

Objective: Find the standard ChatGPT response for view definition requests.

Views examples

1st example - View for book publication example

VPDL Language

Chat Log

Main prompt input	Output
<p>Given the view description and the following PlantUML metamodels, please give me the view definition written in VPDL</p> <p>View description: The Book metamodel has details about each chapter, while the Publication has more information about the publisher and publishing date. I want a view of my Publication that contains information regarding the chapters, both all chapters of the book and the first chapter.</p> <pre> Book PlantUML @startuml class Book [[marker:/org.eclipse.emf.ecore.diagnostic/Book_Publication/metamodels/Book.ecore?uri=platform:/resource/Book_Publication/metamodels/Book.ecore%23//Book]] { title: EString authorName: EString } class Chapter [[marker:/org.eclipse.emf.ecore.diagnostic/Book_Publication/metamodels/Book.ecore?uri=platform:/resource/Book_Publication/metamodels/Book.ecore%23//Chapter]] { title: EString nbPages: EInt } Book *--> "chapters" Chapter @enduml Publication PlantUML @startuml class Publication [[marker:/org.eclipse.emf.ecore.diagnostic/Book_Publication/metamodels/Publication.ecore?uri=platform:/resource/Book_Publication/metamodels/Publication.ecore%23//Publication]] { </pre>	<p>To create a view definition in VPDL (Viewpoint Definition Language) based on your description and the given PlantUML metamodels, we will define a view that integrates the Publication metamodel with the Book metamodel, specifically incorporating details about the chapters of the book, including all chapters and highlighting the first chapter. Below is the VPDL definition:</p> <pre> view PublicationWithChapters { import metamodel Book from "platform:/resource/Book_Publication/metamodels/Book.ecore" import metamodel Publication from "platform:/resource/Book_Publication/metamodels/Publication.ecore" mapping PublicationToBook { source Publication::Publication target Book::Book map title to title map author to authorName } </pre>

<pre> title: EString author: EString publisher: EString year: EInt } @enduml </pre>	<pre> include Book::Chapter as AllChapters in view { source Book::Book.chapters } include Book::Chapter as FirstChapter in view { source Book::Book.chapters[0] } viewAttributes { attribute viewTitle: EString attribute viewAuthor: EString attribute viewPublisher: EString attribute viewYear: EInt attribute allChapters: List<Book::Chapter> attribute firstChapter: Book::Chapter } map { viewTitle = Publication::Publication.title viewAuthor = Publication::Publication.author viewPublisher = Publication::Publication.publisher viewYear = Publication::Publication.year allChapters = Book::Book.chapters firstChapter = Book::Book.chapters[0] } } ... </pre>
---	---

	<p><i>Explanation:</i></p> <p><i>Import Statements: Import the Book and Publication metamodels from their respective URIs.</i></p> <p><i>Mapping Definitions:</i></p> <p><i>Define a mapping between Publication and Book where the title and author attributes of Publication are mapped to the title and authorName attributes of Book.</i></p> <p><i>Include Statements:</i></p> <p><i>Include all chapters from the Book metamodel (Book::Chapter as AllChapters).</i></p> <p><i>Include the first chapter from the Book metamodel (Book::Chapter as FirstChapter).</i></p> <p><i>View Attributes: Define attributes for the view to hold the information about the publication and the chapters.</i></p> <p><i>Map Statements: Map the attributes from the Publication and Book metamodels to the view attributes.</i></p> <p><i>This VPDL definition provides a structured way to create a view of Publication that includes detailed information about the chapters of the book, fulfilling the requirements specified in your description.</i></p>
--	--

Link for the Chat:

<https://chatgpt.com/share/6aa14345-4145-4473-801e-9dc24e5b5cae>

QVT Language

Main prompt input	Output
<p><i>Given the view description and the following PlantUML metamodels, please give me the view definition written in QVT</i></p> <p><i>{Same as VPDL example}</i></p>	<p><i>To create a QVT (Query/View/Transformation) view definition that merges information from the `Book` and `Publication` metamodels and includes details about the chapters, including all chapters and the first chapter specifically, we can follow these steps:</i></p> <ol style="list-style-type: none"> <i>1. **Define the source and target models**.</i> <i>2. **Map the relevant elements from the source model (Book) to the target model (Publication)**.</i> <i>3. **Include transformation rules to handle the chapters</i>

*information**.*

Here is a QVT-O (Operational Mappings) transformation that accomplishes this:

```
```qvto
```

```
modeltype BookModel uses
"platform:/resource/Book_Publication/metamodels/Book
.ecore";
```

```
modeltype PublicationModel uses
"platform:/resource/Book_Publication/metamodels/Publi
cation.ecore";
```

```
transformation BookToPublication(in bookModel:
BookModel, out pubModel: PublicationModel);
```

```
main() {
```

```
 bookModel.rootObjects()[Book]->map
BookToPublication();
```

```
}
```

```
mapping Book::Book to Publication::Publication : pub {
```

```
 title := self.title;
```

```
 author := self.authorName;
```

```
 publisher := ""; // Assume publisher information
needs to be filled or can be derived from elsewhere
```

```
 year := 0; // Assume publication year needs to
be filled or can be derived from elsewhere
```

```
 chapters := self.chapters->collect(c | c.map
ChapterToPublicationChapter());
```

```
 firstChapter := self.chapters->first().map
ChapterToPublicationChapter();
```

```
}
```

```

mapping Book::Chapter to
Publication::PublicationChapter : pubChapter {

 title := self.title;

 nbPages := self.nbPages;

}

```

```

helper Book::Book::firstChapter() : Chapter {

 return self.chapters->first();

}

...

```

### Explanation:

#### 1. **Model Types**:

- `'BookModel'` and `'PublicationModel'` are defined to use the respective metamodels from the provided URIs.

#### 2. **Transformation**:

- The `'BookToPublication'` transformation is defined with the `'bookModel'` as the input model and the `'pubModel'` as the output model.

- The `'main'` function starts the transformation by mapping the root objects of the `'bookModel'` that are of type `'Book'` to the `'Publication'` type in the `'pubModel'`.

#### 3. **Mapping Rules**:

- `'Book::Book'` is mapped to `'Publication::Publication'`. The title and author attributes are directly mapped, while placeholders are used for publisher and year (assuming these need to be filled in later or derived from other information).

- The `'chapters'` attribute is mapped by

	<p>collecting all <code>`Chapter`</code> objects from the <code>`Book`</code> and mapping each to a <code>`PublicationChapter`</code> in the <code>`Publication`</code>.</p> <p>- <code>`firstChapter`</code> is a specific transformation that maps only the first chapter from the <code>`Book`</code> to a <code>`PublicationChapter`</code> in the <code>`Publication`</code>.</p> <p>4. <b><code>**Helper**</code></b>:</p> <p>- The <code>`firstChapter`</code> helper function returns the first chapter of the book.</p> <p>This QVT transformation captures the requirements of the view by including both all chapters and specifically the first chapter of the book into the publication model. Adjustments can be made based on more specific requirements for publisher and year attributes.</p>
--	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Link for the Chat:

<https://chatgpt.com/share/0323f8af-a16f-4d55-be57-1c0e4f4898b7>

## 2nd example - EA Application

VPDL Language

Chat Log

Main prompt	Example outputs
<p>Given the view description and the following PlantUML metamodels, please give me the view definition written in VPDL.</p> <p>View description: The considered view combines an Enterprise Architecture model and a Requirements model (specified in a CSV file that can be opened in a spreadsheet such as Excel). We want to use Requirements to comprehend the enterprise elements better.</p> <pre> Enterprise Architecture PlantUML @startuml class EnterpriseArchitecture { } abstract class Architecture { } class BusinessArchitecture { } class DataArchitecture { } class TechnologyArchitecture { } class Driver { } class Goal { } class Objective { } class OrganizationUnit {     headcount: EString } class Actor {     FTEs: EString     actorGoal: EString     actorTasks: EString } class Role {     estimatedFTEs: EString } class DataEntity {     dataEntityCategory: DataEntityCategory     privacyClassification: EString     retentionClassification: EString } class LogicalApplicationComponent { } class Function { } class BusinessService { } class Process {     processCriticality: EString     isAutomated: EBoolean     processVolumetrics: EString } class PlatformService {     categoryTRM: EString } </pre>	<p>The view definition in VPDL for combining an Enterprise Architecture model and a Requirements model, based on the provided PlantUML metamodels, is as follows:</p> <pre> ```vpdl  view EnterpriseArchitectureRequirementsView {      description "This view combines an Enterprise Architecture model and a Requirements model to use Requirements for better comprehension of enterprise elements."      // Enterprise Architecture Metamodel Elements      metamodel EnterpriseArchitecture {          classes {              Architecture,              BusinessArchitecture,              DataArchitecture,              TechnologyArchitecture,              Driver,              Goal,              Objective,              OrganizationUnit,              Actor,              Role,              DataEntity,              LogicalApplicationComponent,              Function,              BusinessService,              Process, </pre>

```

class PhysicalTechnologyComponent {
 productName: EString
 moduleName: EString
 vendor: EString
 version: EString
 categoryTRM: EString
}
class Product {
}
class Measure {
}
class ServiceQuality {
}
class Contract {
 behaviorCharacteristics: EString
 serviceNameCaller: EString
 serviceNameCalled: EString
 serviceQualityCharacteristics: EString
 availabilityQualityCharacteristics: EString
 servicesTimes: EString
 manageabilityCharacteristics: EString
 serviceabilityCharacteristics: EString
 performanceCharacteristics: EString
 responseCharacteristics: EString
 reliabilityCharacteristics: EString
 qualityOfInformationRequired: EString
 contractControlRequirements: EString
 resultControlRequirements: EString
 recoverabilityCharacteristics: EString
 locatabilityCharacteristics: EString
 securityCharacteristics: EString
 privacyCharacteristics: EString
 integrityCharacteristics: EString
 credibilityCharacteristics: EString
 localizationCharacteristics: EString
 internationalizationCharacteristics: EString
 interoperabilityCharacteristics: EString
 scalabilityCharacteristics: EString
 portabilityCharacteristics: EString
 extensibilityCharacteristics: EString
 capacityCharacteristics: EString
 throughput: EString
 throughputPeriod: EString
 growth: EString
 growthPeriod: EString
 peakProfileShortTerm: EString
 peakProfileLongTerm: EString
}
class Event {
}
class Control {
}
class Element {
 name: EString
 description: EString
 sourceDescr: EString
 ownerDescr: EString
 ID: EString
}
class Container {
 name: EString

```

```

PlatformService,

PhysicalTechnologyComponent,

Product,

Measure,

ServiceQuality,

Contract,

Event,

Control,

Element,

Container,

Label,

Location,

Capability,

Principle,

Constraint,

Assumption,

Requirement,

Gap,

WorkPackage,

LogicalDataComponent,

PhysicalDataComponent,

ApplicationArchitecture,

PhysicalApplicationComponent,

LogicalTechnologyComponent,

StrategicArchitecture

}

enums {

```



<pre>         id: EString         description: EString     }     class Label {         name: EString         id: EString         description: EString     }     class Location {     }     class Capability {         businessValue: EString         increments: EString     }     }     abstract class StrategicElement {     }     class Principle {         principleCategory: PrincipleCategory         priority: EString         statementOfPrinciple: EString         rationale: EString         implication: EString         metric: EString     }     class Constraint {     }     class Assumption {     }     class Requirement {         statementOfRequirement: EString         rationale: EString         acceptanceCriteria: EString     }     }     class Gap {     }     class WorkPackage {         workPackageCategory:         WorkPackageCategory     }     class LogicalDataComponent {     }     class PhysicalDataComponent {     }     class ApplicationArchitecture {     }     class PhysicalApplicationComponent {         lifeCycleStatus: LifeCycleStatus         initialLiveDate: EDate         dateOfLastRelease: EDate         dateOfNextRelease: EDate         retirementDate: EDate         availabilityQualityCharacteristics: EString         servicesTimes: EString         manageabilityCharacteristics: EString         serviceabilityCharacteristics: EString         performanceCharacteristics: EString         reliabilityCharacteristics: EString         recoverabilityCharacteristics: EString         locatabilityCharacteristics: EString         securityCharacteristics: EString         privacyCharacteristics: EString </pre>	<pre>         PrincipleCategory,         StandardsClass,         LifeCycleStatus,         DataEntityCategory,         WorkPackageCategory     }      relationships {          EnterpriseArchitecture [architectures *] -&gt;         Architecture,          EnterpriseArchitecture [containers *] -&gt;         Container,          EnterpriseArchitecture [labels *] -&gt; Label,          BusinessArchitecture [drivers *] -&gt; Driver,          BusinessArchitecture [goals *] -&gt; Goal,          BusinessArchitecture [objectives *] -&gt;         Objective,          BusinessArchitecture [units *] -&gt;         OrganizationUnit,          BusinessArchitecture [actors *] -&gt; Actor,          BusinessArchitecture [roles *] -&gt; Role,          BusinessArchitecture [functions *] -&gt; Function,          BusinessArchitecture [services *] -&gt;         BusinessService,          BusinessArchitecture [processes *] -&gt; Process,          BusinessArchitecture [controls *] -&gt; Control,          BusinessArchitecture [events *] -&gt; Event,          BusinessArchitecture [locations *] -&gt; Location,          BusinessArchitecture [products *] -&gt; Product,          BusinessArchitecture [contracts *] -&gt; Contract,          BusinessArchitecture [measures *] -&gt; </pre>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

```

 integrityCharacteristics: EString
 credibilityCharacteristics: EString
 localizationCharacteristics: EString
 internationalizationCharacteristics: EString
 interoperabilityCharacteristics: EString
 scalabilityCharacteristics: EString
 portabilityCharacteristics: EString
 extensibilityCharacteristics: EString
 capacityCharacteristics: EString
 throughput: EString
 throughputPeriod: EString
 growth: EString
 growthPeriod: EString
 peakProfileShortTerm: EString
 peakProfileLongTerm: EString
}
class LogicalTechnologyComponent {
 categoryTRM: EString
}
class StrategicArchitecture {
}
enum PrincipleCategory {
 GuidingPrinciple: GuidingPrinciple
 BusinessPrinciple: BusinessPrinciple
 DataPrinciple: DataPrinciple
 ApplicationPrinciple: ApplicationPrinciple
 IntegrationPrinciple: IntegrationPrinciple
 TechnologyPrinciple: TechnologyPrinciple
}
enum StandardsClass {
 NonStandard: NonStandard
 Proposed: Proposed
 Provisional: Provisional
 Standard: Standard
 PhasingOut: PhasingOut
 Retired: Retired
}
abstract class Standard {
 standardClass: StandardsClass
 standardCreationDate: EDate
 lastStandardCreationDate: EDate
 nextStandardCreationDate: EDate
 retireDate: EDate
}
abstract class Service {
}
abstract class ApplicationComponent {
}
class InformationSystemService {
}
enum LifeCycleStatus {
 Proposed: Proposed
 InDevelopment: InDevelopment
 Live: Live
 PhasingOut: PhasingOut
 Retired: Retired
}
enum DataEntityCategory {
 Message: Message
 InternallyStoredEntity: InternallyStoredEntity
}

```

```

Measure,

 BusinessArchitecture [servicesQuality *] ->
ServiceQuality,

 DataArchitecture [entities *] -> DataEntity,

 DataArchitecture [logicalComponents *] ->
LogicalDataComponent,

 DataArchitecture [physicalComponents *] ->
PhysicalDataComponent,

 TechnologyArchitecture [platformServices *] ->
PlatformService,

 TechnologyArchitecture [physicalComponents
*] -> PhysicalTechnologyComponent,

 TechnologyArchitecture [logicalComponents *]
-> LogicalTechnologyComponent,

 Driver [addressesDrivers *] -> Goal,

 Driver [isMotivatedByDrivers *] ->
OrganizationUnit,

 Goal [realizesGoals *] -> Objective,

 Goal [isRealizedThroughObjectives *] ->
Measure,

 OrganizationUnit
[isOwnedAndGovernedByOrganizationUnits *] ->
Service,

 OrganizationUnit [containsActors *] -> Actor,

 OrganizationUnit [ownsFunctions *] ->
Function,

 OrganizationUnit [involvesOrganizationUnits *]
-> Process,

 OrganizationUnit [producesProducts *] ->
Product,

 OrganizationUnit [containsOrganizationUnits *]
-> Location,

 Actor [isSuppliedByActors *] -> DataEntity,

 Actor [isConsumedByActors *] -> DataEntity,

 Actor [supportsActors *] -> Function,

```

```

abstract class DataComponent {
}
abstract class TechnologyComponent {
}
enum WorkPackageCategory {
 WorkPackage: WorkPackage
 WorkStream: WorkStream
 Project: Project
 Program: Program
 Portofolio: Portofolio
}
Architecture <|-- BusinessArchitecture
Architecture <|-- DataArchitecture
Architecture <|-- TechnologyArchitecture
Element <|-- Driver
Element <|-- Goal
Element <|-- Objective
Element <|-- OrganizationUnit
Element <|-- Actor
Element <|-- Role
Element <|-- DataEntity
Element <|-- LogicalApplicationComponent
ApplicationComponent <|--
LogicalApplicationComponent
Element <|-- Function
Standard <|-- Function
Element <|-- BusinessService
Service <|-- BusinessService
Element <|-- Process
Standard <|-- Process
Element <|-- PlatformService
Service <|-- PlatformService
Element <|-- PhysicalTechnologyComponent
TechnologyComponent <|--
PhysicalTechnologyComponent
Element <|-- Product
Element <|-- Measure
Element <|-- ServiceQuality
Element <|-- Contract
Element <|-- Event
Element <|-- Control
Element <|-- Location
Element <|-- Capability
Element <|-- StrategicElement
StrategicElement <|-- Principle
StrategicElement <|-- Constraint
StrategicElement <|-- Assumption
StrategicElement <|-- Requirement
StrategicElement <|-- Gap
StrategicElement <|-- WorkPackage
Element <|-- LogicalDataComponent
DataComponent <|-- LogicalDataComponent
Element <|-- PhysicalDataComponent
DataComponent <|-- PhysicalDataComponent
Architecture <|-- ApplicationArchitecture
Element <|-- PhysicalApplicationComponent
ApplicationComponent <|--
PhysicalApplicationComponent
Element <|-- LogicalTechnologyComponent
TechnologyComponent <|--
LogicalTechnologyComponent

```

```

Actor [performsTaskInRoles *] -> Role,

Actor [involvesActors *] -> Process,

Actor [isResolvedByActors *] -> Event,

Actor [isGeneratedByActors *] -> Event,

Actor [containsActors *] -> Location,

Role [canBeAccessedByRoles *] -> Function,

DataEntity [consumesEntities *] -> Service,

DataEntity [providesEntities *] -> Service,

DataEntity [encapsulatesDataEntities *] ->
LogicalDataComponent,

DataEntity
[isProcessesByLogicalApplicationComponents *] ->
LogicalApplicationComponent,

LogicalApplicationComponent
[isRealizedThroughLogicalApplicationComponent *] ->
Service,

LogicalApplicationComponent
[extendsLogicalApplicationComponents *] ->
PhysicalApplicationComponent,

Function
[providesGovernedInterfaceToAccessFunctions *] ->
Service,

Function [supportsProcesses *] -> Process,

BusinessService [supportsObjective *] ->
Objective,

Process [isRealizedByProcesses *] -> Service,

Process
[ensuresCorrectOperationOfProcesses *] -> Control,

Process [isResolvedByProcesses *] -> Event,

Process [isProducedByProcesses *] ->
Product,

PlatformService [suppliesPlatformServices *]
-> LogicalTechnologyComponent,

PhysicalTechnologyComponent
[realizesApplicationComponents *] ->
PhysicalApplicationComponent,

Measure [isTrackedAgainstMeasures *] ->

```

Architecture <|-- StrategicArchitecture  
 Standard <|-- Service  
 Standard <|-- ApplicationComponent  
 Service <|-- InformationSystemService  
 Element <|-- InformationSystemService  
 Standard <|-- DataComponent  
 Standard <|-- TechnologyComponent  
 EnterpriseArchitecture \*--> "architectures \*" Architecture  
 EnterpriseArchitecture \*--> "containers \*" Container  
 EnterpriseArchitecture \*--> "labels \*" Label  
 BusinessArchitecture \*--> "drivers \*" Driver  
 BusinessArchitecture \*--> "goals \*" Goal  
 BusinessArchitecture \*--> "objectives \*" Objective  
 BusinessArchitecture \*--> "units \*" OrganizationUnit  
 BusinessArchitecture \*--> "actors \*" Actor  
 BusinessArchitecture \*--> "roles \*" Role  
 BusinessArchitecture \*--> "functions \*" Function  
 BusinessArchitecture \*--> "services \*" BusinessService  
 BusinessArchitecture \*--> "processes \*" Process  
 BusinessArchitecture \*--> "controls \*" Control  
 BusinessArchitecture \*--> "events \*" Event  
 BusinessArchitecture \*--> "locations \*" Location  
 BusinessArchitecture \*--> "products \*" Product  
 BusinessArchitecture \*--> "contracts \*" Contract  
 BusinessArchitecture \*--> "measures \*" Measure  
 BusinessArchitecture \*--> "servicesQuality \*" ServiceQuality  
 DataArchitecture \*--> "entities \*" DataEntity  
 DataArchitecture \*--> "logicalComponents \*" LogicalDataComponent  
 DataArchitecture \*--> "physicalComponents \*" PhysicalDataComponent  
 TechnologyArchitecture \*--> "platformServices \*" PlatformService  
 TechnologyArchitecture \*--> "physicalComponents \*" PhysicalTechnologyComponent  
 TechnologyArchitecture \*--> "logicalComponents \*" LogicalTechnologyComponent  
 Driver "addressesDrivers \*" -- "createsGoals \*" Goal  
 Driver "isMotivatedByDrivers \*" -- "motivatesOrganizationUnits \*" OrganizationUnit  
 Driver --> "decomposesDriver" Driver  
 Goal "realizesGoals \*" -- "isRealizedThroughObjectives \*" Objective  
 Goal --> "decomposesGoal" Goal  
 Objective "setsPerformanceCriteriaForObjectives \*" -- "isTrackedAgainstMeasures \*" Measure  
 Objective --> "decomposesObjective" Objective  
 OrganizationUnit  
 "isOwnedAndGovernedByOrganizationUnits \*" -- "ownsAndGovernsServices \*" Service  
 OrganizationUnit "belongsTo" -- "containsActors \*" Actor  
 OrganizationUnit "isOwnedByUnit" -- "ownsFunctions \*" Function  
 OrganizationUnit "involvesOrganizationUnits \*" -- "participatesInProcesses \*" Process  
 OrganizationUnit "isProducedByOrganizationUnits \*" -- "producesProducts \*" Product  
 OrganizationUnit "containsOrganizationUnits \*" -- "operatesInLocation" Location  
 Actor "isSuppliedByActors \*" -- "suppliesEntities \*"

Service,  
 ServiceQuality [meetsQualities \*] -> Service,  
 ServiceQuality [meetsServiceQuality \*] -> Contract,  
 Contract  
 [isGovernedAndMeasuredByContracts \*] -> Service,  
 Event [resolvesEvents \*] -> Service,  
 Element [ownedElements \*] -> Label,  
 Container [subContainers \*] -> Container,  
 Container [containers \*] -> Label,  
 Label [subLabels \*] -> Label,  
 Location [isHotedInLocation] -> PhysicalDataComponent,  
 Location [isHostedInLocation \*] -> PhysicalApplicationComponent,  
 Capability [deliversCapabilities \*] -> WorkPackage,  
 LogicalDataComponent  
 [extendsLogicalDataComponents \*] -> PhysicalDataComponent,  
 PhysicalDataComponent  
 [encapsulatesPhysicalApplicationComponents \*] -> PhysicalApplicationComponent,  
 ApplicationArchitecture  
 [logicalApplicationComponents \*] -> LogicalApplicationComponent,  
 ApplicationArchitecture  
 [physicalApplicationComponents \*] -> PhysicalApplicationComponent,  
 ApplicationArchitecture  
 [informationSystemServices \*] -> InformationSystemService,  
 PhysicalApplicationComponent  
 [communicatesWith \*] -> PhysicalApplicationComponent,  
 PhysicalApplicationComponent  
 [encapsulatesPhysicalDataComponents \*] -> PhysicalDataComponent,  
 PhysicalApplicationComponent

```

DataEntity
Actor "isConsumedByActors *" -- "consumesEntities *"
DataEntity
Actor "supportsActors *" -- "interactsWithFunctions *"
Function
Actor "isAssumedByActors *" -- "performsTaskInRoles *"
Role
Actor "involvesActors *" -- "participatesInProcesses *"
Process
Actor --> "consumesServices *" Service
Actor "isResolvedByActors *" -- "resolvesEvents *"
Event
Actor "isGeneratedByActors" -- "generatesEvents *"
Event
Actor "containsActors *" -- "operatesInLocation"
Location
Actor "isPerformedByActors *" -- "performsFunctions *"
Function
Actor --> "decomposesActors *" Actor
Role "canBeAccessedByRoles *" -- "accessesFunctions *"
Function
Role --> "decomposesRole" Role
DataEntity "consumesEntities *" --
"isAccessedByServices *" Service
DataEntity "providesEntities *" --
"isUpdatedThroughServices *" Service
DataEntity "encapsulatesDataEntities *" --
"residesWithinLogicalDataComponent"
LogicalDataComponent
DataEntity "operatesOnDataEntities *" --
"isProcessesByLogicalApplicationComponents *"
LogicalApplicationComponent
DataEntity --> "decomposeEntity" DataEntity
DataEntity --> "relatesTo *" DataEntity
LogicalApplicationComponent
"isRealizedThroughLogicalApplicationComponent *" --
"implementsServices *" Service
LogicalApplicationComponent
"extendsLogicalApplicationComponents *" --
"isExtendedByPhysicalApplicationComponents *"
PhysicalApplicationComponent
LogicalApplicationComponent --> "communicatesWith *"
LogicalApplicationComponent
LogicalApplicationComponent -->
"decomposesLogicalApplicationComponent"
LogicalApplicationComponent
Function
"providesGovernedInterfaceToAccessFunctions *" --
"isBoundedByServices *" Service
Function "decomposesFunctions *" --
"supportsProcesses *" Process
Function "orchestratesFunctions *" --
"isRealizedByProcesses *" Process
Function --> "decomposesFunction" Function
Function --> "communicatedWithFunctions *" Function
BusinessService --> "supportsObjective *" Objective
Process "isRealizedByProcesses *" --
"orchestratesServices *" Service
Process "supportsProcesses *" --
"decomposesServices *" Service
Process "ensuresCorrectOperationOfProcesses *" --

```

```

[isRealizedByPhysicalTechnologyComponents *] ->
PhysicalTechnologyComponent,

LogicalTechnologyComponent
[isImplementedOnLogicalTechnologyComponents *] ->
Service,

StrategicArchitecture [capabilities *] ->
Capability,

StrategicArchitecture [strategicElements *] ->
StrategicElement,

Service [isProvidedToActors *] -> Actor,

Service [consumesServices *] -> Service

}

}

// ReqIF Metamodel Elements

metamodel ReqIF {

classes {

AccessControlledElement,

Identifiable,

AttributeValueXHTML,

AttributeValue,

SpecElementWithAttributes,

AttributeDefinitionXHTML,

AttributeDefinition,

SpecType,

ReqIFContent,

ReqIF,

ReqIFHeader,

ReqIFToolExtension,

SpecObject,

SpecObjectType,

```

*"isGuidedByControls \*\* Control  
 Process "isResolvedByProcesses \*\* -- "resolvesEvents  
 \*\* Event  
 Process "isGeneratedByProcesses \*\* --  
 "generatesEvents \*\* Event  
 Process "isProducedByProcesses \*\* --  
 "producesProducts \*\* Product  
 Process --> "decomposesProcess" Process  
 PlatformService "suppliesPlatformServices \*\* --  
 "isSuppliedByLogicalTechnologyComponents \*\*  
 LogicalTechnologyComponent  
 PhysicalTechnologyComponent -->  
 "realizesApplicationComponents \*\*  
 PhysicalApplicationComponent  
 PhysicalTechnologyComponent  
 "isRealizedByPhysicalTechnologyComponents \*\* --  
 "extendsLogicalTechnologyComponents \*\*  
 LogicalTechnologyComponent  
 PhysicalTechnologyComponent  
 "containsPhysicalTechnologyComponents \*\* --  
 "isHostedInLocation \*\* Location  
 PhysicalTechnologyComponent -->  
 "decomposesPhysicalTechnologyComponent"  
 PhysicalTechnologyComponent  
 PhysicalTechnologyComponent -->  
 "isDependentOnPhysicalTechnologyComponents \*\*  
 PhysicalTechnologyComponent  
 Measure "isTrackedAgainstMeasures \*\* --  
 "setsPerformanceCriteriaForServices \*\* Service  
 Measure --> "decomposesMeasure" Measure  
 ServiceQuality "meetsQualities \*\* -- "appliesToServices  
 \*\* Service  
 ServiceQuality "meetsServiceQuality \*\* --  
 "appliesToContracts \*\* Contract  
 Contract "isGovernedAndMeasuredByContracts \*\* --  
 "governsAndMeasuresBusinessServices \*\* Service  
 Event "resolvesEvents \*\* --  
 "isResolvedByBusinessServices \*\* Service  
 Element "ownedElements \*\* -- "category \*\* Label  
 Container \*--> "subContainers \*\* Container  
 Container "containers \*\* -- "labels \*\* Label  
 Label \*--> "subLabels \*\* Label  
 Location "isHotedInLocation" --  
 "containsPhysicalDataComponents \*\*  
 PhysicalDataComponent  
 Location "isHostedInLocation \*\* --  
 "containsPhysicalApplicationComponents \*\*  
 PhysicalApplicationComponent  
 Location --> "decomposesLocation" Location  
 Capability "deliversCapabilities \*\* -- "isDeliveredBy"  
 WorkPackage  
 LogicalDataComponent  
 "extendsLogicalDataComponents \*\* --  
 "isExtendedByPhysicalDataComponents \*\*  
 PhysicalDataComponent  
 PhysicalDataComponent -->  
 "decomposesPhysicalDataComponent"  
 PhysicalDataComponent  
 PhysicalDataComponent -->  
 "encapsulatesPhysicalApplicationComponents \*\*  
 PhysicalApplicationComponent*

*Specification,  
 SpecificationType,  
 SpecHierarchy,  
 DatatypeDefinition,  
 SpecRelation,  
 SpecRelationType,  
 RelationGroup,  
 RelationGroupType,  
 DatatypeDefinitionXHTML,  
 AlternativeID,  
 AttributeDefinitionBoolean,  
 AttributeDefinitionSimple,  
 DatatypeDefinitionBoolean,  
 DatatypeDefinitionSimple,  
 AttributeValueBoolean,  
 AttributeValueSimple,  
 AttributeDefinitionDate,  
 DatatypeDefinitionDate,  
 AttributeValueDate,  
 AttributeDefinitionEnumeration,  
 DatatypeDefinitionEnumeration,  
 EnumValue,  
 EmbeddedValue,  
 AttributeValueEnumeration,  
 AttributeDefinitionInteger,  
 DatatypeDefinitionInteger,  
 AttributeValueInteger,  
 AttributeDefinitionReal,*

```

ApplicationArchitecture *-->
"logicalApplicationComponents *"
LogicalApplicationComponent
ApplicationArchitecture *-->
"physicalApplicationComponents *"
PhysicalApplicationComponent
ApplicationArchitecture *-->
"informationSystemServices *"
InformationSystemService
PhysicalApplicationComponent --> "communicatesWith
*" PhysicalApplicationComponent
PhysicalApplicationComponent -->
"encapsulatesPhysicalDataComponents *"
PhysicalDataComponent
PhysicalApplicationComponent -->
"isRealizedByPhysicalTechnologyComponents *"
PhysicalTechnologyComponent
PhysicalApplicationComponent -->
"decomposesPhysicalApplicationComponent"
PhysicalApplicationComponent
LogicalTechnologyComponent
"isImplementedOnLogicalTechnologyComponents *" --
"providesPlatformForServices *" Service
LogicalTechnologyComponent -->
"decomposesLogicalTechnologyComponent"
LogicalTechnologyComponent
LogicalTechnologyComponent -->
"isDependentOnLogicalTechnologyComponents *"
LogicalTechnologyComponent
StrategicArchitecture *--> "capabilities *" Capability
StrategicArchitecture *--> "strategicElements *"
StrategicElement
Service --> "isProvidedToActors *" Actor
Service --> "consumesServices *" Service
Service --> "decomposesServices *" Service
@enduml

```

ReqIF metamodel

@startuml

```

abstract class AccessControlledElement {
 editable: EBoolean
}
abstract class Identifiable {
 desc: EString
 identifier: ID
 lastChange: DateTime
 longName: EString
}
class AttributeValueXHTML {
 simplified: EBoolean
}
abstract class AttributeValue {
}
abstract class SpecElementWithAttributes {
}
class AttributeDefinitionXHTML {
}
abstract class AttributeDefinition {
}
abstract class SpecType {
}

```

```

DatatypeDefinitionReal,
AttributeValueReal,
AttributeDefinitionString,
DatatypeDefinitionString,
AttributeValueString,
XhtmlContent,
DateTime,
ID
}

relationships {

 Identifiable [alternativeID] -> AlternativeID,

 AttributeValueXHTML [definition] ->
AttributeDefinitionXHTML,

 AttributeValueXHTML [theOriginalValue] ->
XhtmlContent,

 AttributeValueXHTML [theValue] ->
XhtmlContent,

 SpecElementWithAttributes [values *] ->
AttributeValue,

 AttributeDefinitionXHTML [type] ->
DatatypeDefinitionXHTML,

 AttributeDefinitionXHTML [defaultValue] ->
AttributeValueXHTML,

 SpecType [specAttributes *] ->
AttributeDefinition,

 ReqIFContent [datatypes *] ->
DatatypeDefinition,

 ReqIFContent [specTypes *] -> SpecType,

 ReqIFContent [specObjects *] -> SpecObject,

 ReqIFContent [specRelations *] ->
SpecRelation,

 ReqIFContent [specifications *] ->

```

```

class ReqIFContent {
}
class ReqIF {
 lang: LangType
}
class ReqIFHeader {
 comment: EString
 creationTime: DateTime
 identifier: ID
 repositoryId: EString
 reqIFToolId: EString
 reqIFVersion: EString
 sourceToolId: EString
 title: EString
}
class ReqIFToolExtension {
}
class SpecObject {
}
class SpecObjectType {
}
class Specification {
}
class SpecificationType {
}
class SpecHierarchy {
 tableInternal: EBoolean
}
abstract class DatatypeDefinition {
}
class SpecRelation {
}
class SpecRelationType {
}
class RelationGroup {
}
class RelationGroupType {
}
class DatatypeDefinitionXHTML {
}
class AlternativeID {
 identifier: EString
}
class AttributeDefinitionBoolean {
}
abstract class AttributeDefinitionSimple {
}
class DatatypeDefinitionBoolean {
}
abstract class DatatypeDefinitionSimple {
}
class AttributeValueBoolean {
 theValue: EBoolean
}
abstract class AttributeValueSimple {
}
class AttributeDefinitionDate {
}
class DatatypeDefinitionDate {
}
class AttributeValueDate {
}

```

```

Specification,

 ReqIFContent [specRelationGroups *] ->
RelationGroup,

 ReqIF [theHeader] -> ReqIFHeader,

 ReqIF [coreContent] -> ReqIFContent,

 ReqIF [toolExtensions *] ->
ReqIFToolExtension,

 SpecObject [type] -> SpecObjectType,

 Specification [type] -> SpecificationType,

 Specification [children *] -> SpecHierarchy,

 SpecHierarchy [object] -> SpecObject,

 SpecHierarchy [children *] -> SpecHierarchy,

 SpecHierarchy [editableAtts *] ->
AttributeDefinition,

 SpecRelation [target] -> SpecObject,

 SpecRelation [source] -> SpecObject,

 SpecRelation [type] -> SpecRelationType,

 RelationGroup [specRelations *] ->
SpecRelation,

 RelationGroup [type] -> RelationGroupType,

 RelationGroup [sourceSpecification] ->
Specification,

 RelationGroup [targetSpecification] ->
Specification,

 AttributeDefinitionBoolean [type] ->
DatatypeDefinitionBoolean,

 AttributeDefinitionBoolean [defaultValue] ->
AttributeValueBoolean,

 AttributeValueBoolean [definition] ->
AttributeDefinitionBoolean,

 AttributeDefinitionDate [type] ->
DatatypeDefinitionDate,

 AttributeDefinitionDate [defaultValue] ->
AttributeValueDate,

```



<pre>         theValue: DateTime     }     class AttributeDefinitionEnumeration {         multiValued: EBoolean     }     class DatatypeDefinitionEnumeration {     }     class EnumValue {     }     class EmbeddedValue {         key: EBigInteger         otherContent: EString     }     class AttributeValueEnumeration {     }     class AttributeDefinitionInteger {     }     class DatatypeDefinitionInteger {         max: EBigInteger         min: EBigInteger     }     class AttributeValueInteger {         theValue: EBigInteger     }     class AttributeDefinitionReal {     }     class DatatypeDefinitionReal {         accuracy: EBigInteger         max: EDouble         min: EDouble     }     class AttributeValueReal {         theValue: EDouble     }     class AttributeDefinitionString {     }     class DatatypeDefinitionString {         maxLength: EBigInteger     }     class AttributeValueString {         theValue: EString     }     class XhtmlContent {         / xhtmlSource: EString     }     class DateTime {         java.util.GregorianCalendar     }     class ID {         java.lang.String     }     Identifiable &lt; -- AccessControlledElement     AttributeValue &lt; -- AttributeValueXHTML     Identifiable &lt; -- SpecElementWithAttributes     AttributeDefinition &lt; -- AttributeDefinitionXHTML     AccessControlledElement &lt; -- AttributeDefinition     Identifiable &lt; -- SpecType     SpecElementWithAttributes &lt; -- SpecObject     SpecType &lt; -- SpecObjectType     SpecElementWithAttributes &lt; -- Specification     SpecType &lt; -- SpecificationType </pre>	<pre>         AttributeValueDate [definition] -&gt;         AttributeDefinitionDate,          AttributeDefinitionEnumeration [type] -&gt;         DatatypeDefinitionEnumeration,          AttributeDefinitionEnumeration [defaultValue </pre>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

*AccessControlledElement* <|-- *SpecHierarchy*  
*Identifiable* <|-- *DatatypeDefinition*  
*SpecElementWithAttributes* <|-- *SpecRelation*  
*SpecType* <|-- *SpecRelationType*  
*Identifiable* <|-- *RelationGroup*  
*SpecType* <|-- *RelationGroupType*  
*DatatypeDefinition* <|-- *DatatypeDefinitionXHTML*  
*AttributeDefinitionSimple* <|-- *AttributeDefinitionBoolean*  
*AttributeDefinition* <|-- *AttributeDefinitionSimple*  
*DatatypeDefinitionSimple* <|--  
*DatatypeDefinitionBoolean*  
*DatatypeDefinition* <|-- *DatatypeDefinitionSimple*  
*AttributeValueSimple* <|-- *AttributeValueBoolean*  
*AttributeValue* <|-- *AttributeValueSimple*  
*AttributeDefinitionSimple* <|-- *AttributeDefinitionDate*  
*DatatypeDefinitionSimple* <|-- *DatatypeDefinitionDate*  
*AttributeValueSimple* <|-- *AttributeValueDate*  
*AttributeDefinition* <|-- *AttributeDefinitionEnumeration*  
*DatatypeDefinition* <|-- *DatatypeDefinitionEnumeration*  
*Identifiable* <|-- *EnumValue*  
*AttributeValue* <|-- *AttributeValueEnumeration*  
*AttributeDefinitionSimple* <|-- *AttributeDefinitionInteger*  
*DatatypeDefinitionSimple* <|-- *DatatypeDefinitionInteger*  
*AttributeValueSimple* <|-- *AttributeValueInteger*  
*AttributeDefinitionSimple* <|-- *AttributeDefinitionReal*  
*DatatypeDefinitionSimple* <|-- *DatatypeDefinitionReal*  
*AttributeValueSimple* <|-- *AttributeValueReal*  
*AttributeDefinitionSimple* <|-- *AttributeDefinitionString*  
*DatatypeDefinitionSimple* <|-- *DatatypeDefinitionString*  
*AttributeValueSimple* <|-- *AttributeValueString*  
*Identifiable* \*--> "alternativeID" *AlternativeID*  
*AttributeValueXHTML* --> "definition"  
*AttributeDefinitionXHTML*  
*AttributeValueXHTML* \*--> "theOriginalValue"  
*XhtmlContent*  
*AttributeValueXHTML* \*--> "theValue" *XhtmlContent*  
*SpecElementWithAttributes* \*--> "values" \*"  
*AttributeValue*  
*AttributeDefinitionXHTML* --> "type"  
*DatatypeDefinitionXHTML*  
*AttributeDefinitionXHTML* \*--> "defaultValue"  
*AttributeValueXHTML*  
*SpecType* \*--> "specAttributes" \*" *AttributeDefinition*  
*ReqIFContent* \*--> "datatypes" \*" *DatatypeDefinition*  
*ReqIFContent* \*--> "specTypes" \*" *SpecType*  
*ReqIFContent* \*--> "specObjects" \*" *SpecObject*  
*ReqIFContent* \*--> "specRelations" \*" *SpecRelation*  
*ReqIFContent* \*--> "specifications" \*" *Specification*  
*ReqIFContent* \*--> "specRelationGroups" \*"  
*RelationGroup*  
*ReqIF* \*--> "theHeader" *ReqIFHeader*  
*ReqIF* \*--> "coreContent" *ReqIFContent*  
*ReqIF* \*--> "toolExtensions" \*" *ReqIFToolExtension*  
*ReqIFToolExtension* \*--> "extensions" \*" *EObject*  
*SpecObject* --> "type" *SpecObjectType*  
*Specification* --> "type" *SpecificationType*  
*Specification* \*--> "children" \*" *SpecHierarchy*  
*SpecHierarchy* --> "object" *SpecObject*  
*SpecHierarchy* \*--> "children" \*" *SpecHierarchy*  
*SpecHierarchy* --> "editableAtts" \*" *AttributeDefinition*  
*SpecRelation* --> "target" *SpecObject*

SpecRelation --> "source" SpecObject SpecRelation --> "type" SpecRelationType RelationGroup --> "specRelations *" SpecRelation RelationGroup --> "type" RelationGroupType RelationGroup --> "sourceSpecification" Specification RelationGroup --> "targetSpecification" Specification AttributeDefinitionBoolean --> "type" DatatypeDefinitionBoolean AttributeDefinitionBoolean *--> "defaultValue" AttributeValueBoolean AttributeValueBoolean --> "definition" AttributeDefinitionBoolean AttributeDefinitionDate --> "type" DatatypeDefinitionDate AttributeDefinitionDate *--> "defaultValue" AttributeValueDate AttributeValueDate --> "definition" AttributeDefinitionDate AttributeDefinitionEnumeration --> "type" DatatypeDefinitionEnumeration AttributeDefinitionEnumeration *--> "defaultValue" AttributeValueEnumeration DatatypeDefinitionEnumeration *--> "specifiedValues *" EnumValue EnumValue *--> "properties" EmbeddedValue AttributeValueEnumeration --> "values *" EnumValue AttributeValueEnumeration --> "definition" AttributeDefinitionEnumeration AttributeDefinitionInteger --> "type" DatatypeDefinitionInteger AttributeDefinitionInteger *--> "defaultValue" AttributeValueInteger AttributeValueInteger --> "definition" AttributeDefinitionInteger AttributeDefinitionReal --> "type" DatatypeDefinitionReal AttributeDefinitionReal *--> "defaultValue" AttributeValueReal AttributeValueReal --> "definition" AttributeDefinitionReal AttributeDefinitionString --> "type" DatatypeDefinitionString AttributeDefinitionString *--> "defaultValue" AttributeValueString AttributeValueString --> "definition" AttributeDefinitionString XhtmlContent *--> "xhtml" EObject @enduml	
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--

Link for the Chat:

<https://chatgpt.com/share/a4fcfb36-7086-4328-a9b9-00c69d3dade6>

## QVT Language

Main prompt	Example outputs
-------------	-----------------

Given the view description and the following PlantUML metamodels, please give me the view definition written in QVT

{Same as VPDL example}

To define the QVT (Query/View/Transformation) view for combining the Enterprise Architecture model and the Requirements model, we need to map the elements from the two models together. The QVT Relations (QVT-R) transformation language is commonly used for defining transformations between models.

Here's a simplified QVT-R transformation that captures the essence of mapping elements from the Enterprise Architecture model to the Requirements model:

```
``qvt

modeltype EA uses
EnterpriseArchitecture('path/to/EnterpriseArchitecture.e
core');

modeltype ReqIF uses ReqIF('path/to/ReqIF.ecore');

transformation EAtoReqIF(in ea: EA, out reqif: ReqIF);

main() {

 // Map Enterprise Architecture Elements to
 Requirements

 ea.architectures -> map
 ArchitectureToSpecObject();

 ea.containers -> map
 ContainerToSpecObject();

 ea.labels -> map LabelToSpecObject();

 ea.elements -> map ElementToSpecObject();
}

mapping ArchitectureToSpecObject(in arch:
EA::Architecture) : ReqIF::SpecObject {

 identifier := arch.ID;

 longName := arch.name;
```

```

 desc := arch.description;

 lastChange := now();

 type := objectType;
 }

mapping ContainerToSpecObject(in container:
EA::Container) : ReqIF::SpecObject {

 identifier := container.id;

 longName := container.name;

 desc := container.description;

 lastChange := now();

 type := objectType;
}

mapping LabelToSpecObject(in label: EA::Label) :
ReqIF::SpecObject {

 identifier := label.id;

 longName := label.name;

 desc := label.description;

 lastChange := now();

 type := objectType;
}

mapping ElementToSpecObject(in element:
EA::Element) : ReqIF::SpecObject {

 identifier := element.ID;

 longName := element.name;

 desc := element.description;

 lastChange := now();

 type := objectType;
}

```

```
}
```

```
mapping OrganizationUnitToSpecObject(in orgUnit:
EA::OrganizationUnit) : ReqIF::SpecObject {
```

```
 identifier := orgUnit.ID;
```

```
 longName := orgUnit.name;
```

```
 desc := orgUnit.description;
```

```
 lastChange := now();
```

```
 type := objectType;
```

```
}
```

```
mapping ActorToSpecObject(in actor: EA::Actor) :
ReqIF::SpecObject {
```

```
 identifier := actor.ID;
```

```
 longName := actor.name;
```

```
 desc := actor.description;
```

```
 lastChange := now();
```

```
 type := objectType;
```

```
}
```

```
mapping RoleToSpecObject(in role: EA::Role) :
ReqIF::SpecObject {
```

```
 identifier := role.ID;
```

```
 longName := role.name;
```

```
 desc := role.description;
```

```
 lastChange := now();
```

```
 type := objectType;
```

```
}
```

```
mapping DataEntityToSpecObject(in dataEntity:
```

```
EA::DataEntity) : ReqIF::SpecObject {
```

```
 identifier := dataEntity.ID;
```

```
 longName := dataEntity.name;
```

```
 desc := dataEntity.description;
```

```
 lastChange := now();
```

```
 type := objectType;
```

```
}
```

```
mapping LogicalApplicationComponentToSpecObject(in
comp: EA::LogicalApplicationComponent) :
ReqIF::SpecObject {
```

```
 identifier := comp.ID;
```

```
 longName := comp.name;
```

```
 desc := comp.description;
```

```
 lastChange := now();
```

```
 type := objectType;
```

```
}
```

```
// Continue with other mappings as needed...
```

```
helper objectType() : ReqIF::SpecObjectType {
```

```
 result := ReqIF::SpecObjectType {
```

```
 identifier := 'ObjectTypeID',
```

```
 longName := 'ObjectTypeName',
```

```
 desc := 'Description of the SpecObjectType',
```

```
 lastChange := now()
```

```
 };
```

```
}
```

```
...
```

	<p><b>### Explanation:</b></p> <ul style="list-style-type: none"> <li>- <b>Model Types</b>: <code>`EA`</code> and <code>`ReqIF`</code> model types are defined using the respective Ecore models.</li> <li>- <b>Transformation</b>: The <code>`EAtoReqIF`</code> transformation maps elements from the Enterprise Architecture model (<code>`EA`</code>) to the Requirements model (<code>`ReqIF`</code>).</li> <li>- <b>Main Function</b>: The <code>`main()`</code> function maps different EA elements to ReqIF <code>`SpecObject`</code> elements.</li> <li>- <b>Mapping Functions</b>: Specific mapping functions such as <code>`ArchitectureToSpecObject`</code>, <code>`ContainerToSpecObject`</code>, etc., are defined to map EA elements to ReqIF <code>`SpecObject`</code> elements.</li> <li>- <b>Helper Functions</b>: Helper functions like <code>`objectType()`</code> are used to set the type for <code>`SpecObject`</code>.</li> </ul> <p>This QVT transformation provides a foundational structure. You may need to customize it further based on the specific attributes and relationships in your models.</p>
--	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Link for the Chat:

<https://chatgpt.com/share/a1d99059-3249-4b8e-b132-a7c1077e9ed1>

## 3rd example - View for Evolution

### VPDL Language

#### Chat Log

Main prompt input	Output
Given the view description and the following PlantUML metamodels, please give me the view definition written in VPDL View description:I want to follow the evolution of a engineering system. It shows to different versions of the same system modeled using the CAEX metamodel. Based on the changes performed in the	To define the view in VPDL (Viewpoint Definition Language) based on the provided CAEX and Traces metamodels, we need to outline the necessary elements and their relationships. The view description specifies that we need to show different versions of the



models, the changes are captured into a model that conforms to a Traces metamodel.

Caex PlantUML

```
@startuml class AttributeDataType {
java.lang.String } class AttributeFamily { } class
AttributeNameMapping { roleAttributeName: String
systemUnitAttributeName: String } class Attribute {
defaultValue: String value: String
attributeDataType: AttributeDataType
refAttributeType: String unit: String } class
AttributeLib { } class AttributeValueRequirement {
name: String } class CAEXBasicObject {
changeMode: ChangeMode } class CAEXFile {
superiorStandardVersion: String[] fileName: String
schemaVersion: String } class CAEXObject { iD: String
name: String } enum ChangeMode {
state: state create: create delete: delete
change: change } class ChangeModeObject {
org.eclipse.emf.common.util.Enumerator } class
DocumentRoot { mixed: EFeatureMapEntry[] } class
ExternalInterface { } class ExternalReference {
alias: String path: String } class InstanceHierarchy
{ } class InterfaceClassLib { } class InterfaceClass {
refBaseClassPath: String } class InterfaceFamily { }
class InterfaceIDMapping { roleInterfaceID:
String systemUnitInterfaceID: String } class
InternalElement { refBaseSystemUnitPath: String }
class InternalLink { refPartnerSideA: String
refPartnerSideB: String } class Mapping { } class
NominalScaled { requiredValue: String[] } class
OrdinalScaled { requiredMaxValue: String
requiredValue: String requiredMinValue: String }
class RefSemantic { correspondingAttributePath:
String } class RoleClassLib { } class RoleClass {
refBaseClassPath: String } class RoleFamily { } class
RoleRequirements { refBaseRoleClassPath:
String } class SourceDocumentInformation {
lastWritingDateTime: DateTime originID: String
originName: String originProjectID: String
originProjectTitle: String originRelease: String
originVendor: String originVendorURL: String
originVersion: String } class SupportedRoleClass {
refRoleClassPath: String } class SystemUnitClassLib { }
class SystemUnitClass { refBaseClassPath: String }
class SystemUnitFamily { } class UnknownType {
requirements: String } class Copyright { value:
String changeMode: ChangeMode } class Description
{ value: String changeMode: ChangeMode
} class Revision { revisionDate: DateTime
oldVersion: String newVersion: String
authorName: String comment: String } class
Version { value: String changeMode:
ChangeMode } class AdditionalInformation { } Attribute
<|-- AttributeFamily CAEXBasicObject <|--
AttributeNameMapping CAEXObject <|-- Attribute
CAEXObject <|-- AttributeLib CAEXBasicObject <|--
AttributeValueRequirement CAEXBasicObject <|--
CAEXFile CAEXBasicObject <|-- CAEXObject
InterfaceClass <|-- ExternalInterface CAEXBasicObject
<|-- ExternalReference CAEXObject <|--
```

same engineering system, capturing the changes between these versions using the Traces metamodel. Here's how you can write the view definition in VPD:

```
```vpdl
```

```
view engineeringSystemEvolution {
```

```
description "This view shows different versions
of the same engineering system modeled using the
CAEX metamodel. Changes between the versions are
captured using the Traces metamodel."
```

```
// Define the metamodels involved
```

```
metamodel CAEX {
```

```
import "path/to/caex.ecore"
```

```
}
```

```
metamodel Traces {
```

```
import "path/to/traces.ecore"
```

```
}
```

```
// Elements from the CAEX metamodel
```

```
element CAEXFile from CAEX
```

```
element InstanceHierarchy from CAEX
```

```
element SystemUnitClassLib from CAEX
```

```
element SystemUnitClass from CAEX
```

```
element InternalElement from CAEX
```

```
element RoleClassLib from CAEX
```

```
element RoleClass from CAEX
```

```
element Attribute from CAEX
```

```

InstanceHierarchy CAEXObject <|-- InterfaceClassLib
CAEXObject <|-- InterfaceClass InterfaceClass <|--
InterfaceFamily CAEXBasicObject <|--
InterfaceIDMapping SystemUnitClass <|--
InternalElement CAEXObject <|-- InternalLink
CAEXBasicObject <|-- Mapping CAEXBasicObject <|--
RefSemantic CAEXObject <|-- RoleClassLib
CAEXObject <|-- RoleClass RoleClass <|-- RoleFamily
CAEXBasicObject <|-- RoleRequirements
CAEXBasicObject <|-- SupportedRoleClass
CAEXObject <|-- SystemUnitClassLib CAEXObject <|--
SystemUnitClass SystemUnitClass <|--
SystemUnitFamily CAEXBasicObject <|-- Revision
GenericElement <|-- AdditionalInformation
AttributeNameMapping --> "roleAttribute" Attribute
AttributeNameMapping --> "systemUnitAttribute"
Attribute Attribute *--> "refSemantic" RefSemantic
Attribute *--> "constraint" AttributeValueRequirement
Attribute *--> "attribute" Attribute Attribute -->
"referenceAttribute" Attribute AttributeLib *-->
"attributeType" Attribute AttributeValueRequirement
*--> "ordinalScaledType" OrdinalScaled
AttributeValueRequirement *--> "nominalScaledType"
NominalScaled AttributeValueRequirement *-->
"unknownType" UnknownType CAEXBasicObject *-->
"description" Description CAEXBasicObject *-->
"version" Version CAEXBasicObject *--> "revision"
Revision CAEXBasicObject *--> "copyright" Copyright
CAEXBasicObject *--> "additionalInformation"
AdditionalInformation CAEXFile *-->
"sourceDocumentInformation"
SourceDocumentInformation CAEXFile *-->
"externalReference" ExternalReference CAEXFile *-->
"instanceHierarchy" InstanceHierarchy CAEXFile *-->
"interfaceClassLib" InterfaceClassLib CAEXFile *-->
"roleClassLib" RoleClassLib CAEXFile *-->
"systemUnitClassLib" SystemUnitClassLib CAEXFile
*--> "attributeTypeLib" AttributeLib DocumentRoot *-->
"xMLNSPrefixMap" EStringToStringMapEntry
DocumentRoot *--> "xSISchemaLocation"
EStringToStringMapEntry DocumentRoot *--> "/"
cAEXFile" CAEXFile InstanceHierarchy *-->
"internalElement" InternalElement InterfaceClassLib
*--> "interfaceClass" InterfaceClass InterfaceClass
*--> "attribute" Attribute InterfaceClass --> "baseClass"
InterfaceClass InterfaceClass *--> "interfaceClass"
InterfaceClass InterfaceClass *--> "externalInterface"
InterfaceClass InterfaceIDMapping --> "roleInterface"
ExternalInterface InterfaceIDMapping -->
"systemUnitInterface" ExternalInterface InternalElement
*--> "roleRequirements" RoleRequirements
InternalElement --> "baseSystemUnit" SystemUnitClass
InternalLink --> "partnerSideA" InterfaceClass
InternalLink --> "partnerSideB" InterfaceClass Mapping
*--> "attributeNameMapping" AttributeNameMapping
Mapping *--> "interfaceIDMapping"
InterfaceIDMapping RoleClassLib *--> "roleClass"
RoleClass RoleClass *--> "attribute" Attribute
RoleClass *--> "externalInterface" ExternalInterface
RoleClass --> "baseClass" RoleClass RoleClass *-->
"roleClass" RoleClass RoleRequirements *-->

```

```

// Elements from the Traces metamodel

element Event from Traces

element Log from Traces

element EventType from Traces

element Attributes from Traces


// Define the relationships to be shown

relationship "caex_file_to_instance_hierarchy"
from CAEXFile.to(InstanceHierarchy)

relationship
"instance_hierarchy_to_internal_element" from
InstanceHierarchy.to(InternalElement)

relationship
"system_unit_class_lib_to_system_unit_class" from
SystemUnitClassLib.to(SystemUnitClass)

relationship
"system_unit_class_to_internal_element" from
SystemUnitClass.to(InternalElement)

relationship "role_class_lib_to_role_class"
from RoleClassLib.to(RoleClass)

relationship "internal_element_to_attribute"
from InternalElement.to(Attribute)


// Changes captured in the Traces metamodel

relationship "log_to_event" from Log.to(Event)

relationship "event_to_attributes" from
Event.to(Attributes)


// Define the view

view {

// Include elements and relationships from
CAEX metamodel

include element CAEXFile

include element InstanceHierarchy

```

```

"attribute *" Attribute RoleRequirements *-->
"externalInterface *" InterfaceClass RoleRequirements
*--> "mappingObject" Mapping RoleRequirements -->
"roleClass" RoleClass SupportedRoleClass *-->
"mappingObject" Mapping SupportedRoleClass -->
"roleClass" RoleClass SystemUnitClassLib *-->
"systemUnitClass *" SystemUnitClass SystemUnitClass
*--> "attribute *" Attribute SystemUnitClass *-->
"externalInterface *" InterfaceClass SystemUnitClass
*--> "internalElement *" InternalElement
SystemUnitClass *--> "supportedRoleClass *"
SupportedRoleClass SystemUnitClass *--> "internalLink
*" InternalLink SystemUnitClass --> "baseClass"
SystemUnitClass SystemUnitClass *-->
"systemUnitClass *" SystemUnitClass @enduml Traces
PlantUML @startuml class Event { set: EBoolean
timestamp: EDate          dataValue: EString
eventType: EventType } class Log { resource: EString }
abstract class Attributes { name: EString } class
FeatureAndClass { } class Feature { } class JavaClass {
} class ID { } class ReferenceValue { } enum EventType
{
    SET: SET      UNSET: UNSET  ADD: ADD
    REMOVE: REMOVE  ADD_MANY:
ADD_MANY      REMOVE_MANY: REMOVE_MANY
    MOVE: MOVE    RESOLVE: RESOLVE }
Attributes <|-- FeatureAndClass Attributes <|-- Feature
Attributes <|-- JavaClass Attributes <|-- ID Attributes <|--
ReferenceValue Event --> "featureAndClass"
FeatureAndClass Event --> "id" ID Event -->
"referencevalue" ReferenceValue Log *--> "event *"
Event Log *--> "attributes *" Attributes FeatureAndClass
--> "class" JavaClass FeatureAndClass --> "feature"
Feature @enduml

```

```

include element SystemUnitClassLib

include element SystemUnitClass

include element InternalElement

include element RoleClassLib

include element RoleClass

include element Attribute


include relationship
"caex_file_to_instance_hierarchy"


include relationship
"instance_hierarchy_to_internal_element"


include relationship
"system_unit_class_lib_to_system_unit_class"


include relationship
"system_unit_class_to_internal_element"


include relationship
"role_class_lib_to_role_class"


include relationship
"internal_element_to_attribute"


// Include elements and relationships from
Traces metamodel to capture changes


include element Log

include element Event

include element EventType

include element Attributes


include relationship "log_to_event"

include relationship "event_to_attributes"


// Specific to the view

// Show evolution of the system

```

```
element Evolution {  
  
    include CAEXFile  
  
    include InstanceHierarchy  
  
    include SystemUnitClassLib  
  
    include SystemUnitClass  
  
    include InternalElement  
  
    include RoleClassLib  
  
    include RoleClass  
  
    include Attribute  
  
  
    include Log  
  
    include Event  
  
    include EventType  
  
    include Attributes  
  
  
    include relationship  
    "caex_file_to_instance_hierarchy"  
  
    include relationship  
    "instance_hierarchy_to_internal_element"  
  
    include relationship  
    "system_unit_class_lib_to_system_unit_class"  
  
    include relationship  
    "system_unit_class_to_internal_element"  
  
    include relationship  
    "role_class_lib_to_role_class"  
  
    include relationship  
    "internal_element_to_attribute"  
  
  
    include relationship "log_to_event"  
  
    include relationship "event_to_attributes"  
  
}
```

	<pre> } } ... </pre> <p><i>This VPDL definition outlines the elements and relationships from both the CAEX and Traces metamodels that are necessary to visualize the evolution of an engineering system and capture the changes between different versions. Adjust the import paths for the metamodels according to their actual locations.</i></p>
--	--

Link for the Chat:

<https://chatgpt.com/share/f7bcb7e8-b193-43c4-8837-addb500a0bb6>

QVT Language

Main prompt input	Output
<p><i>Given the view description and the following PlantUML metamodels, please give me the view definition written in QVT.</i></p> <p><i>View description: I want to follow the evolution of a engineering system. It shows to different versions of the same system modeled using the CAEX metamodel. Based on the changes performed in the models, the changes are captured into a model that conforms to a Traces metamodel.</i></p> <p><i>{Same as VPDL example}</i></p>	<p><i>To define a view that captures the changes between different versions of a CAEX model and records these changes in a Traces model, we can use QVT (Query/View/Transformation). Specifically, we'll use QVT-R (QVT-Relations) to specify how elements from the CAEX metamodel map to the Traces metamodel based on their changes.</i></p> <p><i>Here's a basic QVT-R transformation that captures this view definition:</i></p> <pre> ``qvtr modeltype caex uses 'path/to/caex.ecore'; modeltype traces uses 'path/to/traces.ecore'; transformation CaptureChanges(in oldCAEX: caex, in newCAEX: caex, out traceModel: traces); </pre>

```

-- Mapping for CAEXFile

top relation CAEXFileToLog {

  checkonly domain oldCAEX oldFile: CAEXFile {

    fileName = oldFileName;

  }

  checkonly domain newCAEX newFile: CAEXFile {

    fileName = newFileName;

  }

  enforce domain traceModel log: Log {

    resource = newFileName;

  }

  where {

    oldFileName != newFileName;

  }

}

-- Mapping for Attribute

top relation AttributeToEvent {

  checkonly domain oldCAEX oldAttr: Attribute {

    iD = oldID;

    value = oldValue;

  }

  checkonly domain newCAEX newAttr: Attribute {

    iD = newID;

    value = newValue;

  }

  enforce domain traceModel event: Event {

    eventType = if oldValue <> newValue then

```

```

EventType::SET else EventType::UNSET endif;

    dataValue = newValue;

    timestamp = now(); -- Assuming a function
'now()' that returns the current timestamp

    id = newID;
}

where {

    oldID = newID;

    oldValue <> newValue;

}

}

-- Mapping for InternalElement
top relation InternalElementToEvent {

    checkonly domain oldCAEX oldElem: InternalElement
    {

        iD = oldID;

        refBaseSystemUnitPath = oldPath;

    }

    checkonly domain newCAEX newElem:
InternalElement {

        iD = newID;

        refBaseSystemUnitPath = newPath;

    }

    enforce domain traceModel event: Event {

        eventType = if oldPath <> newPath then
EventType::MOVE else EventType::UNSET endif;

        dataValue = newPath;

        timestamp = now();

        id = newID;

```

```

}

where {

    oldID = newID;

    oldPath <> newPath;

}

}

```

-- Additional mappings for other elements as necessary

end transformation;

...

In this QVT-R transformation:

1. *****CAEXFileToLog****: This relation compares `CAEXFile` elements from the old and new models. If the `fileName` has changed, it records this in the `Log` of the `Traces` model.*
2. *****AttributeToEvent****: This relation compares `Attribute` elements based on their `iD`. If the `value` of the attribute has changed, it creates an `Event` in the `Traces` model.*
3. *****InternalElementToEvent****: This relation captures changes in `InternalElement` elements, specifically changes in their `refBaseSystemUnitPath`.*

Each `Event` captures the type of change (`SET`, `UNSET`, `MOVE`), the new value, and a timestamp. Additional relations can be added to handle other types of elements and changes as needed.

Note: The paths to the Ecore metamodels (`path/to/caex.ecore` and `path/to/traces.ecore`) need to be replaced with the actual paths where these metamodels are located. The `now()` function is assumed to provide the current timestamp and would need to be implemented or replaced with an appropriate

	<p><i>mechanism for obtaining the current time.</i></p> <p><i>This QVT-R transformation defines a view that captures and logs the changes between two versions of a CAEX model, conforming to the Traces metamodel.</i></p>
--	---

Link for the Chat:

<https://chatgpt.com/share/c844d46a-0e9a-4fa5-9194-c804a2bc70b9>

4th example - View for Safety

VPDL Language

Main prompt input	Output
<p><i>Given the view description and the following PlantUML metamodels, please give me the view definition written in VPDL</i></p> <p><i>View description: We propose to build a view that aggregates together all the models seen so far. This allows the system engineer to transparently point to the relevant information (spread in different models) while also allowing him/her to have a better vision of the full picture.</i></p> <p><i>We add a new bidirectional association designComponent, between Trace and Component, which allows the system engineer to navigate from a runtime trace back to the component that emitted it, and from a component to all its emitted traces.</i></p> <p><i>We also rename the features leftLinkEnd (which holds</i></p>	<p><i>Based on the provided view description and PlantUML metamodels, here's the VPDL (Viewpoint Definition Language) for the proposed view:</i></p> <pre> ...vpdl view AggregatedView { // Importing the Traceability metamodel import TraceabilityModel </pre>

references to *B* variables) and *rightLinkEnd* (references to events and traces) to specification and events, making the view more useful to the engineer. We perform such a renaming by filtering the existing features and creating new associations;

```
Traceability PlantUML
@startuml
class TraceabilityModel {
    name: String
}
class TraceabilityLink {
    id: Integer
    name: String
    description: String
}
class TraceabilityLinkEnd {
    name: String
}
class TraceabilityContext {
    name: String
    description: String
}
class TraceabilityContextData {
    name: String
    value: String
}
TraceabilityModel *--> "traceabilityLinks *"
TraceabilityLink
TraceabilityLink *--> "rightLinkEnd *"
TraceabilityLinkEnd
TraceabilityLink *--> "leftLinkEnd *" TraceabilityLinkEnd
TraceabilityLink *--> "traceabilityContext"
TraceabilityContext
TraceabilityLinkEnd --> "traceableArtefact" EObject
TraceabilityContext *--> "traceabilityContextData *"
TraceabilityContextData
@enduml

B PlantUML
@startuml
class Abstraction {
    name: EString
}
class Machine {
}
class Implementation {
}
class Sees {
}
class Imports {
}
class Values {
}
class ValueExpr {
    value: EString
}
class ConcreteVariables {
}
class Variable {
    name: EString
}
```

```
import TraceabilityLink
import TraceabilityLinkEnd
import TraceabilityContext
import TraceabilityContextData
import EObject

// Importing the B metamodel

import Abstraction
import Machine
import Implementation
import Variable
import Operation
import Expr
import Type
import PrimitiveTypeEnum
import ReturnTypeExpr

// Renaming features

TraceabilityLink {
    -leftLinkEnd as specification
    -rightLinkEnd as events
}

// Creating new bidirectional association
association designComponent {
    Trace [*] <--> [*] Component
}
```

```

}
class ConcreteConstants {
}
class Invariant {
}
class InvariantExpr {
}
class Type {
}
enum PrimitiveTypeEnum {
    BOOL: BOOL
    INT: INT
    NAT: NAT
    NAT1: NAT1
    STRING: STRING
}
class Initialisation {
}
class InitialisationExpr {
}
class Properties {
}
class PropertyExpr {
}
class Range {
    lowerBound: EInt
}
class Assertions {
}
class AssertionExpr {
}
class Definitions {
}
class Definition {
    name: EString
}
class LogicalExpr {
}
enum InequalityOp {
    <: LESS
    <=: LESS_EQ
    >: GREATER
    >=: GREATER_EQ
}
class DefinitionCall {
}
class IntLiteral {
    value: EInt
}
enum BoolLiteralEnum {
    TRUE: TRUE
    FALSE: FALSE
}
class Sets {
}
class Set {
}
class Operations {
}
class Operation {
    name: EString

```

```

// Aggregating relevant information from both
models

aggregation {

    TraceabilityModel {

        name

        traceabilityLinks {

            id

            name

            description

            traceabilityContext {

                name

                description

                traceabilityContextData {

                    name

                    value

                }

            }

            specification {

                name

                traceableArtefact {

                    // Relevant EObject details

                }

            }

        }

        events {

            name

            traceableArtefact {

                // Relevant EObject details

            }

        }

    }

}

```

<pre> } class Body { } class Expr { } class Skip { } class Statement { } class If { } class IfCond { } class Condition { } class Pre { } class PreExpr { } class Var { } class Begin { } class BeginBody { } class Assign { } class Return { } class ReturnExpr { } class ReturnExpr { } class Case { } class CaseExpr { } class Seq { } class FinalExpr { } class Call { } class SimpleCall { } class Arg { } class LocalOperations { } class PrimitiveType { type: PrimitiveTypeEnum } class Ref { } class PropertyTyped { } class PropertyRange { } class AndExpr { } class ImplyExpr { </pre>	<pre> } } } Abstraction { name sees { seems { name } } definitions { name expr args { name } } properties { exprs } operations { name outputs { name } args { name </pre>
---	---

<pre> } class InequalityExpr { op: InequalityOp } class EqualExpr { } class NegExpr { } class BoolTest { } class TypeConstraint { } class ConstantExpr { constant: BoolLiteralEnum } class CondAnd { } class CondEq { } class CondLessThan { } class CondMinus { } class CondNeg { } class BoolLiteral { value: BoolLiteralEnum constant: BoolLiteralEnum } class ReturnTuple { } class ReturnOr { } class Neg { } class ArgMinus { } class StringLiteral { value: EString } class Program { } Abstraction < -- Machine Abstraction < -- Implementation LogicalExpr < -- DefinitionCall LogicalExpr < -- IntLiteral Condition < -- IntLiteral Arg < -- IntLiteral Body < -- Skip Expr < -- Skip Body < -- If Expr < -- If FinalExpr < -- If Body < -- Pre Body < -- Var Expr < -- Var FinalExpr < -- Var Body < -- Begin Expr < -- Assign Statement < -- Assign Expr < -- Return </pre>	<pre> } body } } Machine { name invariant { exprs { variable { name } type } } initialisation { exprs { variable { name } type } } assertions { exprs { variable { name } } } </pre>
--	---

```

FinalExpr <|-- Return
Return <|-- ReturnExpr
Expr <|-- Case
FinalExpr <|-- Case
Body <|-- Seq
BeginBody <|-- Seq
BeginBody <|-- FinalExpr
Expr <|-- Call
Statement <|-- Call
Type <|-- PrimitiveType
Type <|-- Ref
LogicalExpr <|-- Ref
Condition <|-- Ref
ReturnExpr <|-- Ref
Arg <|-- Ref
PropertyExpr <|-- PropertyTyped
PropertyExpr <|-- PropertyRange
LogicalExpr <|-- AndExpr
LogicalExpr <|-- ImplyExpr
LogicalExpr <|-- InequalityExpr
LogicalExpr <|-- EqualExpr
LogicalExpr <|-- NegExpr
LogicalExpr <|-- BoolTest
ReturnExpr <|-- BoolTest
LogicalExpr <|-- TypeConstraint
LogicalExpr <|-- ConstantExpr
Condition <|-- CondAnd
Condition <|-- CondEq
Condition <|-- CondLessThan
Condition <|-- CondMinus
Condition <|-- CondNeg
Condition <|-- BoolLiteral
ReturnExpr <|-- BoolLiteral
Arg <|-- BoolLiteral
Return <|-- ReturnTuple
ReturnExpr <|-- ReturnOr
ReturnExpr <|-- Neg
Arg <|-- ArgMinus
Arg <|-- StringLiteral
Abstraction *--> "sees" Sees
Abstraction *--> "concreteConstants"
ConcreteConstants
Abstraction *--> "definitions" Definitions
Abstraction *--> "properties" Properties
Abstraction *--> "operations" Operations
Machine *--> "concreteVariables" ConcreteVariables
Machine *--> "invariant" Invariant
Machine *--> "initialisation" Initialisation
Machine *--> "assertions" Assertions
Machine *--> "sets" Sets
Implementation --> "refine" Machine
Implementation *--> "imports" Imports
Implementation *--> "values" Values
Implementation *--> "localOperations" LocalOperations
Sees --> "sees" Abstraction
Imports --> "imports" Abstraction
Values *--> "exprs" ValueExpr
ValueExpr --> "constant" Variable
ConcreteVariables *--> "variables" Variable
ConcreteConstants *--> "constants" Variable
Invariant *--> "exprs" InvariantExpr

```

```

type
}
}
sets {
exprs {
name
elems {
name
}
}
concreteVariables {
variables {
name
}
}
}
Implementation {
refine
imports {
imports {
name
}
}
values {
exprs {
value

```

```

InvariantExpr --> "variable" Variable
InvariantExpr *--> "type" Type
Initialisation *--> "exprs *" InitialisationExpr
InitialisationExpr --> "variable" Variable
InitialisationExpr *--> "type" Type
Properties *--> "exprs *" PropertyExpr
Properties *--> "expr" * PropertyExpr
PropertyExpr --> "constant" Variable
Range --> "upperBound" Variable
Assertions *--> "exprs *" AssertionExpr
AssertionExpr --> "constant" Variable
AssertionExpr *--> "type" Type
Definitions *--> "exprs *" Definition
Definition *--> "args *" Variable
Definition *--> "expr" LogicalExpr
DefinitionCall --> "def" Definition
DefinitionCall *--> "args *" Arg
Sets *--> "exprs *" Set
Set *--> "name" Variable
Set *--> "elems *" Variable
Operations *--> "operations *" Operation
Operation *--> "outputs *" Variable
Operation *--> "args *" Variable
Operation *--> "body" Body
If *--> "exprs *" IfCond
If *--> "else" Expr
IfCond *--> "condition" Condition
IfCond *--> "then" Expr
Pre *--> "exprs *" PreExpr
Pre *--> "expr" Expr
PreExpr --> "var" Variable
PreExpr *--> "type" Type
Var *--> "vars *" Variable
Var *--> "in" Seq
Begin *--> "expr" BeginBody
Assign --> "var" Variable
Assign *--> "value" ReturnExpr
ReturnTypeExpr --> "var" Variable
ReturnTypeExpr *--> "type" Type
Case --> "var" Variable
Case *--> "cases *" CaseExpr
CaseExpr --> "test" Variable
CaseExpr *--> "expr" Expr
Seq *--> "exprs *" Expr
Call --> "rets *" Variable
Call --> "op" Operation
Call *--> "args *" Arg
SimpleCall --> "op" Operation
SimpleCall *--> "args *" Arg
LocalOperations *--> "operations *" Operation
Ref --> "type" Variable
Ref --> "var" Variable
PropertyTyped *--> "type" Type
PropertyRange *--> "range" Range
AndExpr *--> "exprs *" LogicalExpr
ImplyExpr *--> "left" LogicalExpr
ImplyExpr *--> "right" LogicalExpr
InequalityExpr *--> "left" LogicalExpr
InequalityExpr *--> "right" LogicalExpr
EqualExpr *--> "left" LogicalExpr
EqualExpr *--> "right" LogicalExpr

```

```

constant {
    name
}

}

localOperations {
    operations {
        name
        outputs {
            name
        }
        args {
            name
        }
        body
    }
}

...

```

In this VPDL definition:

1. We import the necessary classes from the Traceability and B metamodels.
2. We rename `leftLinkEnd` to `specification` and `rightLinkEnd` to `events` within the `TraceabilityLink` class.
3. We define a new bidirectional association

<pre> NegExpr *--> "expr" LogicalExpr BoolTest *--> "expr" LogicalExpr TypeConstraint --> "var" Variable TypeConstraint *--> "type" Type CondAnd *--> "exprs *" Condition CondEq *--> "left" Condition CondEq *--> "right" Condition CondLessThan *--> "left" Condition CondLessThan *--> "right" Condition CondMinus *--> "left" Condition CondMinus *--> "right" Condition CondNeg *--> "expr" Condition ReturnTuple --> "vars *" Variable ReturnTuple *--> "value" LogicalExpr ReturnOr *--> "exprs *" ReturnExpr Neg *--> "expr" ReturnExpr ArgMinus *--> "left" Arg ArgMinus *--> "right" Arg Program *--> "machines *" Machine Program *--> "implementations *" Implementation @enduml </pre>	<p><code>`designComponent`</code> between <code>`Trace`</code> and <code>`Component`</code>.</p> <p>4. We aggregate relevant information from both models, providing a comprehensive view as described.</p>
---	---

Link for the Chat:

<https://chatgpt.com/share/81787fd1-a6ca-4915-ad9b-acd4d0778a72>

QVT Language

Main prompt input	Output
<p>Given the view description and the following PlantUML metamodels, please give me the view definition written in QVT.</p> <p>View description: We propose to build a view that aggregates together all the models seen so far. This allows the system engineer to transparently point to the relevant information (spread in different models) while also allowing him/her to have a better vision of the full picture.</p> <p>We add a new bidirectional association <code>designComponent</code>, between <code>Trace</code> and <code>Component</code>, which allows the system engineer to navigate from a runtime trace back to the component that emitted it, and from a component to all its emitted traces.</p> <p>We also rename the features <code>leftLinkEnd</code> (which holds references to <code>B</code> variables) and <code>rightLinkEnd</code> (references to events and traces) to <code>specification</code> and <code>events</code>, making the view more useful to the engineer. We perform such a renaming by filtering the existing features and creating new associations;</p> <p>{Same as VPDL example}</p>	<p>To define the view transformation in QVT that aggregates the information from different models and applies the specified modifications, we'll need to use the following QVT constructs: transformation, mapping, relation, domain, and object templates. The transformation will include creating the bidirectional association and renaming features.</p> <p>Here is the QVT code:</p> <pre> ``qvt modeltype Traceability "Traceability.ecore"; modeltype B "B.ecore"; modeltype Aggregated "Aggregated.ecore"; transformation AggregatedView(in traceabilityModel : </pre>

Traceability, in bModel : B, out aggregatedModel : Aggregated);

main() {

// Create a new AggregatedModel element

*aggregatedModel :=
Aggregated::AggregatedModel {*

// Aggregate TraceabilityModel elements

this.aggregateTraceability(traceabilityModel);

// Aggregate BModel elements

this.aggregateBModel(bModel);

};

}

mapping

*Traceability::TraceabilityModel::aggregateTraceability(tr
aceabilityModel : Traceability::TraceabilityModel) :
Aggregated::AggregatedModel {*

name := traceabilityModel.name;

// Aggregate TraceabilityLink elements

*traceabilityLinks :=
traceabilityModel.traceabilityLinks->map this
aggregateTraceabilityLink();*

}

mapping

*Traceability::TraceabilityLink::aggregateTraceabilityLink(
traceabilityLink : Traceability::TraceabilityLink) :
Aggregated::AggregatedLink {*

id := traceabilityLink.id;

name := traceabilityLink.name;

description := traceabilityLink.description;

```
// Renaming leftLinkEnd to specification and  
rightLinkEnd to events
```

```
specification := traceabilityLink.leftLinkEnd;
```

```
events := traceabilityLink.rightLinkEnd;
```

```
// Add bidirectional association  
designComponent between Trace and Component
```

```
designComponent :=  
traceabilityLink.traceableArtefact->select(e |  
e.ocllsKindOf(Component))->map this  
createDesignComponent();
```

```
}
```

```
mapping
```

```
Traceability::TraceabilityLinkEnd::aggregateTraceability  
LinkEnd(traceabilityLinkEnd :  
Traceability::TraceabilityLinkEnd) :  
Aggregated::AggregatedLinkEnd {
```

```
name := traceabilityLinkEnd.name;
```

```
traceableArtefact :=  
traceabilityLinkEnd.traceableArtefact;
```

```
}
```

```
mapping
```

```
Traceability::TraceabilityContext::aggregateTraceability  
Context(traceabilityContext :  
Traceability::TraceabilityContext) :  
Aggregated::AggregatedContext {
```

```
name := traceabilityContext.name;
```

```
description := traceabilityContext.description;
```

```
traceabilityContextData :=  
traceabilityContext.traceabilityContextData->map this  
aggregateTraceabilityContextData();
```

```
}
```

```

mapping
Traceability::TraceabilityContextData::aggregateTraceabilityContextData(traceabilityContextData : Traceability::TraceabilityContextData) : Aggregated::AggregatedContextData {

```

```

    name := traceabilityContextData.name;

```

```

    value := traceabilityContextData.value;

```

```

}

```

```

mapping B::Abstraction::aggregateBModel(abstraction : B::Abstraction) : Aggregated::AggregatedAbstraction {

```

```

    name := abstraction.name;

```

```

    // Map B::Machine and B::Implementation

```

```

    machines := abstraction.sees->select(e | e.ocIsKindOf(B::Machine))->map this
aggregateMachine();

```

```

    implementations := abstraction.refine->select(e | e.ocIsKindOf(B::Implementation))->map this
aggregateImplementation();

```

```

}

```

```

mapping B::Machine::aggregateMachine(machine : B::Machine) : Aggregated::AggregatedMachine {

```

```

    name := machine.name;

```

```

    concreteVariables :=
machine.concreteVariables->map this
aggregateConcreteVariables();

```

```

    invariant := machine.invariant->map this
aggregateInvariant();

```

```

    initialisation := machine.initialisation->map this
aggregateInitialisation();

```

```

}

```

```

mapping
B::Implementation::aggregateImplementation(implementation : B::Implementation) :

```

```

Aggregated::AggregatedImplementation {

    name := implementation.name;

    imports := implementation.imports->map this
aggregateImports();

    values := implementation.values->map this
aggregateValues();

    localOperations :=
implementation.localOperations->map this
aggregateLocalOperations();
}

mapping
B::ConcreteVariables::aggregateConcreteVariables(con
creteVariables : B::ConcreteVariables) :
Aggregated::AggregatedConcreteVariables {

    variables := concreteVariables.variables->map
this aggregateVariable();
}

mapping B::Invariant::aggregateInvariant(invariant :
B::Invariant) : Aggregated::AggregatedInvariant {

    exprs := invariant.exprs->map this
aggregateInvariantExpr();
}

mapping
B::Initialisation::aggregateInitialisation(initialisation :
B::Initialisation) : Aggregated::AggregatedInitialisation {

    exprs := initialisation.exprs->map this
aggregateInitialisationExpr();
}

mapping B::Imports::aggregateImports(imports :
B::Imports) : Aggregated::AggregatedImports {

    imports := imports.imports->map this
aggregateAbstraction();
}

```

```
}
```

```
mapping B::Values::aggregateValues(values :  
B::Values) : Aggregated::AggregatedValues {
```

```
    exprs := values.exprs->map this  
    aggregateValueExpr();
```

```
}
```

```
mapping  
B::LocalOperations::aggregateLocalOperations(localOp  
erations : B::LocalOperations) :  
Aggregated::AggregatedLocalOperations {
```

```
    operations :=  
localOperations.operations->map this  
    aggregateOperation();
```

```
}
```

```
mapping B::Variable::aggregateVariable(variable :  
B::Variable) : Aggregated::AggregatedVariable {
```

```
    name := variable.name;
```

```
}
```

```
mapping  
B::InvariantExpr::aggregateInvariantExpr(invariantExpr :  
B::InvariantExpr) :  
Aggregated::AggregatedInvariantExpr {
```

```
    variable := invariantExpr.variable;
```

```
    type := invariantExpr.type;
```

```
}
```

```
mapping  
B::InitialisationExpr::aggregateInitialisationExpr(initialisa  
tionExpr : B::InitialisationExpr) :  
Aggregated::AggregatedInitialisationExpr {
```

```
    variable := initialisationExpr.variable;
```

```

        type := initialisationExpr.type;
    }

mapping B::ValueExpr::aggregateValueExpr(valueExpr : B::ValueExpr) : Aggregated::AggregatedValueExpr {

    value := valueExpr.value;

    constant := valueExpr.constant;

}

mapping B::Operation::aggregateOperation(operation : B::Operation) : Aggregated::AggregatedOperation {

    name := operation.name;

    outputs := operation.outputs->map this
aggregateVariable();

    args := operation.args->map this
aggregateVariable();

    body := operation.body->map this
aggregateBody();

}

mapping B::Body::aggregateBody(body : B::Body) :
Aggregated::AggregatedBody {

    expr := body.expr->map this aggregateExpr();

}

mapping B::Expr::aggregateExpr(expr : B::Expr) :
Aggregated::AggregatedExpr {

    // Add specific mappings for each type of
expression if necessary

}

mapping
Traceability::TraceableArtefact::createDesignComponent(
traceableArtefact : Traceability::TraceableArtefact) :

```

	<pre>Aggregated::DesignComponent { // Mapping logic for creating designComponent association } ...</pre> <p><i>This QVT transformation code does the following:</i></p> <ol style="list-style-type: none"><i>1. Aggregates elements from both `Traceability` and `B` models into an `Aggregated` model.</i><i>2. Creates a bidirectional association `designComponent` between `Trace` and `Component`.</i><i>3. Renames features `leftLinkEnd` to `specification` and `rightLinkEnd` to `events`.</i><i>4. Ensures that all necessary elements from both source models are mapped to their corresponding elements in the aggregated model.</i> <p><i>Make sure that you have the corresponding Ecore metamodels (`Traceability.ecore`, `B.ecore`, `Aggregated.ecore`) and update the QVT code according to the actual structure and naming conventions of these metamodels.</i></p>
--	---

Link for the Chat:

<https://chatgpt.com/share/c75a819b-ae2c-4189-b502-3e41ad487b92>

Transformations examples

1st example - BibTex2DocBlock

Main prompt input	Output
<p>Given the transformation description and the following PlantUML metamodels, please give me the ATL code for the transformations. Transformation description: The BibTeXXML to DocBook example describes a transformation of a BibTeXXML model to a DocBook composed document. BibTeXXML is an XML-based format for the BibTeX bibliographic tool. DocBook, as for it, is an XML-based format for document composition. The aim of this transformation is to generate, from a BibTeXXML file, a DocBook document that presents the different entries of the bibliographic file within four different sections. The first section provides the full list of bibliographic entries. The second section provides the sorted list of the different authors referenced in the bibliography. The third section presents the titles of the bibliography titled entries (in a sorted way). Finally, the last section provides the list of referenced journals (in article entries). BibTex PlantUML</p> <pre>@startuml class BibTeXFile { } class Author { author: String } abstract class BibTeXEntry { id: String } abstract class AuthoredEntry { } abstract class DatedEntry { year: String } abstract class TitledEntry { title: String } abstract class BookTitledEntry { booktitle: String } class Article { journal: String } class TechReport { } class Unpublished { }</pre>	<p>To transform a BibTeXXML model to a DocBook document using ATL (Atlas Transformation Language), we need to write transformation rules that map elements from the BibTeXXML metamodel to the DocBook metamodel based on the provided transformation description. Below is the ATL code that accomplishes this:</p> <pre>````atl module BibTeX2DocBook; create OUT : DocBook from IN : BibTeX; -- Rule to transform the root BibTeXFile to a DocBook</pre>


```

{
    note: String } class Manual { } class
Proceedings { } class InProceedings { } class Booklet { }
class Book { publisher: String } class InCollection {
} class InBook { chapter: Integer } class Misc { }
abstract class ThesisEntry { school: String }
class PhDThesis { } class MasterThesis { } BibTeXEntry
<|-- AuthoredEntry BibTeXEntry <|-- DatedEntry
BibTeXEntry <|-- TitledEntry BibTeXEntry <|--
BookTitledEntry AuthoredEntry <|-- Article DatedEntry
<|-- Article TitledEntry <|-- Article AuthoredEntry <|--
TechReport DatedEntry <|-- TechReport TitledEntry <|--
TechReport AuthoredEntry <|-- Unpublished TitledEntry
<|-- Unpublished TitledEntry <|-- Manual DatedEntry <|--
Proceedings TitledEntry <|-- Proceedings Proceedings
<|-- InProceedings AuthoredEntry <|-- InProceedings
BookTitledEntry <|-- InProceedings DatedEntry <|--
Booklet AuthoredEntry <|-- Book DatedEntry <|-- Book
TitledEntry <|-- Book Book <|-- InCollection
BookTitledEntry <|-- InCollection Book <|-- InBook
BibTeXEntry <|-- Misc AuthoredEntry <|-- ThesisEntry
DatedEntry <|-- ThesisEntry TitledEntry <|-- ThesisEntry
ThesisEntry <|-- PhDThesis ThesisEntry <|--
MasterThesis BibTeXFile *--> "entries *" BibTeXEntry
AuthoredEntry *--> "authors *" Author @enduml
DocBlock PlantUML @startuml class DocBook { } class
Book { } abstract class TitledElement { title: String
} class Article { } abstract class Section { } class Sect1 {
} class Sect2 { } class Para { content: String }
TitledElement <|-- Article TitledElement <|-- Section
Section <|-- Sect1 Section <|-- Sect2 DocBook *-->
"books *" Book Book *--> "articles *" Article Article *-->
"sections_1 *" Sect1 Section *--> "paras *" Para Sect1
*--> "sections_2 *" Sect2 @enduml

```

```

rule BibTeXFile2DocBook {

    from

    b : BibTeX!BibTeXFile

    to

    d : DocBook!DocBook (

    books <- Sequence {

    thisModule.createBook(b)

    }

    )

}

-- Helper rule to create a Book in DocBook

rule createBook(b : BibTeX!BibTeXFile) : DocBook!Book
{

    to

    book : DocBook!Book (

    articles <- Sequence {

    thisModule.createFullListSection(b),

    thisModule.createAuthorsSection(b),

    thisModule.createTitlesSection(b),

    thisModule.createJournalsSection(b)

    }

    )

}

-- Rule to create the full list of bibliographic entries
section

rule createFullListSection(b : BibTeX!BibTeXFile) :
DocBook!Article {

    to

```

```

        article : DocBook!Article (
            title <- 'Full List of Bibliographic Entries',
            sections_1 <- Sequence {
                thisModule.createEntriesSection(b.entries)
            }
        )
    }

```

-- Rule to create a section for entries

```

rule createEntriesSection(entries :
Collection(BibTeX!BibTeXEntry)) : DocBook!Sect1 {
    to
        sect : DocBook!Sect1 (
            title <- 'Entries',
            paras <- entries->collect(e |
thisModule.createEntryPara(e))
        )
    }

```

-- Rule to create a paragraph for each entry

```

rule createEntryPara(entry : BibTeX!BibTeXEntry) :
DocBook!Para {
    to
        para : DocBook!Para (
            content <- entry.id
        )
    }

```

-- Rule to create the authors section

```

rule createAuthorsSection(b : BibTeX!BibTeXFile) :

```

```

DocBook!Article {

    to

    article : DocBook!Article (

        title <- 'Authors',

        sections_1 <- Sequence {

thisModule.createAuthorsListSection(b.entries->select(e
| e.ocllsKindOf(BibTeX!AuthoredEntry)).authors)

        }

    )

}

-- Rule to create a section for authors list

rule createAuthorsListSection(authors :
Collection(BibTeX!Author)) : DocBook!Sect1 {

    to

    sect : DocBook!Sect1 (

        title <- 'Authors',

        paras <- authors->collect(a |
thisModule.createAuthorPara(a))

    )

}

-- Rule to create a paragraph for each author

rule createAuthorPara(author : BibTeX!Author) :
DocBook!Para {

    to

    para : DocBook!Para (

        content <- author.author

    )

}

```

-- Rule to create the titles section

```
rule createTitlesSection(b : BibTeX!BibTeXFile) :  
DocBook!Article {
```

```
    to
```

```
    article : DocBook!Article (
```

```
        title <- 'Titles',
```

```
        sections_1 <- Sequence {
```

```
            thisModule.createTitlesListSection(b.entries->select(e |  
            e.ocllsKindOf(BibTeX!TitledEntry)).title)
```

```
        }
```

```
    )
```

```
}
```

-- Rule to create a section for titles list

```
rule createTitlesListSection(titles : Collection(String)) :  
DocBook!Sect1 {
```

```
    to
```

```
    sect : DocBook!Sect1 (
```

```
        title <- 'Titles',
```

```
        paras <- titles->collect(t |  
        thisModule.createTitlePara(t))
```

```
    )
```

```
}
```

-- Rule to create a paragraph for each title

```
rule createTitlePara(title : String) : DocBook!Para {
```

```
    to
```

```
    para : DocBook!Para (
```

```
        content <- title
```

```

    )

}

-- Rule to create the journals section

rule createJournalsSection(b : BibTeX!BibTeXFile) :
DocBook!Article {

    to

    article : DocBook!Article (

        title <- 'Journals',

        sections_1 <- Sequence {

thisModule.createJournalsListSection(b.entries->select(
e | e.ocllsKindOf(BibTeX!Article)).journal)

        }

    )

}

-- Rule to create a section for journals list

rule createJournalsListSection(journals :
Collection(String)) : DocBook!Sect1 {

    to

    sect : DocBook!Sect1 (

        title <- 'Journals',

        paras <- journals->collect(j |
thisModule.createJournalPara(j))

    )

}

-- Rule to create a paragraph for each journal

rule createJournalPara(journal : String) : DocBook!Para
{

```

	<pre>to para : DocBook!Para (content <- journal) } ...</pre> <p><i>This ATL transformation code maps elements from the BibTeXML model to the DocBook model according to the described transformation rules. It creates different sections in the DocBook for bibliographic entries, authors, titles, and journals. Each section is populated with the corresponding information from the BibTeXML model.</i></p>
--	--

Link for the Chat:

<https://chatgpt.com/share/a6685320-3d10-44b5-b7a8-fdf903e712f7>

2nd example - Class2Relational

Main prompt input	Output
<p>Given the transformation description and the following PlantUML metamodels, please give me the ATL code for the transformations.</p> <p>Transformation description: The Class to Relational example describes the simplified transformation of a class schema model to a relational database model.</p> <pre> Class PlantUML @startuml class NamedElt { name: EString } class Classifier { } class DataType { } class Class { isAbstract: EBoolean } class Attribute { multiValued: EBoolean } class Boolean { } class Integer { } class String { } NamedElt < -- Classifier Classifier < -- DataType Classifier < -- Class NamedElt < -- Attribute Class --> "super" * Class Class "owner" *-- "attr" Attribute Attribute --> "type" Classifier @enduml Relational PlantUML @startuml class Named { name: EString } class Table { } class Column { } class Type { } Named < -- Table Named < -- Column Named < -- Type Table "owner" *-- "col" * Column Table "keyOf" -- "key" * Column Column --> "type" Type </pre>	<p>Here's the ATL code for the transformation from the Class model to the Relational model based on the provided PlantUML metamodels:</p> <pre> ``atl module Class2Relational; create OUT : Relational from IN : Class; -- Helper to map Class DataTypes to Relational Types helper context Class!Classifier def: mapType() : Relational!Type = if self.ocllsTypeOf(Class!Boolean) then Relational!Type.allInstances()->select(t t.name = 'Boolean')->asOrderedSet()->first() else if self.ocllsTypeOf(Class!Integer) then Relational!Type.allInstances()->select(t t.name = 'Integer')->asOrderedSet()->first() else if self.ocllsTypeOf(Class!String) then Relational!Type.allInstances()->select(t t.name = 'String')->asOrderedSet()->first() else null endif; -- Rule to transform NamedElt to Named rule NamedElt2Named { from </pre>

@enduml

```
s : Class!NamedElt
to
t : Relational!Named (
name <- s.name
)
}

-- Rule to transform Class to Table
rule Class2Table {
from
s : Class!Class
to
t : Relational!Table (
name <- s.name,
col <- s.attr->collect(a |
thisModule.Attribute2Column(a))
)
}

-- Rule to transform Attribute to Column
rule Attribute2Column {
from
s : Class!Attribute
to
t : Relational!Column (
name <- s.name,
type <- s.type.mapType()
)
}
```

```
}
```

```
-- Create Type instances for Boolean, Integer, and  
String
```

```
rule CreateBooleanType {
```

```
    from
```

```
    s : Class!Boolean
```

```
    to
```

```
    t : Relational!Type (
```

```
        name <- 'Boolean'
```

```
    )
```

```
}
```

```
rule CreateIntegerType {
```

```
    from
```

```
    s : Class!Integer
```

```
    to
```

```
    t : Relational!Type (
```

```
        name <- 'Integer'
```

```
    )
```

```
}
```

```
rule CreateStringType {
```

```
    from
```

```
    s : Class!String
```

```
    to
```

```
    t : Relational!Type (
```

```
        name <- 'String'
```

	<pre>) } ... </pre> <p>### Explanation:</p> <p>1. Helper Function (<code>mapType</code>): This helper function maps the data types from the class model (<code>Boolean</code>, <code>Integer</code>, <code>String</code>) to the corresponding relational types.</p> <p>2. Rule <code>NamedElt2Named</code>: This rule transforms any <code>NamedElt</code> elements from the class model to <code>Named</code> elements in the relational model.</p> <p>3. Rule <code>Class2Table</code>: This rule transforms <code>Class</code> elements from the class model to <code>Table</code> elements in the relational model. It also maps attributes (<code>attr</code>) of the class to columns in the table by calling the <code>Attribute2Column</code> rule.</p> <p>4. Rule <code>Attribute2Column</code>: This rule transforms <code>Attribute</code> elements from the class model to <code>Column</code> elements in the relational model and maps the data type of the attribute to the corresponding relational type.</p> <p>5. Rules <code>CreateBooleanType</code>, <code>CreateIntegerType</code>, and <code>CreateStringType</code>: These rules create instances of the <code>Type</code> class in the relational model for the <code>Boolean</code>, <code>Integer</code>, and <code>String</code> types respectively.</p> <p>Make sure to validate the generated ATL code in your environment and adjust any details that might be specific to your ATL configuration or version.</p>
--	--

Link for the Chat:

<https://chatgpt.com/share/afbb58ff-cf72-4d68-a199-d49d86d7822b>

3rd example - Families2Persons

Main prompt input	Output
<p>Given the transformation description and the following PlantUML metamodels, please give me the ATL code for the transformations.</p> <p>Transformation description: This is the source code of the ATL Use Case: "Families to Persons". This use case describes a very simple model transformation example.</p> <p>Initially we have a text describing a list of families. We want to transform this into another text describing a list of persons.</p> <pre> Families PlantUML @startuml class Family { lastName: String } class Member { firstName: String } Family "familyFather" *-- "father" Member Family "familyMother" *-- "mother" Member Family "familySon" *-- "sons *" Member Family "familyDaughter" *-- "daughters *" Member @enduml @startuml abstract class Person { fullName: String } class Male { } class Female { } Person < -- Male Person < -- Female @enduml </pre>	<p>To create an ATL transformation from the Families metamodel to the Persons metamodel, we'll define a transformation that maps each family member to a person and constructs their full names based on their relationships within the family.</p> <p>Here is the ATL code for the transformation:</p> <pre> ``atl module Families2Persons; create OUT : Persons from IN : Families; helper def : concatNames(lastName : String, firstName : String) : String = lastName.concat(" ").concat(firstName); rule FamilyToPersons { from f : Families!Family to father : Persons!Male (fullName <- thisModule.concatNames(f.lastName, f.familyFather.firstName)), mother : Persons!Female (fullName <- thisModule.concatNames(f.lastName, f.familyMother.firstName)), sons : Sequence { </pre>

```

s : Persons!Male |

s.fullName <-
thisModule.concatNames(f.lastName, son.firstName)

} <- f.familySon,

daughters : Sequence {

d : Persons!Female |

d.fullName <-
thisModule.concatNames(f.lastName,
daughter.firstName)

} <- f.familyDaughter
}

```

...

Explanation:

1. **Module Declaration**:

```

````atI

module Families2Persons;

create OUT : Persons from IN : Families;

...

```

2. **Helper Definition**:

```

````atI

helper def : concatNames(lastName : String,
firstName : String) : String =

lastName.concat(' ').concat(firstName);

...

```

This helper function concatenates the last name and the first name to form the full name.

3. **Transformation Rule**:

``atl

```
rule FamilyToPersons {  
    from  
  
    f : Families!Family  
  
    to  
  
    father : Persons!Male (  
  
        fullName <-  
thisModule.concatNames(f.lastName,  
f.familyFather.firstName)  
  
    ),  
  
    mother : Persons!Female (  
  
        fullName <-  
thisModule.concatNames(f.lastName,  
f.familyMother.firstName)  
  
    ),  
  
    sons : Sequence {  
  
        s : Persons!Male |  
  
            s.fullName <-  
thisModule.concatNames(f.lastName, son.firstName)  
  
        } <- f.familySon,  
  
    daughters : Sequence {  
  
        d : Persons!Female |  
  
            d.fullName <-  
thisModule.concatNames(f.lastName,  
daughter.firstName)  
  
        } <- f.familyDaughter  
  
    }  
    ...  
}
```

This rule maps each `Family` instance to `Person` instances. Specifically:

- The `father` and `mother` are mapped to `Male` and `Female` instances respectively.

	<p>- The `sons` and `daughters` are mapped to `Male` and `Female` instances respectively, creating sequences of `Male` and `Female` persons from the list of sons and daughters.</p> <p>This ATL code will transform each family in the source model into corresponding person objects in the target model, preserving their relationships and constructing full names appropriately.</p>
--	---

Link for the Chat:

<https://chatgpt.com/share/d867e21c-8de6-44a0-a51a-2201f6d7a1d4>

4th example - RSS2Atom

Main prompt input	Output
<p>Given the transformation description and the following PlantUML metamodels, please give me the ATL code for the transformations.</p> <p>Transformation description: RSS is a format for syndicating news and the content of news-like sites. Atom is an XML-based file format intended to allow lists of information, known as "feeds", to be synchronised between publishers and consumers. This transformation permits to get an ATOM model from an RSS model.</p> <pre> RSS PlantUML @startuml class RSS { version: String } class Channel { title: String link: String description: String language: String copyright: String managingEditor: String webmaster: String generator: String docs: String ttl: Integer rating: String skipHours: Integer[] pubDate: String skipDays: String[] lastBuildDate: String } class Item { title: String link: String description: String pubDate: String author: String comments: String guid: String } class Image { url: String title: String link: String description: String width: Integer height: Integer } class TextInput { title: String description: String name: String </pre>	<p><i>To transform an RSS model into an ATOM model using the ATL (Atlas Transformation Language), you can follow the provided metamodels. Here's an example of the ATL code that performs the transformation:</i></p> <pre> '''atl module RSStoATOM; create OUT : ATOM from IN : RSS; -- Transformation rules -- Transform RSS to ATOM rule RSStoATOM { from rss : RSS!RSS to atom : ATOM!ATOM (title <- rss.channel.title, id <- rss.channel.link, -- Assuming link as unique ID subtitle <- rss.channel.description, rights <- rss.channel.copyright, icon <- rss.channel.image.url, logo <- rss.channel.image.url, lastUpdate <- rss.channel.lastBuildDate, links <- rss.channel.link, -- Assuming link as the main link generator <- </pre>

```

        link: String
    }
    class Cloud {
        domain: String
        port: Integer
        path: String
        registerProcedure: String
        protocol: String
    }
    class Category {
        domain: String
        value: String
    }
    class Enclosure {
        url: String
        lenght: Integer
        type: String
    }
    class Source {
        url: String
        value: String
    }
}
RSS "rss" *-- "channel" Channel
Channel "channel" *-- "image" Image
Channel "channel" *-- "textInput" TextInput
Channel "channel" *-- "cloud" Cloud
Channel "channel" *-- "category" Category
Channel "channel" *-- "items" Item
Item *--> "source" Source
Item *--> "enclosure" Enclosure
Item "items" *-- "category" Category
@enduml

ATOM PlantUML
@startuml
class ATOM {
    title: String
    id: String
    subtitle: String
    rights: String
    icon: String
    logo: String
    lastUpdate: String
}
class Entry {
    title: String
    id: String
    rights: String
    summary: String
    published: String
    lastUpdate: String
}
class Source {
    id: String
    icon: String
    logo: String
    rights: String
    title: String
    subtitle: String
    lastUpdate: String
}

```

```

thisModule.Generator(rss.channel.generator),

    categories <- rss.channel.category,

    authors <- Sequence {
thisModule.Author(rss.channel.managingEditor) },

    contributors <- Sequence {
thisModule.Contributor(rss.channel.webmaster) },

    entrie <- rss.channel.items

    )
}

-- Transform Channel Item to Entry

rule ItemToEntry {

    from

    item : RSS!Item

    to

    entry : ATOM!Entry (

        title <- item.title,

        id <- item.guid,

        rights <- item.comments, -- Assuming
comments as rights

        summary <- item.description,

        published <- item.pubDate,

        lastUpdate <- item.pubDate, -- Assuming
pubDate as lastUpdate

        links <- item.link,

        source <- item.source,

        content <-
thisModule.Content(item.description), -- Assuming
description as content

        categories <- item.category,

        authors <- Sequence {
thisModule.Author(item.author) }

```

```

abstract class Content {
    type: String
    mode: String
    text: String
}
class InLineTextContent {
}
class InLineXHTMLContent {
}
class InLineOtherContent {
}
class OutOfLineContent {
    src: String
}
class Generator {
    uri: String
    version: String
    name: String
}
class Category {
    term: String
    scheme: String
    label: String
}
class Link {
    rel: String
    hrefl: String
    type: String
    hreflang: String
    title: String
    lenght: Integer
}
class Person {
    name: String
    uri: String
    email: String
}
class Author {
}
class Contributor {
}
Content <|-- InLineTextContent
Content <|-- InLineXHTMLContent
Content <|-- InLineOtherContent
Content <|-- OutOfLineContent
Person <|-- Author
Person <|-- Contributor
ATOM "atom" *-- "links" ** Link
ATOM *--> "generator" Generator
ATOM "atom" *-- "categories" ** Category
ATOM "atom" *-- "authors" ** Author
ATOM *--> "contributors" ** Contributor
ATOM "atom" *-- "entrie" ** Entry
Entry "entry" *-- "links" ** Link
Entry *--> "source" Source
Entry *--> "content" Content
Entry "entry" *-- "categories" ** Category
Entry *--> "authors" ** Author
Entry *--> "contributors" ** Contributor
Source *--> "links" ** Link
Source *--> "generator" Generator

```

```

)
}

-- Helper to create Generator

helper def: Generator(name : String) : ATOM!Generator
=

    if name.ocllsUndefined() then

        null

    else

        ATOM!Generator {

            name <- name,

            version <- " -- Assuming version is not
provided

        }

        endif;

-- Helper to create Author

helper def: Author(name : String) : ATOM!Author =

    if name.ocllsUndefined() then

        null

    else

        ATOM!Author {

            name <- name

        }

        endif;

-- Helper to create Contributor

helper def: Contributor(name : String) :
ATOM!Contributor =

    if name.ocllsUndefined() then

```

```
Source *--> "contributors *" Contributor
Source *--> "categories *" Category
Source *--> "author" Author
@enduml
```

```
null
```

```
else
```

```
ATOM!Contributor {
```

```
name <- name
```

```
}
```

```
endif;
```

```
-- Helper to create Content
```

```
helper def: Content(text : String) : ATOM!Content =
```

```
ATOM!InLineTextContent {
```

```
type <- 'text',
```

```
mode <- 'escaped',
```

```
text <- text
```

```
};
```

```
...
```

This ATL code defines the transformation rules and helpers needed to convert an RSS model into an ATOM model. Here are the main points of the transformation:

1. The 'RSSToATOM' rule transforms an 'RSS' instance into an 'ATOM' instance.

2. The 'ItemToEntry' rule transforms 'Item' instances from the RSS feed into 'Entry' instances in the ATOM feed.

3. Helpers ('Generator', 'Author', 'Contributor', and 'Content') are defined to handle specific elements and ensure they are transformed correctly.

You may need to adjust this code depending on specific details and requirements of your transformation, but this provides a foundational example to get started.

Link for the Chat:

<https://chatgpt.com/share/1efcf86a-8f2d-4551-9077-b9e6a683267f>

5th example - Tree2List

Main prompt input	Output
<p>Given the transformation description and the following PlantUML metamodels, please give me the ATL code for the transformations.</p> <p>Transformation description: This transformation presents a basic example where a tree is transformed into a list. This kind of transformation is usually made by an imperative Depth First Traversal algorithm.</p> <pre> Tree PlantUML @startuml abstract class TreeElement { name: EString } class Node { } class Leaf { size: LeafSize } enum LeafSize { small: small medium: medium big: big } TreeElement < -- Node TreeElement < -- Leaf Node *--> "children *" TreeElement @enduml List PlantUML @startuml abstract class AbstractElement { name: EString } class RootElement { } class CommonElement { } AbstractElement < -- RootElement AbstractElement < -- CommonElement RootElement *--> "elements *" CommonElement @enduml </pre>	<p><i>To transform a tree structure into a list structure using ATL (Atlas Transformation Language), we will define the necessary rules to map each element from the Tree metamodel to the corresponding element in the List metamodel. Here is the ATL code for this transformation:</i></p> <pre> ``atl module TreeToListTransformation; create OUT : List from IN : Tree; -- Rule to transform the root of the tree into the root of the list rule TreeRootToListRoot { from t : Tree!Node to l : List!RootElement (name <- t.name, elements <- t.children->collect(c thisModule.TreeElementToListElement(c))) } -- Abstract rule to transform a TreeElement into an AbstractElement abstract rule TreeElementToListElement { from </pre>

```

        t : Tree!TreeElement

        to

        l : List!CommonElement (

            name <- t.name

        )

    }

-- Rule to transform a Node into a CommonElement

rule NodeToCommonElement extends
TreeElementToListElement {

    from

    t : Tree!Node

    to

    l : List!CommonElement (

        name <- t.name,

        elements <- t.children->collect(c |
thisModule.TreeElementToListElement(c))

    )

}

-- Rule to transform a Leaf into a CommonElement

rule LeafToCommonElement extends
TreeElementToListElement {

    from

    t : Tree!Leaf

    to

    l : List!CommonElement (

        name <- t.name

    )

}

```


-- Helper function to traverse the tree in a Depth First Traversal manner

*helper def : traverseTree(te : Tree!TreeElement) :
Sequence(List!CommonElement) =*

if te.oclIsTypeOf(Tree!Node) then

*Sequence{thisModule.TreeElementToListElement(te)}->
append(te.children->collect(c |
thisModule.traverseTree(c))->flatten())*

else

Sequence{thisModule.TreeElementToListElement(te)}

endif;

...

Explanation of the ATL Code

1. **Module Declaration:**

``atl

module TreeToListTransformation;

...

2. **Create Statement:**

``atl

create OUT : List from IN : Tree;

...

*This declares that we are transforming elements from
the Tree metamodel (input) to the List metamodel
(output).*

3. **TreeRootToListRoot Rule:**

```

``atl

rule TreeRootToListRoot {

    from

    t : Tree!Node

    to

    l : List!RootElement (

        name <- t.name,

        elements <- t.children->collect(c |
thisModule.TreeElementToListElement(c))

    )

}

...

```

This rule transforms the root node of the tree into the root element of the list. It maps the `name` attribute and recursively collects the transformed child elements.

4. **Abstract Rule TreeElementToListElement**:

```

``atl

abstract rule TreeElementToListElement {

    from

    t : Tree!TreeElement

    to

    l : List!CommonElement (

        name <- t.name

    )

}

...

```

This abstract rule is a generic mapping from any `TreeElement` to a `CommonElement` in the list.

5. ****NodeToCommonElement Rule****:

``atl

*rule NodeToCommonElement extends
TreeElementToListElement {*

from

t : Tree!Node

to

l : List!CommonElement (

name <- t.name,

*elements <- t.children->collect(c |
thisModule.TreeElementToListElement(c))*

)

}

...

*This rule extends the abstract rule to handle nodes
specifically, recursively collecting child elements.*

6. ****LeafToCommonElement Rule****:

``atl

*rule LeafToCommonElement extends
TreeElementToListElement {*

from

t : Tree!Leaf

to

l : List!CommonElement (

name <- t.name

)

}

...

*This rule extends the abstract rule to handle leaves,
mapping the `name` attribute.*

	<p>7. <i>**Helper Function traverseTree**:</i></p> <pre> ``atl helper def : traverseTree(te : Tree!TreeElement) : Sequence(List!CommonElement) = if te.ocllsTypeOf(Tree!Node) then Sequence{thisModule.TreeElementToListElement(te)}-> append(te.children->collect(c thisModule.traverseTree(c))->flatten()) else Sequence{thisModule.TreeElementToListElement(te)} endif; ... This helper function is used for Depth First Traversal of the tree, transforming each element and its children recursively. This ATL code performs the transformation as described, mapping a tree structure to a list structure with a depth-first traversal approach. </pre>
--	---

Link for the Chat:

<https://chatgpt.com/share/4d6b27f8-9d1e-4aab-a57f-66de9225b6c9>