Repository-Centric Process Metamodel

Metamodel definition and Concrete Syntaxes
Internal Report IRIT
Reference: IRIT/RR

Jacob Geisel and Brahim Hamid
IRIT, University of Toulouse
118 Route de Narbonne
31062 Toulouse Cedex 9, France
{geisel,hamid}@irit.fr

July 6, 2015

Abstract

Development processes for software construction are common knowledge and mainstream practice in most development organizations. Unfortunately, these processes offer little support in order to meet security requirements and are rarely formalized. As a consequence, increased risks of security vulnerabilities that are introduced into software in various stages of development exist. Secure software (or software security) engineering aims to avoid security vulnerabilities in software by considering security aspects from the very beginning and throughout the life cycle. From another perspective, formalizing processes offers the ability to teach and communicate them and to reason about them. In this paper, we propose an abstract syntax, by means of an OMG-style metamodel, for constructing the modeling language for security-oriented process models, build in an incremental fashion. The abstract syntax is based on the requirements for the process modeling language, describing various concerns, such as security engineering concepts, repository interactions and reuse. In addition to the abstract syntax, we propose an EBNF-style syntaxt and an implementation of a textual sytax with the Eclipse Xtext Framework.

Keywords: Process Modeling, Secure Software Engineering, Model-Driven Engineering, Reuse

Contents

Chapter 1

Metamodel

1.1 Working Example: Simplified V Modell XT

To illustrate the concepts presented in this chapter, we will use the working example described in the following.

The *V-Modell XT*is a high level framework and model for planning and realizing projects developed by the German government. It is the successor of the established *V-Modell 97*. The *V-Modell XT* allows to be tailored to specific needs of projects (e.g., size, budget, time constraints).

For demonstration purposes and better understandability, we simplify the process and focus on the software development part. Decision making and management phases and activities up to specifications upstream to the system development, as well as product maintenance are not treated. Figure 1.1 represents the software development part of the *V-Modell* adapted by the U.S. Department of Transportation and gives an appropriate overview on the process. We will use this simplified *V-Modell* along the presentation of

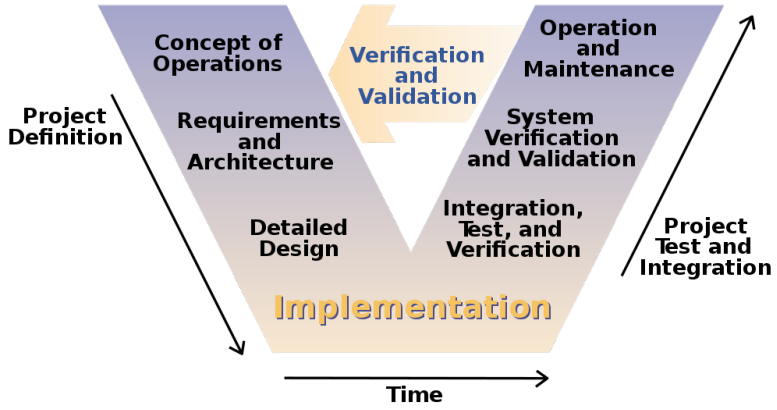

Figure 1.1: Simplified V-Modell by the US DoT

our approach, especially the metamodel, for illustration purposes. As we will see, some

concepts presented in this chapter, mainly those related to security and repository-centric engineering are not natively captured in the used *V-Modell*. For those concepts we indicate the usage and their use will be demonstrated in Chapter **??** and Chapter **??**

1.2 Metamodel Description

The Repository-Centric Process Metamodel (RCPM) is divided into the following six sub-packages: *(1)* Core, regrouping the basic concepts; *(2)* Process, for the concepts related to process engineering; *(3)* Safety, for safety related concepts; *(4)* Security, for security phases, activities; *(5)* Repository, for reuse of artifacts and interactions with a pattern repository and *(6)* Types, for typing process elements and enforcing reuse of process elements.

Figure 1.2: Metamodel Packages

1.2.1 Core Concepts

Core Package. The Core Package contains the elements which are used as top-level elements throughout the other packages and contain the basic attributes of all elements. It consists of three concepts

- *Element.* An `Element` is an abstraction of all concepts of the metamodel.
- *DescribableElement.* A `DescribableElement` is an abstraction of all Concepts which have a description, a name, etc. It is comparable to the `SPEM2.0::DescribableElement` or equivalent concepts in other metamodels. It contains attributes common to all concepts of the metamodel which need a description in form of a name and/or a detailed description.
- *Association.* The `Association` concepts is an abstraction of all relationships of the metamodel.

Process Package. The Process Package contains all the concepts used for process engineering, the basic concepts (Figure 1.3), concepts of a work breakdown structure

(Figure 1.4) and a breakdown structure (Figure 1.5) and concepts needed for detailing activities (Figure 1.6).

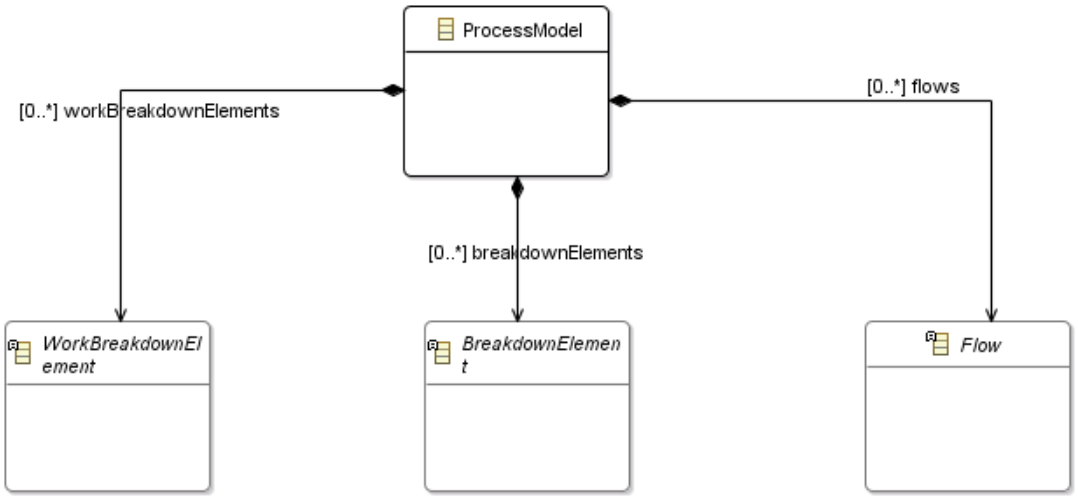

Figure 1.3: Metamodel Process Package: Process Base

- *ProcessElement.* A `ProcessElement` is a `DescribableElement` that represents an abstract generalization of all elements in RCPM process package. It can be considered as equivalent to the `SPEM2.0::ProcessElement`. Any element (excepting `Associations`) in a process structure is a `ProcessElement`. The base concept of `ProcessElement` is inspired by equivalent concepts in SPEM2 and OPF.

- *BreakdownElement.* A `BreakdownElement` is a special `ProcessElement`. Any of its concrete sub-concepts can be used to compose an `Activity`. It is inspired by the `SPEM2.0::MethodContent` and represents elements that intervene in as input/out, actor or support.

- *WorkBreakdownElement.* A `WorkBreakdownElement` is an abstract generalization for any `ProcessElement` that is part of a work breakdown structure. `WorkBreakdownElements` are interconnected by `Flows` to define a work sequence. A `WorkBreakdownElement` can contain additional attributes such as planning data to estimate time consumption. The `WorkBreakdownElement` concepts is inspired by multiple concepts from SPEM2 and OPF.

- *ProcessAssociation* A `ProcessAssociation` is an `Association` and it sub-concepts are used to link elements internally in the breakdown structure, especially in `Activities`.

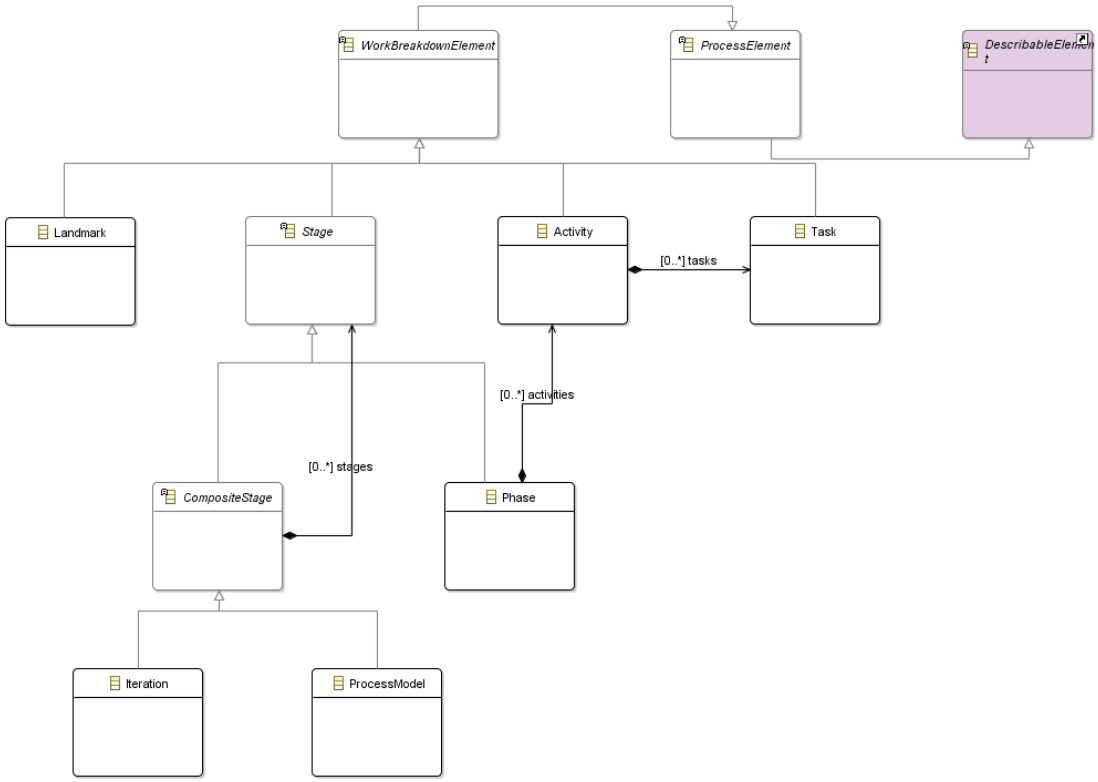

Figure 1.4: Metamodel Process Package: Work Breakdown Structure

- *ProcessModel.* A `Process` is a special `CompositeStage` that describes a structure for particular types of development projects or parts of them. `ProcessModels` are represented in RCPM as a set of `WorkBreakdownElements`. It is a combination of the `SPEM2.0::ActivityKind` stereotype «Process» and the base element of a UML model.

- *Stage.* A `Stage` is an abstract `WorkBreakdownElement`, which represents a generalization of all kinds of pieces of processes or a `ProcessModel` itself.

- *CompositeStage.* A `CompositeStage` is an abstract `Stage` element, which presents a generalization of all kinds of composite stage elements. The concept of `CompositeStage` is derived from `SPEM2.0::Activity.nestedBreakdownElements`.

- *Phase.* A `Phase` represents a significant period in a project, ending with the major management checkpoint, milestone, or set of deliverable. It is composed by activities and represents a certain significance in defining work breakdowns. It is inspired by the `SPEM2.0::ActivityKind` stereotype «Phase».

- *Iteration.* An `Iteration` groups a set of nested `Phases` that are repeated more than once. It represents an important structuring element to organize work in

repetitive cycles. The concept of `Iteration` can be associated with different rules in different methods. It is inspired by the `SPEM2.0::ActivityKind` stereotype «Iteration».

- *Activity.* An `Activity` is a `WorkBreakdownElement` that defines basic units of work within a Process. In other words, every process is composed by activities. It is composed by `WorkProduct`, `Role` and `Task`, which are associated by `Performer`, `Responsible` and `WorkDirection`.

- *Task.* A `Task` is a special `WorkBreakdownElement` that represents the work that should be done in an `Activity`. The `Task` needs to be related to the `Role` (`WorkProduct`) via the `Performer` (`WorkDirection`) `Association` and is stored in the nestedBreakdownElement composition of the `Activity`. A `Task` can be composed of certain number of `Steps` to achieve a result.

- *Step.* A `Step` is the detailed description of actions done in a `Task`.

- *Flow.* A `Flow` is an abstract `Association` as a generation of all the flow elements.

- *ControlFlow.* A `ControlFlow` is a `Flow` that presents the continuation of one `WorkBreakdownElement` to another `WorkBreakdownElement`. It is inspired by the `SPEM2.0::WorkSequence`.

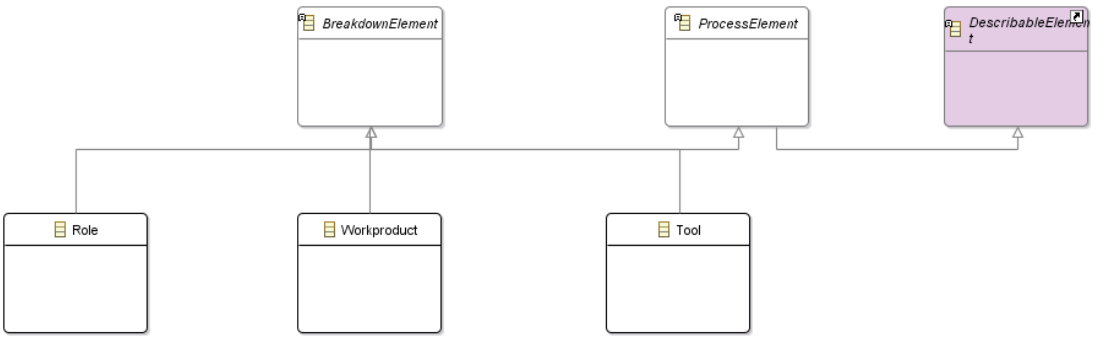

Figure 1.5: Metamodel Process Package: Breakdown Structure

- *Role* A `Role` is a `BreakdownElement` that represents a `Performer` of a `Task`, or a `Responsible` for a `WorkProduct`. The Role and Task and Role and Work Product are related via a `Performer` and `Responsible`, respectively. The `Role` concept is inspired by `SPEM2.0::RoleDefintion`.

- *WorkProduct* A `WorkProduct` is a `BreakdownElement` that represents an input and/or output for a `Task`. The `WorkProduct` is related to the `Task` via the `WorkDirection`. `WorkProduct` concept is inspired by `SPEM2.0::ToolDefintion`.

- *Tool* A `Tool` is a `BreakdownElement` that can be used to specify a tool's participation in a `Task`. The `Tool` concept is inspired by `SPEM2.0::ToolDefintion`.

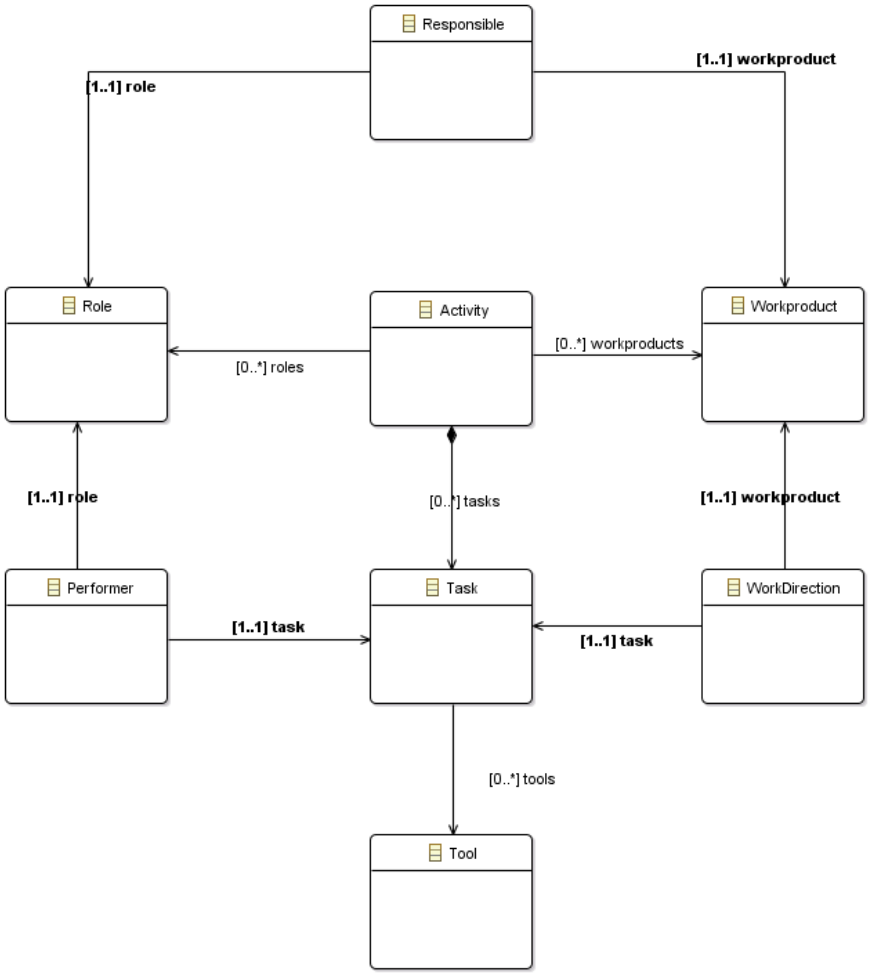

Figure 1.6: Metamodel Process Package: Activities

- *Performer.* `Performer` is an `Association` that represents a relationship between `Task` instances and `Role` instances. An instance of `Performer` links a `Role` instance to a `Task` instance.
- *WorkDirection.* `WorkDirection` is an `Association` that represents a relationship between `Task` instances and `WorkProduct` instances. An instance of `WorkDirection` links a `Task` instance to one `WorkProduct`.
- *Responsible.* `Responsible` is an `Association` that represents a relationship between `Role` instances and `WorkProduct` instances. An instance of `Responsible` links a `Role` instance to a `WorkProduct`.

1.2.2 Repository-Centric Engineering and Safety Life Cycle support

Extension to Process Package.

- *Landmark.* Landmark is a `WorkBreakdownElement` that represents the non-stage `ProcessElement`. It is the generation of all the landmark elements.
- *Partition.* `Partition` is a `CompositeStage` designed to represent parallel development partition in a process.
- *PartitionMark.* `PartitionMark` is a `Landmark` element that is used to show the beginning of certain partitions.
- *IntegrationMark.* `IntegrationMark` is a `Landmark` element that is used to show the end of certain partitions.
- *Milestone.* A `Milestone` describes a significant event in a development project, such as a major decision, completion of a deliverable, or meeting of a major dependency (like completion of a project phase). Because `Milestone` is commonly used to refer to both the event itself and the point in time at which the event is scheduled to happen, it is represented as a `Landmark`. The `Milestone` concept is inspired by `SPEM2.0::Milestone`, as well from concepts in RUP.

Safety Engineering Package. Based on the extended Process Package, the Safety Engineering Package regroups recurring Safety Engineering Concepts and extends and enhances basic process concepts (Figure 1.7).

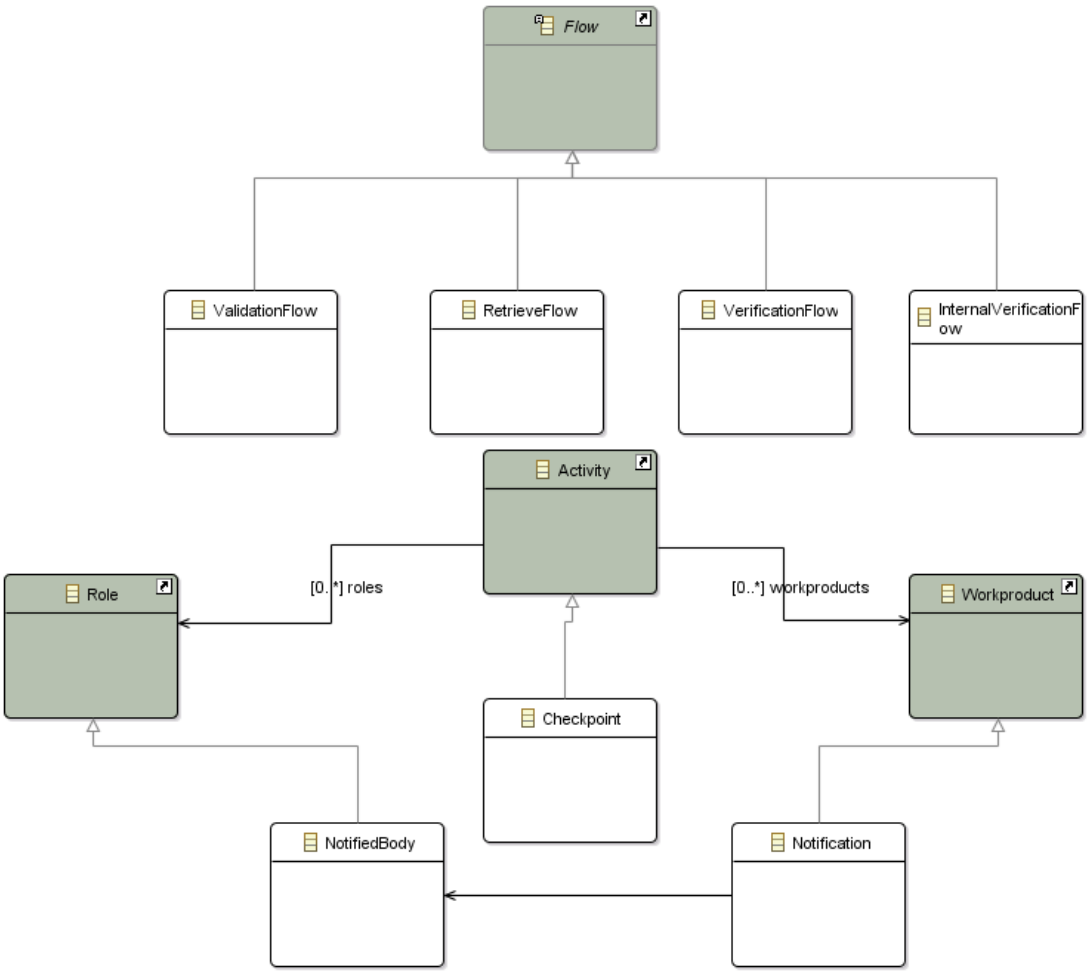

Figure 1.7: Metamodel Safety Package

- *Checkpoint.* A `Checkpoint` is defined as an `Activity` or `Phase` which represents the safety checks at different levels of the process. It represents a point when examination should happen with some activities or as one activity.

- *InternalVerificationFlow.* An `InternalVerificationFlow` is a `Flow` representing the internal verification relationship of one `WorkBreakdownElement` to another `WorkBreakdownElement`. Internal Verification Flow represents the internal verification that are performed before starting a new development phase. These actions are generally performed by a group independent to the design team and are restricted to the left branch of the safety life cycle (V-Modell).

- *VerificationFlow.* `VerificationFlow` is a `Flow` that represents the external verification relationship of one `WorkBreakdownElement` to another `WorkBreakdownElement`. External Verification Flow represents the normal verification performed at the right

branch of Safety Life Cycle (V-Modell). These actions are performed by the verification team and start at the end of the implementation phase.

- *ValidationFlow.* A `Validation Flow` is a `Flow` that represents the validation relationship between two `WorkBreakdownElements`. In a safety life cycle V-Modell, validation holds at the end of the implementation phase to confirm that the installed and commissioned SIFs (Safety Instrumented Functions) meet the Safety Requirements Specification (SRS). Although it is a safety life cycle concept, validation is kept general, meaning it can also concern different perspectives (e.g., dependability validation, security validation).

- *RetrieveFlow.* A `Retrieve Flow` is a `Flow` that represents the retrieve relationship from `Checkpoints` to `Phases` or `Activities`. The retrieve action will be proceeded when the `Checkpoints` do not pass the examination. The process will turn back to the previous `WorkBreakdownElement` to reexamine and/or redo the work.

- *Notification* A `Notification` is sent to inform `Roles` on the outcome of `Activities` or `Phases`. The `Notification`, a specialized `WorkProduct`, is the result of a `Checkpoint`, such as a safety audit.

- *NotifiedBody* A `NotifiedBody`, especially used during checkpoints for verification or validation, is sent a `Notification` on the outcome of the `Checkpoint`. A `NotifiedBody` can, for example, be an external safety assessor, receiving the output of the safety audit.

Repository Package. The Repository Package specializes some Process Concepts, extending them for Repository-Centric Software/System Engineering (Figure 1.8).

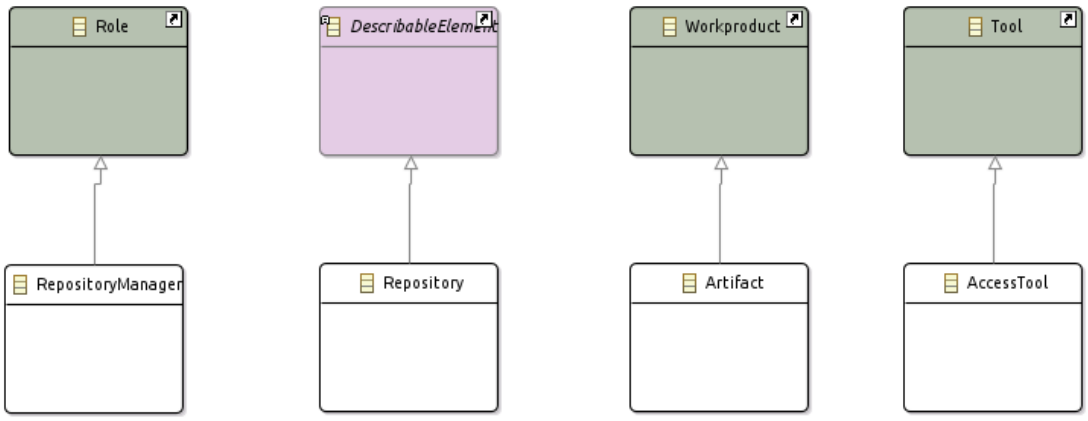

Figure 1.8: Metamodel Repository Package

- *Repository.* A `Repository` is an element that represents the repository used in a development process.

- *Artifact.* A `Artifact` represent a modeling artifact as it can be used in different Repository Software/System Engineering approaches (e.g., Pattern-Based).
- *RepositoryManager.* A `RepositoryManager` is a special `Role`, who manages the artifacts contained in a repository.
- *RepositoryTask* A `RepositoryTask` is a `Task` related to interactions with the `Repository`, such as accessing the `Repository` and retrieving `Artifacts`.
- *AccessTool.* An `AccessTool` is a specific `Tool` used to deposit and retrieve `Artifacts` to and from a `Repository`.
- *UseAssociation.* A `UseAssociation` is an `Association` representing a relationship between a `RepositoryTask` and a `Repository`. An instance of `UseAssociation` links a `RepositoryTask` instance to a `Repository` instance.

1.2.3 Security-Oriented Process support

Security Engineering Package. The Security Engineering Package regroups recurring Security Concepts, like Activities, Phases or Checkpoints (Figure 1.9). It is based on the Process and the Safety Packages and reuses some of their concepts.

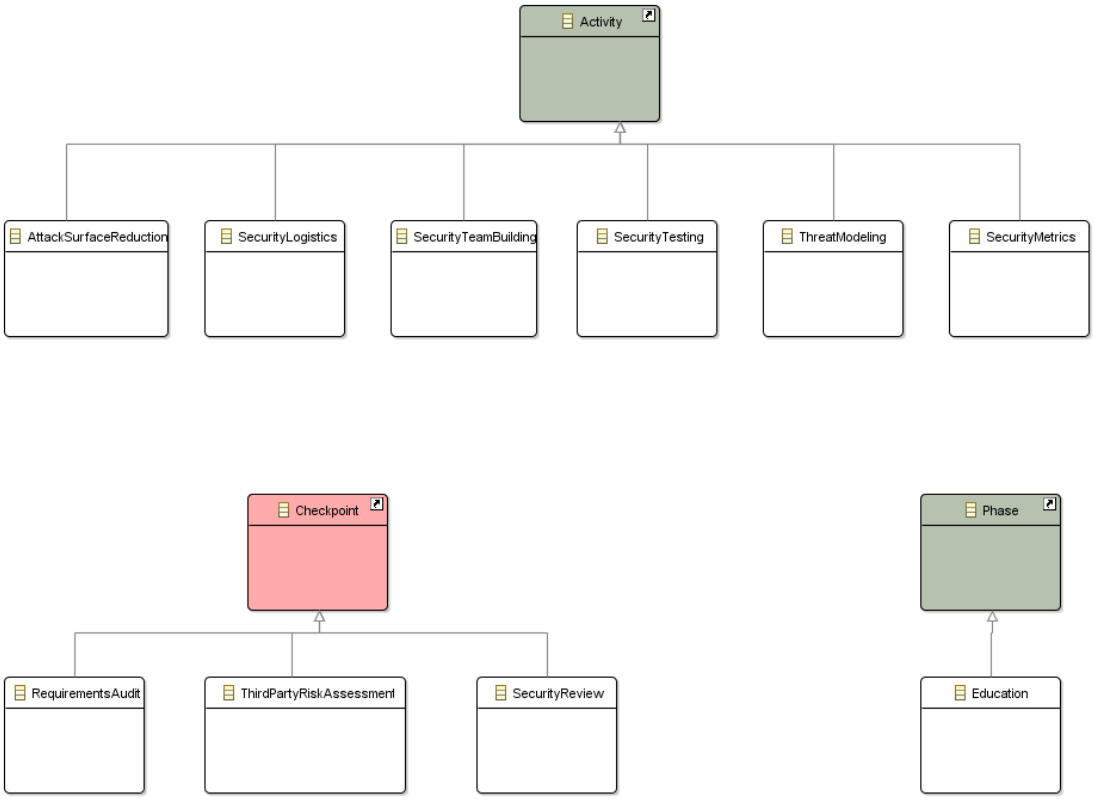

Figure 1.9: Metamodel Security Package

- Recurring Elements
  - *SecurityEngineer.* A `Role` describing a Security Engineer of the system-under-development. It a `Role` common to most security-oriented processes, including CLASP, SDL and Touchpoints.
  - *ThreatModeling.* Recurring `Activity` to define sets of possible attacks on system assets.
  - *SecurityReview.* `Checkpoint` targeting the entire system with a focus on highest-risk components. This is normally a `Checkpoint` at the end of each `Phase`.
  - *AttackSurfaceReduction.* `Activity` to minimize attack surface (reduction of privileges and/or access points).
  - *RiskAssessment.* `Checkpoint` to determine the quantitative or qualitative value of risk related to a concrete situation and a recognized threat.
- Education and Project Inception
  - *Education.* Education `Phase`. Commonly used `Phase` for learning of security aspects. This `Phase` also allows to create common security knowledge in the

team. This is common in security-oriented processes, especially SDL has a focus on this `Phase`.

  - *SecurityTeamBuilding.* `Activity` addressing the set-up of security-oriented responsibilities, project or company-wide. This is part of CLASP and SDL.
  - *SecurityLogistics.* `Activity` addressing logistic aspects (e.g., tools, type of security bugs to be handled). This is an `Activity` proposed especially by SDL and Touchpoints which enforces also team communication.
  - *SecurityMetrics.* `Activity` assessing the security posture of the product as well as enforcing accountability of security issues. All major security approaches offer this `Activity` since it allows to measure progress.

- Analysis and Requirements
  - *AnalysisLevelThreatModeling.* A refinement of `ThreatModeling`, using different approaches on threat modeling, such as use case driven, resource driven and/or knowledge driven, assessing whether known attacks can be valid and useful.
  - *SecurityRequirements.* An `Activity` specialized on defining the security requirements of the system-under-development. These requirements contain legal, financial, contractual and functional security requirements. This `activity` also resolves deficiencies and conflicts between requirement sets.

- Architecture
  - *ThirdPartyRiskAssessment.* Refinement of `RiskAssessment` to analyze weaknesses that arise by using third party software such as off-the-shelf components. This `RiskAssessment` is offered by CLASP, and in SDL in a limited way (e.g., focus on Microsoft specific software)
  - *RequirementsAudit.* `Checkpoint` to audit security (and non-security) requirements in order to assess their completeness. A `Checkpoint` inspired by CLASP, but existing more or less in all major security-oriented process models.
  - *ArchitectureLevelThreatModeling.* A refinement of `ThreatModeling` focusing on threat identification and risk assessment where risks in the system are identified and mitigated.
  - *SecurityArchitecture.* `Activity` to take into account the security requirement in the architecture of the system-under-development.

- Detailed Design
  - *SoftwareAttackSurfaceReduction.* A refinement of the `AttackSurfaceReduction`, which focuses especially on removing unimportant/unused features, reducing privileges and identifying entry points. SDL

- Implementation
  - *CodeInspection.* `Activity` defining the use of automated tools for verification purposes, as well as manual code inspection.
  - *SecurityImplementation.* `Activity` to implement the system using secure coding guidelines and tools to enforce security aspects.
  - *OperationalSecurityGuide.* This `Activity` creates the implementation documentation of the system-under-development. This contains, but is not limited to, configuration requirements, security architecture and security configuration documentation.
- Testing
  - *SecurityTesting.* `Checkpoint` testing the entire system with specific focus on legacy code. This `Activity` can contain black, gray and/or white box testing.
- Release
  - *CodeSigning.* `Activity` to provide stakeholders with a way to validate the origin and integrity of the software.
  - *ConfigurationManagment.* `Activity` dealing with the configuration of access control, logging, and monitoring

1.2.4 Reuse support via Model Libraries

Extension to Process Package. The root concepts of the Process Package `ProcessElement` and `ProcessAssociation` are extended to have additional attributes to be typed by concepts from the `TypeLibrary`.

Types Package. The Types Package is used to define libraries for reuse of process blocks (Figure 1.10) and to create constraints on Breakdown, Work Breakdown (Figure 1.11) and Association Elements.

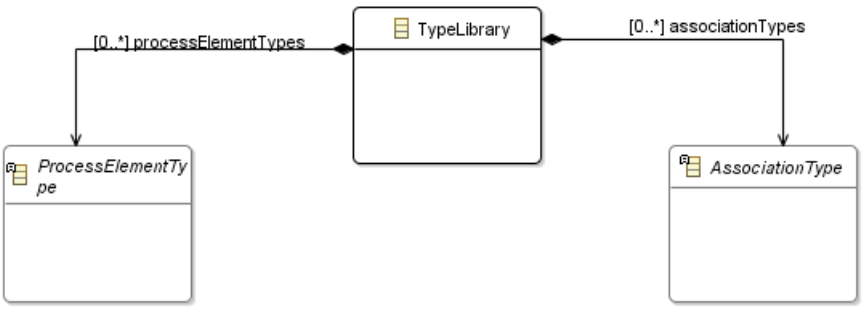

Figure 1.10: Metamodel Types Package (Type Library)

- *TypeLibrary.* A `TypeLibrary` is a library of models enabling reuse of process blocks (e.g. Phases, Activities), containing `ProcessElementTypes` and links (`ProcessAssociationType`) among these types.
- *ProcessElementType.* Generic Process Element Type. Instantiations of sub-concepts are used to type the different `ProcessElements`, like the different `WorkBreakdownElements` and `BreakdownElements`.
- *ProcessAssociationType.* An Element allowing to type different kinds of `ProcessAssociations`.

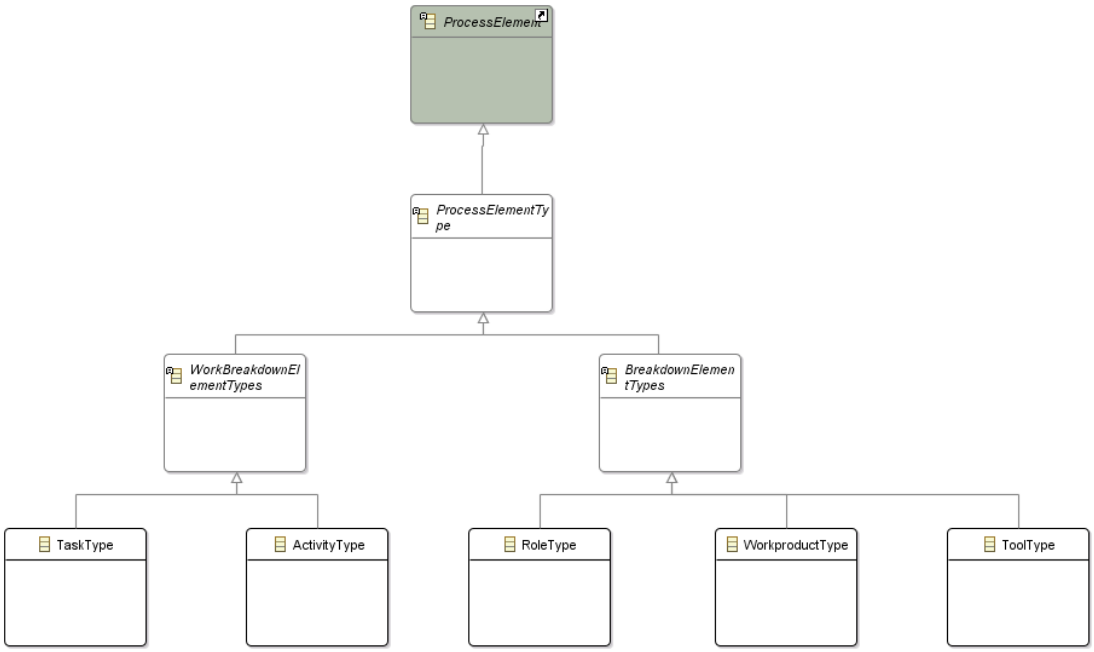

Figure 1.11: Metamodel Types Package (Process Element Types)

- *WorkBreakdownTypes.* An element (and especially its derived elements) allowing to build a Work Breakdown structures for reuse
- *BreakdownTypes.* An element (and especially its derived elements) allowing to build a Breakdown structures for reuse.

Usage of Type Package. The type package allows to create reusable process blocks, using a structure similar to the existing (Work) Breakdown Structure and the Process/Artifact/Safety/Security Concepts, without the need to create a whole process model. These block are stored as process model libraries and can be loaded into existing process models. An example of a type library is the SDL Type Library, containing

for the element types from the SDL Process [**?**]. This process is divided into seven phases (Education, Requirements, Design, ...). Each phase is modeled as a `PhaseType` and the containing `Activities` or security-oriented `Activities`, `Checkpoints` are modeled as their corresponding type. For the Requirements `Phase`, this giving the `PhaseType` *SDLRequirementsPhase* with as `Activities` the `SecurityRequirementsType` *SDLSecurityRequirements*, `ActivityType` *SDLQualityGatesBugBars* and the `RiskAssessmentType* *SDLSecurityAndPrivacyRiskAssessment*.

The elements in existing process models can now be typed by elements from the model library and are thus enhanced by the additional information. In addition, validation can be done to compare the (work) breakdown structure of the process model and the structure of the used blocks from the library to find inconsistencies in the application of the blocks. In this way missing elements in the process (e.g., `Activities`, `Roles`, `WorkProducts`) can be identified.

Two types of assistance are given to the process designer. A simple assistance, which allows, after importing a library, to type, either existing or newly created process elements, with types from the library. A second, more advanced assistance consists of injecting a library to an existing process to have a process model conforming to a security-oriented approach or standard. By using MDE techniques this can be realized by model transformation and can allow a (semi-)automated integration of a complete library.

Chapter 2

ENBF Concrete Syntax

```
ProcessModel_declaration = "ProcessModel" identifier "{"
 "(" Stage {"," Stage} ")",
 "(" BreakdownElement {"," BreakdownElement}")",
 "(" Flow {"," Flow } ")",
 Type {"," Type}
 "}" ";" .

Stage = Phase | CompositeStage .

Phase = Phase_declaration | Safety_Checkpoint | identifier .

Phase_declaration = "Phase" identifier "{"
 "(" Activity {"," Activity} ")",
 Type {"," Type}
 "}" ";" .

CompositeStage = Iteration | ProcessModel .

Iteration = Iteration_declaration | identifier .

Activity = Activity_declaration | Security_ThreatModeling_declaration |
    Security_SecurityArchitecture | [...] .

Activity_declaration = "Activity" identifier "{"
 "(" Task {"," Task} ")",
 "(" WorkProduct {"," WorkProduct} ")",
 "(" Role {"," Role} ")",
 "(" ProcessAssociation {"," ProcessAssociation} ")",
 Type {"," Type}
 "}" ";" .

Task = Task_declaration | identifier .

Task_declaration = "Task" identifier "{"
 "(" Step {"," Step} ")",
 Type {"," Type}
 "}" ";" .
```

```
ProcessAssociation = Performer | Responsible | Workdirection .

Performer = Performer_declaration | identifier .

Performer_declaration = "Performer" identifier "{"
 Role,
 Task,
 Type {"," Type}
 "}" .

Flow = ControlFlow | Safety_Flow .

Safety_Flow = Safety_ValidationFlow | Safety_InternalVerificationFlow |
    Safety_VerificationFlow | Safety_RetrieveFlow .

ControlFlow = ControlFlow_declaration | identifier .

Safety_Checkpoint = Safety_Checkpoint_declaration |
    Security_SecurityRequirementsAudit | identifier .

Security_SecurityRequirementsAudit =
    Security_SecurityRequirementsAudit_Activity_declaration |
    Security_SecurityRequirementsAudit_Phase_declaration | identifier .

Security_SecurityRequirementsAudit_Phase_declaration = "
    SecurityRequirementsAudit_Phase" identifier "{"
 "(" Activity {"," Activity} ")",
 Type {"," Type}
 "}" ";" .

Role = Role_declaration | Security_SecurityEngineer |
    Repository_RepositoryMananger identifier .

Security_SecurityEngineer = Security_SecurityEngineer_declaration |
    identifier .

Security_SecurityEngineer_declaration = "SecurityEngineer" identifier "{"
 Type {"," Type}
 "}" ";" .

[...]

identifier = "a..z,A..Z,$,_" { "a..z,A..Z,$,_,0..9,unicode character over
     00C0" } .

string = "''" { character } "''" .

character = "based on the unicode character set" .
```

Listing 2.1: Extracts from the textual syntax of RCPM

Chapter 3

Xtext Concrete Syntax

3.1 Xtext grammar of RCPM (Process Model Part)

```
grammar org.semcomdt.rcpm.Xtext with org.eclipse.xtext.common.Terminals

import "http://www.semcomdt.org/rcpm"
import "http://www.semcomdt.org/rcpm/process" as process
import "http://www.semcomdt.org/rcpm/types" as types
import "http://www.eclipse.org/emf/2002/Ecore" as ecore
import "http://www.semcomdt.org/rcpm/core" as core
import "http://www.semcomdt.org/rcpm/security" as security
import "http://www.semcomdt.org/rcpm/repository" as repository
import "http://www.semcomdt.org/rcpm/safety" as safety

ModelP returns ModelP:
  {ModelP}
    (elements+=ProcessElement (elements+=ProcessElement)*  )? ;

ProcessElement returns process::ProcessElement:
  //ProcessPackage
  BreakdownElement |  WorkBreakdownElement | Repository
  ;

Stage returns process::Stage:
  ProcessModel | Phase | Iteration |
  //Safety
  Checkpoint |
  //Security
   Education
  ;

BreakdownElement returns process::BreakdownElement:
  Role | WorkProduct | Tool |
  //Safety
  Notification | NotifiedBody |
  //Security
  SecurityEngineer |
```

```
  //Repository
  RepositoryManager | Artifact | AccessTool
  ;

Flow returns process::Flow:
  ControlFlow |
  //Safety
  InternalVerificationFlow | VerificationFlow | ValidationFlow |
      RetrieveFlow
  ;

WorkBreakdownElement returns process::WorkBreakdownElement:
  Landmark | Activity | Task  | Stage |
  //Safety
  //Checkpoint |
  //Security
  SecurityReview | RequirementsAudit |
  SecurityArchitecture | AttackSurfaceReduction |
      ArchitectureLevelThreatModeling |
  ThirdPartyRiskAssessment | SecurityMetrics | SecurityLogistics |
      SecurityTeamBuilding | RiskAssessment |
  //Repository
  ArtifactRetrieval | ArtifactSearch
  ;

ActivityAssociation returns process::ActivityAssociation:
  Performer | Responsible | WorkDirection;

EString returns ecore::EString:
  STRING | ID;

ProcessModel returns process::ProcessModel:
  {process::ProcessModel}
  'ProcessModel'
  name=EString
  '{'
    (('description' description=EString)? &
    ('stages' '{' stages+=[process::Stage|EString] ( "," stages+=[process
        ::Stage|EString])* '}' )? &
    ('breakdownElements' '{' breakdownElements+=[process::
        BreakdownElement|EString] ( "," breakdownElements+=[process::
        BreakdownElement|EString])* '}' )? &
    ('workBreakdownElements' '{' workBreakdownElements+=[process::
        WorkBreakdownElement|EString] ( "," workBreakdownElements+=[
        process::WorkBreakdownElement|EString])* '}' )? &
    ('flows' '{' flows+=Flow ( "," flows+=Flow)* '}' )? &
    ('type' '(' type+=[types::ProcessElementType|EString] ( "," type+=[
        types::ProcessElementType|EString])* ')' )?)
  '}';

Role returns process::Role:
  {process::Role}
  'Role'
```

```
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::RoleType|EString] ( "," type+=[types::
        RoleType|EString])* ')' )?
  '}';

WorkProduct returns process::WorkProduct:
  {process::WorkProduct}
  'WorkProduct'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::WorkProductType|EString] ( "," type+=[types
        ::WorkProductType|EString])* ')' )?
  '}';

Tool returns process::Tool:
  {process::Tool}
  'Tool'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::ToolType|EString] ( "," type+=[types::
        ToolType|EString])* ')' )?
  '}';

Landmark returns process::Landmark:
  {process::Landmark}
  'Landmark'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::ProcessElementType|EString] ( "," type+=[
        types::ProcessElementType|EString])* ')' )?
  '}';

Activity returns process::Activity:
  {process::Activity}
  'Activity'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::ActivityType|EString] ( "," type+=[types::
        ActivityType|EString])* ')' )?
    ('roles' '(' roles+=[process::Role|EString] ( "," roles+=[process::
        Role|EString])* ')' )?
    ('workProducts' '(' workProducts+=[process::WorkProduct|EString] (
        "," workProducts+=[process::WorkProduct|EString])* ')' )?
    ('tasks' '{' tasks+=[process::Task|EString] ( "," tasks+=[process::
        Task|EString])* '}' )?
    ('associations' '{' associations+=ActivityAssociation ( ","
        associations+=ActivityAssociation)* '}' )?
```

```
  '}';

Task returns process::Task:
  {process::Task}
  'Task'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::TaskType|EString] ( "," type+=[types::
        TaskType|EString])* ')' )?
    ('tools' '(' tools+=[process::Tool|EString] ( "," tools+=[process::
        Tool|EString])* ')' )?
  '}';

Phase returns process::Phase:
  {process::Phase}
  'Phase'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::PhaseType|EString] ( "," type+=[types::
        PhaseType|EString])* ')' )?
    ('activities' '{' activities+=[process::Activity|EString] ( ","
        activities+=[process::Activity|EString])* '}' )?
  '}';

Iteration returns process::Iteration:
  {process::Iteration}
  'Iteration'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::ProcessElementType|EString] ( "," type+=[
        types::ProcessElementType|EString])* ')' )?
    ('stages' '{' stages+=[process::Stage|EString] ( "," stages+=[process
        ::Stage|EString])* '}' )?
  '}';

ControlFlow returns process::ControlFlow:
  {process::ControlFlow}
  'ControlFlow'
  '('
    ( previous=[process::WorkBreakdownElement|EString])? ','
    ( next=[process::WorkBreakdownElement|EString])?
  ')';

Performer returns process::Performer:
  'Performer'
  '('
    role=[process::Role|EString] ','
    task=[process::Task|EString]
  ')';
```

```
Responsible returns process::Responsible:
  'Responsible'
  '('
    role=[process::Role|EString] ','
    workProduct=[process::WorkProduct|EString]
  ')';

WorkDirection returns process::WorkDirection:
  'WorkDirection'
  '('
    (direction=WorkDirectionKind) ','
    task=[process::Task|EString] ','
    workProduct=[process::WorkProduct|EString]
  ')';

enum WorkDirectionKind returns process::WorkDirectionKind:
        IN = 'IN' | OUT = 'OUT' | INOUT = 'INOUT';

//Safety
RetrieveFlow returns safety::RetrieveFlow:
  {safety::RetrieveFlow}
  'RetrieveFlow'
  '('
    ( previous=[process::WorkBreakdownElement|EString])? ','
    ( next=[process::WorkBreakdownElement|EString])?
  ')';

ValidationFlow returns safety::ValidationFlow:
  {safety::ValidationFlow}
  'ValidationFlow'
  '('
    ( previous=[process::WorkBreakdownElement|EString])? ','
    ( next=[process::WorkBreakdownElement|EString])?
  ')';

VerificationFlow returns safety::VerificationFlow:
  {safety::VerificationFlow}
  'VerificationFlow'
  '('
    ( previous=[process::WorkBreakdownElement|EString])? ','
    ( next=[process::WorkBreakdownElement|EString])?
  ')';

InternalVerificationFlow returns safety::InternalVerificationFlow:
  {safety::InternalVerificationFlow}
  'InternalVerificationFlow'
  '('
    ( previous=[process::WorkBreakdownElement|EString])? ','
    ( next=[process::WorkBreakdownElement|EString])?
  ')';

Checkpoint returns safety::Checkpoint:
  {safety::Checkpoint}
```

```
  'Checkpoint'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::ProcessElementType|EString] ( "," type+=[
        types::ProcessElementType|EString])* ')' )?
    ('activities' '{' activities+=[process::Activity|EString] ( ","
        activities+=[process::Activity|EString])* '}' )?
    ('roles' '(' roles+=[process::Role|EString] ( "," roles+=[process::
        Role|EString])* ')' )?
    ('workProducts' '(' workProducts+=[process::WorkProduct|EString] (
        "," workProducts+=[process::WorkProduct|EString])* ')' )?
    ('tasks' '{' tasks+=[process::Task|EString] ( "," tasks+=[process::
        Task|EString])* '}' )?
    ('associations' '{' associations+=ActivityAssociation ( ","
        associations+=ActivityAssociation)* '}' )?
  '}';


Notification returns safety::Notification:
  {safety::Notification}
  'Notification'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::ProcessElementType|EString] ( "," type+=[
        types::ProcessElementType|EString])* ')' )?
    ('notifiedBody' '(' notifiedBody=[safety::NotifiedBody|EString]')')
  '}';

NotifiedBody returns safety::NotifiedBody:
  {safety::NotifiedBody}
  'NotifiedBody'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::ProcessElementType|EString] ( "," type+=[
        types::ProcessElementType|EString])* ')' )?
  '}';

//Security
//Phases
Education  returns security::Education:
  {security::Education}
  'Education'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::PhaseType|EString] ( "," type+=[types::
        PhaseType|EString])* ')' )?
    ('activities' '{' activities+=[process::Activity|EString] ( ","
        activities+=[process::Activity|EString])* '}' )?
  '}';
```

```
//Checkpoints
SecurityReview returns security::SecurityReview:
  {security::SecurityReview}
  'SecurityReview'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::ProcessElementType|EString] ( "," type+=[
        types::ProcessElementType|EString])* ')' )?
    ('activities' '{' activities+=[process::Activity|EString] ( ","
        activities+=[process::Activity|EString])* '}' )?
    ('roles' '(' roles+=[process::Role|EString] ( "," roles+=[process::
        Role|EString])* ')' )?
    ('workProducts' '(' workProducts+=[process::WorkProduct|EString] (
        "," workProducts+=[process::WorkProduct|EString])* ')' )?
    ('tasks' '{' tasks+=[process::Task|EString] ( "," tasks+=[process::
        Task|EString])* '}' )?
    ('associations' '{' associations+=ActivityAssociation ( ","
        associations+=ActivityAssociation)* '}' )?
  '}';

RequirementsAudit returns security::RequirementsAudit:
  {security::RequirementsAudit}
  'RequirementsAudit'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::ProcessElementType|EString] ( "," type+=[
        types::ProcessElementType|EString])* ')' )?
    ('activities' '{' activities+=[process::Activity|EString] ( ","
        activities+=[process::Activity|EString])* '}' )?
    ('roles' '(' roles+=[process::Role|EString] ( "," roles+=[process::
        Role|EString])* ')' )?
    ('workProducts' '(' workProducts+=[process::WorkProduct|EString] (
        "," workProducts+=[process::WorkProduct|EString])* ')' )?
    ('tasks' '{' tasks+=[process::Task|EString] ( "," tasks+=[process::
        Task|EString])* '}' )?
    ('associations' '{' associations+=ActivityAssociation ( ","
        associations+=ActivityAssociation)* '}' )?
  '}';


RiskAssessment returns security::RiskAssessment:
  {security::RiskAssessment}
  'RiskAssessment'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::ProcessElementType|EString] ( "," type+=[
        types::ProcessElementType|EString])* ')' )?
    ('activities' '{' activities+=[process::Activity|EString] ( ","
        activities+=[process::Activity|EString])* '}' )?
```

```
    ('roles' '(' roles+=[process::Role|EString] ( "," roles+=[process::
        Role|EString])* ')' )?
    ('workProducts' '(' workProducts+=[process::WorkProduct|EString] (
        "," workProducts+=[process::WorkProduct|EString])* ')' )?
    ('tasks' '{' tasks+=[process::Task|EString] ( "," tasks+=[process::
        Task|EString])* '}' )?
    ('associations' '{' associations+=ActivityAssociation ( ","
        associations+=ActivityAssociation)* '}' )?
  '}';

//Activities
AttackSurfaceReduction returns security::AttackSurfaceReduction:
  {security::AttackSurfaceReduction}
  'AttackSurfaceReduction'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::ProcessElementType|EString] ( "," type+=[
        types::ProcessElementType|EString])* ')' )?
    ('roles' '(' roles+=[process::Role|EString] ( "," roles+=[process::
        Role|EString])* ')' )?
    ('workProducts' '(' workProducts+=[process::WorkProduct|EString] (
        "," workProducts+=[process::WorkProduct|EString])* ')' )?
    ('tasks' '{' tasks+=[process::Task|EString] ( "," tasks+=[process::
        Task|EString])* '}' )?
    ('associations' '{' associations+=ActivityAssociation ( ","
        associations+=ActivityAssociation)* '}' )?
  '}';

ArchitectureLevelThreatModeling returns security::
    ArchitectureLevelThreatModeling:
  {security::ArchitectureLevelThreatModeling}
  'ArchitectureLevelThreatModeling'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::ArchitectureLevelThreatModelingType|EString
        ] ( "," type+=[types::ArchitectureLevelThreatModelingType|EString
        ])* ')' )?
    ('roles' '(' roles+=[process::Role|EString] ( "," roles+=[process::
        Role|EString])* ')' )?
    ('workProducts' '(' workProducts+=[process::WorkProduct|EString] (
        "," workProducts+=[process::WorkProduct|EString])* ')' )?
    ('tasks' '{' tasks+=[process::Task|EString] ( "," tasks+=[process::
        Task|EString])* '}' )?
    ('associations' '{' associations+=ActivityAssociation ( ","
        associations+=ActivityAssociation)* '}' )?
  '}';

SecurityArchitecture returns security::SecurityArchitecture:
  {security::SecurityArchitecture}
  'SecurityArchitecture'
  name=EString
```

```
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::ProcessElementType|EString] ( "," type+=[
        types::ProcessElementType|EString])* ')' )?
    ('roles' '(' roles+=[process::Role|EString] ( "," roles+=[process::
        Role|EString])* ')' )?
    ('workProducts' '(' workProducts+=[process::WorkProduct|EString] (
        "," workProducts+=[process::WorkProduct|EString])* ')' )?
    ('tasks' '{' tasks+=[process::Task|EString] ( "," tasks+=[process::
        Task|EString])* '}' )?
    ('associations' '{' associations+=ActivityAssociation ( ","
        associations+=ActivityAssociation)* '}' )?
  '}';

ThirdPartyRiskAssessment returns security::ThirdPartyRiskAssessment:
  {security::ThirdPartyRiskAssessment}
  'ThirdPartyRiskAssessment'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::ProcessElementType|EString] ( "," type+=[
        types::ProcessElementType|EString])* ')' )?
    ('roles' '(' roles+=[process::Role|EString] ( "," roles+=[process::
        Role|EString])* ')' )?
    ('workProducts' '(' workProducts+=[process::WorkProduct|EString] (
        "," workProducts+=[process::WorkProduct|EString])* ')' )?
    ('tasks' '{' tasks+=[process::Task|EString] ( "," tasks+=[process::
        Task|EString])* '}' )?
    ('associations' '{' associations+=ActivityAssociation ( ","
        associations+=ActivityAssociation)* '}' )?
  '}';

SecurityTeamBuilding returns security::SecurityTeamBuilding:
  {security::SecurityTeamBuilding}
  'SecurityTeamBuilding'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::ProcessElementType|EString] ( "," type+=[
        types::ProcessElementType|EString])* ')' )?
    ('roles' '(' roles+=[process::Role|EString] ( "," roles+=[process::
        Role|EString])* ')' )?
    ('workProducts' '(' workProducts+=[process::WorkProduct|EString] (
        "," workProducts+=[process::WorkProduct|EString])* ')' )?
    ('tasks' '{' tasks+=[process::Task|EString] ( "," tasks+=[process::
        Task|EString])* '}' )?
    ('associations' '{' associations+=ActivityAssociation ( ","
        associations+=ActivityAssociation)* '}' )?
  '}';

SecurityLogistics returns security::SecurityLogistics:
  {security::SecurityLogistics}
  'SecurityLogistics'
```

```
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::ProcessElementType|EString] ( "," type+=[
        types::ProcessElementType|EString])* ')' )?
    ('roles' '(' roles+=[process::Role|EString] ( "," roles+=[process::
        Role|EString])* ')' )?
    ('workProducts' '(' workProducts+=[process::WorkProduct|EString] (
        "," workProducts+=[process::WorkProduct|EString])* ')' )?
    ('tasks' '{' tasks+=[process::Task|EString] ( "," tasks+=[process::
        Task|EString])* '}' )?
    ('associations' '{' associations+=ActivityAssociation ( ","
        associations+=ActivityAssociation)* '}' )?
  '}';

SecurityMetrics returns security::SecurityMetrics:
  {security::SecurityMetrics}
  'SecurityMetrics '
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::ProcessElementType|EString] ( "," type+=[
        types::ProcessElementType|EString])* ')' )?
    ('roles' '(' roles+=[process::Role|EString] ( "," roles+=[process::
        Role|EString])* ')' )?
    ('workProducts' '(' workProducts+=[process::WorkProduct|EString] (
        "," workProducts+=[process::WorkProduct|EString])* ')' )?
    ('tasks' '{' tasks+=[process::Task|EString] ( "," tasks+=[process::
        Task|EString])* '}' )?
    ('associations' '{' associations+=ActivityAssociation ( ","
        associations+=ActivityAssociation)* '}' )?
  '}';

//Roles
SecurityEngineer returns security::SecurityEngineer:
  {security::SecurityEngineer}
  'SecurityEngineer'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::SecurityEngineerType|EString] ( "," type+=[
        types::SecurityEngineerType|EString])* ')' )?
  '}';

//Repository
Repository returns repository::Repository:
  {repository::Repository}
  'Repository'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::SecurityEngineerType|EString] ( "," type+=[
        types::SecurityEngineerType|EString])* ')' )?
```

```
  '}';

RepositoryManager returns repository::RepositoryManager:
  {repository::RepositoryManager}
  'RepositoryManager'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::SecurityEngineerType|EString] ( "," type+=[
        types::SecurityEngineerType|EString])* ')' )?
  '}';

Artifact returns repository::Artifact:
  {repository::Artifact}
  'Artifact'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::WorkProductType|EString] ( "," type+=[types
        ::WorkProductType|EString])* ')' )?
  '}';

AccessTool  returns repository::AccessTool:
  {repository::AccessTool}
  'AccessTool'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::ToolType|EString] ( "," type+=[types::
        ToolType|EString])* ')' )?
  '}';

ArtifactRetrieval returns repository::ArtifactRetrieval:
  {repository::ArtifactRetrieval}
  'ArtifactRetrieval'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::TaskType|EString] ( "," type+=[types::
        TaskType|EString])* ')' )?
    ('tools' '(' tools+=[process::Tool|EString] ( "," tools+=[process::
        Tool|EString])* ')' )?
    ('repository' '(' repository=[repository::Repository|EString]')' )?
  '}';

ArtifactSearch returns repository::ArtifactSearch:
  {repository::ArtifactSearch}
  'ArtifactSearch'
  name=EString
  '{'
    ('description' description=EString)?
    ('type' '(' type+=[types::TaskType|EString] ( "," type+=[types::
        TaskType|EString])* ')' )?
```

```
    ('tools' '(' tools+=[process::Tool|EString] ( "," tools+=[process::
        Tool|EString])* ')' )?
    ('repository' '(' repository=[repository::Repository|EString]')' )?
  '}';
```

Listing 3.1: Extract from the Xtext grammar of RCPM

3.2 Xtext grammar of RCPM (Model Library Part)

```
grammar org.semcomdt.rcpm.XtextLib with org.eclipse.xtext.common.
    Terminals

import "http://www.eclipse.org/emf/2002/Ecore" as ecore
import "http://www.semcomdt.org/rcpm/types" as types
import "http://www.semcomdt.org/rcpm/process" as process
import "http://www.semcomdt.org/rcpm"

ModelL returns ModelL:
  {ModelL}
    (library=TypeLibrary)
    (elements+=ProcessElementType ( elements+=ProcessElementType)*)?
  ;

TypeLibrary returns types::TypeLibrary:
  {types::TypeLibrary}
  'TypeLibrary'
  name=EString
  '{'
    ('processElementTypes' processElementTypes+=[types::
        ProcessElementType|EString] ( "," processElementTypes+=[types::
        ProcessElementType|EString])* )?
    ('associationTypes' associationTypes+=FlowType ( "," associationTypes
        +=FlowType)*  )?
  '}';

ProcessElementType returns types::ProcessElementType:
  WorkBreakdownElementType | BreakdownElementType | StepType |
      RepositoryType
  ;

WorkBreakdownElementType returns types::WorkBreakdownElementType:
  TaskType | ActivityType | PhaseType |
  //safety
  CheckpointType |
  //security
  ThirdPartyRiskAssessmentType | ArchitectureLevelThreatModelingType |
      EducationType | RequirementsAuditType |
  //repository
  RepositoryTaskType
;
```

```
BreakdownElementType returns types::BreakdownElementType:
  RoleType | WorkProductType | ToolType |
  //safety

  //security
  SecurityEngineerType |
  //repository
  RepositoryManagerType | AccessToolType |  ArtifactType
;

StepType returns types::StepType:
  {types::StepType}
  'StepType'
  name=EString
  '{'
    ('description' description=EString)?
  '}';

AssociationType returns types::AssociationType:
  ActivityAssociationType | FlowType
;

FlowType returns types::FlowType:
  ControlFlowType
;

ControlFlowType returns types::ControlFlowType:
  {types::ControlFlowType}
  'ControlFlowType'
  '('
    ( previous=[types::WorkBreakdownElementType|EString])? ','
    ( next=[types::WorkBreakdownElementType|EString])?
  ')';

EString returns ecore::EString:
  STRING | ID;

PhaseType returns types::PhaseType:
  {types::PhaseType}
  'PhaseType'
  name=EString
  '{'
    ('description' description=EString)?
    ('activityTypes' activityTypes+=[types::ActivityType|EString] ( ","
        activityTypes+=[types::ActivityType|EString])* )?
  '}';

EducationType returns types::EducationType:
  {types::EducationType}
  'EducationType'
  name=EString
  '{'
    ('description' description=EString)?
```

```
    ('activityTypes' activityTypes+=[types::ActivityType|EString] ( ","
        activityTypes+=[types::ActivityType|EString])* )?
  '}';

RoleType returns types::RoleType:
  {types::RoleType}
  'RoleType'
  name=EString
  '{'
    ('description' description=EString)?
  '}';

SecurityEngineerType returns types::SecurityEngineerType:
  {types::SecurityEngineerType}
  'SecurityEngineerType'
  name=EString
  '{'
    ('description' description=EString)?
  '}';

WorkProductType returns types::WorkProductType:
  {types::WorkProductType}
  'WorkproductType'
  name=EString
  '{'
    ('description' description=EString)?
  '}';

TaskType returns types::TaskType:
  {types::TaskType}
  'TaskType'
  name=EString
  '{'
    ('description' description=EString)?
    ('stepTypes' stepTypes+=[types::StepType|EString] ( "," stepTypes+=[
        types::StepType|EString])* )?
  '}';

ToolType returns types::ToolType:
  {types::ToolType}
  'ToolType'
  name=EString
  '{'
    ('description' description=EString)?
  '}';

ActivityType returns types::ActivityType:
  {types::ActivityType}
  'ActivityType'
  name=EString
  '{'
    ('description' description=EString)?
```

```
    ('taskTypes' taskTypes+=[types::TaskType|EString] ( "," taskTypes+=[
        types::TaskType|EString])* )?
    ('roleTypes' '(' roleTypes+=[types::RoleType|EString] ( "," roleTypes
        +=[types::RoleType|EString])* ')' )?
    ('workProductTypes' '(' workProductTypes+=[types::WorkProductType|
        EString] ( "," workProductTypes+=[types::WorkProductType|EString])
        * ')' )?
    ('activityAssociationTypes' '{' activityAssociationTypes+=
        ActivityAssociationType ( "," activityAssociationTypes+=
        ActivityAssociationType)* '}' )?
  '}';

ActivityAssociationType returns types::ActivityAssociationType:
  PerformerType | ResponsibleType | WorkDirectionType
;

PerformerType returns types::PerformerType:
  {types::PerformerType}
  'PerformerType'
  '(' ( roleType=[types::RoleType|EString]) ','
  ( taskType=[types::TaskType|EString]) ')'
;

enum WorkDirectionKind returns process::WorkDirectionKind:
        IN = 'IN' | OUT = 'OUT' | INOUT = 'INOUT';

WorkDirectionType returns types::WorkDirectionType:
  {types::WorkDirectionType}
  'WorkDirectionType'
  '('
  (direction=WorkDirectionKind) ','
  ( workProductType=[types::WorkProductType|EString]) ','
  ( taskType=[types::TaskType|EString]) ')'
;

ResponsibleType returns types::ResponsibleType:
  {types::ResponsibleType}
  'ResponsibleType'
  '(' (roleType=[types::RoleType|EString]) ','
  (workProductType=[types::WorkProductType|EString]) ')'
;

//safety
CheckpointType returns types::CheckpointType:
  {types::CheckpointType}
  'CheckpointType'
  name=EString
  '{'
    ('description' description=EString)?
    ('taskTypes' taskTypes+=[types::TaskType|EString] ( "," taskTypes+=[
        types::TaskType|EString])* )?
    ('roleTypes' '(' roleTypes+=[types::RoleType|EString] ( "," roleTypes
        +=[types::RoleType|EString])* ')' )?
```

```
    ('workProductTypes' '(' workProductTypes+=[types::WorkProductType|
        EString] ( "," workProductTypes+=[types::WorkProductType|EString])
        * ')' )?
    ('activityTypes' activityTypes+=[types::ActivityType|EString] ( ","
        activityTypes+=[types::ActivityType|EString])* )?
    ('activityAssociationTypes' '{' activityAssociationTypes+=
        ActivityAssociationType ( "," activityAssociationTypes+=
        ActivityAssociationType)* '}' )?
  '}';

//security
RequirementsAuditType returns types::RequirementsAuditType:
  {types::RequirementsAuditType}
  'RequirementsAuditType'
  name=EString
  '{'
    ('description' description=EString)?
    ('taskTypes' taskTypes+=[types::TaskType|EString] ( "," taskTypes+=[
        types::TaskType|EString])* )?
    ('roleTypes' '(' roleTypes+=[types::RoleType|EString] ( "," roleTypes
        +=[types::RoleType|EString])* ')' )?
    ('workProductTypes' '(' workProductTypes+=[types::WorkProductType|
        EString] ( "," workProductTypes+=[types::WorkProductType|EString])
        * ')' )?
    ('activityTypes' activityTypes+=[types::ActivityType|EString] ( ","
        activityTypes+=[types::ActivityType|EString])* )?
    ('activityAssociationTypes' '{' activityAssociationTypes+=
        ActivityAssociationType ( "," activityAssociationTypes+=
        ActivityAssociationType)* '}' )?
  '}';

ThirdPartyRiskAssessmentType returns types::ThirdPartyRiskAssessmentType:
  {types::ThirdPartyRiskAssessmentType}
  'ThirdPartyRiskAssessmentType'
  name=EString
  '{'
    ('description' description=EString)?
    ('taskTypes' taskTypes+=[types::TaskType|EString] ( "," taskTypes+=[
        types::TaskType|EString])* )?
    ('roleTypes' '(' roleTypes+=[types::RoleType|EString] ( "," roleTypes
        +=[types::RoleType|EString])* ')' )?
    ('workProductTypes' '(' workProductTypes+=[types::WorkProductType|
        EString] ( "," workProductTypes+=[types::WorkProductType|EString])
        * ')' )?
    ('activityAssociationTypes' '{' activityAssociationTypes+=
        ActivityAssociationType ( "," activityAssociationTypes+=
        ActivityAssociationType)* '}' )?
  '}';

ArchitectureLevelThreatModelingType returns types::
    ArchitectureLevelThreatModelingType:
  {types::ArchitectureLevelThreatModelingType}
  'ArchitectureLevelThreatModelingType'
```

```
  name=EString
  '{'
    ('description' description=EString)?
    ('taskTypes' taskTypes+=[types::TaskType|EString] ( "," taskTypes+=[
        types::TaskType|EString])* )?
    ('roleTypes' '(' roleTypes+=[types::RoleType|EString] ( "," roleTypes
        +=[types::RoleType|EString])* ')' )?
    ('workProductTypes' '(' workProductTypes+=[types::WorkProductType|
        EString] ( "," workProductTypes+=[types::WorkProductType|EString])
        * ')' )?
    ('activityAssociationTypes' '{' activityAssociationTypes+=
        ActivityAssociationType ( "," activityAssociationTypes+=
        ActivityAssociationType)* '}' )?
  '}';

//Repository
RepositoryTaskType returns types::RepositoryTaskType:
  {types::RepositoryTaskType}
  'RepositoryTaskType'
  name=EString
  '{'
    ('description' description=EString)?
    ('repositoryType' respositoryType=[types::RepositoryType|EString])?
    ('stepTypes' stepTypes+=[types::StepType|EString] ( "," stepTypes+=[
        types::StepType|EString])* )?
  '}';

AccessToolType returns types::AccessToolType:
  {types::AccessToolType}
  'AccessToolType'
  name=EString
  '{'
    ('description' description=EString)?
  '}';

RepositoryManagerType returns types::RepositoryManagerType:
  {types::RepositoryManagerType}
  'RepositoryManagerType'
  name=EString
  '{'
    ('description' description=EString)?
  '}';

ArtifactType returns types::ArtifactType:
  {types::ArtifactType}
  'ArtifactType'
  name=EString
  '{'
    ('description' description=EString)?
  '}';

RepositoryType returns types::RepositoryType:
  {types::RepositoryType}
```

```
'RepositoryType'
name=EString
'{'
  ('description' description=EString)?
'}';
```

Listing 3.2: Extract from the Xtext grammar of RCPM