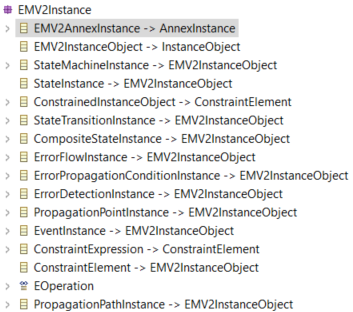
# Instantiation of EMV2 in AADL V2.2

The AADL V2.2 core instance model provides an API for instantiating annexes. For each component instance the registered method instantiateAnnex is invoked by the core instantiator. This is done in a prefix order, i.e., for subcomponents before the enclosing component.

For EMV2 we have the class EMV2AnnexInstantiator as extension of AnnexInstantiator, which provides this method.

## EMV2Instance Ecore Model

The following figure shows the classes defined in the Ecore model.



* EMV2AnnexInstance is a subclass of AnnexInstance. Every component instance with an EMV2 subclause contains an instance of this class. The EMV2AnnexInstance object contains instances of EMVInstanceObjects associated with the component instance.
* EMV2InstanceObject is an abstract class to represent all EMV2 specific instance objects that are contained in an EMV2AnnexInstance. All classes below are subclasses of EMV2InstanceObject.
* StateMachineInstance represents an instance of an error behavior state machine associated with a component. It includes a reference to the current error behavior state.
* StateInstance represents a state within an error behavior state machine.
* ConstrainedInstanceObject represents type constraints on incoming and outgoing error propagations. It includes a reference to the respective feature instance, or propagation point instance and a set of type tokens as constraint. It is also used in composite state conditions – in this case the instance object refers to the state instance associated with the referenced subcomponent.
* StateTransitionInstance represents a error behavior transition. It contains an instance of the condition expression as well as the source and target state instances. Each transition branch is instantiated as a separate StateTransitionInstance.
* CompositeStateInstance represents a composite state declaration in EMV2. It contains an instance of the condition expression as well as a reference to the composite state instance for which the condition is defined.
* ErrorPropagationInstance represents an error propagation. It is a subclass of ConstrainedInstanceObject.
* ErrorFlowInstance represents error flows with incoming and outgoing propagations represented by constrained instance objects.
* ErrorPropagationConditionInstance represents outgoing error propagation condition declarations. It includes acceptable current states as inStates (a list of references to state instances), the incoming error propagation condition as condition instance, and constrained instance object to represent the resulting outgoing error propagation.
* ErrorDetectionInstance represents error detection condition declarations. It includes acceptable current states, the incoming error propagation condition as condition instance, and a reference to the feature instance of the reporting trigger port and the error code value.
* PropagationPointInstance to represent user defined propagation points.
* EventInstance to represent error events, recover events, and repair events.
* Constraint representing a multi-operand logic operation on constrained instance objects or subconstraint expressions.
* ConstraintElement as an abstract class to encompass constraints and constrained instance objects.
* PropagationPathInstance to represent propagation paths along connections and bindings between outgoing and incoming error propagations as represented by constrained instance objects.

## EMV2 Instantiation Order

For each component instance an EMV2AnnexInstance object is created. The EMV2AnnexInstantiator assumes that EMV2 instantiation is invoked on subcomponent instances before their enclosing component instances. This is necessary to handle resolution of references to EMV2 instance objects in condition expressions and in property associations.

For each component instance an EMV2AnnexInstance is added.

Then EMV2InstanceObjects are added into the EMV2AnnexInstance in the following order: Propagation point instances, error propagation instances, error flow instances, error behavior event instances, state machine instances with respective state instances for each component with a use behavior clause, error state transition instances, composite state instances, outgoing propagation condition instances, and error detection instances.

## Condition Expressions and Outgoing Propagations

Condition expressions appear in transitions, outgoing propagation conditions, detection conditions, and composite state conditions. Outgoing propagation conditions also specify the error type to be propagated out.

We do instantiate condition expressions as well as outgoing propagation typed token assignments.

We use ConstrainedInstanceObject to represent elements of condition expressions and the outgoing propagation assignments.

ConstrainedInstanceObjects consist of:

* instanceObject: reference to an instance object through which the propagation occurs – this is a feature instance, propagation point instance, or in case of data/bus access or bindings the component instance. When used in composite state conditions it can refer to a StateInstance of a subcomponent.
* Constraint: a set of TypeToken objects that represent the types acceptable in a condition element – in the case of an action a single type token in the set or an empty set in case of pass through.
* bindingKind: to represent a binding related access propagation involving the component itself as instanceObject reference.

Condition expressions are represented by ConstraintExpression objects that include a logical operator and the elements it applies to. They can be ConstraintExpression or ConstrainedInstanceObject objects. The super class for both is ConstraintElement.

## Propagation Path Instances

Propagation path instances are created for connection instances, user defined propagation path instances, and for bindings.

When we create propagation path instances, we take into account any typed token constraints involved in the propagation path, where possible.

* In the case of outgoing propagation conditions, we have two cases: explicit assignment of a typed token for outgoing propagation and no explicit typed token assignment.
  + In the case of an explicit typed token assignment, we create a propagation path instance from the constrained instance object representing the outgoing propagation assignment to any condition element that references the connection target and whose constraint accepts the outgoing typed token. The enclosing outgoing propagation condition, state transition, or composite state instance of the condition element represents the component internal incoming to outgoing propagation. If there is no outgoing propagation condition involving the connection target then we find error path and error sinks with matching incoming propagations and use its constrained instance object as propagation path target.
  + In the case of no explicit typed token assignment, the actual token is determined during analysis/simulation and the propagation path instance between error propagation instances can be followed.
* In the case of error flows, we have two cases: explicit and no explicit assignment of outgoing typed tokens.
  + In the case of an explicit typed token assignment, we create a propagation path instance from the constrained instance object representing the outgoing typed token assignment to any condition element that references the connection target and whose constraint accepts the outgoing typed token. If there is no outgoing propagation condition involving the connection target then we find error path and error sinks with matching incoming propagations and use its constrained instance object as propagation path target.
  + In the case of no explicit typed token assignment, the actual token is determined during analysis/simulation and the propagation path instance between error propagation instances can be followed.

We also create propagation path instances between outgoing error propagation instances and incoming error propagation instances.

## Property Associations