Time Conscious Objects™ : A Domain-Specific Framework and Generator

Jorn Bettin
jorn.bettin@softmetaware.com

Jeff Hoare
jeff.hoare@softmetaware.com

SoftMetaWare
PO Box 617
Waiheke Island
Auckland, New Zealand

ABSTRACT

In most business software systems the time dimension of business objects plays a significant role. Time is a crosscutting concern that is hard to separate from other business logic. We have developed a toolkit that allows existing business application systems to be extended with "time-conscious" behavior in a non-intrusive way by factoring out all aspects of time-related behavior into a framework and a set of classes that is distinct from the existing code base. The Time Conscious Objects™ (TCO™) toolkit is currently implemented in Java™, but through the use of generation technology the toolkit can easily be made available in any language that supports polymorphism.

Categories and Subject Descriptors

D.2.2 [**Design Tools and Techniques**]: Model Driven Architecture - d*omain-specific architectures.*

General Terms

Design, Standardization, Languages.

Keywords

Model Driven Architecture (MDA), domain-specific languages, domain-driven development, time, temporal, versioning, undo, redo, auditability, logging.

1. INTRODUCTION

We start with a few definitions that provide us with a useful terminology for the domain.

Definition: the *creation time* of an object is a timestamp (date and time) taken when the object was constructed within the system.

Definition: In the context of this article we use the term *interval* consistently with (a) the mathematical definition of "a set of real numbers between two numbers either including or excluding one or both of them" and (b) the physical definition of "a space of time between events". When talking about intervals, we use the terminology of *start time* and *end time* to describe the timestamps (date and time) which define the start and end of the interval.

Definition: an object with *lifetime behavior* is an object that knows about its state and its state changes over time.

Definition: the *lifetime* of an object is the interval during which the object is considered to exist by the business logic of the system. The concept of lifetime extends to object states. Objects and object states may exist in the system at points in time when they are "not yet active" or when they are "no longer active".

Constraints built into lifetime behavior ensure that a system object can't violate the physical properties of time:

1. The start time for new object states is always greater than, or equal to the creation time of the state.
2. The set of lifetimes of all the states of an object always forms a contiguous interval.

As opposed to "real-time" systems, business application systems typically do not have real-time information exchange with real-world objects, which leads to the following important definition.

Definition: an object with *baseline behavior* is an object that knows about its current and historic representations in the system.

The difference between lifetime behavior and baseline behavior may seem to be subtle, but it is critical in accurately modeling time within software systems. When lifetime behavior is combined with baseline behavior

- the first constraint on lifetime behavior is removed, as the history of "back-dated" changes is accurately recorded as part of baseline behavior, ensuring auditability.
- it results in the ability to accurately model the lifecycle of real-world objects in the context of a system where information exchange with the real world only occurs at discrete points in time.

Definition: a *time-conscious object* is an object that implements aspects of the notion of time in its behavior. An object with *time-conscious behavior* is an object with lifetime behavior, or baseline behavior, or the combination of both sets of behavior.

2. TYPICAL APPLICATION SCENARIOS

The TCO framework has the potential to drastically simplify application code that involves the aspect of state changes of business objects, in particular in those cases where both lifetime and baseline behavior are required. Consequently time-conscious objects can have a major impact on the quality and maintainability of applications by eliminating the complexity of dealing with the time dimension. This is the case for example in all time-based billing systems in the telecommunications and utility industries, in the insurance industry, and in the modeling of "parts" in ERP and manufacturing systems, where product composition changes over time, as new [product] components get developed and are phased into production.

The TCO toolkit has been explicitly designed to allow non-intrusive and incremental integration into existing systems. It

OOPSLA'03, October 26–30, 2003, Anaheim, California, USA.
ACM 1-58113-751-6/03/0010.

allows existing classes to be extended with time-conscious behavior, which then enables application code in the next higher architectural layer to be refactored and simplified. Existing business logic can be incrementally modified to take advantage of time-conscious behavior in the extended classes.

Structurally TCO can be divided into three components:

1. A small framework that provides support for the notion of intervals, lifetime behavior, and baseline behavior,
2. A tool that allows existing class structures to be annotated with high-level design information related to the required level of time-conscious behavior - giving the designer the ability to trade-off granularity of time-consciousness against performance requirements,
3. A generator which automates the extension of existing class structures and the integration with the TCO framework.

At compile-time and run-time only the first and the third component are relevant, and of course the generated glue code which is encapsulated in a separate small subsystem for each time-conscious class.

The only assumption made about the architecture of existing applications is that classes which need to be made time-conscious provide modifier and accessor methods to the properties that the user intends to declare as time-conscious.

TCO is geared towards use in MDA® [1] approaches where system functionality is expressed in Platform Independent Models (PIMs) at a high level of abstraction [2]. Time-conscious behavior is specified in a PIM by tagging classes as having lifetime and/or baseline behavior. The TCO toolkit then uses this information to generate the code required to hook into the TCO framework. To take into account practical performance and object size implications, the commercial version of TCO will allow users to tag individual properties (attributes and associations) of classes as time-conscious, enabling the TCO generator to produce code that is optimized for a specific context.

An interesting side effect of incrementally refactoring existing code to leverage time-conscious objects is the impact on the business object model of the system in question. In each case where previously the business object model contained a pattern of class `Foo` and a related class `FooVersion` with an aggregation link between the two classes, the introduction of time-conscious objects eliminates the need for class `FooVersion` by providing a time-conscious extension `TcFoo` of class `Foo`. As a consequence new code can take advantage of time-conscious behaviour in `TcFoo`, but all existing code and all code that is not interested in the time dimension works with existing class `Foo`.

3. FUNCTIONALITY

The TCO framework provides users with the ability to specify and retrieve property values using standard modifier and accessor methods in combination with appropriate specification of associated intervals. This is the trivial part of TCO.

The interesting part of TCO consists of functionality to iterate over intervals and baselines, and to compare the properties of two objects over time, iterating over the results of the comparison. Let the intervals over which objects **a** and **b** are valid within the business logic of a system be denoted by sets **A** and **B** correspondingly. The result of a comparison of two objects with lifetime behavior is shown in figure 1. The graphical notation we use for open and closed ends of intervals is derived from the usual mathematical notation of $[a_1, a_2]$ for the closed interval that includes a_1 and a_2, and $[a_1, a_2)$ for the half open interval that includes a_1 and excludes a_2 etc. The p_i in figure 1 denote the property values of an object over specific intervals.

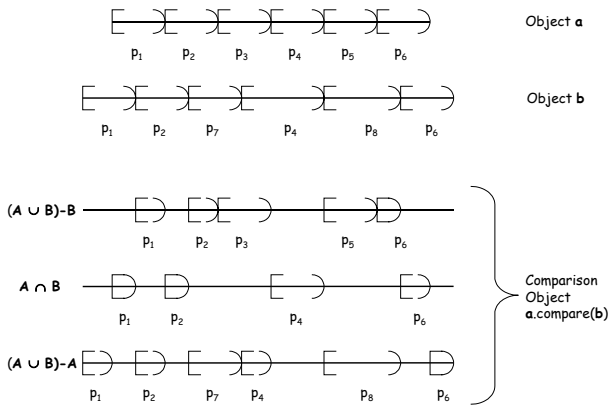

Figure 1

The TCO framework provides the ability to selectively iterate over the differences, commonalities, or all state changes across **a** and **b**. TCO emphasizes intervals and the ability to manipulate state information associated with intervals in an intuitive way. For calculations involving the time dimension, TCO allows switching to a perspective where the flow of time is the primary concern, and where object structure is a secondary concern.

TCO can be used to standardize and replace existing implementations of "versioning" functionality. Even if an existing implementation is standardized, TCO adds significant value by raising the level of abstraction of the "versioning API".

A related topic is "undo/redo" functionality. TCO can simplify the implementation of such functionality by allowing users to work with objects that represent snapshots in time. When handling object structures, the user can construct a snapshot by specifying the desired point in time, without having to be concerned with the start time and end time of individual object states. In the scenario of reactivating old object states, TCO preserves full auditability.

4. FURTHER WORK

Currently further work is in progress to integrate time-conscious objects with other concerns such as persistence and distribution. The intention is to create the ability to easily integrate with the persistence mechanisms of existing systems and to extend the interfaces of time-conscious objects with behavior that is relevant when large time-conscious objects need to be processed in a distributed environment. The future direction of TCO will be determined by the feedback we get from users and potential users. Different types of systems may have specific time-dimension related requirements that will either be addressed by expanding the base functionality of TCO, or by creating domain-specific variants of TCO.

5. REFERENCES

[1] Model Driven Architecture. www.omg.org/mda/

[2] J. Bettin. Raising the level of abstraction of design models. OOPSLA 2001 Companion, (October 2001).