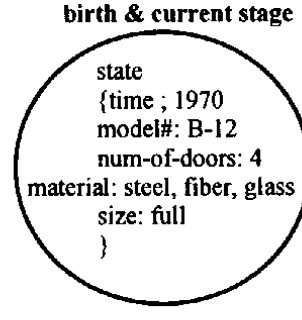
ROF(Engine)	*ROF(Body)*	*ROF(Wheel)*
Instance-variables	Instance-variables	Instance-variables
{	{	{
time: 1965	time: 1966	time:1960
model#,	model#,	model#,
serial#,	num-of-doors,	size
engine-shape,	color,	
num-of-pistons,	material,	
piston-shape,	size	
net-weight,		
carburetor-type,		
designed-by }	}	}
Methods	**Methods**	**Methods**
{run:	{weld-it:,	{mount-it:
}	color-it:}	}

Fig. 4. ROF's of engine family, body family and wheel family.

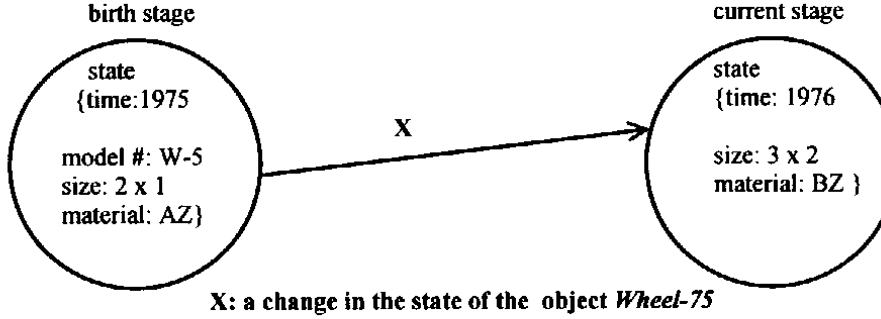


X: a change in the structure of the object *ENG-67*

(a) Evolution of the temporal object *ENG-67*



(b) The temporal object *Body-70*



X: a change in the state of the object *Wheel-75*

(c) Evolution of the temporal object *Wheel-75*

Fig. 5. Evolution of the temporal objects ENG-67, body-70 and wheel-75.

Figure 5(a) shows the evolution of the temporal object *ENG-67* of the *Engine* family. The current stage of the temporal object is added to the temporal object when the instance variable *carburetor-type* is replaced by the instance variable *fuel-injector* in the

structure of the temporal object. This change is a structural change to the temporal object. Fig. 5(b) shows only one stage (here the birth stage is also its current stage) of the temporal object *Body-70*. Fig. 5(c) shows the two stages of the temporal object *Wheel-75*. The current stage of this temporal object is constructed when the state of the object changes. These changes are in the value of the instance variables *size* and *material* at the time instance *1976*.

The complex family *Vehicle* shown in Fig. 6 is an aggregation of the three simple families. Fig. 7 shows the *ROF* of the complex family. The family is constructed in the system at the time instance *1970* (see Fig. 7). Existence of the aggregated families (*Engine, Body* and *Wheel*) is validated by the system at the construction time of the complex family (for details see [10], [17]). Fig. 6 shows the first temporal complex object (TCO) *Vehicle-77* of the complex family. The TCO is constructed by using temporal objects (subobjects) *ENG-67, Body-70* and *Wheel-75*. In the first subobject, *ENG-67.1968* means all stages of the subobject in the time interval *[1967, 1968]* are used in the TCO, where *1967* is the time instance when the subobject was constructed. Both stages of the subobject fall in the time interval *[1967, 1968]*. For the subobject *Wheel-75, Wheel-75.1975* is mentioned in the TCO. Therefore, only the birth stage of the subobject is used in the TCO as only the birth stage lies in the time interval *[1975,1975]*.

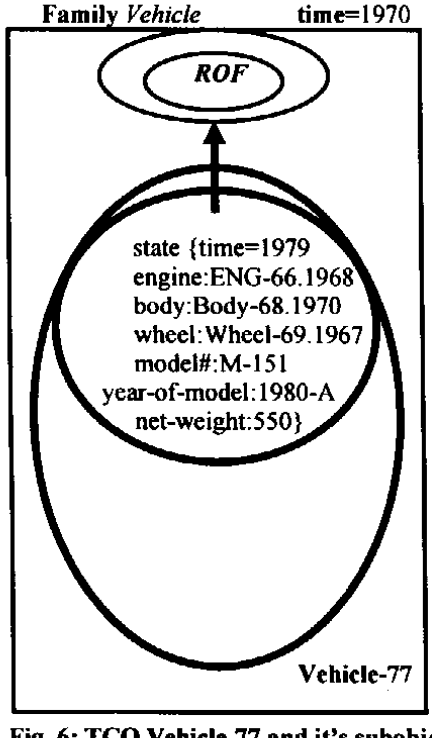


Fig. 6: TCO Vehicle-77 and it's subobjects.

ROF(Vehicle)
Aggregation-of:
{*Engine, Body, Wheel*}
Instance-Variables:
{ time=1970
model#,
year-of-model,
net-weight}
Methods:
{*assemble, test-it*}

Fig. 7. ROF of the family vehicle.

3. Inheritance Hierarchy and Delegation Hierarchy Models

After describing the basic elements of the TOS, we now give a brief description of the existing knowledge sharing mechanisms (i.e., the inheritance and delegation)of the object-oriented paradigm and their formal hierarchy models. These hierarchy models are proposed by Stein in [19]. It is also formally proved that both mechanisms are equivalent in terms of power. Both hierarchy models handle only simple inheritance (one parent of a sub-class) and single-parent delegation, details of the models and the related proofs can be seen in [19]. In the next two sections we briefly describe the inheritance hierarchy model **I** and the delegation hierarchy model **D** used to model objects in the class-based approach and the prototype-based approach, respectively. Note that we use the two terms hierarchy model **I** (or simply the model **I**) and hierarchy model **D** (or simply the model **D**) for the terms the inheritance hierarchy model **I** and the delegation hierarchy model **D**, respectively, later in this paper.

3.1 The inheritance hierarchy model

The inheritance hierarchy model **I** for objects of a universe of discourse is defined as follows:

I = $\{C, I, Y, W\}$ where
C is a set of classes in a class-hierarchy,
I is a set of instances,
Y is a set of attributes (instance variables and methods), and
W is a set of attribute values in Y.

Each class structure (or structure) has two sub-structures: locally defined sub-structure which is defined in the sub-class (or the child class) and sub-structure which is inherited from a super-class (or a parent class) [19]. The sub-structure which is defined locally in a class is referred to as *class-attributes*. The sub-structure which is inherited from a superclass is referred to as *instance-template*. In the hierarchy model **I**, the structure of each object of a class $c \in C$ is defined by the two parameters: *instance-template* and *class-attributes* where *class-attributes* are inherited from its superclass *super(c)*. The *attributes* of a class c can be written as the set *attributes(c)* = *class-attributes(c)* $\cup$ *instance-template(c)*. The values of the *attributes* are taken from the set $V(c) = \{val_c(y) \mid y \in attributes(c)\}$ where $c \in C$. An instance $i \in I$ of a class is defined by two sets *class(i)* and *value V(i)*, the values it assigns to the attributes in the instance template of its class. Other details of this hierarchy model can be seen in [19].

3.2 The delegation hierarchy model

The delegation hierarchy model **D** for objects of a universe of discourse is defined as follows:

D = $\{O, X, U\}$ where
O is a set of objects,
X is a set of attributes (instance variables and methods), and
U is a set of storage for the values of the instance variables of set X.

Each element of the set O is a prototype which is denoted by *prototype(o)* where $o \in O$. The attributes of the object o are denoted as *attributes(o)* and their corresponding values are denoted as follows:

$$V(o) = \{val_o(x) \mid x \in attributes(o)\}.$$

If the prototype o is the root of the hierarchy, then attributes of *the proto(o)* will be an empty set. Other details of both models can be seen in [19].

4. The Share-kno Hierarchy Model

A model of a system (or a universe of discourse) is an abstraction of the system in order to understand the system before implementing it [6], [15]. For example, in [6] a model of a language $\boldsymbol{L}$ is defined by 2-tuple $<A, T>$ where A is its *universe* (or universe of discourse) which is a non-empty set, and T is a set of *relations*. We refer to these components A and T as *elements* of the model $\boldsymbol{L}$. We say that a model M_1 is *more knowledgeable* (or is a *super-model*) than a model M_2, *iff* the model M_2 is a *sub-model* of the model M_1, and we denote them as $M_2 \Rightarrow M_1$. In other words, if one or more elements of a model M_2 are subsets of the corresponding elements of another model M_1 [6], then $M_2 \Rightarrow M_1$. If every element of a model M_1 is *equal* (or the same) to the corresponding element of a model M_2, then we say that both models are *equally knowledgeable*, and they are denoted as $M_1 = M_2$. Note that the relationship $\Rightarrow$ between two models also includes the relationship "=" (*equally knowledgeable*).

Now we propose a hierarchy model **SK** for temporal object hierarchy of the TOS. This proposed model is referred to as *Share-kno* (Share Knowledge) because the model fixes the knowledge sharing pattern among the objects in the hierarchy. We denote the *Share-kno* hierarchy model as **SK**. As we have mentioned earlier, Stein's inheritance hierarchy and delegation hierarchy models support only simple inheritance and single-parent delegation cases, receptively. Our hierarchy model **SK** of the TOS also supports and models knowledge sharing among temporal objects of simply families (see Section 2 for the definition of a simple family). This makes the model **SK** compatible with the Stein's inheritance hierarchy and delegation hierarchy models. In defining the model **SK** and for theorem proving in this paper, we use the same notations and conventions which are used in [19].

The Share-kno (**SK**) hierarchy model for temporal objects of a universe of a discourse in the TOS is defined as follows:

SK = {*F, TO, S*} where
F is a set union of sets *ROF*'s of simple families,
TO is a set of temporal objects, and
S is a set of stages in the TOS.

The sets *F, TO*, and *S* are referred to as *elements* of the hierarchy model **SK**. Similarly, the members of the sets {*C, I, Y, W*} and {*O, X, U*} are the elements of the hierarchy model **I** and hierarchy model **D**, respectively (see the previous section).

Basic axiom of the hierarchy model **SK** is defined as follows:

Axiom: The set *S* of stages of a system can be partitioned into three mutually disjoint subsets as follows:

$$S = S_{sr} \cup S_{st} \cup S_{sr+st} \text{ where}$$

S_{sr} is a set of stages which are constructed due to the structural changes to temporal objects of the system,

S_{st} is a set of stages which are constructed due to the stature changes to temporal objects of the system, and

S_{sr+st} is a set of stages which are constructed due to both structural and stature changes if both changes occur at the same time to temporal objects of the system.

These three subsets of the set *S* are mutually disjoint, that is,

$$S_{sr} \cap S_{st} \cap S_{sr+st} = \phi.$$

A temporal object $TO_i \in TO$ which was defined in a family $f \in F$, is a set of temporally ordered stages. The *j*-th stage $S_{j,i} \in S$ of the temporal object TO_i can be a member of the set S_{sr}, S_{st}, or S_{sr+st} based on the type of the stage.

On the first instance we take the hierarchy modelI proposed by Stein and prove that this hierarchy model is *less knowledgeable* than our proposed hierarchy model **SK**. In order to prove this we assume that if both models (the modelI and the model **SK**) are model objects of the same universe of discourse, then the elements of the hierarchy model I are proper or improper subsets of the corresponding elements of the hierarchy model **SK**. We prove this by simulating the elements of the first model with the elements of the second model.

4.1 Basis of the proof

In the hierarchy model I, at some given time instance, the structure of an object represents accumulative effect of all structural changes which occurred *after* defining the class of the object. In this model, details of the structural changes to a specific object are not traceable. Also, if both the structure and the state of an object change at the same time instance, then the model I is incapable to capture this type of change in a single update operation. On the other hand, the model **SK** can handle such type of changes to

an object, the set S_{sr+st} of stages represents those changes in a system. In other words, the model **SK** can handle all the types of changes (described in Axiom) to an object and later each change can be traced and identified, whereas in the modelI it is not possible to do so.

In the next section we formally prove that the hierarchy modelI is *less knowledgeable* than the hierarchy model **SK**, if both models have been used to model objects of the same universe of discourse. To prove this we define a function σ which simulates the elements of the model I and shows that they are subsets (proper or improper) of the corresponding elements of the model **SK**.

5. Proof of $\sigma(\mathrm{I}) \Rightarrow$ SK

We define a function σ which simulates the elements of the hierarchy modelI = $\{C, I, Y, W\}$ with the elements of the hierarchy model **SK** = $\{F, TO, S\}$ to show that the hierarchy model I is *less knowledgeable* (or a *sub-model*), than the hierarchy model **SK** if both hierarchy models are two different snapshots of objects of the same universe of discourse at the same time instance, i.e., $\sigma(\mathrm{I}) \Rightarrow$ **SK**. The function σ is defined as follows:

(i) $\sigma(\phi) = \phi$

(ii) $\forall x \in Y$, the set $\sigma(x) \in S_{sr} \vee \sigma(x) \in F$

(iii) $\forall u \in W$, the set $\sigma(u) \in S_{st}$

(iv) $\forall c \in C$, the set $\sigma(c)$ is defined as follows:
$\sigma(c) \subseteq f$ where $f \in F$ and $\sigma(attributes(c)) = \{\cup (\sigma(x) \mid x \in$ *class-attributes*(c)$\} \cup$
$\sigma(x) \mid x \in$ *instance-template(c)*$\}$. Here the set $\{\cup (\sigma(x) \mid x \in$ *class-attributes(c)*$\} \subseteq S^{f}_{sr} \cup f$ if x does not belong to current stage of any temporal object, and $\{\cup \sigma(x) \mid x \in$ *instance-template(c)*$\} \subseteq S^{f}_{sr}$ if x belongs to current stage, $S^{f}_{sr} \subseteq S_{sr}$ and the subset S^{f}_{sr} belongs to a family f. Note that *attributes(c)* is *union* of the sets *class-attributes(c)* and *instance-template(c)*, and the set S^{f}_{sr} denotes a set of stages which are defined due to the structural changes to temporal objects of the family f. The set *state(c)* is defined as follows:
$\sigma(state(c)) = \{\cup \sigma(v) \mid v \in state(c)\} \subseteq S^{f}_{st}$ where $\forall x \in attributes(\sigma(c))$
$val_{\sigma(c)}(x) = val_c(x)$, and $S^{f}_{st} \subseteq S_{st}$. Note that the set S^{f}_{st} is a set of stages which are defined due to the stature changes to temporal objects of the family f.

(v) $\forall i \in I$, $\sigma(i)$ is defined *as* $\sigma(i) = \sigma(class(i)) \subseteq (TO)_f$, where $(TO)_f$ is the j-th temporal object of a family f in the hierarchy model **SK**. Here $class(i) =$ $attributes(class(i)) \cup V(class(i))$ (see Section 3.1). The state of the temporal object $(TO)_f$ corresponds to the i-th instance in the hierarchy model I, and

$\sigma(attribute(class(i)))$ $=\{\cup \sigma(x) \mid x \in class\text{-}attributes(c) \wedge c \in super(class(i))\}$ $\subseteq \cup S^{f}_{i,j}$ for all $1 \leq i \leq n$, where each member of the set $\cup S^{f}_{i,j}$ belongs to the set S^{f}_{sr}, and the stage $S^{f}_{i,j}$ is i-th stage of the j-th temporal object belonging to the family f.

$\sigma(state\ (class(i))) = \{\sigma(v) \mid v \in state\ (i)\} \subseteq \cup S^{f}_{k,j}$ for all $1 \leq k \leq m$, where each member of the set $\cup S^{f}_{k,j}$ belongs to the set S^{f}_{st}, and the stage $S^{f}_{k,j}$ is the k-th stage of the j-th temporal object in the family f.

We assume there are p number of stages of the temporal object $(TO_j)_f$, which are members of the set of stages S^{f}_{sr+st} where the set S^{f}_{sr+st} is the set of stages S_{sr+st} of the family f. The instance $i \in I$ is unable to simulate those p number of stages since we know that the hierarchy modelI does not support a change to an object if the change occurs simultaneously to both parameters (i.e., the structure and the state) of the object (see Section 4.1). Therefore, the instance i of the hierarchy model **I** can simulate at the most $(m+n)$ number of stages if total number of stages of the temporal object $(TOi)_f$ is $(m+n+p)$. It means that the hierarchy modelI does not own some special type of instances that can be simulated with p number of stages of the model **SK**.

(vi) By using the function σ, we show that existence of the following relationships among the elements of both hierarchy models (**I** and **SK**).

$(S_{st} \cup S_{sr} \cup S_{sr+st} \cup F) \supseteq C$ as $(S_{st} \cup S_{sr} \cup F) = C$ .....(1)

$TO \supseteq \sigma(c) \cup \{\sigma(i) \mid i \in I\}$ .....(2)

$(S_{st} \cup S_{sr+st}) \supseteq I$ as $S_{st} = I$, it is true from (1) .....(3)

$(S_{sr} \cup S_{sr+st} \cup F) \supseteq Y$ as $(S_{sr} \cup F) = Y$ .....(4)

$(S_{st} \cup S_{sr+st}) \supseteq W$ as $S_{st} = W$ .....(5)

Now we first state the theorem for the proposed relationship between the modelI and the model **SK**, followed by its proof.

Theorem: Suppose both hierarchy models, **I** and **SK**, represent objects hierarchies of the same universe of discourse and both hierarchies are modeled at the same time instance t_0. A new object of the universe of discourse is defined in both hierarchy models at the same time instance $t_1 > t_0$. If both versions (th atemporal version O_i in the model **I** and the temporal version TO_i in the model **SK**) of the new object represent the same entity of the universe of discourse, then the temporal version TO_i of the new object is *equally or more knowledgeable* than the non-temporal version O_i of the new object at a time instance $t_2 > t_1$ in their respective hierarchies.

Proof: Since at the time instance t_1 both versions (O_i and TO_i) of the new object are defined in their respective hierarchies, therefore, at the time instance t_1 the following expressions will be true:

(attributes(O_i) = structure(TO_i)) and *(state(O_i) = state(TO_i))*. Because the object did not change since its birth, the relationship between the class structure of the object O_i and set of its attributes, i.e., *attributes(O_i)* = $\{x \mid x \in$ *class-attributes(c)*, $O_i \in c\} \cup \{y \mid y \in$ *instance-template(c)*$\}$ where *attributes(O_i)* $\subseteq Y' \subseteq Y$ (see Section 3.1) is true.
The following relationship is also true for both versions of the new object.
structure(TO_i) = $\{x \mid x \in S_{sr}\} \cup \{y \mid y \in f, TO_i \in f\} \cup \{z \mid z \in S_{sr+st}\}$ where *structure(TO_i)* $\subseteq (S_{sr} \cup S_{sr+st} \cup f)$ and from relationships (1) and (4) we conclude $Y' \subseteq \{S_{sr} \cup S_{sr+st} \cup f\}$.
state(O_i) = $V(O_i) = \{val_{Oi}(x) \mid x \in$ *attributes(O_i)*$\} \subseteq W' \subseteq W$ where $i \in c$, and
state(TO_i) = $\{x \mid x \in S_{st}\} \cup \{y \mid y \in S_{sr+st}\} \subseteq (S_{st} \cup S_{sr+st})$.
From relationships (4) and (5) we conclude $W' \subseteq (S_{st} \cup S_{sr+st})$. Hence, at the time instance t_1, both versions of the object are *equally knowledgeable.*

Now suppose that at the time instance $t_2 > t_1$, the new object suffered a change, then the change will affect both versions of the object in their respective hierarchies. The change will fall in one the following three types of changes. In the following paragraphs we describe the three possibilities of the change to the new object.

(i) **Stature Change:** If the change to the new object is a stature change at the time instance t_2, then *(attribute(O_i) = structure(TO_i))*. Since the change did not affect to the structure of the object, therefore, *(state(O_i)* $\subseteq$ *state(TO_i))* where *state(O_i)* is a new version of the state of the object O_i in the hierarchy modelI. In the model **SK**, *state(TO_i)* $\subseteq S_{st}$, where the temporal object $TO_i = \{S_{1,i}\ S_{2,i}\}$ (by the definition of a temporal object) at time instance t_2, and the stage $S_{2,i} \in S_{st}$ is the current stage of the temporal object (the new object). The current stage reflects the stature change to the new object in the hierarchy model **SK**. Hence, we conclude that both object versions (i.e., O_i and TO_i) are *equally knowledgeable* at the time instance t_2.

(ii) **Structural Change:** If the change to the new object at the time instance t_2 is a structural change, then the change has further two possibilities, and they are discussed as follows:

(a) If the change adds instance variables and/or methods to the object's structure, then the change is an increment to the knowledge of both versions of the object. So, at the time instance t_2, *attribute(O_i) = structure(TO_i)* and *state(O_i) = state(TO_i)*, where *attribute(O_i)* is an updated version of the *class-attributes*, *structure(TO_i)* $\in S_{sr}$, and *state(TO_i)* $\in S_{st}$. The cardinality of the set *class-attributes* at the time instance t_1 is less than the cardinality of the set *class-attributes* at the time instance t_2 for the object version O_i. Similarly, for the object version TO_i the cardinality of the *structure(TO_i)* at the time instance t_1 is also less than the cardinality of the *structure(TO_i)* at the time instance t_2. The hierarchy model **SK** keeps both previous knowledge and current

knowledge of every object of the model, while the hierarchy model **I** keeps only updated version of the *class-attributes* and does not keep the history of all changes to the *class-attributes*. Therefore, we conclude that in this case, the object version TO_i in the hierarchy model **SK** is *more knowledgeable* than the object version O_i in the hierarchy model **I** at the time instance t_2, because the previous record of the set *class-attributes* is lost *after* recording the change.

(b) If the change deletes instance variables and/or methods from the structure of the new object, then in this case the previous knowledge of the object in the hierarchy model **I** is lost *after* recording the change. Therefore, the cardinality of the *class-attributes* at the time instance t_1 will be greater than cardinality of the *class-attributes* at the time instance t_2. Since the hierarchy model **SK** keeps record of all changes to an object, therefore, the cardinality of the *structure*(TO_i) *before* recording of the change is the same because the model SK does not delete any information. So, *at* and *after* time instance t_2, *attribute* $(O_i) \subseteq$ *structure*(TO_i) and *state*$(O_i) \subseteq$ *state*(TO_i), because the deleted instance variables and/or methods are not available in the *attribute*(O_i) *at* and *after* the time instance t_2. Since the object version TO_i, retains both *present* and *past* knowledge of the object in the form of the current stage and previous stage(s), respectively. Therefore, we conclude that the object version TO_i in the hierarchy model **SK** is *more knowledgeable* than the second object version O_i in the hierarchy model **I**.

(iii) Structural and Stature Change: If the above two types of changes (i.e., (i) Stature Change and (ii) Structural Change) occur simultaneously to the new object at the same time instance t_2, the hierarchy model **I** is unable to accommodate this type of change, whereas the hierarchy model **SK** can handle this type of change to the object by creating a new stage and appending it to the temporal object. The new stage will be a member of the set S_{sr+st}. *After* recording the change in the form of a stage, the following inequality will be true.

$$structure(TO_i) \subseteq S_{sr} \cup S_{sr+st} \cup f.$$

We conclude that in this case the object version TO_i in the hierarchy model **SK** is *more knowledgeable* than its other version O_i in the hierarchy model **I**.

Therefore, it is concluded from the above three cases that the objects in the hierarchy model **SK** are *equally or more knowledgeable* than the objects in the hierarchy model **I** at the time instance t_2. This completes proof of the theorem.

Similarly, by following the steps of the above proof we can also prove that objects in the hierarchy model **SK** are *equally or more knowledgeable* than objects in the delegation hierarchy model **D** given in [19].

6. Conclusions and Future Work

In this paper, we have introduced a hierarchy model **SK** for objects of the TOS, and for knowledge sharing mechanism among the objects. We have also proved that objects of the hierarchy model **SK** are *equally or more knowledgeable* than objects of both inheritance hierarchy model proposed by Stein [19]. Also, the model **SK** can handle and record a type of changes (the third type of changes described in Section 5) to objects of the real-world which the other two models cannot handle.

We have implemented the kernel of the TOS, i.e., the *Object Manager* (OM) [18] using a prototype-based object-oriented language SELF [5], [20]. The object manager provides facilities of defining temporal objects and stages. Now we are in the phase of developing the other modules: *Family Manager*, *Storage Manager* and *Index Manager*, of the TOS.

The hierarchy model **SK** for objects of the TOS, which have been proposed in this paper, handles only simple families and their objects. We are working to enhance the present hierarchy model **SK** so that it could handle other elements of the TOS such as offstage objects (see [9] for details) and complex families and their objects. After the enhancement the hierarchy model **SK** will be able to model all elements of the TOS, and it will provide a sound mathematical footing to the TOS.

Acknowledgment. We like to thank the anonymous revisers for their valuable comments. We also like to thank Dr. Zafar U. Singhera and Mr. Habeebullah S. Khan for revering the early drafts of this paper.

References

[1] Agrawal *et al*, R. "Object Versioning in Ode." *Proceedings of Data Engineering*, (1991), 446-455.

[2] Aksit, M., Dijkstra, W. J. and Tripathi, A. "Atomic Delegation: Object-Oriented Databases." *IEEE Software Journal*, (March 1991), 84-92.

[3] Almarode, L. "Rule-Based Delegation for Prototypes." *Proceedings of the ACM International Conference on Object-oriented Programming Languages, Systems and Applications(OOPSLA'89)*, (1989), 363-370.

[4] Borning, A. H. "Classes Versus Prototypes in Object-oriented Languages." *Proceedings ACM/IEEE Fall Joint Conference*, (November 1986), 36-40.

[5] Chambers, C., Unger, D. and Lee, E. "An Efficient Implementation of SELF Dynamically-Typed Object-oriented Language Based on Prototypes." *Proceedings of ACM International Conference on Object-oriented Programming Languages, Systems and Applications (OOPSLA'86)*, (1986), 49-70.

[6] Chang C. C. and Keisler, H. J. *Model Theory. 3rd ed.*, North-Holland Publishers, 1990.

[7] Clifford, J. "A Model for Historical Databases." *Proceedings of Logical Bases for Databases*, Toulouse, France, 1982.

[8] Dutta, S. "Generalized Events in Temporal Databases." *Proceeding of the 5th IEEE Conference on Data Engineering*, (1989), 118-126.

[9] Fotouhi, F., Shah, A., Ahmed, I. and Grosky, W. "TOS: A Temporal Object-oriented System." *The Journal of Database Management*, 5, No. 4 (1994), 3-14.

[10] Fotouhi, F., Shah, A. and Grosky, W. "Complex Objects in the Temporal Object System." *IEEE Post-Proceeding of the 4-th International Conference on Computing & Information*, (1992), 381-384.

[11] Lieberman, H. "Using Prototypical Objects to Implementation Shared Behavior in Object-oriented Systems." *Proceedings of the ACM International Conference on Object-oriented Programming Languages, Systems and Applications (OOPSLA'86)*, (1986), 214-223.

[12] Ling, D. H. O. and Bell, D. A. "Taxonomy of Time Models in Databases." *Information and Software Technology*, 32, No. 3 (1990), 215-224.

[13] Ling, D. H. O. and Bell, D. A. "Modeling and Managing Time in Database Systems." *The Computer Journal*, 35, No. 4 (1992), 332-340.

[14] Nguyen G. and Rieu, D. "Schema Evolution in Object-oriented Database Systems." *Data & Knowledge Engineering Journal, North-Holland*, 4, No. 1 (1989), 43-67.

[15] Rumbough, J., *et al. Object-oriented Modeling and Design.* Prentice-Hall, 1991.

[16] Shah, A. "TOS: A Temporal Object System." *Ph.D. Dissertation*, Wayne State University,Detroit, Michigan, 1992.

[17] Shah, A., Fotouhi, F. and Grosky, W. "Renovation of Complex Objects in the Temporal Object System." *Proceedings of IEEE International Phoenix Conference on Computers and Communications*, Scottsdale, Arizona, (1993), 203-209.

[18] Shah, A., Fotouhi, F., and Grosky, W. "Operators Implementation of the Temporal Object System." The *Australian Computer Journal.* (Communicated)

[19] Stein, L A. "Delegation is Inheritance. " *Proceedings of the ACM International Conferenceon Object-oriented Programming Languages, Systems and Applications (OOPSLA'87)*, (1987), 138-146.

[20] Ungar, U. and Smith, R. B. "SELF: The Power of Simplicity." *Proceedings of the ACM International Conference on Object-oriented Programming Languages, Systems and Applications (OOPSLA'87)*, (1987), 227-242.

"شيرنو" طريقة للتشارك في المعرفة في النظام الموضوعي الزمني "توس"

عباد شاه*، فرشاد فتوحي و ويلم جورشكي**

**قسم علوم الحاسب، كلية علوم الحاسب والمعلومات، جامعة الملك سعود،*
ص.ب ٥١١٧٨، الرياض ١١٥٤٣، المملكة العربية السعودية
*** قسم علوم الحاسب، جامعة ولاية وايني، الولايات المتحدة الأمريكية*

(قدّم للنشر في ١٩٩٧٦/٣/٨م؛ وقبل للنشر في ١٩٩٧/٦/٣م)

ملخص البحث . يتعامل النظام الموضوعي الزمني "توس" مع التغييرات الناتجة في تراكيب وحالة المواضيع بصورة مطردة تراعي عامل الزمن. ينتهج "توس نهجاً جديداً في تمثيل المواضيع والمشاركة بينها في المعرفة. هذا النهج الجديد يخلط بين أسلوبي "الفصائل" و "الأمثلة" مما يجعله أكثر مرونة من النهجين.

في هذا البحث نقترح طريقة جديدة للمشاركة في المعرفة وترتيب المواضيع في نظام "توس": "شيرنو". أيضاً نثبت في هذا البحث أن نموذج شيرنو يحتوي على قدر من المعرفة أكبر من الذي يوفره نموذج "الإرث" ونموذج "التفويض" المستخدمة في الأنظمة السابقة.