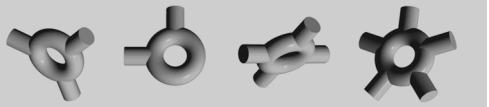
elements



Dynamic Data Package

TARMS Inc.

September 07, 2000

Typeset in LaTeX.

Contents

List of Figures

List of Tables

Package Description

Dynamic data extends the concept of versioned data, where changes in versions are expected while a system is running. Static versioned data, in contrast, is data that can only be updated while the system is not in use.

To prevent simultaneous attempts to update dynamic data, a system of locks is used. Only the owner of a lock can update the data. As well as a domain of identification — the domain that allocated the object's identifier — a dynamic data object has a domain of ownership — the domain that has the right to allocate locks. Versions of object families may pass from domain to domain during ownership changes.

Dynamic data needs to account for the possibility of holdovers, updates that occur within one day, but whose effects need to be held over until the next day as far as processing and consolidation are concerned.

Dynamic data objects usually pass through a “life-cycle” where the object, once created, goes through a series of modifications and enrichments before expiring. As an example, a deal might be entered, go to back office for confirmation, go through a series of settlement steps and then mature.

Life-cycles are usually specified by the business rules of an institution and can, in theory, be exactly specified. A basic form of life-cycle is supplied in this package, which allows the life-cycle stages to be named.

1 Use Cases

1.1 Edit Object

Multiple simultaneous updates to a dynamic data object, shared between several systems, need to be prevented.

If there is no architectural locking mechanism, then a locking mechanism is needed. Before editing a dynamic data object, a lock needs to be obtained on the object. When the object has been edited and committed, the lock can be removed.

1.2 Holdovers

Back office processing is cut-off at a given time each day. This is to allow end of day processing to be done. If trading occurs after the cut-off time, these trades are ‘held over’ for processing the next day.The period between the termination of back office processing and end of day is the *holdover period*.

During the holdover period, any new deals are held over until the next end of day (although the deals still have a trade date of the current date), rather than being incorporated into end of day processing.

If a deal made before the holdover period is modified during the holdover period, then the last version of the deal made before the period started is included at end of day. The new version(s) of the deal are included as modifications to the deal during the next day. A deal can be modified several times during the holdover period; the current version is used for modification.

1.3 Domain Transfers

Domains tend to represent geographically distant entities. It should be possible for a system in one domain to modify an object held in another domain. In the absence of a suitable architectural locking scheme, the object needs to be transferred

from the current owning domain to the modifying domain, so that lock grants and updates can be handled locally.

2 Interfaces

2.1 DynamicData

The DynamicData interface is an extension of the basic Version interface. DynamicData objects have an owning domain, the domain that holds the rights to either modify the object or pass modification rights to another domain.

DynamicData objects also draw a distinction between the *current* version of an object and the *active* version of the object. The current version is the latest version of that object, irrespective of whether or not it is being used. The active version is the version that would be used at the current end of day. If the current object is not the active object, the current object is held over until the next business day. The distinction between active and current version of an object allows transactions to occur during the current business day which will not be processed until the next business day, even though they are treated as if they had been done on the previous business day.

2.1.1 Relationships

	Class	Description	Notes
⇑	Version		
↓	DynamicDataModel §4.2		
↔	LockModel §4.3	lockedObject 0..1	

⇑:Inherits ↓:Realized by ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.1.2 Operations

Domain owner() owner

Owning domain. Returns the owning domain for this object.

Lock lock() lock

Associated lock. Returns the lock associated with this object, or nil if there is no lock.

Boolean isActive() isActive

Currently active version? Returns true if this is the active version of the object.

Version activeVersion() activeVersion

Currently active version. Returns the active version of this object. If there is no explicitly marked active version, then the current version is returned.

LifeCycleState lifeCycleState() lifeCycleState

The life cycle state. Returns the current life cycle state of this piece of dynamic date.

2.2 LifeCycleState

Various objects can be in various states throughout their lifetime. The state an object is in at a given point in time (the lifeCycleState) can be used to determine actions which can be performed on that object.

For example, deals can be in various states from the time they are dealt to their maturity. If a deal does not have the minimum information required to allow a deal to be settled, then a deal could be said to be in an 'incomplete' lifeCycleState. By knowing a deal's lifeCycleState, actions that can be performed on this deal can then be determined.

This interface provides an abstract interface for the provision of life cycle states. Since life cycle states need to be named, the interface inherits from Identifiable. Life cycle states are normally simple state markers and can be generally implemented as attributes; this interface, therefore, has value semantics.

2.2.1 Relationships

	Class	Description	Notes
⇑	Identifiable		
⇑	ValueSemantics		
↓	BasicLifeCycleStateModel §4.1		

⇑:Inherits ↓:Realized by

2.3 Lock

The Lock interface provides basic information on a lock. Locks are used to indicate that a DynamicData §2.1 object is locked for modification. Locks have an expiry

time, after which the lock is no longer valid and is eliminated from the granting Locksmith §3.2 and requesting DynamicData object.

2.3.1 Relationships

	Class	Description	Notes
↓	LockModel §4.3		
↔	LockingCacheModel §5.1	locks 1..1	
↔	DynamicDataModel §4.2	lock 0..1	

↓:Realized by ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.3.2 Operations

DynamicData lockedObject() lockedObject

Returns the DynamicData §2.1 object that has been locked.

Locksmith lockGranter() lockGranter

Returns the Locksmith §3.2 that granted the lock.

Timestamp lockedAt() lockedAt

Time locked. Returns a time stamp for when the object was locked.

Timestamp expiresAt() expiresAt

Expiry time. Returns the time at which the lock expires.

3 Architectural Service Interfaces

3.1 DynamicDataDepot

A DynamicDataDepot is an extension of the Depot interface that can also provide information as to the active version of an object family, as well as the current version.

3.1.1 Relationships

	Class	Description	Notes
⇑	Depot		
↓	LockingCacheModel §5.1		

⇑:Inherits ↓:Realized by

3.1.2 Operations

Version activeVersion(Keyable key) activeVersion
key: Keyable The key for the family.
Raises: NotFoundException

Active version of a versioned family. Returns the active version of the family of objects that have the supplied key. If no object is marked as active (or the object is not DynamicData) then the current version is returned. If no object with this identifier can be found, then the NotFoundException is raised.

3.2 Locksmith

The Locksmith interface provides basic functionality for centralizing and managing locking functions. Before being modified, DynamicData §2.1 objects request a lock from a Locksmith §3.2, so that multiple updates are prevented.

3.2.1 Relationships

	Class	Description	Notes
↓	LockingCacheModel §5.1		
↔	LockModel §4.3	lockGranter 1..1	

↓:Realized by ↔:Association →:Navigable ◊:Aggregate ♦:Composite

3.2.2 Operations

DynamicData lockRequest(DynamicData requestor) lockRequest
requestor: DynamicData The object that is to be locked.
Raises: LockException, NotFoundException

Get a lock on an object. Returns the version of the requestor that has a lock on it. In some cases, when ownership is transferred from a different domain, a new version of the dynamically modifiable object will be created. This object is the object that will be locked. If the object is not maintained by the depot, then a NotFoundException is raised. If the object has already been locked, then a LockException is raised.

unlockRequest(DynamicData requestor) unlockRequest
requestor: DynamicData The object that has finished with its lock.

Raises: LockException, NotFoundException

Unlock a dynamic data object. Remove the lock from the requestor. If this object is not managed by this depot, then raise a NotFoundException. If there is no lock on the requestor, then raise a LockException.

4 Classes

4.1 BasicLifeCycleStateModel

A simple implementation of the LifeCycleState §2.2 interface. This implementation allows the life cycle states to be named.

4.1.1 Relationships

	Class	Description	Notes
↑	LifeCycleState §2.2		

↑:Realizes

4.1.2 Attributes

identifier: String The identifier naming the life cycle state.

4.1.3 Operations

Reportable validate() validate

Validate the Life Cycle State Model
The Life Cycle State Model is invalid if:

- The identifier is null

4.2 DynamicDataModel

The DynamicDataModel class is a concrete implementation of the DynamicData interface.

This class is a subclass of VersionModel and, like VersionModel, is intended to be used as part of some larger object that also implements the DynamicData interface by passing messages through to a held DynamicDataModel.

4.2.1 Relationships

	Class	Description	Notes
⇑	VersionModel		
↑	DynamicData §2.1		
↔	DomainModel	owner 1..1	→
↔	Lock §2.3	lock 0..1	→

⇑:Inherits ↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

4.2.2 Attributes

isActive: Boolean = false This attribute is set to true if this object is the currently active version of the object. Otherwise, this object is set to false.

lifeCycleState: LifeCycleState The current life-cycle state of this object.

4.3 LockModel

A LockModel is a concrete implementation of the Lock interface.

4.3.1 Relationships

	Class	Description	Notes
↑	Lock §2.3		
↔	DynamicData §2.1	lockedObject 0..1	→
↔	Locksmith §3.2	lockGranter 1..1	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

4.3.2 Attributes

lockedAt: Timestamp The date and time at which the lock was granted.

expiresAt: Timestamp The date and time at which the lock expires.

5 Architectural Services

5.1 LockingCacheModel

The LockingDepotModel class is essentially a depot that can grant locks and keep track of which version of an object is the active object. Both the DynamicDataDe-

pot and Locksmith interfaces are implemented. Centralizing storage and locking information creates a single point for handling the manipulation of dynamic data.

As with all implementations of the Depot interface, the exact implementation of the associated operations are architecture-specific.

5.1.1 Relationships

	Class	Description	Notes
⇑	DepotModel		
↑	Locksmith §3.2		
↑	DynamicDataDepot §3.1		
↔	Lock §2.3	locks 0..n	→

⇑:Inherits ↑:Realizes ↔:Association →:Navigable ◊:Aggregate ⧫:Composite

6 Exceptions

6.1 LockException

A LockException is raised when an attempt to lock or unlock a piece of DynamicData fails.

6.1.1 Operations

DynamicData requestor() requestor

Lock (or unlock) requestor. Return the DynamicData §2.1 object that requested a lock or unlock.

Lock lock() lock

Existing lock. Return the lock, if there is one, that prevents this lock request. Return nil if there is no existing lock.

7 Associations

Table 1: Dynamic Data— Associations

Association Role	Class	Card.	Notes
locks			

Table 1: . . . continued

Association Role	Class	Card.	Notes
	Lock §2.3	0..n	→
	LockingCacheModel §5.1	1..1	
lockedObject			
	DynamicData §2.1	0..1	→
	LockModel §4.3	0..1	
owner			
owner	DomainModel	1..1	→
local version	DynamicDataModel §4.2	0..n	
lockGranter			
	Locksmith §3.2	1..1	→
	LockModel §4.3	1..1	
lock			
	Lock §2.3	0..1	→
	DynamicDataModel §4.2	0..1	

→:Navigable ◊:Aggregate ♦:Composite

7.1 locks

Role: *Navigable* Lock, 0..n.
Role: LockingCacheModel, 1..1.

7.2 lockedObject

Role: *Navigable* DynamicData, 0..1.
Role: LockModel, 0..1.
Ownership of a locked object.

7.3 owner

Role: owner *Navigable* DomainModel, 1..1.
Role: local version DynamicDataModel, 0..n.
The owner of this version.

7.4 lockGranter

Role: *Navigable* Locksmith, 1..1.
Role: LockModel, 1..1.

The locksmith that granted a lock

7.5 lock

Role: *Navigable* Lock, 0..1.
Role: DynamicDataModel, 0..1.
The current lock on this object.

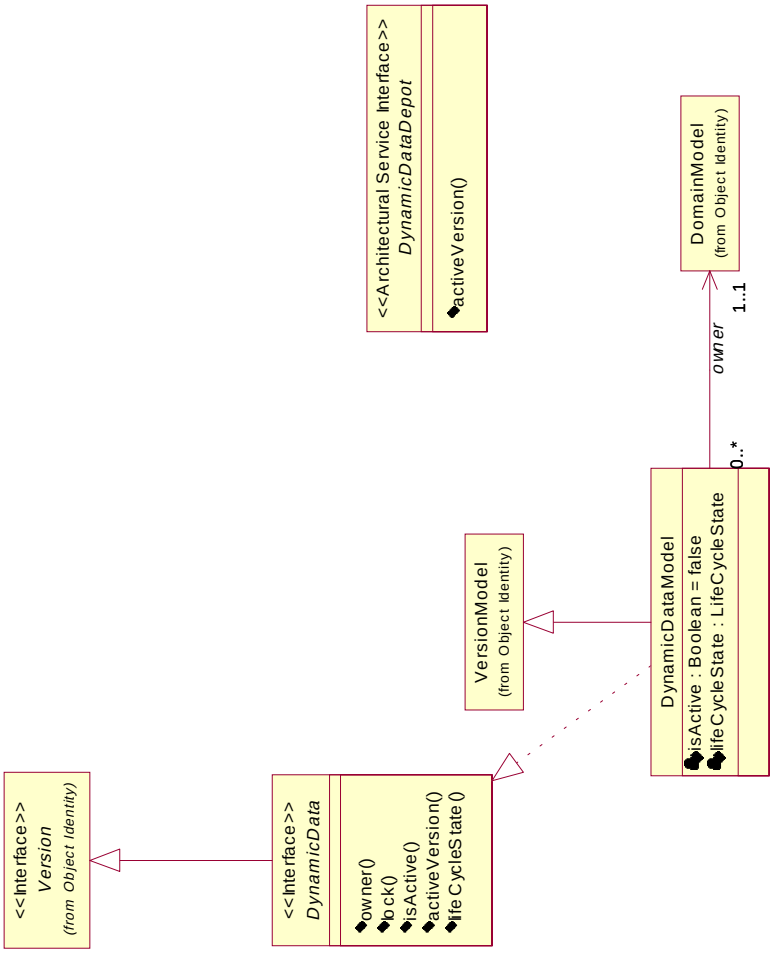


Figure 1: Class Diagram— Dynamic Data

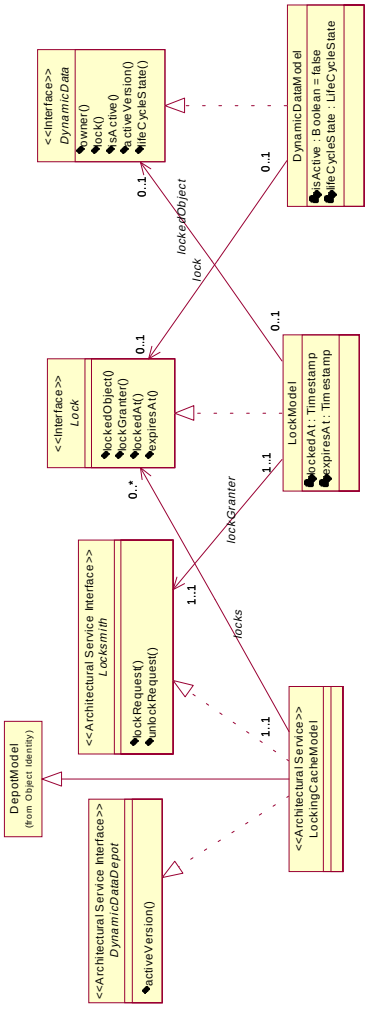


Figure 2: Class Diagram— Locking

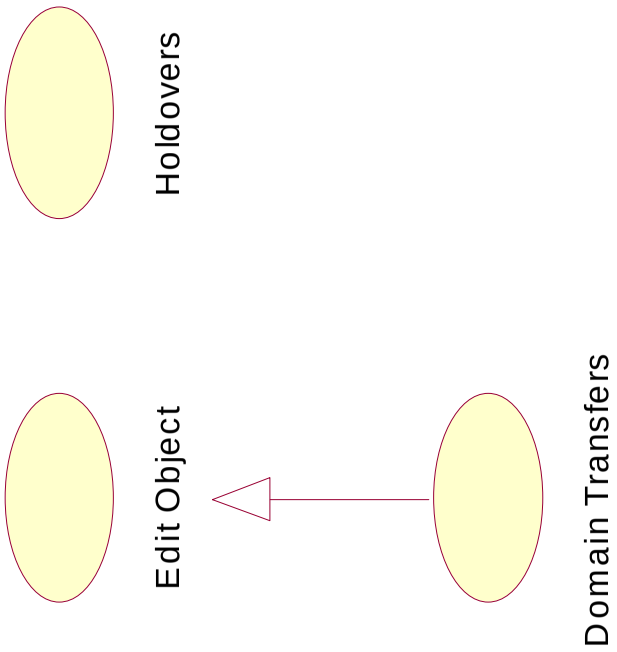


Figure 3: Class Diagram— Examples

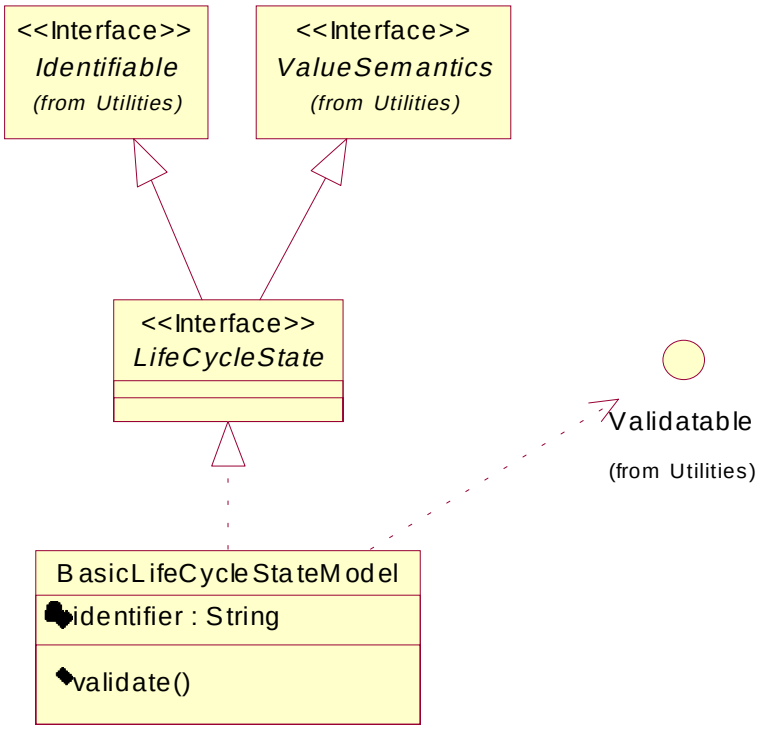


Figure 4: Class Diagram— Life Cycles

References