ATL rules examples from AO-ADL to UML 2.0 transformations

AO-ADL Architecture to UML 2.0 Model

```
-- RULE: Architecture_to_Model
--
-- PRECONDITION : It is supposed that there is only one configuration to be
-- transformed. All the rest of configurations will be ignored.
------------------------------------------------------------------------------
rule Architecture_to_Model {
  from
    aoadl : schema!AO_ADL (aoadl.hasConfigurations())
  using {
    configuration : schema!Configuration = aoadl.getConfigurations() -> first();

    -- Transform interfaces using Interface Transformation Rule.
    umlInterfaces : Sequence(UML2!Interface) = configuration.getUsedInterfaces()
      -> collect(i | thisModule.Interface_to_Interface(i));

    -- Transform components using Component Transformation Rule.
    umlComponents : Sequence(UML2!Component) = configuration.getComponents()
      -> collect(comp | comp.componentTypeToComponents(umlInterfaces));

    -- Use base binding information from connectors to connect components.
    connectors : Sequence(schema!ConnectorType) = configuration.getConnectors();
    attachments : Sequence(schema!AttachmentType) =
      configuration.getNonAspectualAttachments();
    componentsAndConnectorsTmp : Sequence(TupleType(component : UML2!Component,
      connector : UML2!Connector)) = connectors -> collect(conn |
        conn.connectUMLComponents(umlComponents,
          attachments -> select(a | a.getConnectorType() = conn))
      ) -> flatten() -> excluding(OclUndefined);

    -- Use aspectual binding information from connectors to create <<crosscuts>> dependencies.
    crosscutsDependencies : Sequence(UML2!Dependency) = connectors ->
      collect(conn | conn.crosscutsUMLComponents(umlInterfaces, umlComponents))
      -> flatten() -> excluding(OclUndefined);
    interfacesGeneralizations : Sequence(TupleType(general : UML2!Interface,
      specific : UML2!Interface)) =
      connectors ->  collect(conn | conn.generalizeUMLInterfaces(umlInterfaces))
      -> flatten() -> excluding(OclUndefined) -> asSet() -> asSequence();

    -- Use aspectual binding information from connectors to create sequence diagrams.
    interactions : Sequence(UML2!Package) = connectors ->
      collect(conn |
        let interactionList : Sequence(UML2!PackageableElement) =
          conn.createAspectualInteractions(umlComponents,
            componentsAndConnectorsTmp -> collect(cact | cact.connector)
              -> excluding(OclUndefined),
            umlInterfaces
          ) -> flatten() -> excluding(OclUndefined) in
        if interactionList -> isEmpty() then
          OclUndefined
        else
          thisModule.CreatePackage(conn.instance_name, interactionList)
        endif
      ) -> flatten() -> excluding(OclUndefined);
  }
  to
    model : UML2!Model (
      name <- configuration.getName(),
      packagedElement <- Sequence {
        umlInterfaces,
        umlComponents,
        crosscutsDependencies,
        interactions
      } -> flatten() -> excluding(OclUndefined)
    )
}
```

AO-ADL Interface to UML 2.0 Interface

```
-- RULE: Interface_to_Interface
--------------------------------------------------------------------------------
lazy rule Interface_to_Interface {
  from
    interfaceVar : schema!Interface
  to
    outVar : UML2!Interface (
      name <- interfaceVar.name,
      ownedComment <- Sequence{
        thisModule.getComment(interfaceVar.description, 'Description'),
        thisModule.getComment(interfaceVar.uri, 'URI')
      } -> excluding(OclUndefined),
      ownedOperation <-
        if interfaceVar.operation.oclIsUndefined() then
          Sequence{}
        else
          interfaceVar.operation
            -> collect(o | thisModule.Operation_to_Operation(o))
        endif
    )
}
```

AO-ADL Component to UML 2.0 Component

```
-- This helper returns a sequence of UML2!Component that are the components that
-- map the context schema!ComponentType. This helper generates the components
-- for each instance of the ComponentType
--
-- HELPER      : componentTypeToComponents
-- CONTEXT     : schema!ComponentType
-- PARAMETERS  : umlInterfaces : Sequence(UML2!Interface)
-- RETURN      : Sequence(UML2!Component)
--------------------------------------------------------------------------------
helper context schema!ComponentType
def: componentTypeToComponents(umlInterfaces : Sequence(UML2!Interface)) : Sequence(UML2!Component) =
  let instancesList : Sequence(Integer) =
    thisModule.getMultiplicitySequence(self.multiplicity) in
  let component : schema!Component =
    thisModule.getComponentByUri(self.uri) in
  if (component.oclIsUndefined()) then
    Sequence{}
  else
    if (instancesList -> size() = 0) then
      -- If input model is correct this option will never be chosen
      thisModule.Component_to_Component(component.name, self, component, umlInterfaces)
    else
      if (instancesList -> size() = 1) then
          -- If there only one instance, not add number to the name
        thisModule.Component_to_Component(self.instance_name, self, component, umlInterfaces)
      else
        -- More than one instance, add a number to the name of each one
        instancesList -> collect(c |
          thisModule.Component_to_Component(self.instance_name + c.toString(), self, component,
            umlInterfaces)
        )
      endif
    endif
  endif;
```

AO-ADL Connector to UML 2.0 <<crosscuts>> Dependency

```
-- This helper returns a sequence of UML2!Dependency stereotyped with
-- <<crosscuts>> that represents the crosscutting behaviour contained in context
-- schema!ConnectorType.
--
-- HELPER    : crosscutsUMLComponents
-- CONTEXT    : schema!ConnectorType
-- PARAMETERS : umlInterfaces : Sequence(UML2!Interface)
-- RETURN    : Sequence(UML2!Dependency)
--------------------------------------------------------------------------------
helper context schema!ConnectorType
def: crosscutsUMLComponents(umlInterfaces : Sequence(UML2!Interface),
  umlComponents : Sequence(UML2!Component)) : Sequence(UML2!Dependency) =
  let connector : schema!Connector = thisModule.getConnectorByUri(self.uri) in
  let aspectualBindings : Sequence(schema!AspectualBinding) =
    connector.getAspectualBindings() in
  aspectualBindings -> collect(ab |
    -- Get the UML interfaces of the aspectual roles that participe in the
    -- aspectual binding ab.
    let aspectualInterfaceNames : Sequence(String) = ab.getAspecualInterfaceNames() in
    let aspectualUMLInterfaces : Sequence(UML2!Interface) = umlInterfaces
      -> select(ui | aspectualInterfaceNames -> exists(ain | ain = ui.name)) in
    -- Get the UML element (interface, usage or interface realization) of
    -- components crosscuted by the aspect in the aspectual binding ab.
    let crosscutedRoles : Sequence(TupleType(role : schema!Role, joinpoint : String)) =
      self.getCrosscuttedRoleAndJoinpoint(ab) in
    let crosscutedUMLElements : Sequence(UML2!NamedElement) = crosscutedRoles
      -> collect(cr |
        -- If joinpoint is 'BINDING' the element selected is the UML interface
        if (cr.joinpoint = 'BINDING') then
          umlInterfaces -> select(ui | cr.role.role_specification.extractName() = ui.name)
        else
          let roleAttachments : Sequence(schema!AttachmentType) =
            self.getAttachmentsThroughRole(cr.role) in
          roleAttachments -> collect(ra |
            let component : UML2!Component = umlComponents
              -> select(uc | ra.getComponentName() = uc.name) -> first() in
            -- If joinpoint is 'SOURCE' the element selected is the UML usage
            if (cr.joinpoint = 'SOURCE') then
              component.packagedElement
                -> select(pe | pe.oclIsKindOf(UML2!Usage))
                -> select(usage |
                  usage.client -> collect(c | c.name)
                    -> includes(ra.getInterfaceName())
                ) -> first()
            -- If joinpoint is 'TARGET' the element selected is the UML InterfaceRealization
            else -- (cr.joinpoint = 'TARGET')
              component.ownedAttribute
                -> select(oa | oa.name = ra.getInterfaceName())
                -> collect(port | port.type.interfaceRealization)
                -> first()
            endif
          ) -> flatten()
        endif
      ) -> flatten() in
    aspectualUMLInterfaces -> collect(aui |
      crosscutedUMLElements -> collect(cue |
        thisModule.Crosscuts(ab.name,aui,cue)
      )
    ) -> flatten() -> excluding(OclUndefined)
  ) -> flatten() -> excluding(OclUndefined);
```

AO-ADL Connector to UML 2.0 Interactions

```
-- This helper returns the sequence of UML2!Interaction that represents the
-- aspectual behaviours described in the context schema!ConnectorType and that
-- relate parameter UML components by means of parameter UML connectors.
--
-- HELPER     : createAspectualInteractions
-- CONTEXT    : schema!ConnectorType
-- PARAMETERS : components : Sequence(UML2!Component),
--              connectors : Sequence(UML2!Connector)
-- RETURN     : Sequence(UML2!PackageableElement)
--------------------------------------------------------------------------------
helper context schema!ConnectorType
def: createAspectualInteractions(components : Sequence(UML2!Component),
    connectors : Sequence(UML2!Connector), interfaces : Sequence(UML2!Interface))
  : Sequence(UML2!PackageableElement) =
  let connector : schema!Connector = thisModule.getConnectorByUri(self.uri) in
  let aspectualBindings : Sequence(schema!AspectualBinding) =
    connector.getAspectualBindings() in
  -- For each Aspectual Binding of the schema!Connector
  aspectualBindings -> collect(ab |
     let crosscuttedOps : Set(TupleType(sourcePort : schema!Role,
        targetPort : schema!Role, operation : schema!Operation)) =
      self.mapAspectualBinding(ab) in
    let adviceInfo : Sequence(TupleType(roleName : String, advice : String,
        precondition : String, postcondition : String, operator : schema!Operator,
        order : schema!Order)) =
      ab.getAspectualRoleInfo() in
    let joinpoint : String = ab.getJoinpoint() in
    -- For each operation selected by the pointcut (crosscutted operation)
    crosscuttedOps -> collect(co |
      let sources : Sequence(TupleType(component : UML2!Component, port : UML2!Port)) =
        self.getComponentAndPort(components, co.sourcePort) in
      let targets : Sequence(TupleType(component : UML2!Component, port : UML2!Port)) =
        self.getComponentAndPort(components, co.targetPort) in
      let umlConnector : UML2!Connector = connectors -> select(c |
        c.name = ab.getComponentBindingName()) -> first() in
      -- For each advice apply to those pointcuts
      let aspectuals : Sequence(UML2!Port) =
        adviceInfo -> collect(ai |
          let aspectualRole : schema!Role =
            connector.getRoleByName(ai.roleName, 'ASPECTUAL') in
          let adviceOp : UML2!Operation =
            thisModule.selectOperation(interfaces,
              aspectualRole.role_specification.extractName(), ai.advice) in
          let aspects : Sequence(TupleType(component : UML2!Component, port : UML2!Port)) =
            self.getComponentAndPort(components, aspectualRole) in
          aspects -> collect(a | Tuple{port = a.port, advice = adviceOp,
            precondition = ai.precondition,  postcondition = ai.postcondition})
        ) -> flatten() in
      let crosscuttedOp : UML2!Operation =
        if co.sourcePort.oclIsUndefined() then
          thisModule.selectOperation(interfaces,
            co.targetPort.role_specification.extractName(), co.operation.name)
        else
          thisModule.selectOperation(interfaces,
            co.sourcePort.role_specification.extractName(), co.operation.name)
        endif in
      thisModule.createInteractions(crosscuttedOp, sources, targets, aspectuals,
        joinpoint, adviceInfo -> first().operator, umlConnector)
    ) -> flatten() -> excluding(OclUndefined)
  ) -> flatten() -> excluding(OclUndefined);
```

ATL rules examples from AO-ADL to Theme/UML transformations

AO-ADL to Theme/UML

```
-- RULE: Architecture_to_Model
--
-- PRECONDITION : It is suposed that there is only one configuration to be
-- transformed. All the rest of configurations will be ignored.
------------------------------------------------------------------------------
rule Architecture_to_Model {
  from
    aoadl : schema!AO_ADL (
      aoadl.hasConfigurations()
    )
  using {
    configuration : schema!Configuration = aoadl.getConfigurations() -> first();
    components : Sequence(schema!ComponentType) =
      configuration.getComponents();
    connectors : Sequence(schema!ConnectorType) =
      configuration.getConnectors();
    attachments : Sequence(schema!AttachmentType) =
      configuration.getNonAspectualAttachments();
    aspectualAttachments : Sequence(schema!AttachmentType) =
      configuration.getAspectualAttachments();
  }
  to
    model : UML2!Model (
      name <- configuration.getName(),
      packagedElement <- Sequence {
        components -> collect(comp | comp.componentTypeToPackages()),
        connectors  -> collect(conn |
          thisModule.connectorTypeToPackage(conn,thisModule.getConnectorByUri(conn.uri))
        ),
        attachments -> collect(att | att.mergeComponentAndConnector()
        ),
        aspectualAttachments -> collect(aatt |
         aatt.bindAspectualComponentAndConnector())
      } -> flatten() -> excluding(OclUndefined)
    )
}
```

AO-ADL to Theme/UML

```
-- RULE: Interface_to_Interface
------------------------------------------------------------------------------
lazy rule Interface_to_Interface {
  from
    interfaceVar : schema!Interface
  to
    outVar : UML2!Interface (
      name <- interfaceVar.name,
      ownedComment <- Sequence{
        thisModule.getComment(interfaceVar.description, 'Description'),
        thisModule.getComment(interfaceVar.uri, 'URI')
      } -> excluding(OclUndefined),
      ownedOperation <-
        if interfaceVar.operation.oclIsUndefined() then
          Sequence{}
        else
          interfaceVar.operation
            -> collect(o | thisModule.Operation_to_Operation(o))
        endif
    )
  do {
    thisModule.UMLInterfacesList <-
      thisModule.UMLInterfacesList -> including(outVar);
  }
}
```

AO-ADL to Theme/UML

```
-- RULE: Component_to_Package
--------------------------------------------------------------------------------
lazy rule Component_to_Package {
  from
    packageName : String,
    componentName : String,
    instance : schema!ComponentType,
    component : schema!Component,
    isLast : Boolean
  using {
    interfaces : Sequence(schema!Interface) = component.getInterfaces()
      -> collect(i | i.uri.getInterface())
      -> asSet() -> asSequence()
      -> excluding(OclUndefined);
    umlInterfaces : Sequence(UML2!Interface) = interfaces
      -> collect(i | thisModule.Interface_to_Interface(i));
    umlComponent : UML2!Component =
      thisModule.Component_to_Component(componentName,instance,component);
    interfaceRealizations : Sequence(UML2!InterfaceRealization) =
      -- TODO: Fix this mess
      component.provided_interface
      --> collect(pi | thisModule.CreateInterfaceRealization(pi.getUMLInterface(),umlComponent))
      -> collect(pi | thisModule.CreateInterfaceRealization(
        umlInterfaces -> select(i | i.name = pi.uri.extractName()) -> first(),
        umlComponent))
      -> excluding(OclUndefined);
    interfaceUsages : Sequence(UML2!Usage) =
      -- TODO: Fix this mess
      component.required_interface
      --> collect(ri | thisModule.CreateUsage(ri.getUMLInterface(),umlComponent))
      -> collect(ri | thisModule.CreateUsage(
        umlInterfaces -> select(i | i.name = ri.uri.extractName()) -> first(),
        umlComponent))
      -> excluding(OclUndefined);
  }
  to
    package : UML2!Package (
      name <- packageName,
      packagedElement <- Sequence{umlComponent, umlInterfaces}
        -> flatten() -> excluding(OclUndefined)
    )
  do {
    -- Add the interface realizations to the UML component
    umlComponent.interfaceRealization <- interfaceRealizations;

    -- Add the interface usages to the UML component
    umlComponent.packagedElement <-
      umlComponent.packagedElement -> union(interfaceUsages);

    -- Store created UML2!Package to apply 'theme' stereotype later.
    thisModule.toApplyThemePool <- thisModule.toApplyThemePool
      -> append(Tuple{package = package, template = OclUndefined});

    -- Store the UML2!Package created from the instance to merge with
    -- packages created from instances of connectors which are related to
    -- (except relations of aspectual kind).
    if ((not instance.oclIsUndefined()) and isLast) {
      if (instance.hasNonAspectualAttachments(componentName)) {
        thisModule.componentPackagesToMerge <-
          thisModule.componentPackagesToMerge.including(componentName, package);
      }
    }
  }
}
```

AO-ADL to Theme/UML

```
-- RULE: Connector_to_Package
--------------------------------------------------------------------------------
lazy rule Connector_to_Package {
  from
    connectorName : String,
    instance : schema!ConnectorType,
    connector : schema!Connector
  using {
    componentMappingConnector : UML2!Component =
      thisModule.Connector_to_Component(connectorName,instance,connector);
    -- Get all the interfaces of the provided, required and aspectual roles
    -- of the AO-ADL connector without repeating them
    interfaces : Sequence(schema!Interface) = connector.getRoles()
      -> collect(i | i.role_specification.getInterface())
      -> asSet() -> asSequence()
      -> excluding(OclUndefined);
    umlInterfaces : Sequence(UML2!Interface) = interfaces
      -> collect(i | thisModule.Interface_to_Interface(i));
    interfaceRealizations : Sequence(UML2!InterfaceRealization) =
      connector.provided_role
      -> collect(pr | thisModule.CreateInterfaceRealization(
        umlInterfaces -> select(i | i.name = pr.role_specification.extractName())
          -> first()
        , componentMappingConnector))
      -> excluding(OclUndefined);
    interfaceUsages : Sequence(UML2!Usage) =
      connector.required_role
      -> union(connector.aspectual_role)
      -> collect(rr | thisModule.CreateUsage(
        umlInterfaces -> select(i | i.name = rr.role_specification.extractName())
          -> first()
        ,componentMappingConnector))
      -> excluding(OclUndefined);
    attachments : Sequence(schema!AttachmentType) =
      thisModule.getConnectorNonAspectualAttachments(instance,connectorName);
    aspectualBindings : Sequence(schema!AspectualBinding) =
      thisModule.getConnectorAspectualBindings(connector);
    -- Create one component for each attachment, but just one no matter how
    -- many times it is attached to the connector.
    attachmentsWithoutRepeatedComponents : Sequence(schema!AttachmentType) = attachments
      -> iterate(a; res : Sequence(schema!AttachmentType) = Sequence{} |
        if res -> exists(resElem | resElem.getComponentName() = a.getComponentName()) then
          res
        else
          res -> append(a)
        endif);
    umlComponents : Sequence(UML2!Component) = attachmentsWithoutRepeatedComponents
      -> collect(a | thisModule.Component_to_Component(a.getComponentName(),
        OclUndefined, thisModule.getComponentByUri(a.getComponentType().uri)));
    connexionToComponents : Sequence(UML2!Connector) =
      thisModule.connectComponents(attachments,componentMappingConnector,umlComponents);
    -- UML2!Events created for the same UML2!Connector to be included in the theme
    operationEvents : Sequence(UML2!Event) =
      if componentMappingConnector.ownedConnector.oclIsUndefined() then
        Sequence{}
      else
        componentMappingConnector.ownedConnector
          -> collect(oc | thisModule.umlConnectorEvents.get(oc))
          -> flatten() -> excluding(OclUndefined)
      endif;
    componentBindings : Sequence(schema!Binding) =
      thisModule.getConnectorBindings(connector);
    binding : String = OclUndefined;
    bindInfo : Sequence(TupleType(bindingName : String, role : String, advice : String, binding :
String)) =
      Sequence{};
    componentTmp : UML2!Component = OclUndefined;
    interfaceTmp : UML2!InterfaceCompositeComponent = OclUndefined;
    interfaceRealizationTmp : UML2!InterfaceRealization = OclUndefined;
    usageTmp : UML2!Usage = OclUndefined;
  }
  to
    package : UML2!Package (
      name <- connectorName
```

```
    ,packagedElement <- Sequence{componentMappingConnector,
       umlComponents, umlInterfaces, operationEvents,
      componentBindings
        -> collect(b | thisModule.bindingInteractions.get(b))}
      -> flatten()
      -> excluding(OclUndefined)
  )
do {
  -- Add UML InterfaceRealization to the component that maps AO-ADL connector
  componentMappingConnector.interfaceRealization <- interfaceRealizations;

  -- Add UML Usage to the component that maps AO-ADL connector
  if (componentMappingConnector.packagedElement.oclIsUndefined()) {
    componentMappingConnector.packagedElement <- interfaceUsages;
  } else {
    componentMappingConnector.packagedElement <-
      componentMappingConnector.packagedElement -> union(interfaceUsages);
  }

  -- Add UML connectors between components to the one that maps AO-ADL connector
  componentMappingConnector.ownedConnector <-  connexionToComponents;

  -- Add UML InterfaceRealization and Usage to the UML components connected
  -- to the one that maps the AO-ADL connector
  for (a in attachments) {
    componentTmp <- umlComponents -> select(c | c.name = a.getComponentName()) -> first();
    if ( a.provided_interface.oclIsUndefined() ) {
      interfaceTmp <- a.getComponent().getInterfaceByName(a.required_interface,'REQUIRED');
      if ( not interfaceTmp.oclIsUndefined() ) {
        usageTmp <- thisModule.CreateUsage(
          umlInterfaces
            -> select(i | i.name = interfaceTmp.uri.extractName())
            -> first(),
          componentTmp);
      } else {
        usageTmp <- OclUndefined;
      }
      if (not usageTmp.oclIsUndefined()) {
        if (componentTmp.packagedElement.oclIsUndefined()) {
          componentTmp.packagedElement <- Sequence{usageTmp};
        } else {
          componentTmp.packagedElement <- componentTmp.packagedElement
            -> append(usageTmp) -> excluding(OclUndefined);
        }
      }
    } else {
      interfaceTmp <- a.getComponent().getInterfaceByName(a.provided_interface,'PROVIDED');
      interfaceRealizationTmp <- thisModule.CreateInterfaceRealization(
        umlInterfaces
          -> select(i | i.name = interfaceTmp.uri.extractName())
          -> first(),
        componentTmp);
      if (not interfaceRealizationTmp.oclIsUndefined()) {
        if (componentTmp.interfaceRealization.oclIsUndefined()) {
          componentTmp.interfaceRealization <- Sequence{interfaceRealizationTmp};
        } else {
          componentTmp.interfaceRealization <- componentTmp.interfaceRealization
            -> append(interfaceRealizationTmp) -> excluding(OclUndefined);
        }
      }
    }
  }

  -- Store created UML2!Package to apply 'theme' stereotype later.
  thisModule.toApplyThemePool <- thisModule.toApplyThemePool
    -> append(Tuple{package = package, template = OclUndefined});

  -- Store the UML2!Package created from the instance to merge with
  -- packages created from component instances.
  if (not instance.oclIsUndefined()) {
    if (instance.hasNonAspectualAttachments(connectorName)) {
      thisModule.connectorPackagesToMerge <-
        thisModule.connectorPackagesToMerge.including(connectorName, package);
    }
  }

  -- Store the UML2!Package created from the instance to bind with
```

```
    -- packages created from component instances.
    if (not instance.oclIsUndefined()) {
      if (instance.hasAspectualAttachments(connectorName)) {
        for (ab in aspectualBindings) {
          binding <- instance.generateBinding(ab);
          for (tuple in ab.getAdvices()) {
            bindInfo <- bindInfo
              -> append(Tuple{bindingName = ab.name, role = tuple.roleName,
              advice = tuple.advice, binding = binding});
          }
        }
        thisModule.bindingConnectorPackages <- thisModule.bindingConnectorPackages
          -> including(connectorName,  Tuple{package = package, bindInfo = bindInfo});
      }
    }

    -- Clear the stored Interaction created for this connector
    for (b in componentBindings) {
      thisModule.bindingInteractions <-
        thisModule.bindingInteractions.including(b, OclUndefined);
    }
  }
}
```