

instance. An attribute instance has a zero-to-many association relationship with a data instance that is a holder for instances of data that meet the constraints placed upon them. An attribute instance has a zero-to-many association relationship with access signal instances that are the signal instances holding listeners interested in hearing about accesses to the attribute values. An attribute instance has a zero-to-many association relationship with prechange signal instances that are the signal instances holding listeners interested in hearing about changes to the attribute values before the change occurs. An attribute instance has a zero-to-many association relationship with postchange signal instances that are signal instances holding listeners interested in hearing about changes to the attribute values after the change has occurred.

[0313] An attribute instance may include the following operations: `getValue()` that returns the current value(s), `setValue(Instance)` that removes any previous value and sets it to the new value after checking all constraints, `addValue(Instance)` that add the instance given to the current values after checking all constraints, `removeValue(Instance)` that remove the given instance from the current values after checking to see if the value is being held and checking occurrence constraints, and `clearValues()` that remove all values after checking occurrence constraints.

[0314] A data event is fired whenever a value is added, changed, or removed.

[0315] An operation implementation is a difficult virtual implementation to construct. For each language construct (like branches, loops, comparisons, etc.), there must exist an operation implementation. These operations are added together in a sequence to construct the virtual operation. This is similar to scripting inside an application, or writing source code where one line is evaluated before the next, with some lines of source code skipping or repeating other lines of source code. Once these primary language elements exist, a virtual operation implementation has all the power of a full programming language. However, a virtual operation implementation still has the advantages of drag-and-drop assembly (if the metamodel tool supports it) and certain aspect-programming advantages from being "virtual." Security and logging can be added at runtime to a virtual operation implementation, as can any other functionality modification. New operations can be added to the language by creating a new operation implementation. This is similar to but far beyond operator overloading or standard template library functionality in C++.

[0316] An operation implementation has a one-to-one association with an operation descriptor that is the operation descriptor for which the virtual operation implementation provides access to an implementation. An operation implementation has a zero-to-many association relationship with suboperations that are used to describe the operations that will be executed by this operation in the order in which they occur. An operation implementation has a zero-to-many association relationship with parameter accessors that are used by the operation implementation to impose occurrence, value, and access constraints on the parameter values.

[0317] An operation implementation includes an `execute(ParameterList)` operation that attempts to perform the operation by executing the operation implementation. First all access and precondition constraints are checked for the

operation implementation and parameter implementations. Then the operation is attempted. If the operation implementation is successful, the postcondition constraints are checked for the operation implementation and parameter implementations.

[0318] An operation implementation adds no additional events.

[0319] Examples of the use of an operation implementation sub-implementation for trapping failures are described below.

[0320] A failure trap implementation of the present invention is an example of an operation implementation representing a try-catch block used by some programming languages (Visual Basic refers to these as `onError` blocks). A failure trap implementation serves to capture any failures that occur during execution. Failure descriptors signify errors that must be handled in order for the software to continue executing or errors that explain why execution has been cancelled. In either case, a failure trap allows the program to capture (or trap) a failure and perform an operation to attempt logging or recovery from that error. The finally operation is always executed at the end of the try operation, whether a failure was thrown in the try operation or not. The finally operation implementation is even guaranteed to execute even if a further failure is created in the catch operation implementation that is not caught. It will always be executed if the try operation is executed. Note: precondition constraints can prevent the try operation from executing. When preconditions prevent the try operation from executing, the finally operation is not executed. This is consistent with the statement made earlier: the finally operation will always be executed if the try operation is executed.

[0321] A failure trap implementation has a zero-to-many association relationship with failures that are captured when thrown. Failures may be thrown by an operation implementation, constructor implementation, destructor implementation, any constraint type, and by accessor operations. Failure trap implementations capture these error notifications and perform some action. This action may be corrective or may simply record the event. A failure trap implementation has a one-to-one association relationship with a try operation implementation that is the operation to attempt. A failure trap implementation has a zero-to-one association relationship with a catch operation accessor that is the operation to perform whenever a failure is received. A failure trap implementation has a zero-to-one association relationship with a finally operation accessor that is the operation to perform whenever the try operation has been executed whether the execution of the try operation was successful or not.

[0322] A failure trap includes `Execute(ParameterList)` operation that inherited from `OperationImplementation`.

[0323] Instances of failure trap implementation fire failure events fired whenever a failure is trapped. These events can be useful to ensure that a failure trap operation is correctly handling failures. Corrective actions, assumed to always succeed, that fail to perform can cause serious headaches to debugging. These events allow rapid discovery of failed corrective actions.

[0324] An operation instance of the present invention represents an operation to execute. At this level, an operation