

Tools Output Integration Framework (TOIF)

Request for Comments

Version 1.2

OMG Document Number: sysa/2017-12-01

Standard document URL: <http://www.omg.org/spec/TOIF/1.2/pdf>

Normative Machine Consumable files:

<http://www.omg.org/spec/TOIF/1.2/toif.emof>

<http://www.omg.org/spec/TOIF/1.2/toif.xsd>

Non-normative Machine Consumable files:

<http://www.omg.org/spec/TOIF/1.2/toif.ecore>

<http://www.omg.org/spec/TOIF/1.2/toif.mdxml>

http://www.omg.org/spec/TOIF/1.2/toif_clean_uml.xmi

Copyright © 2017, KDM Analytics, Inc.
Copyright © 2017, Lockheed Martin Corporation
Copyright © 2017, The MITRE Corporation
Copyright © 2017, Model Driven Solutions
Copyright © 2017, 88solutions Corp.
Copyright © 2017, NoMagic, Inc

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 140 Kendrick Street, Needham, MA 02494, U.S.A.

TRADEMARKS

MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™, Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™, MOF™, OMG Interface Definition Language (IDL)™, and OMG SysML™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page <http://www.omg.org>, under Documents, Report a Bug/Issue (<http://www.omg.org/technology/agreement>.)

Table of Contents

1	Submission Specific Material	1
1.1	Submission Preface	1
1.2	Copyright Wavers	1
1.3	IPR Mode	1
1.4	Submitter Representatives	1
1.5	Author Team	1
2	Scope	2
3	Conformance	2
4	References	3
4.1	Normative References	3
4.2	Informative References	3
5	Terms and Definitions	3
6	Symbols	4
7	Additional Information	4
7.1	How to Read this Specification	4
7.2	Acknowledgements	5
8	TOIF Exchange Format	5
8.1	Objectives	5
8.2	TOIF Ecosystem	6
9	TOIF Conceptual Model	9
9.1	Basic Entities and Facts	9
9.2	“Housekeeping” Entities and Facts	17
9.3	Fact-oriented organization of TOIF XMI	20
10	TOIF Logical model	23
10.1	The basic elements of the TOIF XML	23
10.1.1	Finding Class Diagram	23
10.1.1.1	Finding Class	23
10.1.1.2	FindingsReportedAsType Class	25
10.1.1.3	FindingsReportedByGenerator Class	26
10.1.1.4	FindingsDefinedAsCWE Class	27
10.1.1.5	FindingsProducedByAdaptor Class	27
10.1.1.6	FindingHasCodeLocation Class	28
10.1.1.7	FindingReferencesFile Class	28
10.1.1.8	FindingsReportedInBuild Class	29
10.1.2	WeaknessType Class Diagram	29
10.1.2.1	WeaknessTypeIdentifier Class (abstract)	30
10.1.2.2	CWEIdentifier Class	31
10.1.2.3	SFPIIdentifier Class	31

10.1.2.4	SFPCluster Class	31
10.1.2.5	CWEBelongsToSFP Class	32
10.1.2.6	SFPBelongsToCluster Class	32
10.1.3	Weakness Class Diagram	32
10.1.3.1	Weakness Class.....	33
10.1.3.2	WeaknessIsDefinedAsCWE Class.....	34
10.1.3.3	WeaknessHasCodeLocation Class.....	34
10.1.3.4	WeaknessReferencesFile Class	35
10.1.4	Citing Class Diagram	35
10.1.4.1	Citing Class.....	36
10.1.4.2	CitingReferencesWeakness Class	37
10.1.4.3	CitingIsGeneratedAtDate Class.....	37
10.1.4.4	CitingAgent Class (abstract).....	38
10.1.4.5	CitingIsGeneratedByAgent Class	38
10.1.5	Code Location Class Diagram.....	38
10.1.5.1	CodeLocation Class	39
10.1.5.2	CodeLocationReferencesFile Class	40
10.1.6	File Class Diagram	40
10.1.6.1	File Class	41
10.1.6.2	FilesContainedInDirectory Class	42
10.1.6.3	FileBelongsToProject Class	42
10.1.7	Directory Class Diagram.....	43
10.1.7.1	Directory Class.....	43
10.1.7.2	DirectoryBelongsToProject Class.....	44
10.1.7.3	DirectoryIsContainedInDirectory Class.....	44
10.1.8	Semantic Statement Class Diagram	45
10.1.8.1	Statement Class	45
10.1.8.2	StatementIsInvolvedInFinding Class	46
10.1.8.3	StatementIsSinkOfFinding Class	46
10.1.8.4	StatementIsSourceOfFinding Class.....	47
10.1.8.5	StatementHasCodeLocation Class	47
10.1.8.6	StatementIsPrecededByStatement Class	48
10.1.9	Semantic Data Class Diagram	48
10.1.9.1	DataElement Class	48
10.1.9.2	DataIsInvolvedInFinding Class	49
10.1.9.3	DataIsInvolvedInStatement Class	50
10.1.9.4	DataIsDefinedAtCodeLocation Class.....	50
10.2	The housekeeping elements of the TOIF XML	51
10.2.1	Build Class Diagram	51
10.2.1.1	Build Class.....	51
10.2.1.2	BuildIsRelatedToProject Class	53
10.2.2	Housekeeping Class Diagram.....	53
10.2.2.1	BuildIsOrchestratedByTool Class.....	54
10.2.2.2	BuildIsProducedByOrganization Class	55
10.2.2.3	BuildIsOwnedByOrganization Class	55
10.2.2.4	BuildIsGeneratedByPerson Class	56
10.2.2.5	BuildIsSupervisedByPerson Class.....	57
10.2.2.6	BuildIsGeneratedAtDate Class.....	57
10.2.3	Project Class Diagram	57
10.2.3.1	Project Class	58
10.2.3.2	ProjectIsOwnedByOrganization Class.....	60
10.2.3.3	OrganizationIsInvolvedInProjectAsRole Class.....	60

10.2.3.4	PersonsInvolvedInProjectAsRole Class	61
10.2.4	Tools Class Diagram	61
10.2.4.1	Tool Class (abstract)	62
10.2.4.2	ToolsSuppliedByVendor Class.....	62
10.2.4.3	Generator Class	63
10.2.4.4	Adaptor Class.....	63
10.2.4.5	OrchestrationTool Class.....	63
10.2.4.6	Analytics Tool Class.....	64
10.2.4.7	AdaptorSupportsGenerator Class.....	64
10.2.4.8	AdaptorIsCapableOfFindingCWE Class	65
10.2.5	Organization Class Diagram	65
10.2.5.1	Organization Class	66
10.2.5.2	Vendor Class	67
10.2.5.3	OrganizationIsPartOfOrganizationAsRole Class.....	67
10.2.6	Person Class Diagram	67
10.2.6.1	Person Class.....	68
10.2.6.2	PersonsEmployedByOrganizationAsRole Class.....	69
10.2.7	Role Class Diagram	69
10.2.7.1	Role Class.....	69
10.3	The fact-oriented structure of the TOIF XML.....	71
10.3.1	Abstract Structure Class Diagram	71
10.3.1.1	TOIFSegment Class	72
10.3.1.2	TOIFElement Class (abstract).....	72
10.3.1.3	Entity Class (abstract)	72
10.3.1.4	Fact Class (abstract).....	72
10.3.1.5	Attribute Class (abstract).....	73
10.3.1.6	EvidentialRecord Class (abstract)	73
10.3.2	Abstract Types Class Diagram	73
10.3.2.1	Element Class (abstract)	74
10.3.3	Basic Entities Class Diagram	75
10.3.3.1	BasicEntity Class (abstract).....	75
10.3.4	Basic Facts Class Diagrams.....	75
10.3.4.1	FindingFact Class (abstract)	77
10.3.4.2	WeaknessTypeFact Class (abstract).....	77
10.3.4.3	WeaknessFact Class (abstract)	78
10.3.4.4	CodeLocationFact Class (abstract).....	78
10.3.4.5	SemanticFact Class (abstract).....	78
10.3.5	Basic Attributes Class Diagram	78
10.3.5.1	Offset Class	79
10.3.5.2	Checksum Class	79
10.3.5.3	Linenummer Class.....	79
10.3.5.4	Position Class.....	80
10.3.5.5	Name Class	80
10.3.5.6	Version Class.....	80
10.3.5.7	Description Class	80
10.3.5.8	Confidence Class.....	80
10.3.5.9	Criticality Class.....	81
10.3.5.10	Verdict Class	81
10.3.6	Housekeeping Entities Class Diagram	81
10.3.6.1	HousekeepingEntity Class (abstract)	82
10.3.7	Housekeeping Facts Class Diagrams	82
10.3.7.1	ToolFact Class (abstract).....	84

10.3.7.2	BuildFact Class (abstract).....	84
10.3.7.3	ProjectFact Class (abstract)	84
10.3.8	Housekeeping Attributes Class Diagram.....	85
10.3.8.1	Phone Class.....	85
10.3.8.2	Address Class	85
10.3.8.3	EmailAddress Class	86
10.4	Evidential Records in TOIF XML	86
10.4.1	EvidentialRecord Class Diagram	86
10.4.1.1	BuildRecord Class	86
10.4.1.2	CompileRecord Class	87
10.4.1.3	GeneratorRecord Class	87

Table of Figures

Figure 1. Organization of the TOIF specification.....	5
Figure 2. The Flow of the TOIF Protocol and the TOIF Ecosystem.....	8
Figure 3. UML class diagram Finding.....	24
Figure 4. UML class diagram WeaknessType.....	30
Figure 5. UML class diagram Weakness.....	33
Figure 6. UML class diagram Citing.....	36
Figure 7. UML class diagram Code Location.....	39
Figure 8. UML class diagram File.....	41
Figure 9. UML class diagram Directory.....	43
Figure 10. UML class diagram Semantic Statement.....	45
Figure 11. UML class diagram Semantic Data.....	49
Figure 12. UML class diagram Build.....	51
Figure 13. UML class diagram Housekeeping.....	53
Figure 14. UML class diagram Project.....	58
Figure 15. UML class diagram Tools.....	61
Figure 16. UML class diagram Organization.....	66
Figure 17. UML class diagram Person.....	68
Figure 18. UML class diagram Role.....	69
Figure 19. UML class diagram Abstract Structure.....	71
Figure 20. UML class diagram Abstract Types.....	74
Figure 21. UML class diagram Basic entities.....	75
Figure 22. UML class diagram Basic Facts 1.....	76
Figure 23. UML class diagram Basic Facts 2.....	76
Figure 24. UML class diagram Basic Facts 3.....	77
Figure 25. UML class diagram Basic Facts 4.....	77
Figure 26. UML class diagram Basic Attributes.....	79
Figure 27. UML class diagram Housekeeping entities.....	82
Figure 28. UML class diagram Housekeeping Facts 1.....	83
Figure 29. UML class diagram Housekeeping Facts 2.....	83
Figure 30. UML class diagram Housekeeping Facts 3.....	84
Figure 31. UML class diagram Housekeeping Attributes.....	85
Figure 32 UML class diagram EvidentialRecord.....	86

Preface

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at <http://www.omg.org/>.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A Specifications Catalog is available from the OMG website at:

http://www.omg.org/technology/documents/spec_catalog.htm

Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM)

Platform Specific Model and Interface Specifications

- CORBA services
- CORBA facilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
140 Kendrick Street
Building A, Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: pubs@omg.org

Certain OMG specifications are also available as ISO standards. Please consult <http://www.iso.org>

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

Courier - 10 pt. Bold: Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

NOTE: Terms that appear in italics are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

1 Submission Specific Material

1.1 Submission Preface

TOIF XML (XMI) is a common normalized format for representing the findings of static code analysis tools for the purpose of integrating multiple facts related to the system under assessment. This format is described in this specification first as a conceptual model in SBVR Structured English, focusing at the key noun and verb concepts, then by a more specific logical model in MOF/UML which determines the TOIF XML schema. The MOF metamodel is consistent with the SBVR Structured English representation (and can in principle, be systematically derived from it). The key to the TOIF MOF metamodel is that each verb concept is represented by an association class in such a way that the resulting XML has a “triple flavor”. SBVR stands for Semantic for Business Vocabulary and Rules. MOF stands for Meta Object Facility. XML stands for eXtended Markup Language. The acronym XMI stands for XML Metadata Interchange format. XMI is a specific form of XML that is associated with the Model Driven Development approach. XMI has been developed for the purpose of exchanging metadata such as models. XMI is standardized by OMG (current specification is identified as MOF 2.0 / XMI Mapping Specification, v2.1.1, document formal/07-12-01) and ISO (19503:2005).

1.2 Copyright Wavers

KDM Analytics Inc., Lockheed Martin Corporation, The MITRE Corporation, Model Driven Solutions, NoMagic Inc., and 88 Solutions Corp: (i) grants to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version, and (ii) grants to each member of the OMG a nonexclusive, royalty-free, paid up, worldwide license to make up to fifty (50) copies of this document for internal review purposes only and not for distribution, and (iii) has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used any OMG specification that may be based hereon or having conformed any computer software to such specification.

1.3 IPR Mode

The IPR Mode for this specification is: Non-Assertion Covenant

1.4 Submitter Representatives

Dr. Nikolai Mansourov, KDM Analytics, Inc., nick@kdmanalytics.com
Dr. Ben A. Calloni, Lockheed Martin Corporation, ben.a.calloni@lmco.com
Robert A. Martin, The MITRE Corporation, ramartin@mitre.org
Cory Casanave, Model Driven Solutions, cory-c@modeldriven.com
Gary Duncanson, NoMagic, Inc., gary@nomagic.com
Manfred Koethe, 88solutions Corp., koethe@88solutions.com

1.5 Author Team

Dr. Nikolai Mansourov, KDM Analytics Inc., nick@kdmanalytics.com
Dr. Ben A. Calloni, Lockheed Martin Corporation, ben.a.calloni@lmco.com
Manfred Koethe, 88solutions Corp., koethe@88solutions.com
Robert A. Martin, The MITRE Corporation, ramartin@mitre.org
Cory Casanave, Model Driven Solutions, cory-c@modeldriven.com

2 Scope

This document provides specification of the Tools Output Integration Framework (TOIF) XMI schema – the common reporting format of source/machine code weaknesses. TOIF XMI is one of the inputs to the TOIF Analyzer tool that integrates multiple weakness findings and the basic facts related to a system under assessment.

This specification describes TOIF schema at three different levels of abstraction. First, the specification describes the conceptual schema of the TOIF XML as SBVR Structured English focusing at a technology-independent description of the key noun and verb concepts involved in reporting weak ness findings. This conceptual schema defines a common vendor-neutral vocabulary for the TOIF Ecosystem. The conceptual schema that addresses the following concerns:

- Defining TOIF basic facts and entities
- Defining TOIF housekeeping concepts
- Presenting TOIF fact-oriented organization (emphasizing the noun and verb organization of TOIF facts which gives it a characteristic “triple flavor”)

Second, the TOIF specification then describes the MOF/UML metamodel of the TOIF XMI. This metamodel is consistent with the Structred English representation and can be, in principle, produced by a systematic transformation from the conceptual schema. The MOF metamodel determines the TOIF XML/XMI schema which can be derived from the UML model as described in the MOF and XMI specifications.

Third, the specification illustrates the usage of the TOIF XMI schema by providing examples of the TOIF XMI data representation that uses the TOIF XMI schema.

TOIF addresses two types of normalization. First, **syntactic** normalization addresses the differences in reporting formats of static code analysis tools, is addressed by the common TOIF XML schema. Second, the **semantic** normalization addresses the nomenclature of the findings by the static analysis tools is addressed by a mapping to a common nomenclature. The common nomenclature of weaknesses in TOIF is based on the Software Fault Pattern (SFP) system of clusters and individual patterns, and the further mapping to a catalog known as the Common Weakness Enumeration (CWE). The vendor-neutral common nomenclature of weakness types consisting of SFP and CWE is an integral part of the TOIF approach. Both the syntactic and the logical mapping from a proprietary reporting format of a given static analysis tool to TOIF is assumed to be implemented by an Adaptor in a way that is non-intrusive with respect to the static code analysis tool.

3 Conformance

The principle goal of TOIF is the common normalized format for representing the findings of multiple static code analysis tools for the purpose of integrating multiple facts related to the system under assessment.

To be TOIF compliant, a tool must fully support TOIF as one compliance point. An implementation must:

1. Provide the capability to generate XMI documents based on the TOIF XMI schema capturing findings from the existing model of the tool.
2. Provide the capability to import findings via representations based on the TOIF XMI schema and to map the findings into the existing model of the tool.
3. Ensure the mapping from each proprietary weakness type to common TOIF vendor-neutral weakness type based on SFP and CWE.

This compliance point is formally defined as follows. Let’s assume an SCA tool XYZ is capable of producing propriataty weakness findings of types WD_i where $i=1..k$. This means that given a set of files $F_1, …, F_n$ the tool XYZ may produce a set of findings, described by proprietary set report items $RWD_1, …, RWD_m$ such that each RWD_j refers to a certain weakness type RWD_i . The number of findings, m , depends on the presence of weaknesses in the input files F , as well as on the capability of the tool XYZ to identify a finding (true positive) and the capability of the tool XYZ to avoid reporting a false positive. The TOIF mapping is a set of k tuples (where k is the number of all distinct proprietary weakness types for tool XYZ), $\{WD_i, \{CWE_i, SFP_i, SFP\ Cluster-i \}\}$ where

WD_i is the proprietary description of the weakness type by tool XYZ

CWE_i is the CWE identifier aligned with the SFP_i and $SFP\ Cluster-i$ that provides the most specific description of the weakness

SFP_i is the SFP identifier that provides the most specific description of the weakness; the SFP catalog provides mappings from each SFP to a set of relevant CWE

SFP Cluster-i is the SFP Cluster that describes the broad and non-overlapping set of faults to which the weakness type belongs.

According to the TOIF specification, each individual finding RWDj refers only to the CWEj, while the relations between CWEi, SFPi and SFP Cluster-i are defined once at the weakness type level rather than at the finding level.

TOIF mapping constitutes the semantic specification of a **TOIF Adaptor** for tool XYZ. The other part of the specification of the Adaptor is the syntactic specification related to transforming the proprietary syntax of report items RWDj into TOIF data conforming to the TOIF XML.

4 References

4.1 Normative References

The following normative documents contain provisions which, through reference in this text, provide normative context for material in this specification.

- [kdm] Knowledge Discovery Metamodel (KDM), v1.4, <http://www.omg.org/spec/KDM/1.4>
- [sbvr] Semantics for Business Vocabulary and Rules (SBVR), v1.4, <http://www.omg.org/spec/SBVR/1.4/>
- [uml] Unified Modeling Language (UML), v2.5, <http://www.omg.org/spec/UML/2.5>
- [xmi] XML Metadata Interchange (XMI), v2.5.1, <http://www.omg.org/spec/XMI/2.5.1>
- [xml] Extensible Markup Language, v1.1, <http://www.w3.org/TR/xml11>
- [xsd-1] XML Schema Definition Language (XSD) v1.1 Part 1: Structures, <http://www.w3.org/TR/xmlschema11-1>
- [xsd-2] XML Schema Definition Language (XSD) v1.1 Part 2: Datatypes, <http://www.w3.org/TR/xmlschema11-2>

4.2 Informative References

The following non-normative documents contain provisions which, through reference in this text, provide informative context for material in this specification.

- [cwe] Common Weakness Enumeration (CWE) –
 - a repository maintained by MITRE Corporation of known weaknesses in software that can be exploited to modify data, read data, create a denial-of-service that results in unreliable execution, create a denial-of-service that results in resource consumption, execute unauthorized code or commands, gain privileges / assume identity, bypass protection mechanism, and/or hide their activities¹. <<https://cwe.mitre.org>>
 - also, ITU standard: ITU X.1524 Common Weakness Enumeration < <https://www.itu.int/rec/T-REC-X.1524-201203-I/en> >
- Software Fault Patterns (SFP) Catalog –
 - AFRL-RY-WP-TR-2012-0111, V2 - DoD document approved for public release, distribution unlimited;
 - Software Fault Pattern Clusters - a repository maintained by MITRE Corporation of links connecting SFPs and CWEs <<https://cwe.mitre.org/data/definitions/888.html>> ;

5 Terms and Definitions

¹ CWE technical impact enumeration <https://cwe.mitre.org/cwraf/enum_of_ti.html>

For the purposes of this specification, the most of applicable terms and definitions are provided in Section 9 TOIF Conceptual Model.

6 Symbols

List of symbols/abbreviations:

CWE	Common Weakness Enumeration
KDM	Knowledge Discovery Metamodel
SCA	Static Code Analysis
SFP	Software Fault Patterns
TOIF	Tools Output Integration Framework
XMI	XML Metadata Interchange

7 Additional Information

7.1 How to Read this Specification

TOIF Exchange Format is a common normalized format for representing the findings of static code analysis (SCA) tools for the purpose of integrating multiple facts related to the system under assessment.

This specification has the following structure.

Section 8.1 “Objectives” summarizes the key design objectives for the TOIF XML format and its role in the TOIF Ecosystem

Section 9 “TOIF Conceptual Model” presents the conceptual schema for TOIF XMI described in SBVR Structured English as a set of definitions of noun and verb concepts. This section defines a vendor-neutral vocabulary for the entire TOIF Ecosystem. TOIF Conceptual Model provides a technology-neutral vocabulary for TOIF which is then systematically implemented as a MOF/UML metamodel for the purpose of specifying a concrete XML/XMI schema for the TOIF data (the TOIF Exchange Format).

Section 9.1. describes the basic common facts related to weakness findings.

Section 9.2 “Housekeeping considerations for TOIF XMI” describes several “housekeeping” facts that facilitate management of multiple TOIF XMI files during the entire life cycle of the operation of the TOIF framework. The “housekeeping” fact define various meta-data to the basic TOIF facts, mainly related to multiple builds of the system, and versions of the tools used, etc. Such additional information is important to manage TOIF data over the entire life-cycle of the system under assessment as well as in an enterprise context where multiple systems are assessed by multiple teams.

Section 9.3 “Fact-oriented organization of TOIF XMI” elaborates the conceptual model and describes the organization of the TOIF XMI as triples built around the verb concepts with noun concepts as the endpoints.

Section 10 “TOIF Logical Model” presents the MOF/UML metamodel for TOIF which is systematically developed based on the TOIF Conceptual Model as an intermediate step towards the TOIF XMI schema. Both the TOIF Conceptual Model as well as the TOIF Logical Model provide an adequate description of the TOIF XMI schema, so either (or both) can be used to understand TOIF. However it is the TOIF Logical Model that determines the exact structure of the TOIF XMI schema through the rules described in MOF and XMI specifications. Section 10 provides multiple examples of the TOIF XMI data compliant to the TOIF XMI schema. Section 10 has the following organization.

Section 10.1 describes the basic concepts of TOIF represented as MOF/UML metamodel.

Section 10.2 describes the MOF/UML representation of the house-keeping concepts of TOIF.
 Section 10.3 describes the fact-oriented structure of TOIF XML.
 Section 10.4 describes evidential records in TOIF XML.

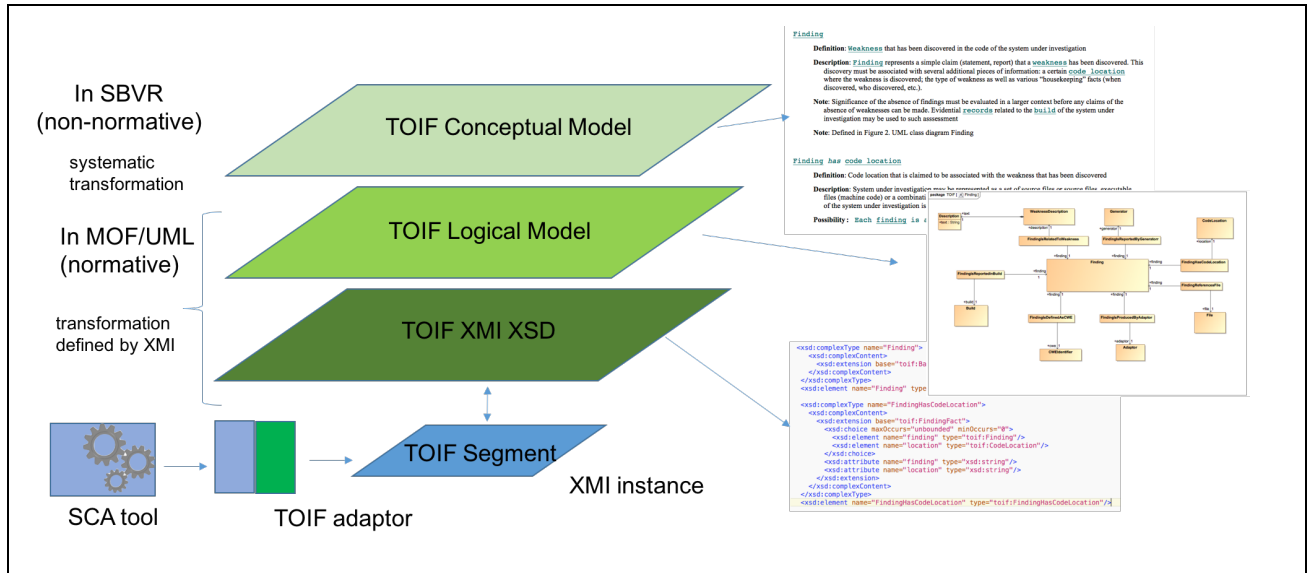


Figure 1. Organization of the TOIF specification

7.2 Acknowledgements

The following companies submitted this specification:

- KDM Analytics Inc
- Lockheed Martin Corporation
- The MITRE Corporation
- Model Driven Solutions
- 88solutions Corp
- NoMagic Inc

8 TOIF Exchange Format

8.1 Objectives

TOIF is a standard vendor-neutral protocol that facilitates information flow from multiple proprietary static code analysis (SCA) tools as producers to various consumer tools that can integrate, collate, store, transform and present findings from multiple SCA tools. TOIF establishes a uniform, vendor-neutral, normalized environment for processing findings from multiple SCA tools and managing findings in enterprise environments. TOIF facilitates managing findings originating from multiple SCA tools over the life-cycle of a system under assessment. TOIF is a normalized fact-oriented protocol which is aligned with other protocols, such as the standard Knowledge Discovery Metamodel (KDM) protocol describing basic facts about the system under assessment, risk analysis interchange protocol, as well as other protocols of the System Assurance Ecosystem, by the Object Management Group (OMG). This alignment establishes a practical integrated environment where multiple tools can contribute software assurance information and facilitates systematic evaluation and measurement of existing static code analysis tools. The core of the TOIF protocol is the specification of the TOIF exchange format.

The key requirement for the TOIF protocol is that no modifications to the source code of the original static code analysis tools is made, in other words, the TOIF shall include an explicit **adaptation step** that is performed outside of an off-the-shelf proprietary static code analysis tool, and that is capable of transforming the original report from such tool into a

normalized TOIF report that performs both **syntactic integration** (normalizing the differences in the output reporting formats of proprietary SCA tools) as well as **semantic integration** (normalizing the meaning of the findings and the location of the findings). The semantic normalization maps the nomenclature of the findings used in a proprietary static analysis tool into a common vendor-neutral nomenclature. The “mapping” artifact is formally described in the Compliance Section, clause 3. The common nomenclature of weaknesses in TOIF is based on the Software Fault Pattern (SFP) system of clusters and individual patterns, and the further mapping to a catalog known as the Common Weakness Enumeration (CWE). The vendor-neutral common nomenclature of weakness types consisting of SFP and CWE is an integral part of the TOIF approach. Both the syntactic and the semantic normalizations of the proprietary reporting format of a given static analysis tool to TOIF is assumed to be implemented by an TOIF Adaptor in a way that is non-intrusive with respect to the static code analysis tool. This consideration is important to the design of the TOIF as it separates the syntactic aspects of integration from the semantic ones. In addition, TOIF extends finding reports from proprietary SCA tools with normalized vendor-neutral meta-information (further referred to as housekeeping), facilitating management of facts, their provenance and attribution over larger life-cycles, independent on any of the SCA tools.

A key requirement of the TOIF is that the production of the normalized TOIF facts shall not depend on the modification of the original SCA tool. The open description of the normalized report format TOIF XMI will encourage the vendors of the commercial SCA tools to support it natively, however regardless of the adoption by the tool vendors of the original tools, such tools can still be integrated into the framework by third parties providing the adaptation capability.

Therefore the second requirement is that the TOIF shall define a TOIF XMI format as the output for the adaptation tools. The TOIF XMI format shall normalize the syntax of the original weakness reports so that the TOIF Consumers can focus at the semantic integration of the findings. The TOIF XMI may be positioned as the markup language such that the predefined markers can be inserted into the original report and the TOIF-compliant consumer will then read the report as it is structured by the markers.

TOIF data are organized as triples, following the verb nad noun phrases in the TOIF Structured English Vocabulary. While the specific embodiment of the TOIF Exchange Format is specified in this document as a TOIF XMI schema (through a MOF/UML metamodel and the MOF/XMI rules that determine a an XML/XMI schema) the “triple flavor” of the TOIF data is designed to support additional formats and technology spaces, including, for example, reasoning tools.

8.2 TOIF Ecosystem

The TOIF exchange protocol assumes several specific capabilities with regards to how TOIF information can be produced and consumed. Thus the TOIF protocol determines a certain ecosystem where there can exist multiple implementations of the TOIF capabilities that satisfy the interfaces defined by the TOIF protocol and that address the different roles within the TOIF protocol.

The operation of the TOIF Ecosystem involves three distinct phases. Phase 1 involves application of one or more static code analysis tools to the system under assessment. Phase 1 also may involve application of a Knowledge Discovery Metamodel (KDM) extractor tool to the same system under assessment in order to generate the basic KDM facts about the system. Within the Phase 1, TOIF Adaptor tools process the proprietary finding reports from each SCA tool, and normalize these report (both syntactically and semantically) into the TOIF format. Since weakness finding reports are produced by multiple off-the-shelf static code analysis tools, phase 1 shall perform normalization of the original reports by tool-specific TOIF Adaptors so that the rest of capabilities in the TOIF Ecosystem can successfully consume reports from multiple TOIF producers.

Phase 1 is often performed as part of the regular build of the system under investigation, in which case this phase would also involve running code compilers and linkers. Regular builds are usually orchestrated by build tools. Extending the orchestration to correctly include SCA tools and TOIF adaptors into the build process is one of the key success factors for running static code analysis programmes and software assurance.

In Phase 2, normalized weakness reports from various tools are integrated into a single, comprehensive report. As the result, an integrated repository of the TOIF facts can be populated.

Phase 3 involves consuming the integrated TOIF weakness finding facts for the purposes of presenting them to human analysts (browsing), analyzing them as the software assurance, entering them as evidence for risk assessment or RMF security control assessment, as well as any other purposes.

The TOIF Ecosystem assumes the following roles:

- SCA tool – provides capability to scan source or machine code of the system under investigation and generate weakness finding reports. An SCA tool usually involves components that perform scanning and parsing of source

code, or perform disassembling of the machine code, implement optimized control and data flow analysis algorithms, often incorporate extensive information about standard software libraries and components, operating systems and compilers, as well as a certain knowledge base of what they consider as weaknesses and the corresponding patterns that can be used to discover at least some of these weaknesses in the code. Effectiveness of an SCA tool is determined by multiple factors

- TOIF Adaptor tool – provides capability to transform the proprietary weakness finding report from a particular SCA tool into a normalized representation determined by the TOIF specification. The most challenging part of implementing a TOIF adaptor is to provide a mapping from proprietary weakness type system used by a particular SCA tool into a normalized system of weakness types in a justifiable and unambiguous way that facilitates further semantic integration of the TOIF finding facts. TOIF specification uses a formalized 3-level hierarchical system of weakness types that involve a combination of the Software Fault Patterns (SFP) catalog and the Common Weakness Enumeration (CWE). The “mapping” artifact is formally defined in Compliance Section of this document, clause 3.
- TOIF producer – a generic term to describe any capability that produces output conformant with the TOIF specification. For example, a combination of an SCA tool and the corresponding TOIF Adaptor can play a role of a TOIF Producer
- KDM tool – provides capability to scan source or machine code of the system under investigation and produce normalized description of this system conformant to the Knowledge Discovery Metamodel (KDM) specification. KDM facts, as we will refer to such normalized description provide a vendor-neutral general-purpose representation of the semantic structure, behavior, and datatype organization of the system under investigation. KDM facts are a form of intermediate representation of the system under assessment. A KDM tool usually involves components that perform scanning and parsing of source code, or perform disassembling of the machine code, may incorporate information about standard software libraries and components, and operating systems. KDM facts may be generated by a Code Compiler. KDM facts can be integrated with the TOIF facts for more powerful analysis of the weakness findings.
- Code Compiler – provides capability to scan source code of the system under investigation and produce linkable object code or executable machine code for the selected platform. A Code Compiler involves a proprietary intermediate representation of each module of the system under assessment from the syntax viewpoint, and from the semantic viewpoint. A Code Compiler usually involves components that perform scanning and parsing of source code, build the intermediate representation(s) of the code, analyze the intermediate representation and generate the machine code. The last component is often called the BackEnd, while the first two components are often referred to as the FrontEnd. The intermediate representation constructed by a code compiler provides valuable information about the system under investigation that may be useful for the purposes of software assurance, however this information is seldom exposed by code compilers and when it is, it is often difficult to utilize it because of its proprietary nature, technology dependencies, and low level. Some compilers may choose to transform their high-fidelity intermediate representation into KDM facts, thus removing the barriers for using this information by other consumers
- Code Linker – provides capability to combine one or more linkable object code files into machine code for the selected execution platform. Code Linker is used in system builds because the executable machine code of the system usually involves a mix of application modules and various third-party libraries, already precompiled for the selected platform.
- Build Tool – provides capability to orchestrate the process of running Code Compilers, with desired options, inputs and outputs, running Code Linkers, packaging the outputs, and performing other desired steps to transform

input source files, precompiled object files and libraries into the output artifacts. Usually a Build Tool is general-purpose, driven by a Build Script that describes the build steps.

- Build Script – description of the build steps to be performed by a Build Tool to perform a build of the system under investigation
- TOIF Orchestration tool – provides capability to orchestrate the process of running SCA tools and their corresponding TOIF Adaptors in alignment with the regular build, i.e. such that each source file is processed by selected SCA tools with desired options, aligned with the options used during the regular build, that an appropriate TOIF Adaptor is called for each SCA tool, that all TOIF output files are appropriately managed; Similarly for machine code analysis, the TOIF Orchestration tool aligns the process of running the selected SCA tools and their TOIF Adaptors on all desired machine code files. From the software assurance evidence perspective, the TOIF Orchestration tool generates the key piece of evidence regarding the coverage of the source and machine code files, correctness of the SCA findings, etc.
- TOIF repository – provides capability to store, manage and query TOIF facts.
- TOIF browser – provides capability to view TOIF related entities and relationships by human analysts in some kind of visual environment.
- TOIF consumer - a generic term to describe any capability that consumes input conformant with the TOIF specification
- TOIF Analytics tool – a generic term to describe any capability that consumes one or more TOIF segments and produces one or more TOIF segments. This may include, for example, a TOIF Integration tool, that consumes partial TOIF segments and produces a single integrated segment, or a TOIF Citing tool that consumes TOIF integrated segment and augments it with some elements.

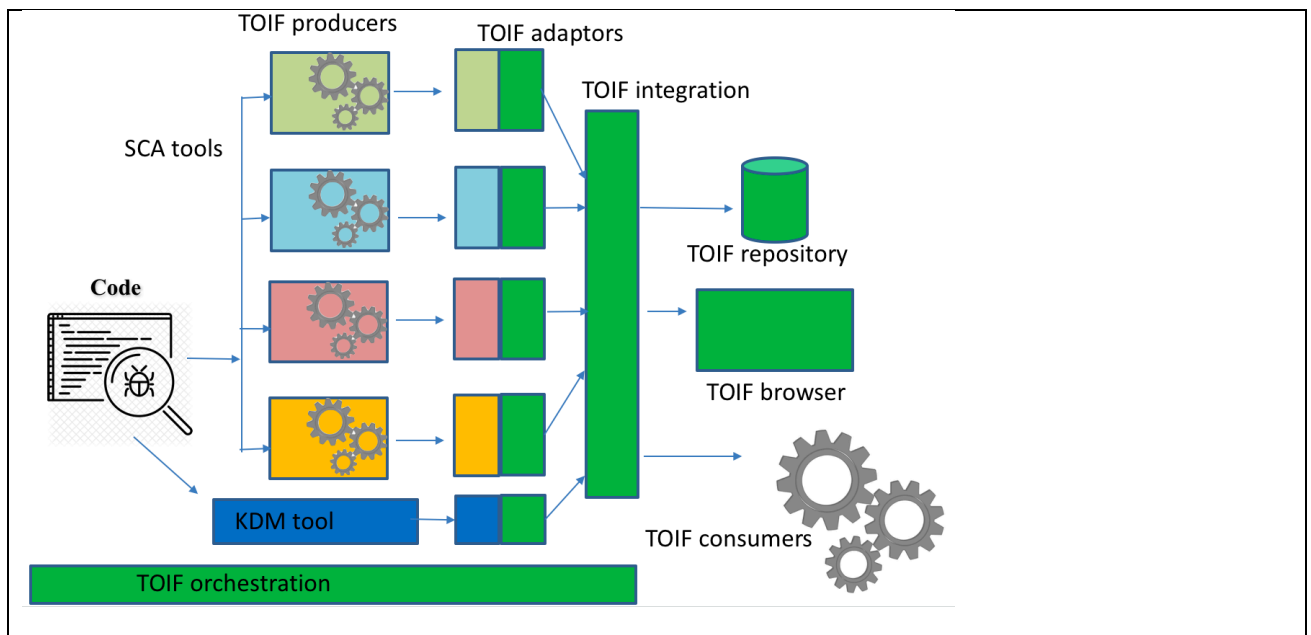


Figure 2. The Flow of the TOIF Protocol and the TOIF Ecosystem

9 TOIF Conceptual Model

This section describes TOIF Exchange Format in SBVR Structured English by focusing at a set of vendor-neutral noun and verb phrases that provide the foundation for the TOIF Ecosystem as its technology neutral vocabulary. The actual TOIF XMI schema is consistently derived from this conceptual model by representing each verb concept as a triple. However the precise details of the TOIF XMI schema are provided by the TOIF MOF/UML metamodel defined in Section 10 together with multiple example of TOIF XMI data compliant with the TOIF XMI schema. The TOIF MOF/UML metamodel determines the TOIF XMI schema through a set of rules described in MOF and XMI specifications.

9.1 Basic Entities and Facts

The conceptual model of the TOIF protocol describes the characteristics of the weakness findings, as they are reported by SCA tools. We also defined the facts where the original weakness findings can be merged with the basic facts about the system under investigation, as defined by the standard Knowledge Discovery Metamodel (KDM).

Weakness

Definition: characteristic or property of software that, in proper conditions, could contribute to the introduction of vulnerabilities within that software

Synonym: weakness of software

Note: Each weakness is categorized by a weakness type. Some weaknesses can be characterized by a certain location in the code of the system under assessment.

Note: A claim of a weakness (of a certain weakness type at a certain code location) can be supported by by one or more findings as well as additional citings.

Vulnerability

Definition: weakness of software, hardware, or online service that can be exploited by a threat

Description: Examples of weaknesses in a system are software and hardware design flaws, poor administrative processes, lack of awareness and education, and advancements in the state of the art or improvements to current practices. Regardless of cause, an exploitation of such vulnerabilities may result in real threats to mission-critical information systems.

Note: Vulnerabilities can be architecture flaws, coding errors, or other implementation errors, or insecure configuration. Vulnerabilities can also result from insufficient or incorrect security documentation, security awareness, or communication.

Finding

Definition: Weakness that has been discovered in the code of the system under investigation

Description: Finding represents a simple claim (statement, report) that a weakness has been discovered. This discovery must be associated with several additional pieces of information: a certain code location where the weakness is discovered; the type of weakness as well as various “housekeeping” facts (when discovered, who discovered, etc.).

Note: Significance of the absence of findings must be evaluated in a larger context before any claims of the absence of weaknesses can be made. Evidential records related to the build of the system under investigation may be used for such assessment

Note: Defined in Figure 3. UML class diagram Finding.

Finding has code location

Definition: Code location that is claimed to be associated with the weakness that has been discovered

Description: System under investigation may be represented as one or more source files , executable files (machine code) or a combination of both. The mechanism to uniquely identify a location within the code of the system under investigation is the foundation for reporting weaknesses.

Possibility: Each finding is associated with one or more code location.

Finding is defined as CWE

Definition: Normalized identifier of the weakness type associated with the finding.

Possibility: a finding may have many CWE identifier

Note: CWE identifier will be added during the adaptation phase.

Note: In the situation when there is an ambiguity in a mapping of a particular finding (type) of a static analysis tool to CWE, multiple CWE identifier will be associated with the corresponding finding.

Note: the TOIF Analyzer may split finding with multiple CWE identifier into several findings with a single CWE identifier

Finding is reported as Weakness Description

Definition: Description of the weakness type other than the normalized identifier associated with the finding.

Description: Weakness description is associated with the finding. Usually this description represents a proprietary report generated by the static code analysis tool (either specific to the weakness type, or specific to the finding).

Finding references File

Finding is produced by Adaptor

Finding is reported by Generator

Finding is reported in Build

Finding has Criticality

Definition: Description of the weakness type other than the normalized identifier associated with the finding.

Description: Weakness description is associated with the finding. Usually this description represents a proprietary report generated by the static code analysis tool (either specific to the weakness type, or specific to the finding).

Finding has Confidence

Definition: Description of the weakness type other than the normalized identifier associated with the finding.

Description: Weakness description is associated with the finding. Usually this description represents a proprietary report generated by the static code analysis tool (either specific to the weakness type, or specific to the finding).

Criticality

Definition: A measure of impact that a certain weakness may cause.

General concept: Percent

Description: 0% - means that a weakness does not cause any impact, while 100% means that the weakness corresponds to a critical vulnerability. Criticality is a natural number from 0 to 100 interpreted as percent.

Confidence

Definition: a measure of confidence of an agent making a claim that the statement is actually true.

General concept: Percent

Description: 0% means that an agent is not confident (the evidence is slim, yet there is something that caused the agent to make the claim). 100% means that the agent is very confident (there is strong evidence supporting the claim). Confidence is a natural number from 0 to 100 interpreted as percent

Weakness is defined as CWE

Weakness has Code Location

Code location

Definition: Location in the code of a system under investigation.

Description: This element is a statement of a location within a system under investigation. The system under investigation may be represented as one or more source files, executable files (machine code) or a combination of both. Location in the code of the system under investigation is defined as a combination of a file and a location within the file. Location in a source file is given as a line number and optionally a position within the line. Location within an executable file is defined as an offset. Multiple Code Location elements may refer to the same logical location, for example when the same set of files is analyzed independently by multiple static code analysis tools and several reports are produced.

Note: In some cases, Code location may refer to the entire file (including situations when the file is empty). In this case, the Code location involves only the reference to a File. In other cases, Code location must involve either a Line number or an Offset. When a Code location involves a Line number, it may additionally involve a position.

Note: this provides a unique reference schema for findings. The corresponding concept in KDM is SourceRef

Note: Defined in Figure 7. UML class diagram Code Location

Code location references file

Definition: Code location is uniquely described as a location within a certain file.

Description: Code location element refers to a location in the code of the system under investigation by describing a combination of a file and a location within the file.

Possibility: Code location references exactly one file

Code location has line number

Definition: Line number that describes a location within a source file.

Description: for locations in source files; this attribute is optional

Possibility: [code location](#) may have [line number](#)

[Code location](#) has [position](#)

Definition: Position of a character within a line number that uniquely describes a location within a source file.

Description: for locations in source files; this attribute is optional

Possibility: [code location](#) may have [position](#)

[Code location](#) has [offset](#)

Definition: Number of a byte in a binary file that uniquely describes a location within an executable file.

Description: for locations in binary files. Code locations in executable files are identified in binary image, therefore offset is the same as the virtual address of a byte in the image. Offset does not represent the offset in the executable file itself.

Possibility: [code location](#) may have [offset](#)

[Line number](#)

Definition: Line number that uniquely describes a location within a file.

General concept: [Natural number](#)

Description: A source file is assumed to be a text file that consists of a sequence of one or more lines, marked by an end-of-line characters. Lines are enumerated from 1. Line number in case of an empty file is not applicable.

[Position](#)

Definition: Number of a character within a line (identified by a line number) that uniquely describes a location within a file.

General concept: [Nonnegative integer number](#)

Description: A line of a source file is assumed to be a sequence of one or more characters different from an end-of-line character. Characters are enumerated from 1. Position in in case of an empty line is not applicable.

[Offset](#)

Definition: Offset of a byte that uniquely describes a location within a binary image.

General concept: [Nonnegative integer number](#)

Description: A binary file is assumed to be a sequence of one or more bytes. Bytes are enumerated from 1. Offset in in case of an empty binary file is not applicable. Offset is the same as the virtual address of a byte in the image. Offset does not represent the offset in the executable file itself.

[Weakness Type Identifier](#)

Definition: A category of weakness.

Note: This is not a designation, but the actual category. The suffix Identifier is added for consistency with “CWE Identifier” and SFP Identifier”, to avoid possible confusion between “CWE” as the entire catalog, “CWE” as a specific category of weakness in the CWE catalog.

Synonym: Weakness Type

Note: Defined in Figure 4. UML class diagram WeaknessType

Weakness Type Identifier has name

Definition: A unique name provided to a weakness type defined by the Common Weakness Enumeration (CWE).

Description: Common Weakness Enumeration (CWE) is a standard that provides a list of weakness types, each identified by a CWE name, for example, “CWE-561”.

Weakness Type Identifier has description

Definition: Description of a Weakness Type Identifier is an informal description of the corresponding weakness type

CWE identifier

Definition: A weakness type defined by the Common Weakness Enumeration (CWE).

Description: Common Weakness Enumeration (CWE) is a catalog that provides a list of weakness types, each identified by a CWE name, for example, “CWE-561”. CWE associates several information blocks with each weakness type, including a long name, an informal description, examples, references to other standards, such as CVE, etc.

Note: This is not a designation, but the actual category. The suffix Identifier is added to avoid possible confusion between “CWE” as the entire catalog, “CWE” as a specific category of weakness in the CWE catalog.

Note: Defined in Figure 4. UML class diagram WeaknessType

SFP Identifier

Definition: A weakness type provided by the Software Fault Patterns (SFP) catalog.

Note: This is not a designation, but the actual category. The suffix Identifier is added to avoid possible confusion between “SFP” as the entire catalog, “SFP” as a specific category of weakness in the SFP catalog.

Description: Software Fault Patterns (SFP) is a standard that provides a list of weakness types, each identified by a SFP name, for example, “SFP-8”.

SFP Cluster

Definition: A weakness category provided by the Software Fault Patterns (SFP) catalog.

Description: Software Fault Patterns (SFP) is a standard that provides a list of weakness clusters, each identified by a unique name, for example, “Authentication”.

Note: Defined in Figure 4. UML class diagram WeaknessType

Weakness Description

Definition: Description of the weakness type other than the normalized identifier associated with the finding.

Description: Weakness description is associated with the finding. Usually this description represents a proprietary report generated by the static code analysis tool (either specific to the weakness type, or specific to the finding).

Note: Weakness Description is a proprietary Weakness Type Identifier. Because it is proprietary, and also because it is defined operationally (by an SCA tool), it is difficult to reason about the exact extent of this Weakness Type.

Weakness description has description

Definition: Text of the weakness description

Description

General concept: Text

File

Definition: A computer resource for recording a collection of related data or program records stored as a unit with a single name.

Description: File in TOIF corresponds to the InventoryItem concept in KDM. In TOIF a File is assumed to represent code of the system under investigation and is usually either a source file (KDM SourceFile) or an executable file (KDM ExecutableFile).

Note : Defined in Figure 8. UML class diagram File.

File has name

Definition: Name of the file

Name

General concept: Text

File has checksum

Definition: Checksum of the file

Note: the ability to compute the checksum of a file that is the source for the particular weakness report depends on the access to this file. In general, the application of the generator is done at a separate phase, therefore the adaptor may not be able to compute this information. However, availability of the checksum will facilitate management of multiple TOIF segment and reduce errors caused by merging unrelated TOIF segment.

Note: the Inventory Model of the KDM Model includes the checksum of each file in the system under assessment.

Checksum

General concept: Integer

Definition: Checksum is a small-sized datum derived from a block of digital data for the purpose of detecting errors which may have been introduced during its transmission or storage or to identify duplicate blocks.

File has version

Note: the ability to compute the version of a file that is the source for the particular weakness report depends on the access to this file. In general, the application of the generator tool is done at a separate phase; therefore the adaptor may not be able to compute this information. However, availability of the version will facilitate management of multiple TOIF segment and reduce errors caused by merging unrelated TOIF segment.

Note: the Inventory Model of the KDM Model includes the version of each file in the system under assessment.

Version

Definition: A unique identifiable state of something.

General concept: State

Note: Version of the subject is designated by a string

File is contained in Directory

File belongs to Project

Directory

Definition: An organizational unit or container, used to organize directories and files into a hierarchical structure.

Note: Defined in Figure 9. UML class diagram Directory

Directory is contained in Directory

Directory has name

Directory belongs to Project

Statement

Definition: An basic identifiable unit of behavior in software such as a source code statement, a basic block, a operator.

Note: this corresponds to KDM ActionElement class

Note: Defined in Figure 10. UML class diagram Semantic Statement

Statement has code location

Statement is involved in Finding

Synonym: Finding is associated with statement

Possibility: each Finding may be associated with many statement

Statement is part of sink of Finding

Note: This is a stronger form of the fact type *Statement is involved in Finding* where the role of Statement is known to be a sink, i.e. a statement that corresponds to the discernable necessary condition of the weakness. For example, a statement that performs access to a buffer is the necessary condition to a buffer overflow weakness, since without an access there is no overflow. Sink is a concept use in the Software Fault Patterns (SFP). Many software faults have discernable Sink and Source statements and a data flow path between them.

Statement is part of source of Finding

Note: This is a stronger form of the fact type *Statement is involved in Finding* where the role of Statement is known to be a source, i.e. a statement that corresponds to the discernable sufficient condition of the weakness. For example, a statement that sets the pointer outside of the available space in a buffer is the sufficient condition to a buffer overflow weakness, provided that there also exists an data flow path to a sink which performs access to the buffer using the same pointer, and the value of the pointer is unchanged along the path. Source is a concept use in the Software Fault Patterns (SFP). A weakness finding may have multiple sources. Many software faults have discernable Sink and Source statements and a data flow path between them.

Statement is preceded by Statement

Data element

Definition: An basic identifiable data item is software such as global and local variables, records, formal parameters and constants.

Note: This corresponds to the KDM DataElement class

Note: Defined in Figure 11. UML class diagram Semantic Data

Data element is defined at Code Location

Data element is involved in Finding

Data element has name

Data element is involved in Statement

Note: This is a stronger form of the fact type Data element *is involved in Finding*, and is usually related to the Source and Sink of the Finding. In this case the Data element is passed along the data flow path from the Statement is known to be a source, i.e. a statement that corresponds to the discernable sufficient condition of the weakness to the sink of the finding. For example, a statement that sets the pointer outside of the available space in a buffer is the sufficient condition to a buffer overflow weakness, provided that there also exists an data flow path to a sink which performs access to the buffer using the same pointer, and the value of the pointer is unchanged along the path. The pointer is then the Data element involved in the Source statement. Data element, Source and Sink is are concept use in the Software Fault Patterns (SFP). A weakness finding may have multiple data elements involved in the data flow path between Source and Sink.

Citing

Definition: An observation related to a weakness that may supply additional information to the weakness and a verdict which is a claim that a weakness is valid or is not valid.

Note: Some citings may be performed by analysts, while other citings may be performed by Analytics Tools, for example based on pattern matching and/or machine learning

Note: Defined in Figure 6. UML class diagram Citing

Citing references Weakness

Citing has Description

Citing has Confidence

Citing is generated at Date

Citing is generated by Citing Agent

Citing has Verdict

Verdict

Definition: An evaluation of the findings and code facts for a particular weakness as represented by one or more findings, CWE Identifier and code location into Code Artifacts. Verdict represents a claim that a weakness is a valid finding (true) or not a valid finding (false)

9.2 “Housekeeping” Entities and Facts

This section describes several “housekeeping” facts that facilitate management of multiple TOIF facts during the entire life cycle of the system under investigation, or multiple systems under investigation within an enterprise.

The key objectives are

- to facilitate management of multiple TOIF Segment generated during the course of the TOIF project
- to reduce the possibility of errors caused by merging unrelated TOIF Segment
- to reduce the possibility of errors caused by merging TOIF Segment with the unrelated KDM model

The audit information includes the following:

- project identifier (unique project name that corresponds to the system under investigation)
- build identifier
- name of the generator tool
- vendor name of the generator tool
- generator tool identifier (unique version of the generator tool)
- adaptor identifier (unique name and version of the adaptor tool)
- person name responsible to the TOIF Segment
- organization responsible for the TOIF Segment
- date when the TOIF Segment was produced

Tool

Definition: Any software that can be used to develop, test, analyze, or maintain a computer program or its documentation.

Note: Defined in Figure 15. UML class diagram Tools

Tool *has* Description

Tool *has* name

Tool *has* version

Generator

Definition: Any capability to scan source or machine code of the system under investigation and generate weakness finding reports.

Description: Generator tool usually involves components that perform scanning and parsing of source code, or perform disassembling of the machine code, implement optimized control and data flow analysis algorithms, often incorporate extensive information about standard software libraries and components, operating systems and compilers, as well as a certain knowledge base of what they consider as weaknesses and the corresponding patterns that can be used to discover at least some of these weaknesses in the code. Effectiveness of Generator tool is determined by multiple factors.

Synonym: SCA tool

Note: Defined in Figure 15. UML class diagram Tools

Adaptor

Definition: Any capability to transform the proprietary weakness finding report from a particular Generator tool into a normalized representation determined by the TOIF specification.

Description: The most challenging part of implementing a TOIF adaptor is to provide a mapping from proprietary weakness type system used by a particular Generator tool into a normalized system of weakness types in a justifiable and unambiguous way that facilitates further semantic integration of the TOIF finding facts. TOIF specification uses a formalized 3-level hierarchical system of weakness types that involve a combination of the Software Fault Patterns (SFP) catalog and the Common Weakness Enumeration (CWE)

Note: Defined in Figure 15. UML class diagram Tools

Adaptor supports Generator

Synonym: Generator requires Adaptor

Adaptor is capable of finding CWE

Synonym: CWE can be reported by Adaptor

Orchestration tool

Definition: Any capability to perform the process of running Generator tools and their corresponding TOIF Adaptors in alignment with the regular build

Description: The responsibility of the Orchestration tool is to make sure that each source file is processed by selected Generator tools with desired options, aligned with the options used during the regular build, that an appropriate TOIF Adaptor is called for each Generator tool, that all TOIF output files are appropriately managed; Similarly for machine code analysis, the TOIF Orchestration tool aligns the process of running the selected Generator tools and their TOIF Adaptors on all desired machine code files. From the software assurance evidence perspective, the TOIF Orchestration tool generates the key piece of evidence regarding the coverage of the source and machine code files, correctness of the weakness findings, etc.

Note: Defined in Figure 15. UML class diagram Tools

Analytics tool

Definition: Any capability to consumes one or more TOIF segments and produces one or more TOIF segments.

Description: Analytics tools may include, for example, a TOIF Integration tool, that consumes partial TOIF segments and produces a single integrated segment, or a TOIF Citing tool that consumes TOIF integrated segment and augments it with some elements.

Note: Defined in Figure 15. UML class diagram Tools

Vendor

Definition: An organization that supplies a Tool used in project

General concept: Organization

Note: Defined in Figure 16. UML class diagram Organization

Tool is supplied by Vendor

Synonym: Vendor supplies Tool

Person

Synonym: Individual, human

Note: Defined in Figure 17. UML class diagram Person

Person has name

Person has email address

Person has phone number

Email address

Phone number

Address

Person is employed by Organization as Role

Note: it is assumed that the dynamics of this relationship is not relevant to the TOIF. So this relationship means that the Person was employed by Organization for the duration of the project.

Organization

Description: An entity comprising multiple persons that have a shared goal and is linked to an external environment

Description: an Organization involved in project

Note: Defined in Figure 16. UML class diagram Organization

Organization has name

Organization has Description

Organization has address

Organization has email address

Organization has phone number

Organization₁ is part of Organization₂ as Role

Role

Note: Defined in Figure 18. UML class diagram Role

Role has name

Role has Description

Project

Description: a TOIF project related to a specific system under investigation

Note: Defined in Figure 14. UML class diagram Project

Project has name

Project has Description

Note: a separate TOIF Segment may be used to own all “housekeeping” elements and their descriptions

Person is involved in Project as Role

Organization is involved in Project as Role

Build

Definition: An engineering activity that involves a series of transformations of the “source code” artifacts into “executables” that can run on a selected computer platform. Build is performed in the context of the system under investigation. Build is a specific event and the corresponding set of artifacts.

Synonym: TOIFBuild

Note: Defined in Figure 12. UML class diagram Build and in Figure 13. UML class diagram Housekeeping

Build has name

Build has description

Definition: Text of the weakness description

Build is related to Project

Build is generated by Person

Build is supervised by Person

Synonym: Person is responsible for TOIF Segment

Build is produced by Organization

Build is owned by Organization

Build is created at Date

Date

Definition: Time stated in terms of year, month, day and possibly also hour, minute

Note: Date may include a Time Zone.

Synonym: Timestamp

Build is orchestrated by Orchestration tool

Synonym: Orchestration tool generated Build

9.3 Fact-oriented organization of TOIF XMI

This section elaborates the conceptual model and describes the fact-oriented organization of the TOIF XMI. More specifically, this section describes a certain abstract structure of the TOIF concepts, by observing that all above concepts of TOIF are either noun concepts (such as Finding), or verb concepts (such as Finding is reported in Build), or take a specific form of an owned attribute (such as File has Name). Noun concepts will be further referred to as Entities (or TOIF Entities). Verb concepts will be further referred to as Facts, or Clauses (or TOIF Facts). Few special purpose verb concepts will be referred to as TOIF Records (or Evidential Records). Finally, we will keep distinguishing the Attributes, as few special verb concepts in the form of TOIF Entity *has* something.

Collectively, TOIF Entities and TOIF Facts/Clauses will be referred as TOIF Elements.

The physical structure of the TOIF XMI shall be based on the abstract structure of the TOIF, i.e. shall be structured as a collection of instances of TOIF Entities, together with their owned Attributes, and instances of TOIF Facts/Clauses and Records. A TOIFSegment is a physical container for a collection of the instances of TOIF Entities together with their owned/unique Attributes, and instances of TOIF Facts/Clauses and TOIF Records. When this does not lead to confusion, the content of a TOIFSegment will be also referred to as “TOIF facts”, meaning the individual noun and verb concepts that are instances of the TOIF concepts from the TOIF specification, taken as the truth.

First, a Segment is the root element for the TOIF XMI file and the container of the TOIF facts and the corresponding entities. TOIF segment is be the main unit of information exchange within the TOIF framework. The TOIF Segment *owns* the corresponding entities and facts. The concept of element ownership is important from the design perspective of the XML/XMI. Eventually, ownership corresponds to the nested XMI tags. It is assumed that every TOIF entity and every TOIF fact is defined by a unique pair of XMI tags. Facts and entities in a TOIF Segment are flat, i.e a TOIF Segment is an ordered list of TOIF entities and facts. The following logical constraints apply:

- C1: TOIF Segment shall own all TOIF entities that are objects of the TOIF facts owned by that Segment
- C2: TOIF entity that is the object of a TOIF fact shall precede that fact in the TOIF Segment.
- C3: TOIF Entity owns all its attributes

The physical organization of the TOIF facts is defined by the following conceptual schema:

Fact

Definition: A general category that includes all verb concepts defined in TOIF, that represent general statements (assertions) about TOIF entities, except the verb concepts that define owned attributes of TOIF Entities.

Synonym: TOIF Fact

Synonym: Clause

Synonym: TOIF Clause

Description: a TOIFSegment describes instances of TOIF Facts. TOIF distinguishes Facts and Evidential Records where a Fact provides a statement related to both basic and housekeeping TOIF Entities, and Evidential Record provides a statement related to the build (orchestration) environment.

Note: Defined in Figure 19. UML class diagram Abstract Structure

Note: Concrete facts are enumerated in Figure 22. UML class diagram Basic Facts 1, Figure 23. UML class diagram Basic Facts 2, Figure 24. UML class diagram Basic Facts 3, Figure 25. UML class diagram Basic Facts 4 and Figure 28. UML class diagram Housekeeping Facts 1, Figure 29. UML class diagram Housekeeping Facts 2, Figure 30. UML class diagram Housekeeping Facts 3

Entity

Definition: A general category that includes all noun concepts defined in TOIF, except ones that define owned attributes of TOIF Entities.

Synonym: a TOIF entity

Description: a TOIFSegment describes instances of TOIF Facts that reference TOIF Entities

Note: TOIF specification describes a number of noun concepts referred to as Entity. TOIF segment enumerates instances of Entity as individual noun concept. Introduction of an individual Entity is considered as the so-called existential fact

Note: Concrete noun concepts that correspond to Entity in TOIF specification are enumerated in Figure 21. UML class diagram Basic entities and Figure 27. UML class diagram Housekeeping entities

Entity is subject of Fact

Synonym: Fact adds information about Entity

Entity is object of Fact

Synonym: Fact references Entity

Attribute

Definition: A general category of noun concepts and the corresponding role concepts in the form of X has Y that describe owned attributes of TOIF Entities

Description: a TOIF attribute

General concept: Fact

Possibility: *an Attribute is owned by exactly one Entity*

Note: Defined in Figure 26. UML class diagram Basic Attributes and Figure 31. UML class diagram Housekeeping Attributes

Attribute is an attribute of Entity

Synonym: Entity owns Attribute

Evidential Record

Definition: A general category of verb concepts that represent evidential record related to the build (orchestration) environment of the system under assessment rather than generic statements about TOIF Basic or Housekeeping Entities

Description: A general category that includes few verb concepts defined in TOIF that have a form of a TOIF fact with one or more additional attributes.

Synonym: TOIF Record

Synonym: Record

Note: Defined in Figure 19. UML class diagram Abstract Structure

Note: Concrete record are enumerated in Figure 32 UML class diagram EvidentialRecord

Example: BuildRecord, CompileRecord, GeneratorRecord

TOIF Segment

Synonym: Segment

Definition: A container for one or more instances of TOIF elements with a shared purpose

TOIF Segment has name

TOIF Segment has Description

Note: Defined in Figure 19. UML class diagram Abstract Structure

TOIF Segment owns Fact

TOIF Segment owns Entity

Necessity: *TOIF Segment owns each Entity that is referenced by Fact that is owned by the TOIF Segment*

10 TOIF Logical model

This section describes the MOF/UML metamodel for TOIF XMI which is developed as an intermediate step from the TOIF Conceptual Model defined in SBVR Structured English as a technology-independent vocabulary for the TOIF Ecosystem towards the TOIF XMI schema. The TOIF MOF/UML metamodel is consistent with the TOIF Conceptual Model. The TOIF XMI schema is derived from the TOIF MOF/UML model by applying the MOF/XMI rules.

The TOIF UML model consists of a single UML package and includes 30 class diagrams to represent the following:

- TOIF basic elements
- TOIF housekeeping elements
- TOIF fact-oriented structure

The TOIF UML model is structured as an explicit set of classes corresponding to the conceptual schema, where each verb concept is represented as a UML association class, and each TOIF attribute is implemented as an owned class, rather than a UML attribute. This determines a certain “triple flavor” of the TOIF XMI data, and facilitates the potential use of other technology spaces, including reasoning tools to handle TOIF data.

The rest of this section has the following organization. Section 10.1 presents 7 UML class diagrams that describe the basic elements of the TOIF XMI, the logical entities and fact types. Section 10.2 presents 8 UML class diagrams that describe the housekeeping elements of the TOIF XML. Section 10.3 presents 8 UML class diagram that describes the physical structure of the TOIF XMI.

10.1 The basic elements of the TOIF XML

This section presents 9 UML class diagrams that represent the basic entities and facts of the TOIF XMI: Finding, Weakness Type Identifier, Code Location, File, Directory, Semantic Statement and Semantic Data.

10.1.1 Finding Class Diagram

This section describes the UML representation of the Finding concept and the corresponding facts that fully describe the finding through several clauses where the finding is the subject and other concepts of TOIF are objects of the clause.

10.1.1.1 Finding Class

The Finding class is the key class of TOIF. Instances of this class represent individual weakness findings reported by static analysis tools for the code of the system under investigation. The Finding class only has a unique id, it does not own any attributes, and is defined through connections to other instances of TOIF through association classes, such as, for example, FindingIDefinedAsCWE, which represents a verb concept “Finding is represented as CWE” and associated the instance of Finding to an instance of a class CWEIdentifier. An instance of Finding is essentially a “hub” that joins multiple related clauses, each providing a facet of information about the finding.

The superclass BasicEntity is defined in Section 10.3.3 Basic Entities Class Diagram

Superclass

BasicEntity

Associations

criticality:Criticality[0..1]	Owned attribute that specifies criticality of the finding in terms of the impact that it may cause.
confidence:Confidence[0..1]	Owned attribute that specifies the confidence in this finding claim.

Constraints

1. Each Finding instance must be the subject of at least one FindingIsReportedAsType clause
2. Each Finding instance must be the subject of at least one FindingIsReportedByGenerator clause
3. Each Finding instance must be the subject of exactly one FindingIsDefinedAsCWE clause
4. Each Finding instance must be the subject of at least one FindingIsProducedByAdaptor clause
5. Each Finding instance must be the subject of at least one FindingHasCodeLocation clause
6. Each Finding instance must be the subject of at least one FindingIsProducedInBuild clause

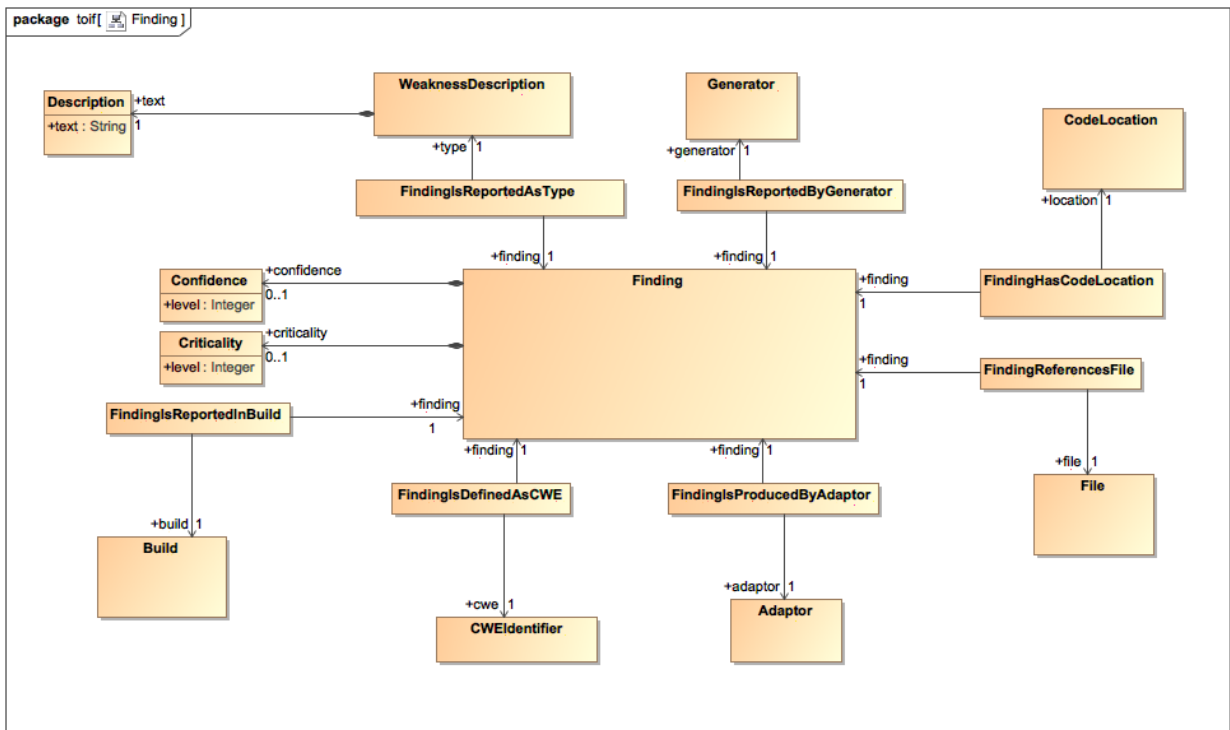


Figure 3. UML class diagram Finding

Example

This example illustrates a single Finding and a complete set of related clauses. Note, that all finding-related clauses are mandatory, except for the FindingReferencesFile. This clause is optional, since the same information is provided by the pair of mandatory clauses FindingHasCodeLocation and CodeLocationReferencesFile.

```

<fact xmi:type="toif:Finding" xmi:id="f0001"/>

<fact xmi:type="toif:CodeLocation" xmi:id="loc10">
  <linenumber linenumber="1856"/>
</fact>

<fact xmi:type="toif:FindingIsReportedAsType"
  finding="f0001" type="wd_t1_1"/>
<fact xmi:type="toif:FindingIsReportedByGenerator"
  finding="f0001" generator="rats_2.3"/>
  
```

```

<fact xmi:type="toif:FindingIsDefinedAsCWE"
  finding="f0001" cwe="CWE-561"/>
<fact xmi:type="toif:FindingIsProducedByAdaptor"
  finding="f0001" adaptor="rats_toif_adaptor_1.1"/>
<fact xmi:type="toif:FindingIsRelatedToBuild"
  finding="f0001" build="b1020171330"/>
<fact xmi:type="toif:FindingReferencesFile"
  finding="f0001" file="f10"/>
<fact xmi:type="toif:FindingHasCodeLocation"
  finding="f0001" location="loc10"/>

<fact xmi:type="toif:WeaknessDescription" xmi:id="wd_t1_1">
  <description text="Weakness that may lead to severe exposure"/>
</fact>
<fact xmi:type="toif:CWEIdentifier" xmi:id="CWE-561">
  <description text="xxxxxx"/>
</fact>
<fact xmi:type="toif:Generator" xmi:id="rats_2.3">
  <name name="RATS"/>
  <description text="xxxxxx"/>
  <version version="2.3"/>
</fact>
<fact xmi:type="toif:Adaptor" xmi:id="rats-toif-adaptor_1.1">
  <name name="RATS-TOIF"/>
  <description text="xxxxxx"/>
  <version version="1.1"/>
</fact>
<fact xmi:type="toif:Build" xmi:id="b1020171330">
  <description text="xxxxxx"/>
</fact>
  <fact xmi:type="toif:CodeLocationReferencesFile"
    finding="loc10" file="f10"/>

<fact xmi:type="toif:File" xmi:id="f10">
  <name name="main.c"/>
</fact>

```

Example

This example illustrates a single Finding with attributes. Other clauses are assumed to be refer to the previous example.

```

<fact xmi:type="toif:Finding" xmi:id="f0001">
  <confidence xmi:type="toif:Confidence" xmi:id="co_f0001" level=90/>
  <criticality xmi:type="toif:Criticality" xmi:id="cr_f0001" level=50/>
</fact>

```

10.1.1.2 FindingsReportedAsType Class

The FindingsReportedAsType class represents a verb concept “Finding is reported as Weakness Description”. This is an important clause that associates an instance of a Finding class to an instance of WeaknessDescription, which is a proprietary weakness type provided by a static code analysis tool.

When one instance of Finding is a subject of more than one FindingsReportedAsType clauses, all corresponding descriptions are assumed to be jointly describing the finding, however no particular order is assumed. This may be utilized by some Adaptor tools to split proprietary reports into parts, some of which may be shared across findings, while others are specific to an individual finding.

Superclass

FindingFact

Associations

type:WeaknessDescription[1]	represents a proprietary weakness type reported by a static code analysis tool (either specific to the weakness type, or specific to the finding)
finding: Finding[1]	Weakness that has been discovered in the code of the system under investigation

Example

For the basic example, see 10.1.1.1

The following example illustrates multiple FindingIsReportedAsType clauses for the same subject, described below as xmi:id "f0001". Note, that several mandatory clauses for the finding are not shown. Note, that the generic WeaknessDescription is shared by two findings "f0001" and "f0002".

```
<fact xmi:type="toif:Finding" xmi:id="f0001"/>
  <fact xmi:type="toif:FindingIsReportedAsType"
    finding="f0001" type="wd_t1_1_generic"/>
  <fact xmi:type="toif:FindingIsReportedAsType"
    finding="f0001" type="wd_t1_1_concrete"/>
<fact xmi:type="toif:WeaknessDescription" xmi:id="wd_t1_1_generic">
  <description text="Unprotected global may lead to severe exposure"/>
</fact>
<fact xmi:type="toif:WeaknessDescription" xmi:id="wd_t1_1_concrete">
  <description text="unprotected global X"/>
</fact>

<fact xmi:type="toif:Finding" xmi:id="f0002"/>
  <fact xmi:type="toif:FindingIsReportedAsType"
    finding="f0002" type="wd_t1_1_generic"/>
```

10.1.1.3 FindingIsReportedByGenerator Class

The FindingIsReportedByGenerator class represents a verb concept "Finding is reported by Generator". This clause provides an association between a weakness finding and the static code analysis tool that has reported this finding. The Generator is represented as an instance of the Generator class, including specific version of the Generator used in the current Build. The clauses represented by TOIF can address situations where multiple versions of Generator were used in the same build, processing same or different files, and in these situations each instance of Finding is associated with a specific instance of Generator, where some of these instances will have the same name, and possibly description, but different version numbers. For more details, see the description of the Generator class.

Situations where one instance of Finding is a subject of more than one FindingIsReportedByGenerator clauses, may occur after merging multiple TOIFSegment in a TOIF repository in which case it may be beneficial to further normalize similar findings reported by multiple version of the same Generator tool by merging them into a single Finding instance.

Superclass

FindingFact

Associations

generator:Generator[1]	Generator tool that discovered weakness
finding: Finding[1]	Weakness that has been discovered in the code of the system under investigation

Example

See 10.1.1.1

10.1.1.4 FindingsDefinedAsCWE Class

The FindingsDefinedAsCWE class represents a verb concept “Finding is defined as CWE”. This clause provides an association between a weakness finding and the normalized weakness type identifier for the weakness – a Common Weakness Enumerated identifier, represented by an instance of CWEIdentifier class.

Superclass

FindingFact

Associations

cwe:CWEIdentifier[1]	A weakness type defined by the Common Weakness Enumeration (CWE).
finding: Finding[1]	Weakness that has been discovered in the code of the system under investigation

Constraints

1. Each Finding must be the subject of exactly one FindingsDefinedAsCWE clause

Example

See 10.1.1.1

10.1.1.5 FindingsProducedByAdaptor Class

The FindingsProducedByAdaptor class represents a verb concept “Finding is produced by Adaptor”. This clause provides a direct association between a weakness finding and the Adaptor tool that transformed the original proprietary finding reported by the Generator into a normalized format and a normalized weakness type identifier. For more details, see description of the Adaptor class.

Situations where one instance of Finding is a subject of more than one FindingsProducedByAdaptor clauses, may occur after merging multiple TOIFSegment in a TOIF repository in which case it may be beneficial to further normalize similar findings reported by multiple version of the same Generator tooland/or multiple versions of the Adaptor tools for the Generator tool by merging them into a single Finding instance.

Superclass

FindingFact

Associations

adaptor:Adaptor[1]	Instance of the Adaptor tool that performed normalization of the original proprietary tool weakness report
finding: Finding[1]	Weakness that has been discovered in the code of the system under investigation

Example

See 10.1.1.1

10.1.1.6 FindingHasCodeLocation Class

The FindingHasCodeLocation class represents a verb concept “Finding has Code Location”. This clause provides an association between a weakness finding and the specific location in the code of the system under investigation, where the original finding was reported by some Generator tool. For more details, see description of the CodeLocation class.

When one instance of Finding is a subject of more than one FindingHasCodeLocation clauses, all corresponding locations are assumed to be jointly describing the finding, however no particular order is assumed. This may be utilized by some Adaptor tools to split proprietary reports involves multiple code locations. For a more semantically accurate descriptions, see descriptions of semantic facts (Semantic Statement and Semantic Data class diagrams).

Superclass

FindingFact

Associations

location:CodeLocation[1]	Location in the code of a system under investigation where weakness is discovered
finding: Finding[1]	Weakness that has been discovered in the code of the system under investigation

Example

See 10.1.1.1

10.1.1.7 FindingReferencesFile Class

The FindingReferencesFile class represents a verb concept “Finding references File”. This clause provides a direct association between a weakness finding and the specific file of the system under investigation, where the original finding was reported by the Generator. For more details, see description of the CodeLocation and File classes.

This is an optional clause, since CodeLocation is already referencing a File.

Superclass

FindingFact

Associations

file:File[1]	File of the system under investigation in which the original weakness has been reported
finding: Finding[1]	Weakness that has been discovered in the code of the system under investigation

Example

See 10.1.1.1

10.1.1.8 FindingsReportedInBuild Class

The FindingsReportedInBuild class represents a verb concept “Finding is reported in Build”. This clause provides a direct association between a weakness finding and the specific Build of the system under investigation, when the original finding was reported by some Generator. For more details, see description of the Build class.

Situations where one instance of Finding is a subject of more than one FindingsReportedInBuild clauses, may occur after merging multiple TOIFSegment in a TOIF repository in which case it may be beneficial to further normalize similar findings reported in multiple builds by merging them into a single instance.

Superclass

FindingFact

Associations

build: Build[1]	TOIF build has a name and description and is related to the project
finding: Finding[1]	Weakness that has been discovered in the code of the system under investigation

Example

See 10.1.1.1

10.1.2 WeaknessType Class Diagram

This section describes the UML representation of the Weakness Type Identifier concept and the corresponding facts.

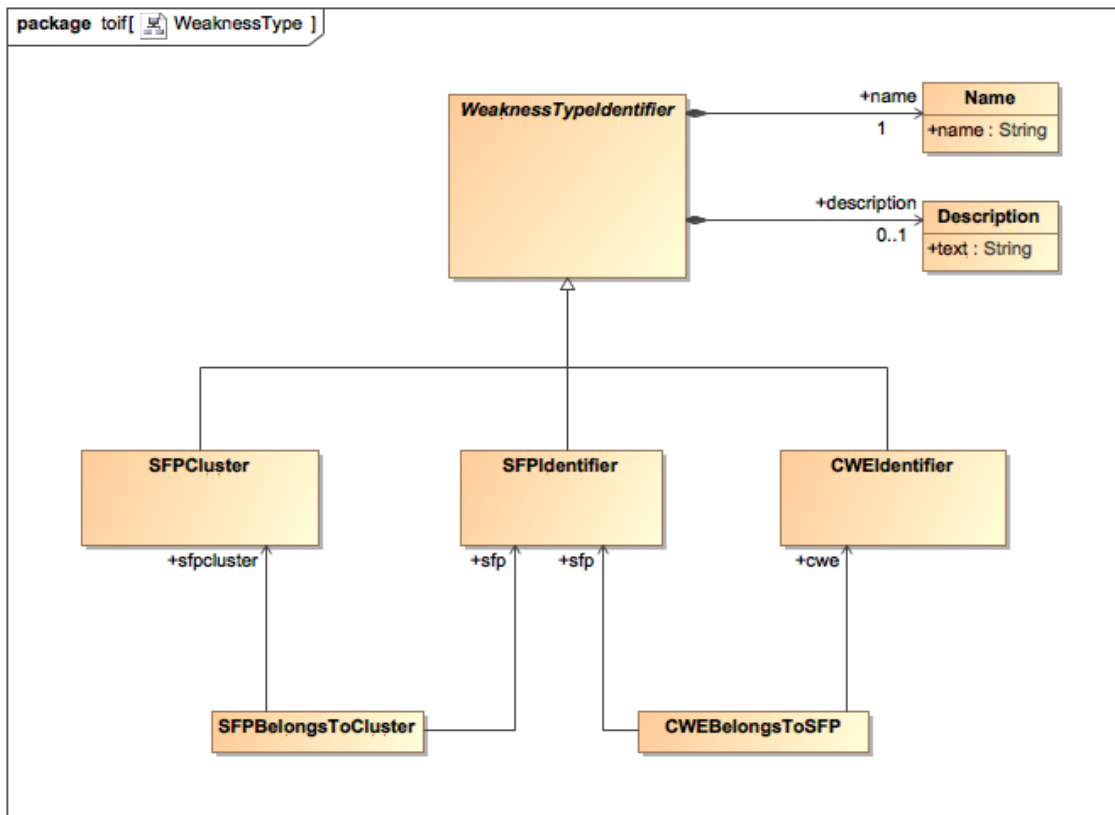


Figure 4. UML class diagram WeaknessType

10.1.2.1 WeaknessTypeIdentifier Class (abstract)

WeaknessTypeIdentifier is one of the key concepts in TOIF since it provides the means to achieve unique identification of weakness finding reports through a normalized standardized type identifier. The WeaknessTypeIdentifier in TOIF is a 3 level hierarchical structure consisting of the so-called SFP Cluster at the top, the SFP Identifier in the middle and the CWE identifier at the bottom. The use of Software Fault Patterns (SFP) catalog to augment CWE provides the means to overcome semantic ambiguity of CWE. Although TOIF uses the CWE identifiers, the allocation of such identifiers must be coordinated with the mappings described in the SFP catalog, including the gaps and ambiguities. This strategy represents a unique formalized approach to normalization of (a discernable subset of) code weaknesses that is also aligned with the automated generation of test cases from the same set of formalizations. On the other hand, TOIF recognizes the importance of using CWE identifiers as the basis for the normalization.

Superclass

BasicEntity

Associations

name:Name[1]	Owned attribute that specifies the name of weakness type identifier
description:Description[0..1]	Owned attribute that provides the text description of the normalized weakness type

Example

```

<fact xmi:type="toif:CWEIdentifier" xmi:id="CWE-561">
  <description text="xxxxxx"/>
</fact>

<fact xmi:type="toif:CWEBelongsToSFP"
cwe="CWE-561" sfp="SFP-8"/>
<fact xmi:type="toif:SFPBelongsToCluster"
sfp="SFP-8" cluster="Authentication"/>

<fact xmi:type="toif:SFPIdentifier" xmi:id="SFP-8">
  <name name="es-ef-pi-eight"/>
  <description description="this is the description that usually comes with
it"/>

<fact xmi:type="toif:SFPCluster" xmi:id="Authentication">
  <name name="Authentication Cluster"/>
  <description description="Description of the cluster"/>
</fact>

```

10.1.2.2 CWEIdentifier Class

CWEIdentifier class represents the CWE Identifier concept.

Superclass

WeaknessTypeIdIdentifier

Example

See 10.1.2.1

10.1.2.3 SFPIdentifier Class

SFPIdentifier class represents the SFP Identifier concept.

Superclass

WeaknessTypeIdIdentifier

Example

See 10.1.2.1

10.1.2.4 SFPCluster Class

SFPCluster class represents the SFP Cluster concept.

Superclass

WeaknessTypeIdIdentifier

Example

See 10.1.2.1

10.1.2.5 CWEBelongsToSFP Class

CWEBelongsToSFP class represents the verb concept CWE belongs to SFP.

Superclass

WeaknessTypeFact

Associations

cwe:CWEIdentifier[1] A weakness type defined by the Common Weakness Enumeration (CWE)

sfp: SFPIIdentifier[1] SFP Identifier which formalizes the mapping to the CWE identifier of the clause

Constraints

1. Each CWEIdentifier belongs to exactly one SFPIIdentifier

Example

See 10.1.2.1

10.1.2.6 SFPBelongsToCluster Class

SFPBelongsToSCluster class represents the verb concept SFP belongs to Cluster.

Superclass

WeaknessTypeFact

Associations

sfp:SFPIIdentifier[1] SFP Identifier that is the subject of the clause

cluster: SFPCluster[1] SFP Cluster to which the SFP Identifier belongs

Constraints

1. Each SFPIIdentifier belongs to exactly one SFPCluster

Example

See 10.1.2.1

10.1.3 Weakness Class Diagram

This section describes the UML representation of Weakness concept and the corresponding facts.

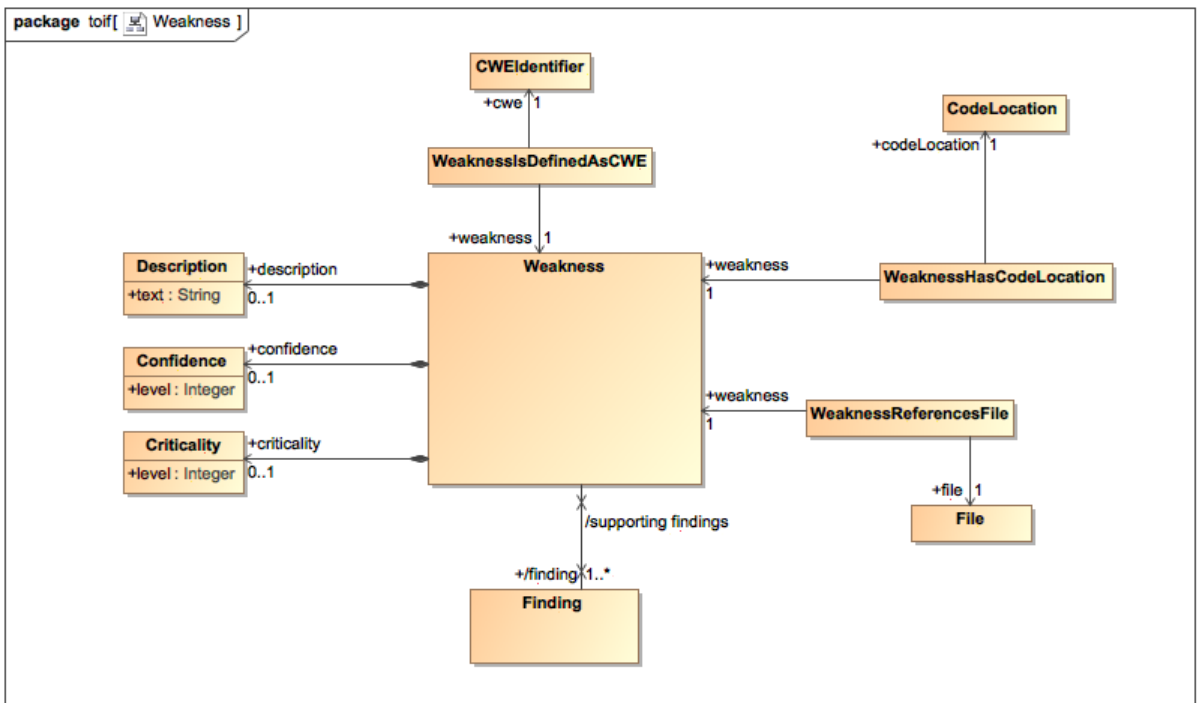


Figure 5. UML class diagram Weakness

10.1.3.1 Weakness Class

Weakness class represents the Weakness concept. Objects of this class are created by TOIF Analytics Tools, and not by the TOIF Adaptor Tools as are the Finding objects. A Weakness object represents a unique weakness in the code of system under assessment, as supported by one or more Findings. This class supports integration and analysis of multiple TOIF reports and serves as the subject for additional statements, mainly the criticality, confidence, and citing statements. Confidence and Criticality statements involve owned attributes of the Weakness class, and Citing statements involve additional instances of a Citing class, described in section 10.1.4.

The Weakness class is an important part of the TOIF interface for the consumer tools and integration tools that can be used for the vulnerability management purposes. Property Finding is provided for illustration purposes only, as it is defined as derived, and the corresponding association is non-navigable in both directions. Relation between Weakness and its supporting Findings is dynamic: while Finding object is related to a particular Build, a Weakness is related to the entire Project, but may have a certain range of builds during which it was present, starting from the build where one or more tools has reported this weakness until the build where none of the tools were any longer reporting this weakness either because it was fixed or for other reasons.

Superclass

BasicEntity

Constraints

1. Each Weakness must be the subject of exactly one WeaknessIsDefinedAsCWE clause
2. Each Weakness must be the subject of exactly one WeaknessHasCodeLocation clause

Associations

description:Description[0..1] Owned attribute that provides than informal text description of the weakness

criticality:Criticality[0..1]	Owned attribute that specifies criticality of the weakness in terms of the impact that it may cause.
confidence:Confidence[0..1]	Owned attribute that specifies the confidence in this weakness claim.

Example

```
<fact xmi:type="toif:Weakness" xmi:id="w0001">
  <criticality xmi:type="Criticality" level=80 />
</fact>
```

```
  <fact xmi:type="toif:WeaknessHasCodeLocation"
    finding="w0001" location="loc10"/>
  <fact xmi:type="toif:WeaknessHasCodeLocation"
    finding="w0001" location="loc10"/>
```

```
<fact xmi:type="toif:CodeLocation" xmi:id="loc10">
  <linenumber linenumber="1856"/>
</fact>
```

```
<fact xmi:type="toif:Finding" xmi:id="f0001"/>
```

For the facts related to the finding id="f0001" refer to example in Section 10.1.1.1

10.1.3.2 WeaknessIsDefinedAsCWE Class

CodeLocationReferencesFile class represents the verb concept "Code Location references File".

Superclass

WeaknessFact

Associations

location:CodeLocation[1] Code Location that is the subject of the clause

file: File[1] File that is referenced by the Code Location of the clause

Example

See 10.1.3.1

10.1.3.3 WeaknessHasCodeLocation Class

CodeLocationReferencesFile class represents the verb concept "Code Location references File".

Superclass

WeaknessFact

Associations

location:CodeLocation[1]	Code Location that is the subject of the clause
file: File[1]	File that is referenced by the Code Location of the clause

Example

See 10.1.3.1

10.1.3.4 WeaknessReferencesFile Class

CodeLocationReferencesFile class represents the verb concept “Code Location references File”.

Superclass

WeaknessFact

Associations

location:CodeLocation[1]	Code Location that is the subject of the clause
file: File[1]	File that is referenced by the Code Location of the clause

Example

See 10.1.3.1

10.1.4 Citing Class Diagram

This section describes the UML representation of Citing concept and the corresponding facts.

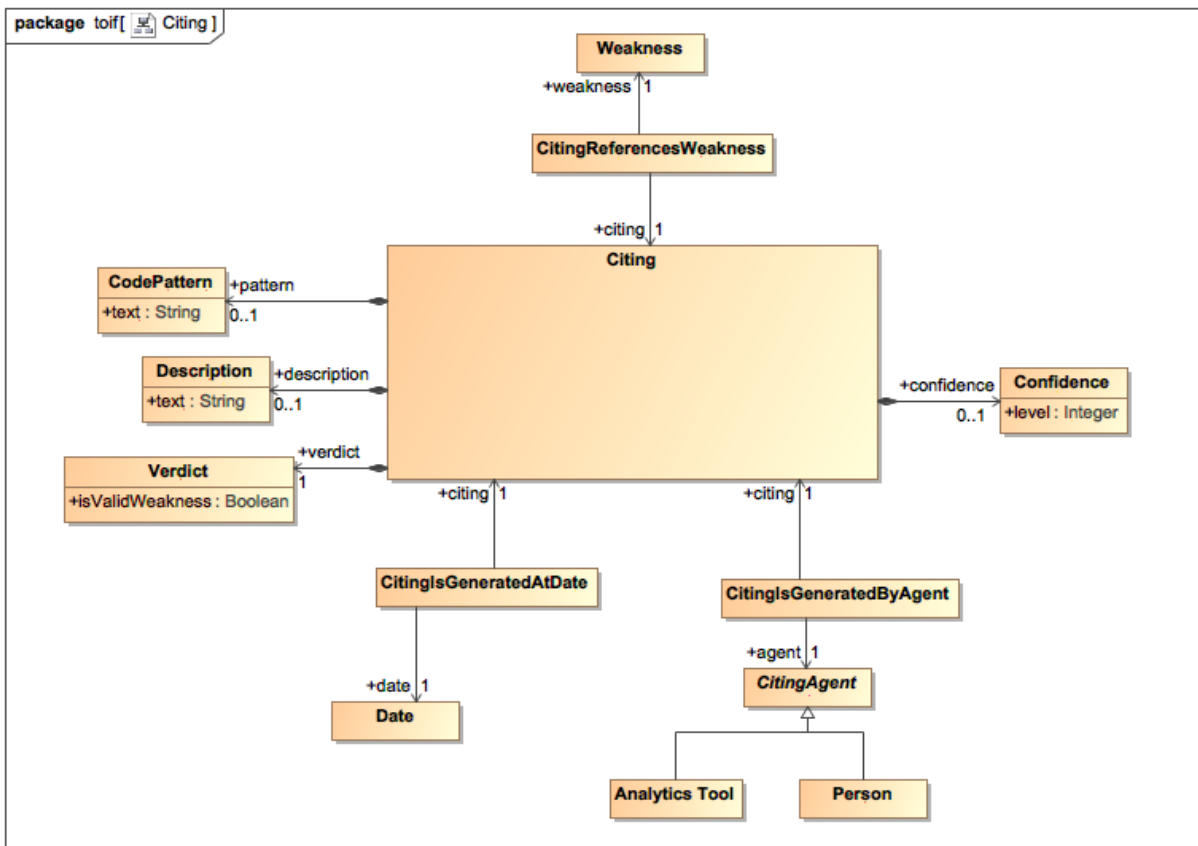


Figure 6. UML class diagram Citing

10.1.4.1 Citing Class

Citing class represents the Citing concept. Objects of this class are created by TOIF Analytics Tools. A Citing object supplies additional statements to some Weakness object, mainly the verdict, confidence as well as some audit trail. This class is an important part of the TOIF interface for the consumer tools and integration tools that can be used for the vulnerability management purposes. CodePattern element can be used to group related Weaknesses that have the same CWE. This mechanism can be used by TOIF Analytics tools to identify the characteristics of a Weakness, and then identify other Weaknesses that have the same characteristics.

Superclass

BasicEntity

Constraints

1. Each Citing must be the subject of exactly one CitingReferencesWeakness clause

Associations

description:Description[0..1]	Owned attribute that provides than informal text description of the weakness
verdict:Verdict	Owned attribute that specifies criticality of the weakness in terms of the impact that it may cause.

confidence:Confidence[0..1]	Owned attribute that specifies the confidence in this weakness claim.
pattern:CodePattern[0..1]	Owned attribute that specifies the confidence in this weakness claim.

Example

```
<fact xmi:type="toif:Citing" xmi:id="c110">
  <verdict xmi:type="Verdict" verdict="true"/>
  <condifence xmi:type="Confidence" level="90" />
</fact>

<fact xmi:type="toif:CitingReferencesWeakness"
  location="c110" file="w0001"/>
```

For the facts related to Weakness id="w0001" refer to example in Section 10.1.3.1

10.1.4.2 CitingReferencesWeakness Class

CitingReferencesWeakness class represents the verb concept "Citing references Weakness".

Superclass

WeaknessFact

Associations

location:CodeLocation[1]	Code Location that is the subject of the clause
file: File[1]	File that is referenced by the Code Location of the clause

Example

See 10.1.4.1

10.1.4.3 CitingIsGeneratedAtDate Class

CitingIsGeneratedAtDate class represents the verb concept "Citing is generated at Date".

Superclass

WeaknessFact

Associations

location:CodeLocation[1]	Code Location that is the subject of the clause
--------------------------	---

file: File[1]

File that is referenced by the Code Location of the clause

Example

See 10.1.4.1

10.1.4.4 CitingAgent Class (abstract)

CitingAgent class represents the common supertype of agents that can generate Citings and is used as the endpoint class of the clause “CitingIsGeneratedByAgent”. Two subtypes of this class are Person and Analytics tool, defined in Housekeeping concepts section.

Superclass

Element

10.1.4.5 CitingIsGeneratedByAgent Class

CitingIsGeneratedByAgent class represents the verb concept “Citing is generated at Agent”.

Superclass

WeaknessFact

Associations

citing:Citing[1]

Citing that is the subject of the clause

agent: CitingAgent[1]

CitingAgent that has generated the Citing of the clause

Example

See 10.1.3.1

10.1.5 Code Location Class Diagram

This section describes the UML representation of the Code Location concept and the corresponding facts.

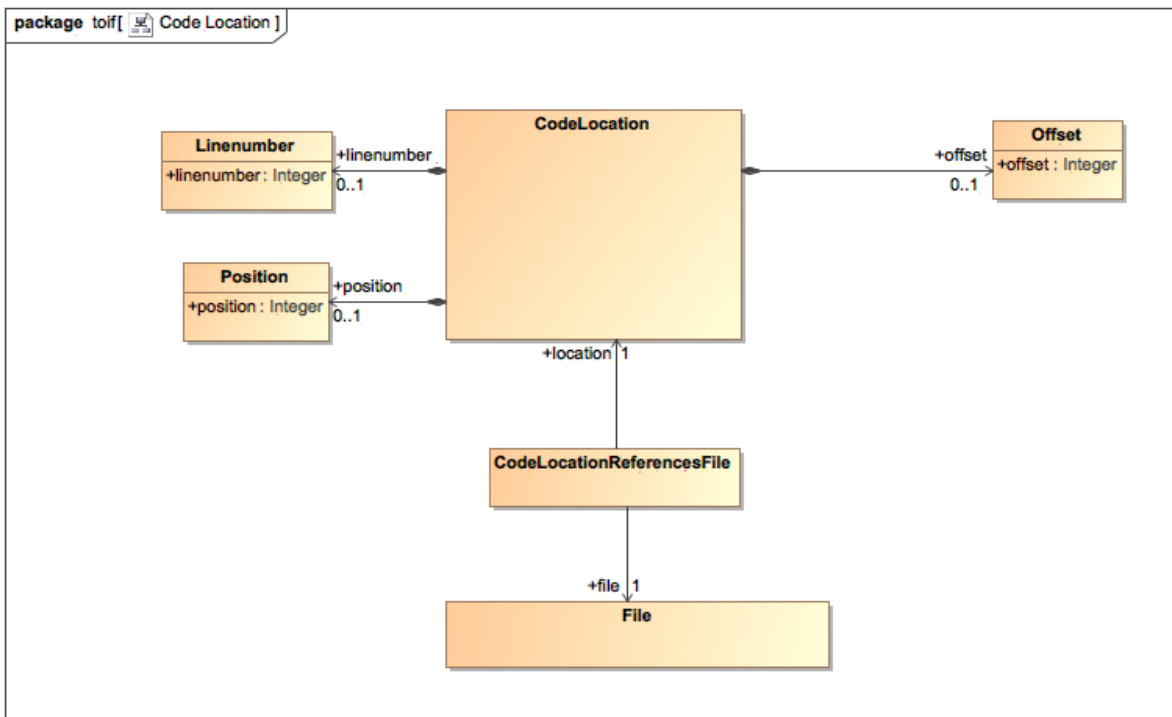


Figure 7. UML class diagram Code Location

10.1.5.1 CodeLocation Class

CodeLocation class represents the Code Location concept.

Superclass

BasicEntity

Constraints

3. Each CodeLocation must define either Linenumber or Offset attribute
4. When CodeLocation defines Position attribute, the CodeLocation must define Linenumber attribute

Associations

linenumber:Linenumber[0..1]	Owned attribute that specifies the linenumber of the Code Location. The Linenumber is taken to be in the File that is referenced by the CodeLocation
position:Position[0..1]	Owned attribute that specifies the position of the Code Location. The position is takes to be within the Linenumber.
offset:Offset[0..1]	Owned attribute that specifes the offset in a binary image. The offset is assumed to be in the File that is referenced by the Code Location.

Example

```

<fact xmi:type="toif:CodeLocation" xmi:id="loc10">
  <linenumber linenumber="1856"/>
</fact>

  <fact xmi:type="toif:CodeLocationReferencesFile"
    location="loc10" file="f10"/>

<fact xmi:type="toif:File" xmi:id="f10">
  <name name="main.c"/>
</fact>

```

10.1.5.2 CodeLocationReferencesFile Class

CodeLocationReferencesFile class represents the verb concept “Code Location references File”.

Superclass

CodeLocationFact

Associations

location:CodeLocation[1]	Code Location that is the subject of the clause
file: File[1]	File that is referenced by the Code Location of the clause

Example

See 10.1.5.1

10.1.6 File Class Diagram

This section describes the UML representation of the File concept and the corresponding facts.

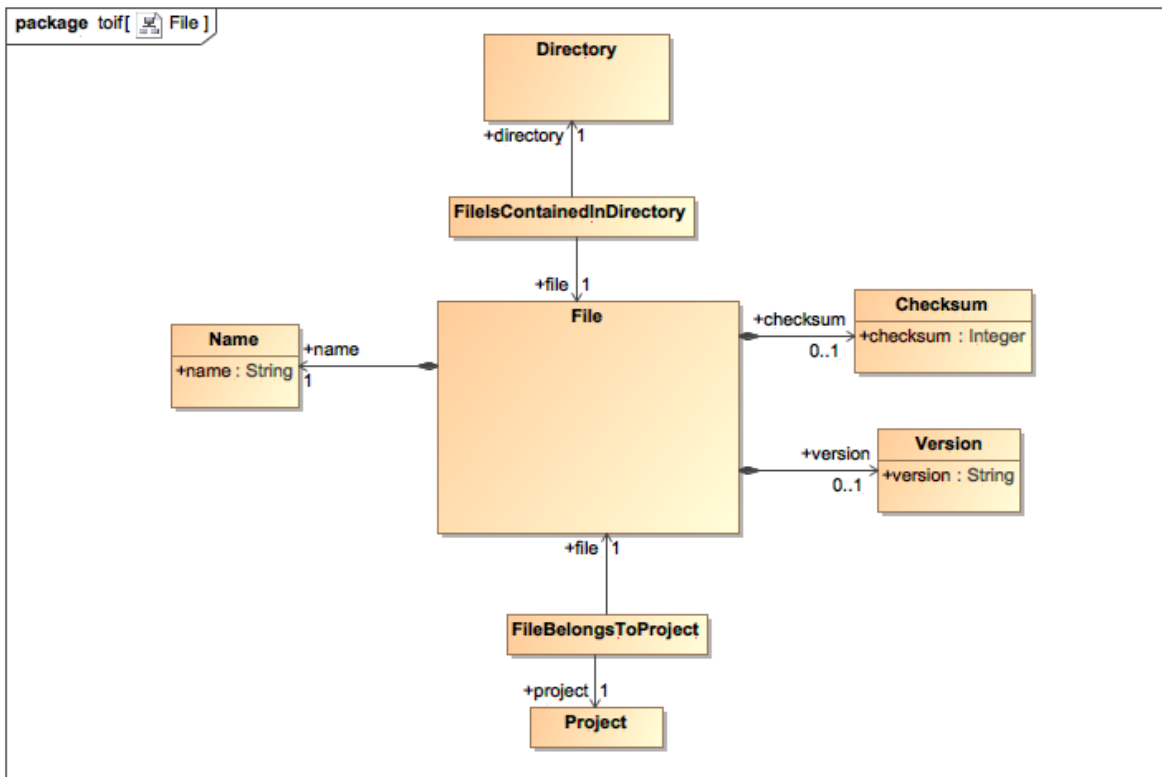


Figure 8. UML class diagram File

10.1.6.1 File Class

File class represents the File concept.

In TOIF corresponds to the InventoryItem concept in KDM. In TOIF a File is assumed to represent code of the system under investigation and is usually either a source file (KDM SourceFile) or an executable file (KDM ExecutableFile).

Superclass

BasicEntity

Associations

name:Name[1]	Owned attribute of the File, specifying its name
version:Version[0..1]	Owned attribute of the File, specifying its version
checksum:Checksum[0..1]	Owned attribute of the File, specifying its checksum

Constraints

1. Each File class must be the subject of at least one FileBelongsToProject clause

Example

```

<fact xmi:type="toif:File" xmi:id="f10">
  <name name="main.c"/>
  <checksum checksum="898432423894723"/>
  <version version="10092017"/>
</fact>
  
```

```

</fact>

    <fact xmi:type="toif:FileIsContainedInDirectory"
        file="f10" directory="d10"/>
    <fact xmi:type="toif:FileBelongsToProject"
        file="f10" project="p10"/>

<fact xmi:type="toif:Directory" xmi:id="d10">
    <name name="applications/src"/>
</fact>
<fact xmi:type="toif:Project" xmi:id="p10">
    <name name="Dispatcher"/>
</fact>

```

10.1.6.2 FileIsContainedInDirectory Class

FileIsContainedInDirectory class represents the verb concept “File is contained in Directory”.

Superclass

CodeLocationFact

Associations

file:File[1]	File that is the subject of the clause
directory: Directory[1]	Directory that the File is contained in

Example

See 10.1.6.1

10.1.6.3 FileBelongsToProject Class

FileBelongsToProject class represents the verb concept “File belongs to Project”.

Superclass

ProjectFact

Associations

file:File[1]	File that is the subject of the clause
project: Project[1]	Project that the File belongs o

Example

10.1.7 Directory Class Diagram

This section describes the UML representation of the Directory concept and the corresponding facts.

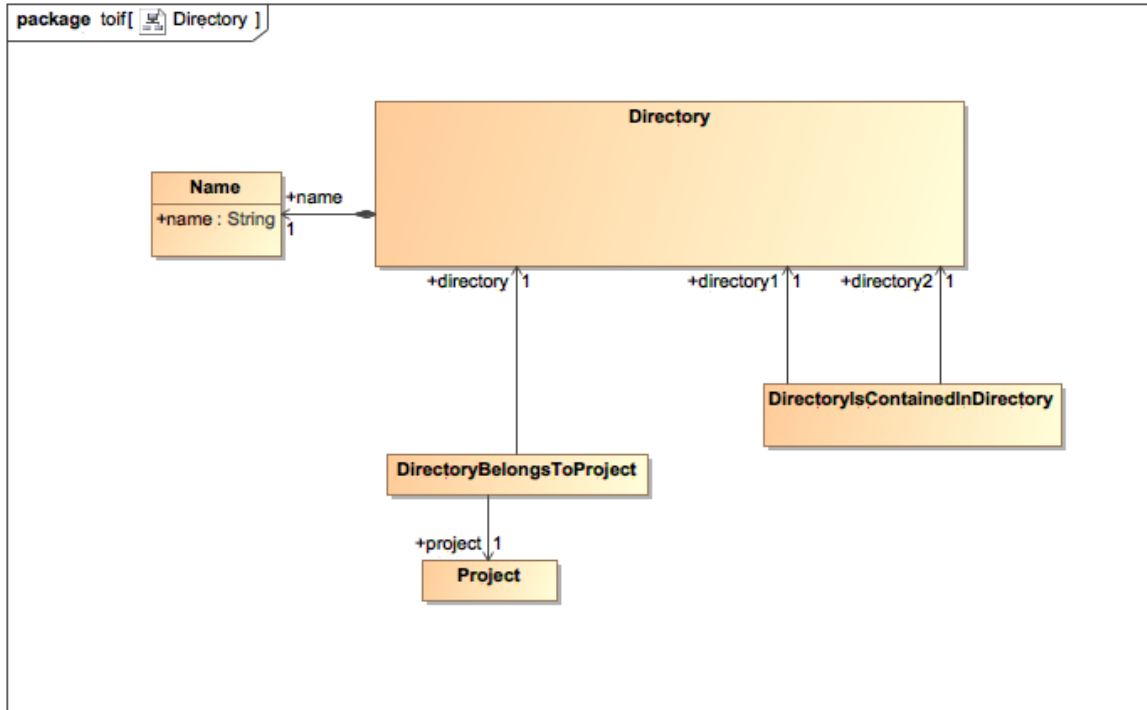


Figure 9. UML class diagram Directory

10.1.7.1 Directory Class

Directory class represents the Directory concept. Directories provide additional context to the TOIF Findings, which reference individual Files. TOIF does not make any assumptions on whether the name of the Directory represents an absolute or a relative path. TOIF also does not make any assumptions regarding the correlation between the names of the Directory and File and relations DirectoryIsContainedInDirectory and FileIsContainedInDirectory.

Superclass

BasicEntity

Associations

name:Name[1] Owned attribute that specifies the name of the Directory

Example

```
<fact xmi:type="toif:Directory" xmi:id="d10">
  <name name="applications/src"/>
</fact>
```

```

<fact xmi:type="toif:DirectoryIsContainedInDirectory"
      directory1="d10" directory2="d20"/>
<fact xmi:type="toif:DirectoryBelongsToProject"
      directory="d10" project="p10"/>

<fact xmi:type="toif:Directory" xmi:id="d20">
  <name name="dispatcher"/>
</fact>
<fact xmi:type="toif:Project" xmi:id="p10">
  <name name="Dispatcher"/>
</fact>

```

10.1.7.2 DirectoryBelongsToProject Class

DirectoryBelongsToProject class represents the verb concept “Directory belongs to Project”.

Superclass

ProjectFact

Associations

directory:Directory[1]	Directory that is the subject of this clause
project: Project[1]	Project that the Directory belongs to

Example

See 10.1.7.1

10.1.7.3 DirectoryIsContainedInDirectory Class

DirectoryIsContainedInDirectory class represents the verb concept “Directory₁ is contained in Directory₂”.

Superclass

CodeLocationFact

Associations

directory1:Directory[1]	Directory that is the subject of this clause
directory2: Directory[1]	Directory in which the subject Directory is contained

Example

See 10.1.7.1

10.1.8 Semantic Statement Class Diagram

This section describes the UML representation of the Statement concept and the corresponding facts.

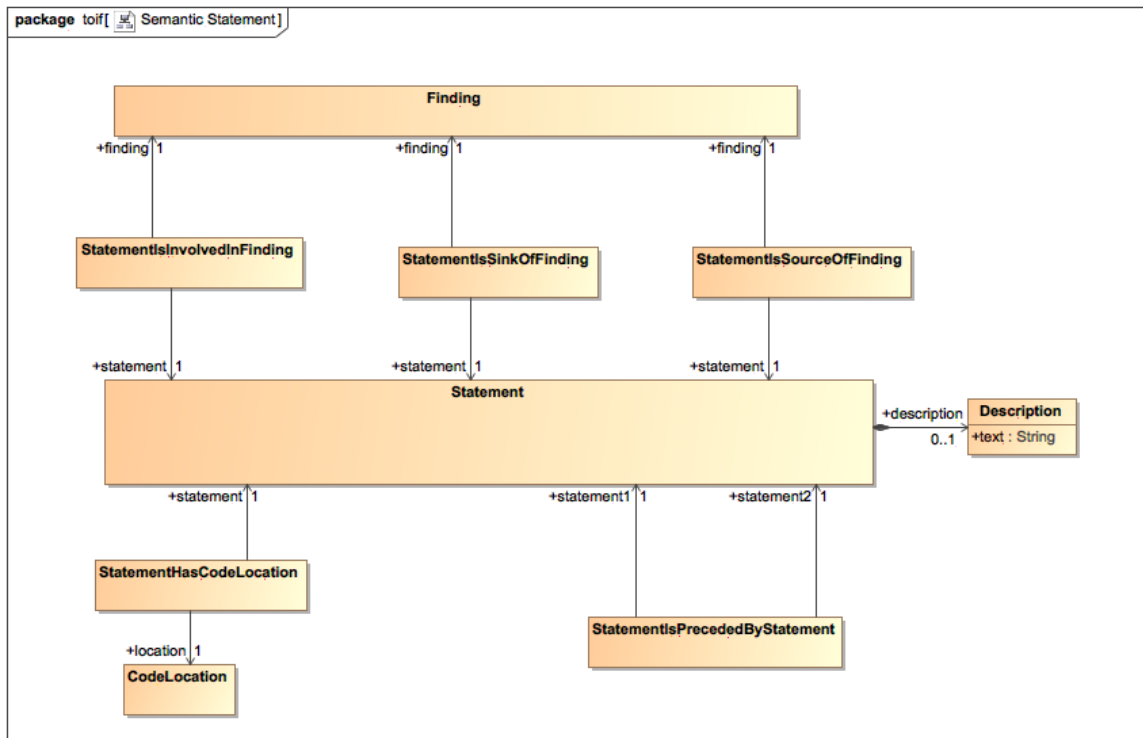


Figure 10. UML class diagram Semantic Statement

10.1.8.1 Statement Class

Statement class represents the Statement concept.

Superclass

BasicEntity

Associations

description:Description[0..1] Owned attribute that an informal text description of the Statement

Constraints

1. Each Statement class must be the subject of at least one StatementHasCodeLocation clause

Example

```

<fact xmi:type="toif:Statement" xmi:id="s10"/>

<fact xmi:type="toif:StatementIsInvolvedInFinding"
  statement="s10" finding="f10"/>

<fact xmi:type="toif:StatementIsSinkOfFinding"

```

```

        statement="s20" finding="f20"/>
<fact xmi:type="toif:StatementIsSourceOfFinding"
    statement="s30" finding="f30"/>

<fact xmi:type="toif:StatementPrecedesStatement"
    statement1="s30" statement2="s20"/>
<fact xmi:type="toif:StatementPrecedesStatement"
    statement1="s10" statement2="s20"/>

<fact xmi:type="toif:StatementHasCodeLocation"
    statement1="s10" location="loc10"/>

<fact xmi:type="toif:Statement" xmi:id="s20">
    <description text="*pHandler( pData, 0x200 );" />
</fact>

<fact xmi:type="toif:Statement" xmi:id="s30"/>

<fact xmi:type="toif:Finding" xmi:id="f10"/>
<fact xmi:type="toif:Finding" xmi:id="f20"/>

<fact xmi:type="toif:CodeLocation" xmi:id="loc10">
    <linenumber linenumber="1856"/>
</fact>

```

10.1.8.2 StatementIsInvolvedInFinding Class

StatementIsInvolvedInFinding class represents the verb concept “Statement is involved in Finding”.

Superclass

SemanticFact

Associations

statement:Statement[1]	Statement that is the subject of the clause
finding: Finding[1]	Finding in which the Statement is involved in

Example

See 10.1.8.1

10.1.8.3 StatementIsSinkOfFinding Class

StatementIsSinkOfFinding class represents the verb concept “Statement is sink of Finding”. A Sink of a Finding corresponds to the necessary condition of the Finding.

Superclass

SemanticFact

Associations

statement:Statement[1]	Statement that is the subject of the clause
finding: Finding[1]	Finding of which the Statement is the Sink

Example

See 10.1.8.1

10.1.8.4 StatementIsSourceOfFinding Class

StatementIsSourceOfFinding class represents the verb concept “Statement is source of Finding”. A Source of a Finding corresponds to one of the sufficient conditions of the Finding.

Superclass

SemanticFact

Associations

statement:Statement[1]	Statement that is the subject of the clause
finding: Finding[1]	Finding of which the Statement is the Source

Example

See 10.1.8.1

10.1.8.5 StatementHasCodeLocation Class

StatementHasCodeLocation class represents the verb concept “Statement has Code Location”.

Superclass

SemanticFact

Associations

statement:Statement[1]	Statement that is the subject of the clause
location: CodeLocation[1]	Code Location to which the Statement refers

Example

See 10.1.8.1

10.1.8.6 StatementIsPrecededByStatement Class

StatementIsPrecededByStatement class represents the verb concept “Statement₁ is preceded by Statement₂”. This fact corresponds to a segment of the data flow path between a Source and a Sink of the Finding.

Superclass

SemanticFact

Associations

statement1:Statement[1]	Statement that is the subject of the clause
statement2: Statement[1]	Statement that precedes Statement1

Example

See 10.1.8.1

10.1.9 Semantic Data Class Diagram

This section describes the UML representation of the Data Element concept and the corresponding facts.

10.1.9.1 DataElement Class

DataElement class represents the Data Element concept.

Superclass

BasicEntity

Associations

name:Name[1]	Owned attribute that specifies the name of the Data Element
description:Description[0..1]	Owned attribute that an informal text description of the Data Element

Example

```
<fact xmi:type="toif:DataElement" xmi:id="d10">
  <name name="X"/>
  <description text="struct pData * X[ MAXDATA];" />
</fact>
```

```
<fact xmi:type="toif:DataIsInvolvedInFinding"
  data="d10" finding="f10"/>
<fact xmi:type="toif:DataIsInvolvedInFinding"
  data="d10" project="f20"/>
```

```
<fact xmi:type="toif:DataIsInvolvedInStatement"
  data="d10" statement="s20"/>
```

```

<fact xmi:type="toif:DataIsInvolvedInStatement"
  data="d10" statement="s30"/>

<fact xmi:type="toif:DataIsDefinedAtCodeLocation"
  data="d10" location="loc10"/>

<fact xmi:type="toif:Statement" xmi:id="s20"/>
<fact xmi:type="toif:Statement" xmi:id="s30"/>

<fact xmi:type="toif:Finding" xmi:id="f10"/>
<fact xmi:type="toif:Finding" xmi:id="f20"/>

<fact xmi:type="toif:CodeLocation" xmi:id="loc10">
  <linenumber linenumber="1856"/>
</fact>

```

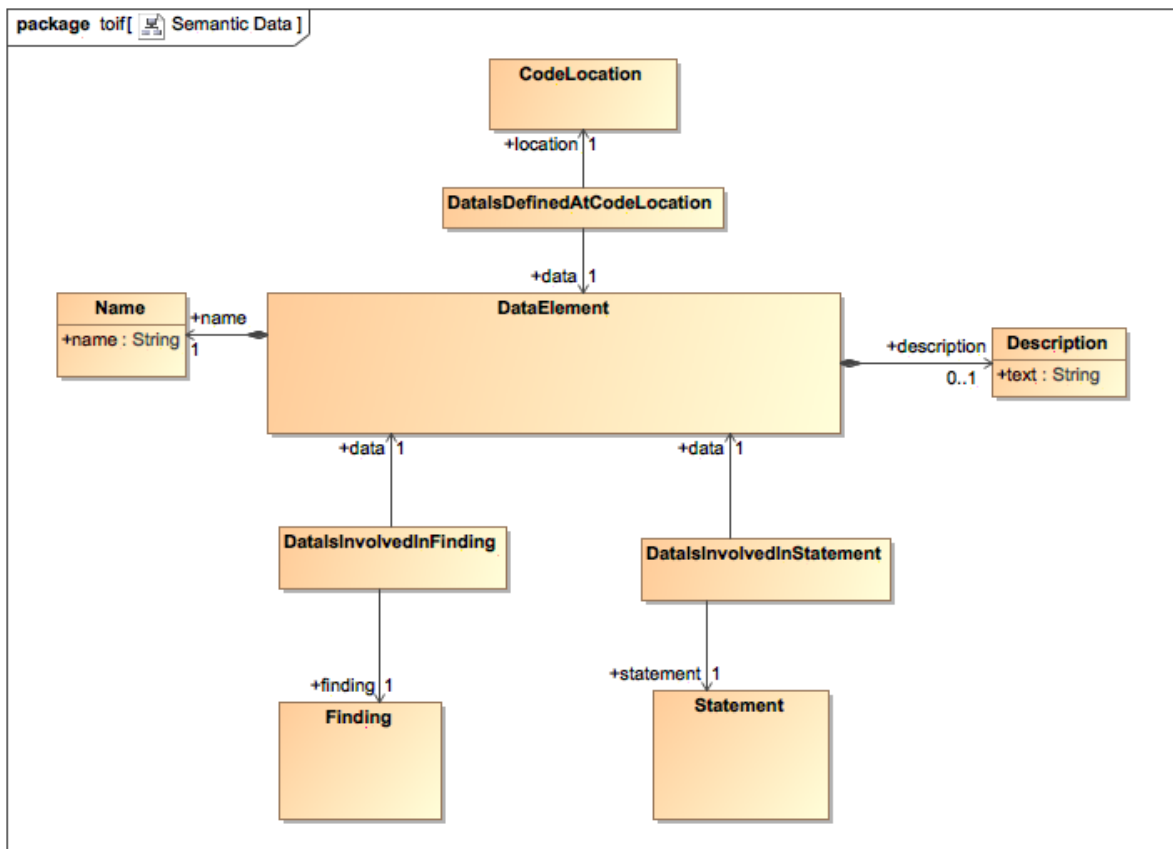


Figure 11. UML class diagram Semantic Data

10.1.9.2 DataIsInvolvedInFinding Class

DataIsInvolvedInFinding class represents the verb concept “Data Element is involved in Finding”.

Superclass

SemanticFact

Associations

data:DataElement[1]	Data Element that is the subject of the clause
finding: Finding[1]	Finding in which the Data Element is involved

Example

See 10.1.9.1

10.1.9.3 DataIsInvolvedInStatement Class

DataIsInvolvedInStatement class represents the verb concept “Data Element is involved in Statement”. Data Element usually corresponds to the data flow path between the Source and the Sink of a Finding.

Superclass

SemanticFact

Associations

data:DataElement[1]	Data Element that is the subject of the clause
statement: Statement[1]	Statement in which the Data Element is involved

Example

See 10.1.9.1

10.1.9.4 DataIsDefinedAtCodeLocation Class

DataIsDefinedAtCodeLocation class represents the verb concept “Data Element is defined at Code Location”.

Superclass

SemanticFact

Associations

data:DataElement[1]	DataElement that is the subject of the clause
location: Location[1]	Code Location at which the Data Element is defined

Example

See 10.1.9.1

10.2 The housekeeping elements of the TOIF XML

This section presents 7 UML class diagrams that describe the housekeeping elements of the TOIF XML: Build, Housekeeping, Project, Tools, Organization, Person, Role.

10.2.1 Build Class Diagram

This section describes the UML representation of the Build concept and illustrates the corresponding facts are described in the Basic Entities and Facts section. This diagram shows how housekeeping facts provide the context for the basic facts and allow management of the TOIF facts in the life-cycle of the system-under investigation, across multiple builds, evolving SCA tools, TOIF Adaptors, and TOIF Orchestration tools, and evolving understanding of the weaknesses. Also in an enterprise environment, source files and build may be shared across multiple systems.

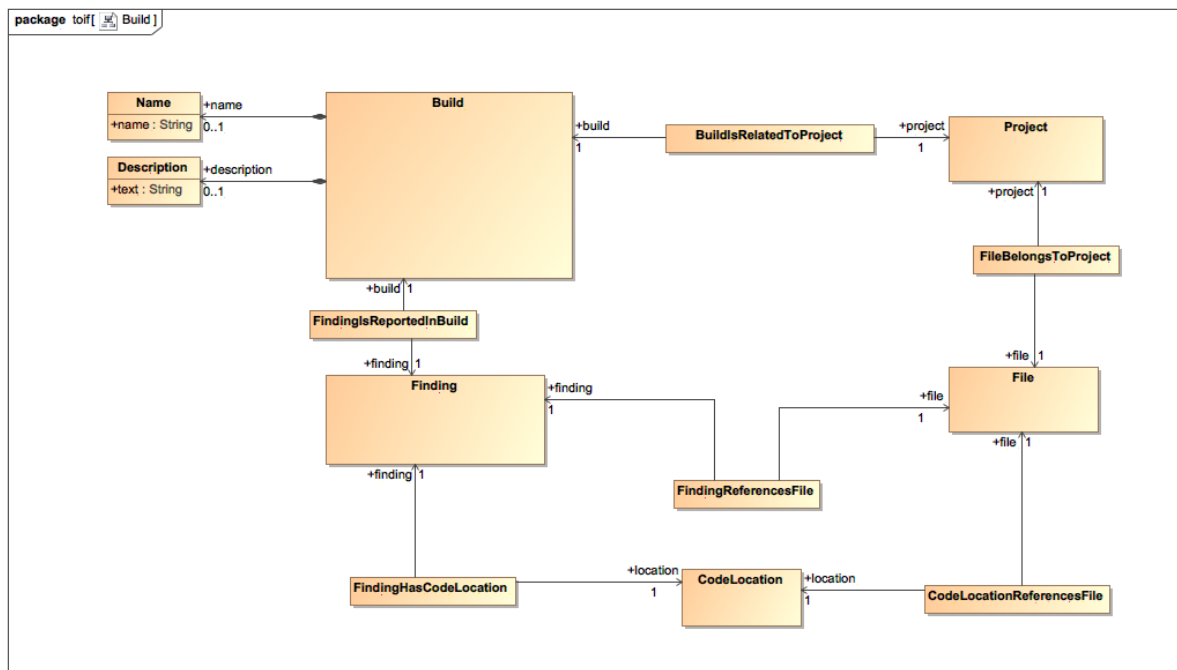


Figure 12. UML class diagram Build

10.2.1.1 Build Class

Build class represents the Build concept. The superclass Housekeeping Entity is defined in Section 10.3.5.8 Housekeeping Entities Class Diagram. Build represents (the results of) one particular build of the system under assessment, for example, corresponding to running a custom “Make” script.

Superclass

HousekeepingEntity

Associations

name:Name[1]	Owned attribute that specifies the name of the Build
description: Description[0..1]	Owned attribute that provides a text description of the Build

Constraints

1. Each Build class must be the subject of at least one BuildIsRelatedToProject clause
2. Each Build class must be the subject of at least one BuildIsGeneratedAtDate clause

Example

The following example illustrates 2 Builds, both belonging to the same Project, and a single Finding that was reported in both builds. The Finding references a File that is related to the same Project. Note that some mandatory clauses are omitted.

```
<fact xmi:type="toif:Build" xmi:id="b1020171330">
  <description text="xxxxxx"/>
</fact>
<fact xmi:type="toif:Build" xmi:id="b1020171340">
  <description text="xxxxxx"/>
</fact>

<fact xmi:type="toif:Project" xmi:id="proj01">
  <description text="xxxxxx"/>
</fact>

<fact xmi:type="toif:Finding" xmi:id="f0001"/>

  <fact xmi:type="toif:FindingIsReportedToBuild"
    finding="f0001" build="b1020171330"/>
  <fact xmi:type="toif:FindingIsReportedToBuild"
    finding="f0001" build="b1020171340"/>

  <fact xmi:type="toif:FindingReferencesFile"
    finding="f0001" file="f10"/>

<fact xmi:type="toif:File" xmi:id="f10">
  <name name="main.c"/>
</fact>

  <fact xmi:type="toif:BuildIsRelatedToProject"
    build=" b1020171330" project=" proj01"/>
  <fact xmi:type="toif:BuildIsRelatedToProject"
    build=" b1020171340" project="proj01"/>

  <fact xmi:type="toif:FileBelongsToProject"
    file=" f10" project="proj01"/>

  <fact xmi:type="toif:BuildIsGeneratedAtDate"
    build=" b1020171330" date="2016-10-24 at 22:30 UTC"/>
  <fact xmi:type="toif:BuildIsGeneratedAtDate"
    build=" b1020171340" date="2016-10-25 at 10:30 UTC"/>
```


10.2.1.2 BuildsRelatedToProject Class

BuildIsRelatedToProject class represents the verb concept “Build is related to Project”.

Superclass

BuildFact

Associations

build: Build[1]	Build that is the subject of the clause
project: Project[1]	Project to which the Build is related

Example

See 10.2.1.1

10.2.2 Housekeeping Class Diagram

This section describes the UML representation of several housekeeping facts related to the Build concept.

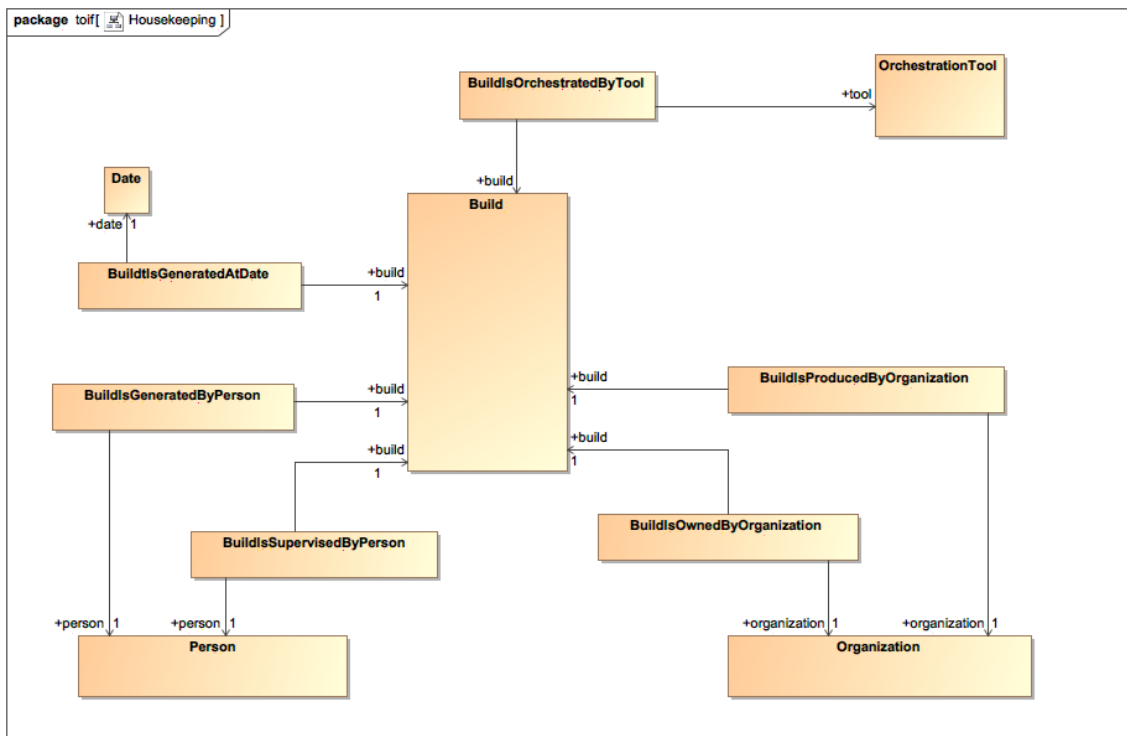


Figure 13. UML class diagram Housekeeping

10.2.2.1 BuildsOrchestratedByTool Class

BuildsOrchestratedByTool class represents the verb concept “Build is orchestrated by Tool”.

Superclass

BuildFact

Associations

build:Build[1]	Build that is the subject of the clause
tool:OrchestrationTool[1]	Orchestration tool that produced the Build

Example

```
<fact xmi:type="toif:Build" xmi:id="b1020171330">
  <description text="xxxxxx"/>
</fact>
<fact xmi:type="toif:Build" xmi:id="b1020171340">
  <description text="xxxxxx"/>
</fact>

<fact xmi:type="toif:Project" xmi:id="proj01">
  <description text="xxxxxx"/>
</fact>

  <fact xmi:type="toif:BuildIsRelatedToProject"
    build=" b1020171330" project=" proj01"/>
  <fact xmi:type="toif:BuildIsRelatedToProject"
    build=" b1020171340" project="proj01"/>

  <fact xmi:type="toif:BuildIsGeneratedAtDate"
    build=" b1020171330" date="2016-10-24 at 22:30 UTC"/>
  <fact xmi:type="toif:BuildIsGeneratedAtDate"
    build=" b1020171340" date="2016-10-25 at 10:30 UTC"/>

<fact xmi:type="toif:OrchestrationTool" xmi:id="bt01">
  <name name="Build Environment 2"/>
  <description text="xxxxxx"/>
  <version version="2.3.04A"/>
</fact>

<fact xmi:type="toif:OrchestrationTool" xmi:id="bt02">
  <name name="Build Environment 2"/>
  <description text="xxxxxx"/>
  <version version="2.3.05B"/>
</fact>

  <fact xmi:type="toif:BuildIsOrchestratedByTool"
    build=" b1020171330" project="bt01"/>
  <fact xmi:type="toif:BuildIsOrchestratedByTool"
    build=" b1020171340" project="bt01"/>

  <fact xmi:type="toif:BuildIsProducedByOrganization"
```

```

        build=" b1020171330" organization="org01"/>
    <fact xmi:type="toif:BuildIsProducedByOrganization"
        build=" b1020171340" organization="org02"/>

<fact xmi:type="toif:Organization" xmi:id="org01">
    <name name="Night Shift Build Group"/>
    <description text="xxxxxx"/>
</fact>
<fact xmi:type="toif:Organization" xmi:id="org02">
    <name name="Day Shift Build Group"/>
    <description text="xxxxxx"/>
</fact>

<fact xmi:type="toif:Person" xmi:id="p88997766">
    <name name="John The LoadBuilder"/>
    <description text="xxxxxx"/>
</fact>

    <fact xmi:type="toif:BuildIsSupervisedByPerson"
        build=" b1020171330" person="p88997766"/>
    <fact xmi:type="toif:BuildIsSupervisedByPerson"
        build=" b1020171340" person="p88997766"/>

```

10.2.2.2 BuildIsProducedByOrganization Class

BuildIsProducedByOrganization class represents the verb concept “Build is produced by Organization”.

Superclass

BuildFact

Associations

build:Build[1]	Build that is the subject of the clause
organization: Organization[1]	Organization which produced the Build

Example

See 10.2.2.1

10.2.2.3 BuildIsOwnedByOrganization Class

BuildIsOwnedByOrganization class represents the verb concept “Build is owned by Organization”.

Superclass

BuildFact

Associations

build:Build[1]	Build that is the subject of the clause
organization: Organization[1]	Organization that owns the Build

Example

The following TOIF facts illustrate ownership of builds, described in the basic example in 10.2.2.1

```
<fact xmi:type="toif:BuildIsOwnedByOrganization"
  build=" b1020171330" organization="org03"/>
<fact xmi:type="toif:BuildIsOwnedByOrganization"
  build=" b1020171340" organization="org03"/>

<fact xmi:type="toif:Organization" xmi:id="org03">
  <name name="McDuck and Sons Assurance"/>
  <description text="xxxxxx"/>
</fact>
```

10.2.2.4 BuildIsGeneratedByPerson Class

BuildIsGeneratedByPerson class represents the verb concept “Build is generated by Person”.

Superclass

BuildFact

Associations

build:Build[1]	Build that is the subject of the clause
person: Person[1]	Person that has generated the Build

Example

```
<fact xmi:type="toif:Build" xmi:id="b_007_025">
  <description text="xxxxxx"/>
</fact>

<fact xmi:type="toif:Person" xmi:id="p007">
  <name name="Scroodge The Inspector Guy"/>
  <description text="xxxxxx"/>
</fact>

<fact xmi:type="toif:BuildIsRelatedToProject"
  build="b_007_025" project=" proj01"/>

<fact xmi:type="toif:BuildIsProducedByPerson"
  build="b_007_025" person="p007"/>
<fact xmi:type="toif:BuildIsSupervisedByPerson"
  build="b_007_025" person="p88997766"/>

<fact xmi:type="toif:BuildIsGeneratedAtDate"
  build="b_007_025" date="2016-11-01 at 15:00 UTC"/>
```

10.2.2.5 BuildIsSupervisedByPerson Class

BuildIsSupervisedByPerson class represents the verb concept “Build is supervised by Person”.

Superclass

BuildFact

Associations

build: Build[1] Build that is the subject of the clause

person: Person[1] Person who has supervised the Build

Example

See 10.2.2.1 and 10.2.2.4

10.2.2.6 BuildIsGeneratedAtDate Class

BuildIsGeneratedAtDate class represents the verb concept “Build is generated at Date”.

Superclass

BuildFact

Associations

build: Build[1] Build that is the subject of the clause

date: Date[1] Date at which the Build was generated

Example

See 10.2.2.1

10.2.3 Project Class Diagram

This section describes the UML representation of the Project concept and the corresponding facts.

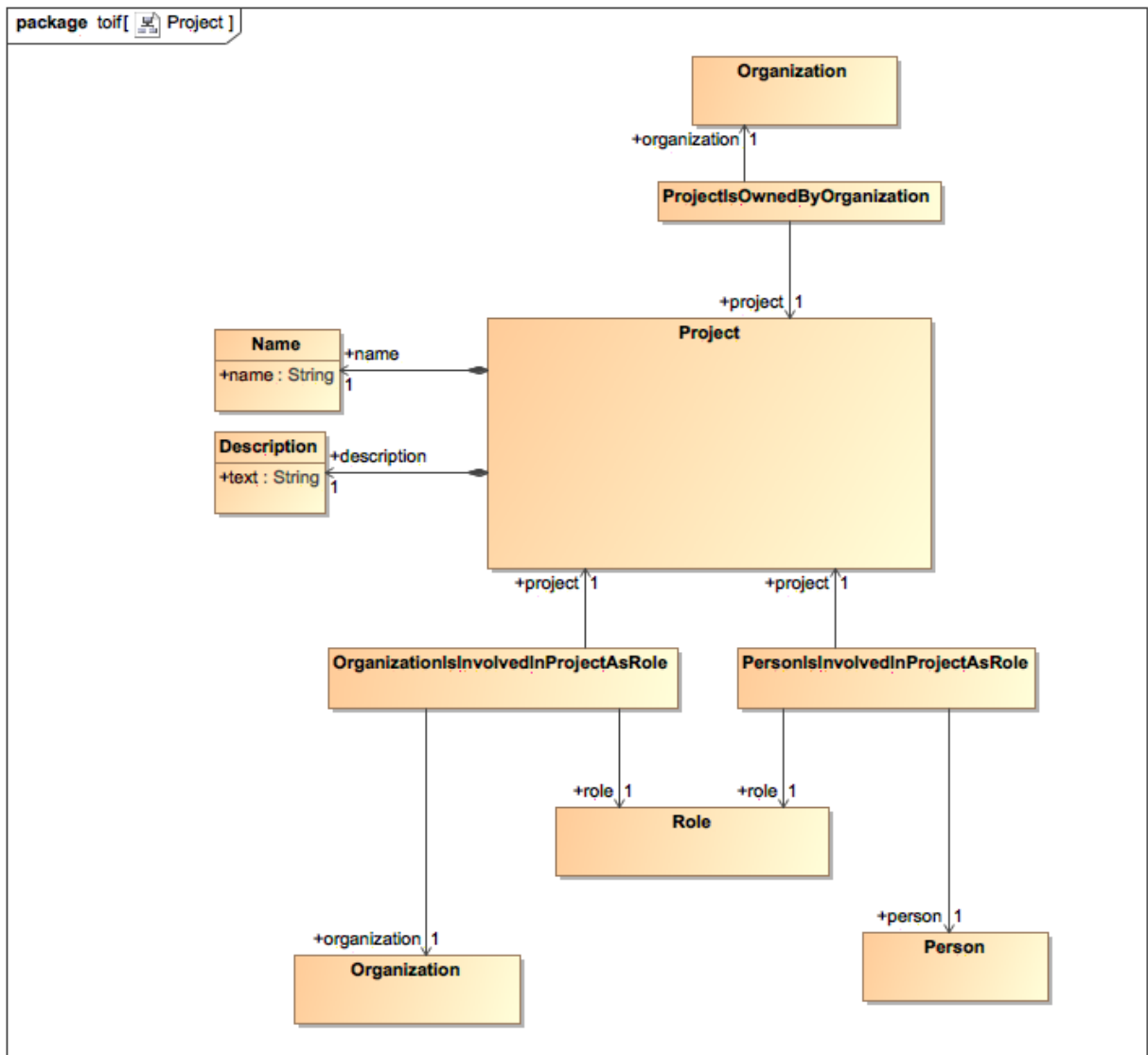


Figure 14. UML class diagram Project

10.2.3.1 Project Class

Project class represents the Project concept. Project corresponds to the system under assessment and is a container for the corresponding File and Directory elements. The corresponding facts are described in sections 10.1.6 and 10.1.7. Project is also a container for the related Builds. The corresponding facts are described in section 10.2.1

Superclass

HousekeepingEntity

Associations

name:Name[1]	Owned attribute that specifies the name of the Project
description: Description[0..1]	Owned attribute that provides an informal text description of the Project

Example

The following is an example how TOIF data is represented in TOIF XMI:

Suppose we want to represent the following statements:

There exists a project TOIF that *has description* “Flying Gizmo Assessment using TOIF”.

There exists an organization that *has name* “McDuck & Sons Assurance”

There exists an organization that *has name* “Gizmo Mfg”

There exists a person that *has name* “Scroodge

There exists a role that *has name* “Prime Investigator”

There exists a role that *has name* “CTO”

There exists a role that *has name* “Third-party Assessor”

Scroodge is employed by McDuck & Sons as CTO

Scroodge is involved in project TOIF as Prime Investigator

Project_TOIF is owned by Gizmo Mfg

McDuck & Sons Assurance is involved in Project_TOIF as Third-party Assessor

The first 7 statements represent the so-called existential facts that introduce entities. Entity project will be referenced by an identifier TOIF, other entities will be referenced by their names. The last 4 facts correspond to the regular TOIF facts that refer to the entities. The facts above are represented in SBVR Structured English. Below is the TOIF XMI representation of these facts.

```
<fact xmi:type="toif:Project" xmi:id="pr1">
  <name name="TOIF"/>
  <description text="Flying Gizo Assessment using TOIF"/>
</fact>

<fact xmi:type="toif:Organization" xmi:id="o1">
  <name name="McDuck and Sons Assurance"/>
</fact>

<fact xmi:type="toif:Organization" xmi:id="o2">
  <name name="Gizmo Mfg"/>
</fact>

<fact xmi:type="toif:Person" xmi:id="p1">
  <name name="Scroodge"/>
</fact>

<fact xmi:type="toif:Role" xmi:id="r1">
  <name name="Prime Investigator"/>
</fact>

<fact xmi:type="toif:Role" xmi:id="r2">
  <name name="CTO"/>
</fact>

<fact xmi:type="toif:Role" xmi:id="r3">
  <name name="Third-party Assessor"/>
</fact>
```

```

<fact xmi:type="toif:PersonIsInvolvedInProjectAsRole" project="pr1" role="r1"
person="p1"/>

<fact xmi:type="toif:PersonIsEmployedByOrganizationAsRole" role="r2"
person="p1" organization="o1"/>

<fact xmi:type="toif:ProjectIsOwnedByOrganization" project="pr1"
organization="o2"/>
<fact xmi:type="toif:OrganizationIsInvolvedInProjectAsRole" organization="o1"
project="pr1" role="r3"/>

```

10.2.3.2 ProjectIsOwnedByOrganization Class

ProjectIsOwnedByOrganization class represents the verb concept “Project is owned by Organization”.

Superclass

ProjectFact

Associations

project:Project[1]	Project that is the subject of the clause
organization: Organization[1]	Organization that owns the Project

Example

See 10.2.3.1

10.2.3.3 OrganizationIsInvolvedInProjectAsRole Class

OrganizationIsInvolvedInProjectAsRole class represents the verb concept “Organization is involved in Project as Role”.

Superclass

ProjectFact

Associations

organization: Organization[1]	Organization that is the subject of the clause
project:Project[1]	Project in which Organization is involved in
role: Role[1]	Role in which the Organization is involved in the Project

Example

See 10.2.3.1

10.2.3.4 PersonsInvolvedInProjectAsRole Class

PersonsInvolvedInProjectAsRole class represents the verb concept “Person is involved in Project as Role”.

Superclass

ProjectFact

Associations

person: Person[1]	Person that is the subject of the clause
project:Project[1]	Project is which the Person is involved
role: Role[1]	Role in which the Person is involved in the Project

Example

See 10.2.3.1

10.2.4 Tools Class Diagram

This section describes the UML representation of the Tool concept, describes several concrete tools that are essential to the TOIF Ecosystem and the corresponding facts.

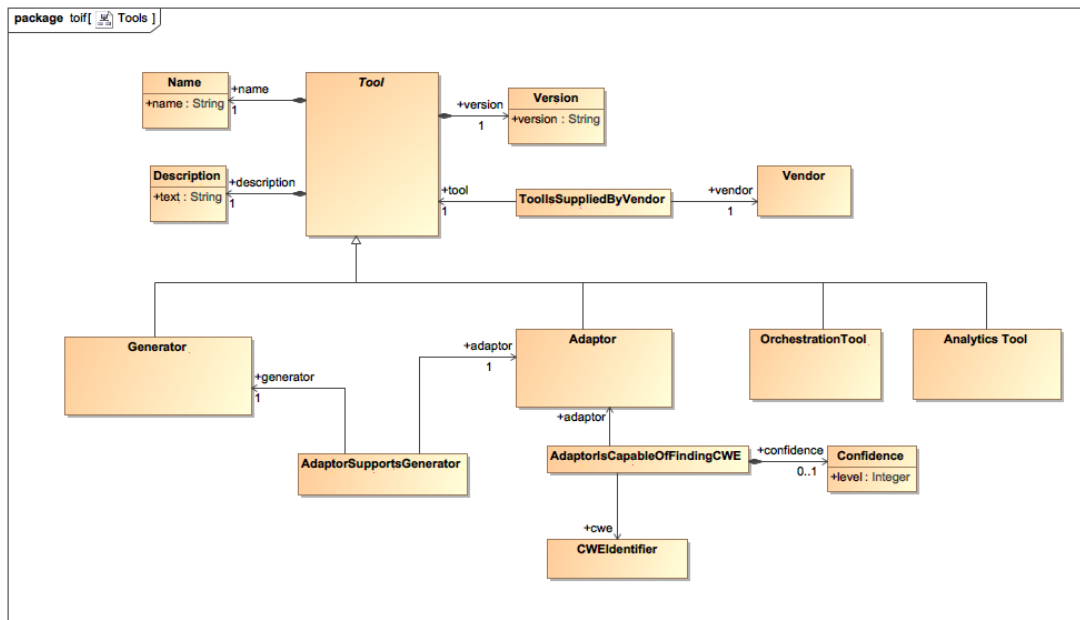


Figure 15. UML class diagram Tools

10.2.4.1 Tool Class (abstract)

Tool class represents the Tool concept.

Superclass

HousekeepingEntity

Associations

name: Name[1]	Owned attribute that specifies the name of the Tool
description: Description[1]	Owned attribute that provides an informal text description of the Tool
version: Version[1]	Owned attribute that specifies the Version of the Tool

Example

10.2.4.2 ToolsSuppliedByVendor Class

ToolsSuppliedByVendor class represents the verb concept “Tool is supported by Vendor”.

Superclass

ToolFact

Associations

tool: Tool[1]	Tool that is the subject of the clause
vendor: Vendor[1]	Vendor which supplies the Tool

Example

```
<fact xmi:type="toif:OrchestrationTool" xmi:id="bt02">
  <name name="Build Environment 2"/>
  <description text="xxxxxx"/>
  <version version="2.3.05B"/>
</fact>
```

```
<fact xmi:type="toif:Vendor" xmi:id="v1">
  <name name="McDuck Tool Craft"/>
</fact>
```

```
<fact xmi:type="toif:Vendor" xmi:id="o1">
  <name name="McDuck and Sons Assurance"/>
</fact>
```

```
<fact xmi:type="toif:Role" xmi:id="r1">
  <name name="Tiger Team"/>
</fact>
```

```
<fact xmi:type="toif:ToolIsSuppliedByVendor" tool="bt02" vendor="v1"/>
<fact xmi:type="toif:OrganizationIsPartOfOrganizationAsRole" organization1="v1"
organization="o1" role="r1"/>
```

10.2.4.3 Generator Class

Generator class represents the Generator concept.

Superclass

Tool

Example

See 10.1.1.1

10.2.4.4 Adaptor Class

Adaptor class represents the Adaptor concept.

Superclass

Tool

Constraints

1. Each Adaptor class must be the subject of at least one AdaptorSupportsGenerator clause
2. Each Adaptor class must be the subject of at least one AdaptorIsCapableOfFindingCWE clause
3. For each instance of Finding that is the subject of the clause FindingIsProducedByAdaptor, where A is the object is the above clause, and the Finding is the subject of the FindingIsDefinedAsCWE where X is the instance of CWEIdentifier that is the object of the above clause, A must be the subject of the clause AdaptorIsCapableOfFindingCWE where X is the object.

Example

See 10.1.1.1

10.2.4.5 OrchestrationTool Class

OrchestrationTool class represents the Orchestration Tool concept.

Superclass

Tool

Example

See 10.2.2.1

10.2.4.6 Analytics Tool Class

Analytics Tool class represents the Analytics Tool concept.

Superclass

Tool

Example

See 10.2.2.1

10.2.4.7 AdaptorSupportsGenerator Class

AdaptorSupportsGenerator class represents the verb concept “Adaptor supports Generator”.

Superclass

ToolFact

Constraints

1. Each Adaptor class must be the subject of at least one AdaptorSupportsGenerator clause

Associations

adaptor:Adaptor[1]	Adaptor that is the subject of the clause
generator: Generator[1]	Generator that is supported by the Adaptor

Example

```
<fact xmi:type="toif:Generator" xmi:id="rats_2.3">
  <name name="RATS"/>
  <description text="xxxxxx"/>
  <version version="2.3"/>
</fact>
<fact xmi:type="toif:Adaptor" xmi:id="rats-toif-adaptor_1.1">
  <name name="RATS-TOIF"/>
  <description text="xxxxxx"/>
  <version version="1.1"/>
</fact>

  <fact xmi:type="toif:AdaptorSupportsGenerator" adaptor="rats-toif-adaptor_1.1"
generator="rats_2.3"/>
```

10.2.4.8 AdaptorIsCapableOfFindingCWE Class

AdaptorIsCapableOfFindingCWE class represents the verb concept “Adaptor is capable of finding CWE”.

Superclass

ToolFact

Associations

adaptor:Adaptor[1]	Adaptor that is the subject of the clause
cwe: CWEIdentifier[1]	CWE Identifier that specifies the type of weakness findings that the Adaptor is capable of producing
Confidence:Confidence[0..1]	Owned attribute that specifies the confidence that the Consumer of TOIF findings has in findings of the CWE of the clause by the tool of the clause

Constraints

1. Each Adaptor class must be the subject of at least one AdaptorIsCapableOfFindingCWE clause
2. For each instance of Finding that is the subject of the clause FindingsProducedByAdaptor, where A is the object is the above clause, and the Finding is the subject of the FindingsDefinedAsCWE where X is the instance of CWEIdentifier that is the object of the above clause, A must be the subject of the clause AdaptorIsCapableOfFindingCWE where X is the object.

Example

```
<fact xmi:type="toif:Adaptor" xmi:id="rats-toif-adaptor_1.1">
  <name name="RATS-TOIF"/>
  <description text="xxxxxx"/>
  <version version="1.1"/>
</fact>

  <fact xmi:type="toif:AdaptorIsCapableOfFindingCWE" adaptor="rats-toif-
adaptor_1.1" cwe="CWE-121"/>
  <fact xmi:type="toif:AdaptorIsCapableOfFindingCWE" adaptor="rats-toif-
adaptor_1.1" cwe="CWE-122"/>
  <fact xmi:type="toif:AdaptorIsCapableOfFindingCWE" adaptor="rats-toif-
adaptor_1.1" cwe="CWE-124"/>
  <fact xmi:type="toif:AdaptorIsCapableOfFindingCWE" adaptor="rats-toif-
adaptor_1.1" cwe="CWE-124">
    <confidence xmi:type="toif:Confidence" level=20/>
  </fact>
```

10.2.5 Organization Class Diagram

This section describes the UML representation of the Organization concept and the corresponding facts.

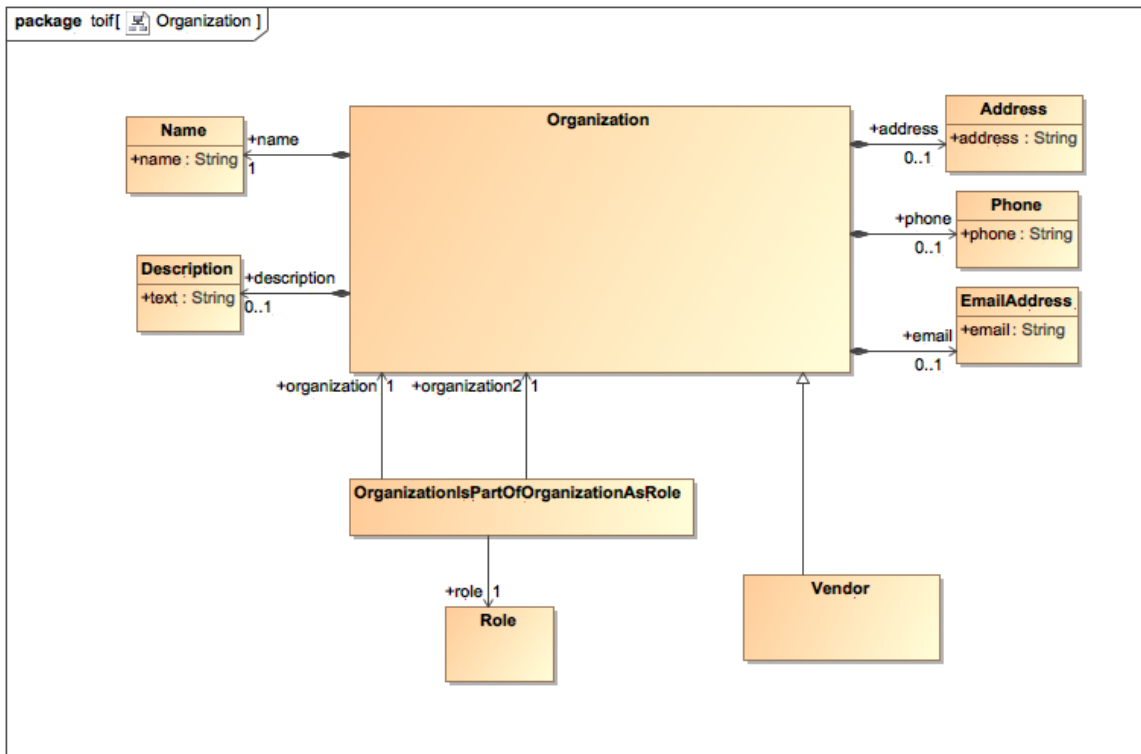


Figure 16. UML class diagram Organization

10.2.5.1 Organization Class

Organization class represents the Organization concept.

Superclass

HousekeepingEntity

Associations

name: Name[1]	Owned attribute that specifies the name of the Organization
description: Description[0..1]	Owned attribute that provides an informal text description of the Organization
address: Address[0..1]	Owned attribute that specifies the Address of the Organization
phone: Phone[0..1]	Owned attribute that specifies the contact Phone number for the Organization
email: EmailAddress[0..1]	Owned attribute that specifies the contact Email Address for the Organization

Example

See 10.2.3.1

10.2.5.2 Vendor Class

Vendor class represents the Vendor concept.

Superclass

Organization

Example

See 10.2.4.2

10.2.5.3 OrganizationIsPartOfOrganizationAsRole Class

OrganizationIsPartOfOrganizationAsRole class represents the verb concept “Organization₁ is part of Organization₂ as Role”.

Superclass

ProjectFact

Associations

organization1:Organization[1]	Organization-1 that is the subject of the clause
organization2: Organization[1]	Organization-2 of which Organization-1 is a part
role: Role[1]	Role in which Organization-1 participates in Organization-2

Example

See 10.2.4.2

10.2.6 Person Class Diagram

This section describes the UML representation of the Person concept and the corresponding facts.

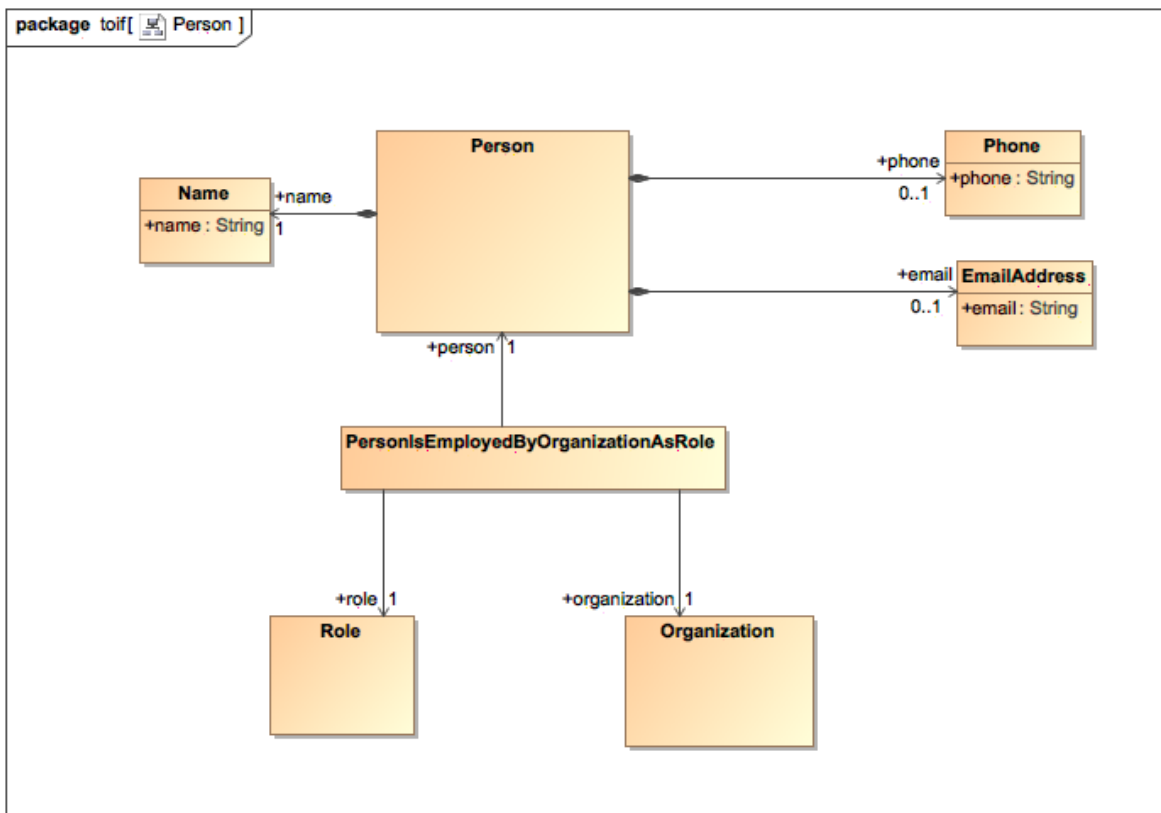


Figure 17. UML class diagram Person

10.2.6.1 Person Class

Person class represents the Person concept.

Superclass

HousekeepingEntity

Associations

name:Name[1]	Owned attribute that specifies the name of the Person
phone: Phone[0..1]	Owned attribute that provides a contact Phone number for the Person
email: EmailAddress[0..1]	Owned attribute that provides a contact Email Address for the Person

Example

See 10.2.3.1

10.2.6.2 PersonsEmployedByOrganizationAsRole Class

PersonsEmployedByOrganizationAsRole class represents the verb concept “Person is employed by Organization as Role”.

Superclass

ProjectFact

Associations

person:Person[1]	Person that is the subject of the clause
organization: Organization[1]	Organization which employs Person
role: Role[1]	Role in which the Person is employed by the Organization

Example

See 10.2.3.1

10.2.7 Role Class Diagram

This section describes the UML representation of the Role concept and the corresponding facts.

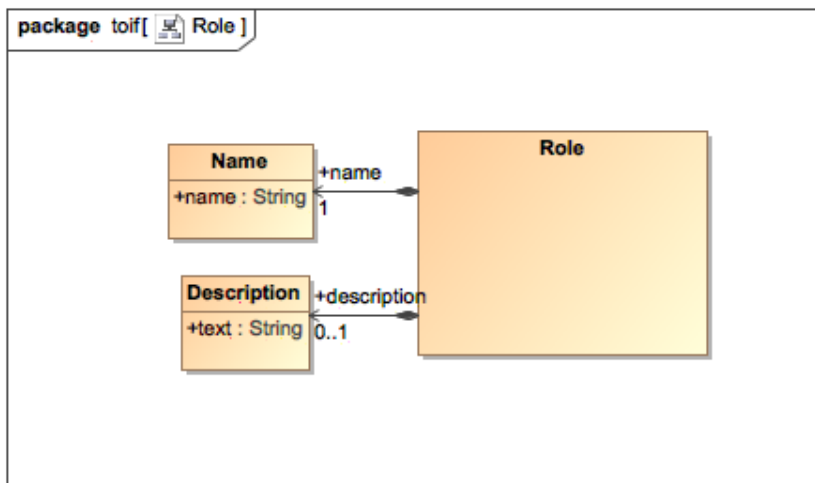


Figure 18. UML class diagram Role

10.2.7.1 Role Class

Role class represents the Role concept. Roles are useful in enterprise context to manage multiple TOIF Builds and multiple TOIF Projects performed by multiple people and over organizations (for example, departments and contractors). Only a name and an uninterpreted description is associated with a role. TOIF does not link Role to any of the standard taxonomy, although an enterprise adopting TOIF may choose to interpret TOIF Roles more formally by referencing some standard taxonomy in the description.

Superclass

HousekeepingEntity

Associations

name:Name[1]

Owned attribute that specifies the name of the Role

description: Description[0..1]

Owned attribute that provides an informal text description of the role

Example

See 10.2.3.1

10.3 The fact-oriented structure of the TOIF XML

This section presents 13 UML diagrams that describes the physical structure of the TOIF XML.

The Figure 19. UML class diagram Abstract Structure represent the physical structure of the TOIF XML. All basic entities are defined as subclasses of class BasicEntity. All logical facts are defined as subclasses of class Fact. Similarly, all housekeeping entities are defined as subclasses of class HousekeepingEntity and all housekeeping facts are defined as subclasses of class Fact. All records are defined as subclasses of class Record.

Classes BasicEntity and HousekeepingEntity are both subclasses of class Entity.

This allows the TOIFSegment class to own an ordered list of both Entities, Facts and Records. Attributes are owned by the entities to which they belong, so they are not owned directly by the TOIFSegment.

Classes Entity, Fact and Record are subclasses of class TOIFElements.

Classes TOIFElements, TOIFSegment and Attribute are subclasses of class Element.

Class Element defines and XML attribute “xmi:id” which is used in the TOIF XML to reference elements. This makes all Entity, Fact, Attribute, Record and TOIFSegment referenceable.

The rest of this section presents 8 UML diagrams that arrange the basic and housekeeping facts, entities and attributes as subclasses of the 3 main structure elements – Entities, Facts and Attributes.

10.3.1 Abstract Structure Class Diagram

This section describes the UML representation of the TOIFSegment concept and the corresponding facts.

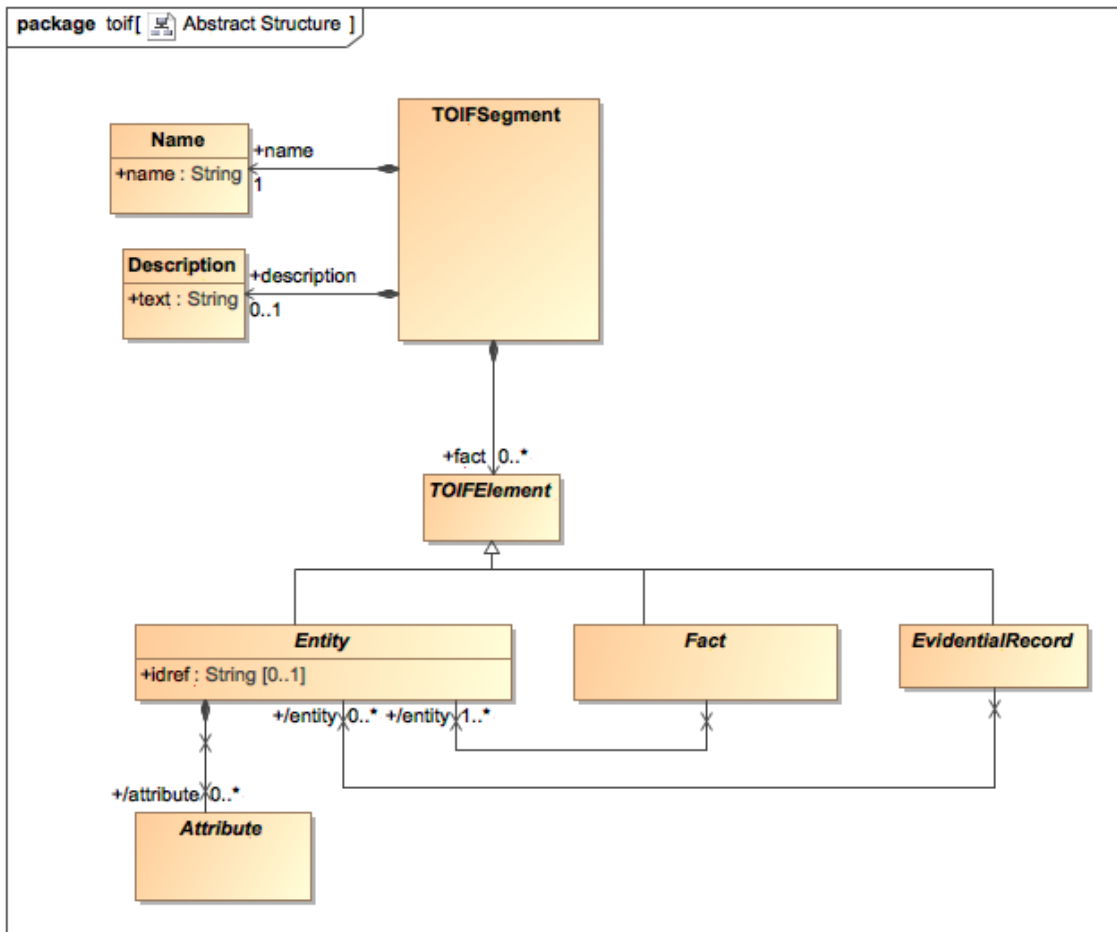


Figure 19. UML class diagram Abstract Structure

10.3.1.1 TOIFSegment Class

TOIFSegment class represents the TOIFSegment concept.

Superclass

Element

Associations

name:Name[1]	Owned attribute that specifies the name of the Segment
description: Description[0..1]	Owned attribute that provides an informal text description of the Segment
fact: TOIFElement[0..*]	Owned TOIFElement that represent the Facts contained in the Segment

10.3.1.2 TOIFElement Class (abstract)

Superclass

Element

10.3.1.3 Entity Class (abstract)

Entity class represents the Entity concept.

Superclass

TOIFElement

Attributes

idref:String[0..1]	Optional attribute reserved for alternative reference schemas for TOIF Entities
--------------------	---

Associations

/attribute:Attribute[0..*]	Owned Attribute of this Entity. This is a derived property. The actual owned attributes are specified by the subclasses of the Entity class
----------------------------	---

10.3.1.4 Fact Class (abstract)

Fact class represents the Fact concept.

Superclass

TOIFElement

Associations

/entity:Entity[1..*]

Entity that are referenced by the Fact (clause). Each clause has a subject represented by an Entity, and one or mode objects. This is a derived property. The actual are specified by the subclasses of the Fact class

10.3.1.5 Attribute Class (abstract)

Attribute class represents the Attribute concept.

Superclass

Element

10.3.1.6 EvidentialRecord Class (abstract)

EvidentialRecord class represents the Evidential Record concept.

Superclass

TOIFElement

Associations

/entity:Entity[1..*]

Entity that are referenced by the Evidential Record. Each references one or mode objects. This is a derived property. The actual are specified by the subclasses of the Fact class. Concrete subclasses of Evidential Record may define owned attributes.

10.3.2 Abstract Types Class Diagram

This section describes the top class of the TOIF metamodel and its direct subclasses.

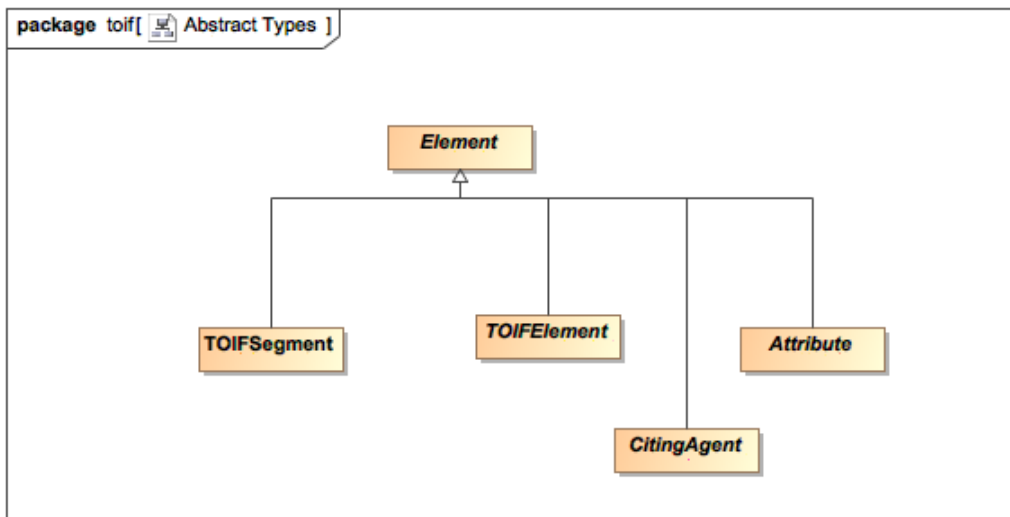


Figure 20. UML class diagram Abstract Types

10.3.2.1 Element Class (abstract)

Superclass

MOF Element

10.3.3 Basic Entities Class Diagram

This section describes the class hierarchy of the basic entities of the TOIF specification.

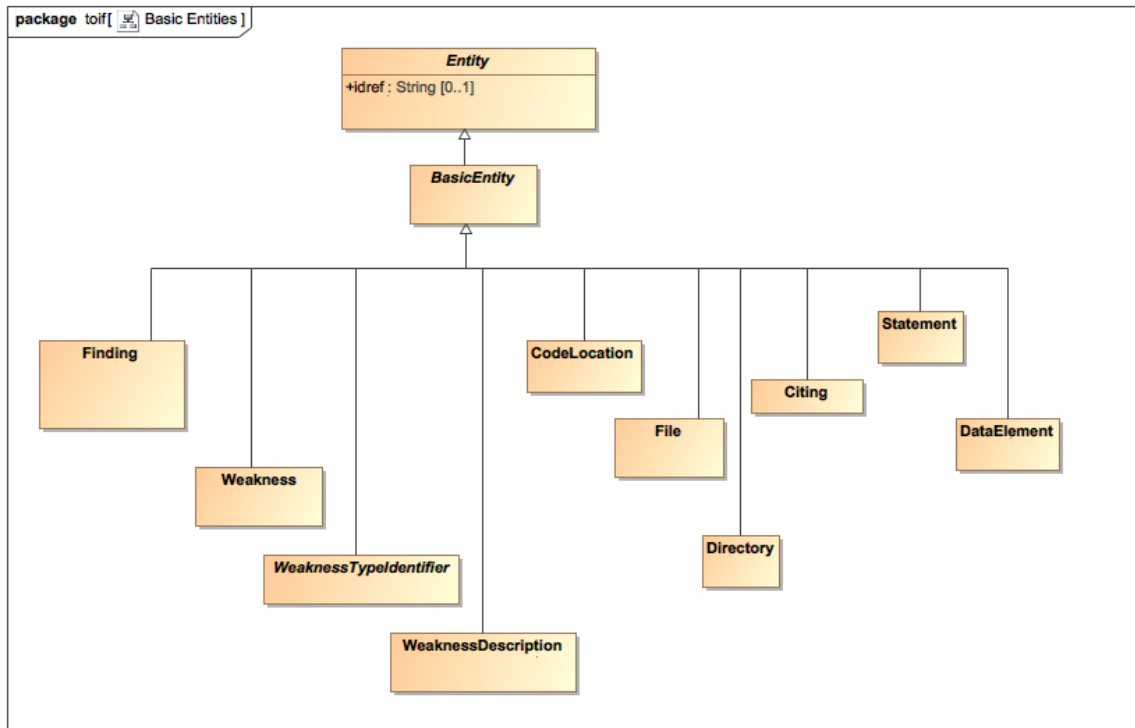


Figure 21. UML class diagram Basic entities

10.3.3.1 BasicEntity Class (abstract)

Superclass

Entity

10.3.4 Basic Facts Class Diagrams

This section describes the class hierarchy of the basic facts of the TOIF specification. Basic facts are grouped into the following four categories: Finding Facts, Weakness Facts, Code Location Facts and Semantic Facts. Each category is represented by own UML diagram. These diagrams introduce 4 abstract superclasses and provide the subclass relationships to the subclasses, defined earlier in Section 10.3.1

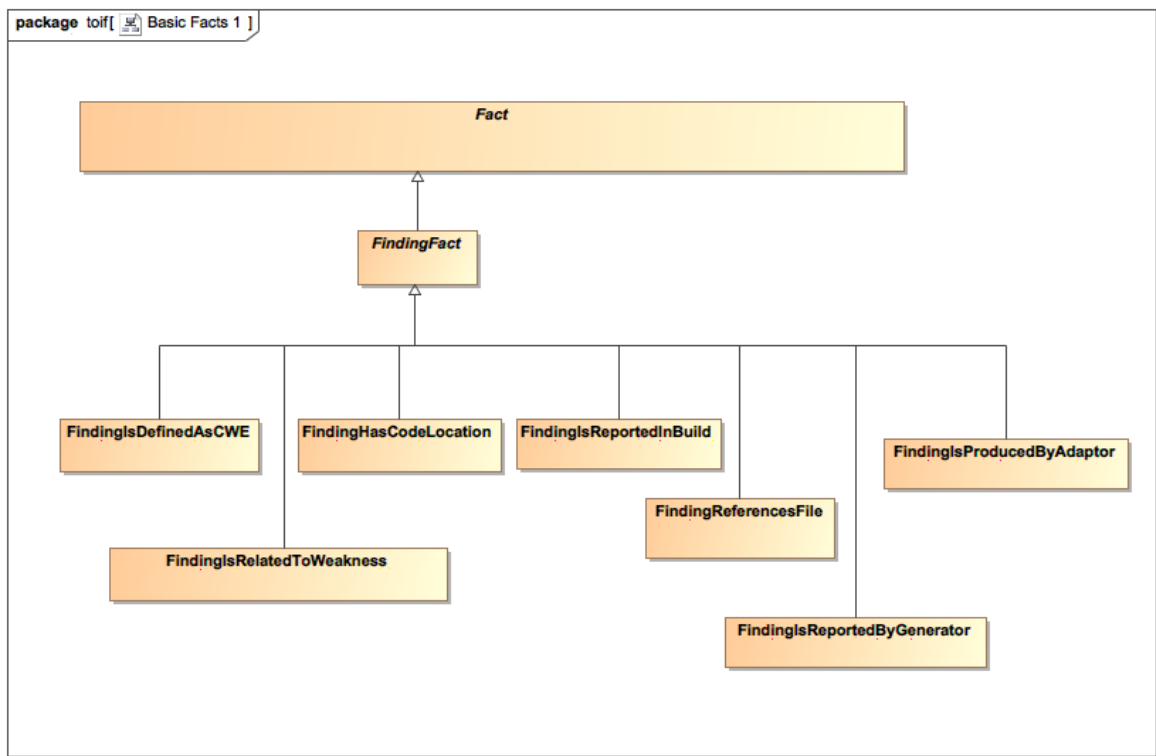


Figure 22. UML class diagram Basic Facts 1

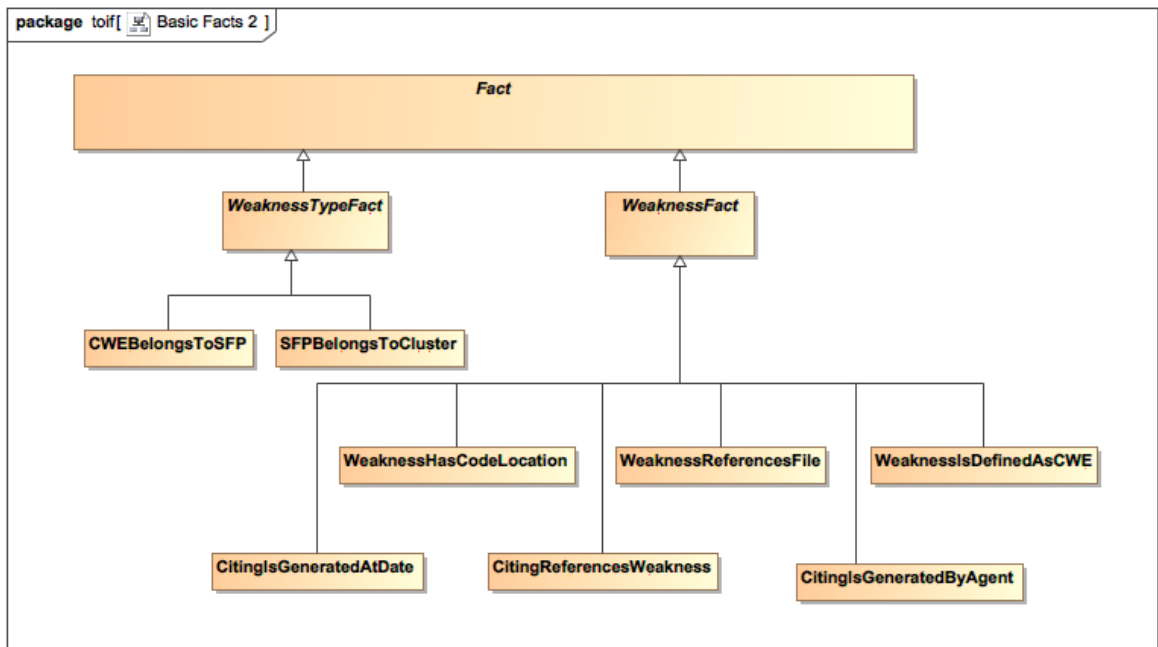


Figure 23. UML class diagram Basic Facts 2

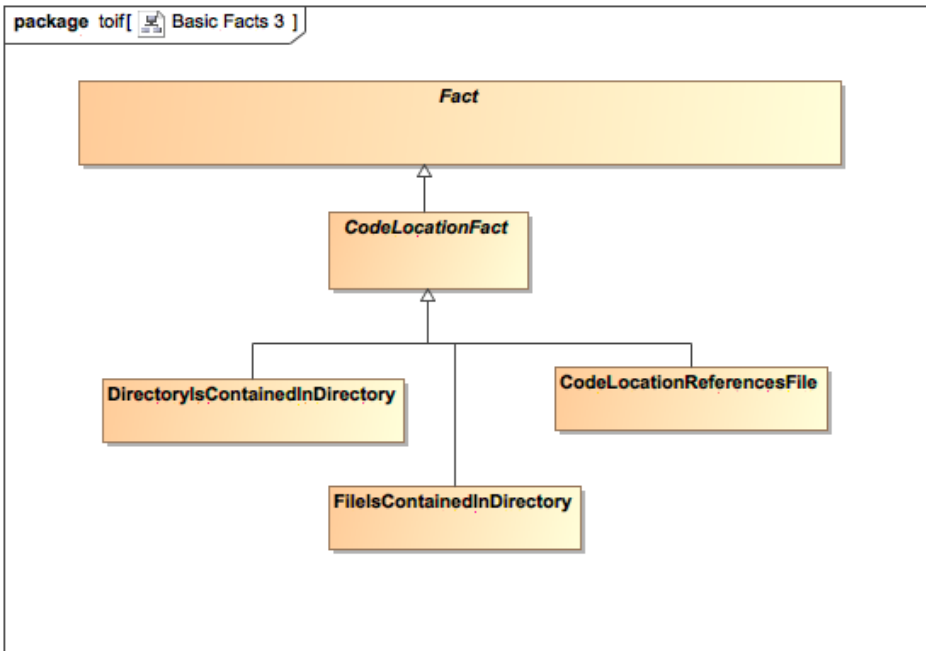


Figure 24. UML class diagram Basic Facts 3

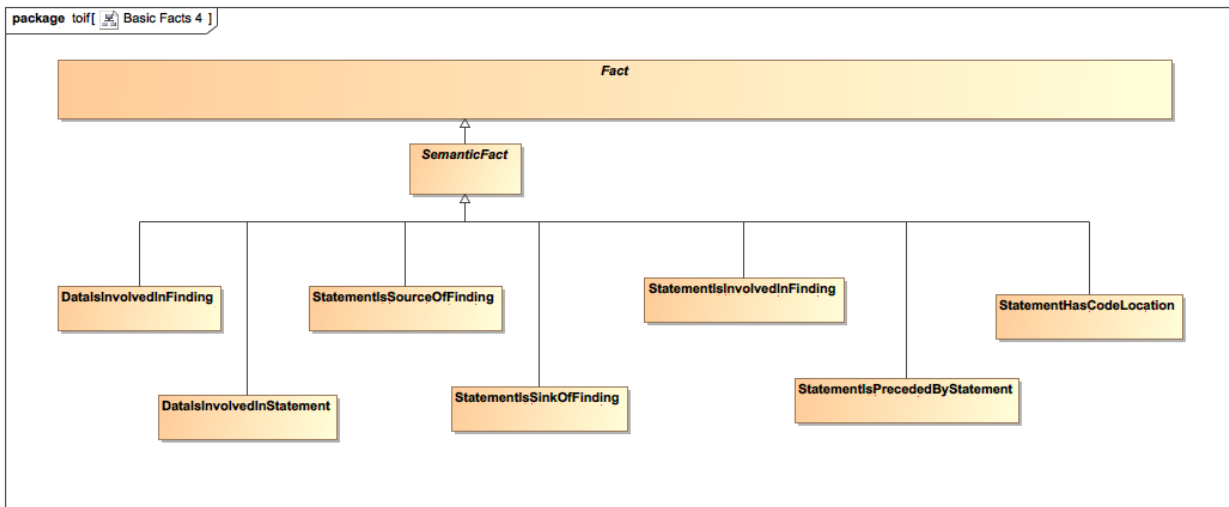


Figure 25. UML class diagram Basic Facts 4

10.3.4.1 FindingFact Class (abstract)

Superclass

Fact

10.3.4.2 WeaknessTypeFact Class (abstract)

Superclass

Fact

10.3.4.3 WeaknessFact Class (abstract)

Superclass

Fact

10.3.4.4 CodeLocationFact Class (abstract)

Superclass

Fact

10.3.4.5 SemanticFact Class (abstract)

Superclass

Fact

10.3.5 Basic Attributes Class Diagram

This section describes the class hierarchy of the attributes of the TOIF specification. More attributes are defined in the Housekeeping Attributes Class Diagram. TOIF does not make distinction between Basic Attributes and Housekeeping Attributes. The two diagrams are split mostly for the presentation purposes, as the attributes described in the Basic Attribute diagram are used both by the Basic as well as the Housekeeping entities.

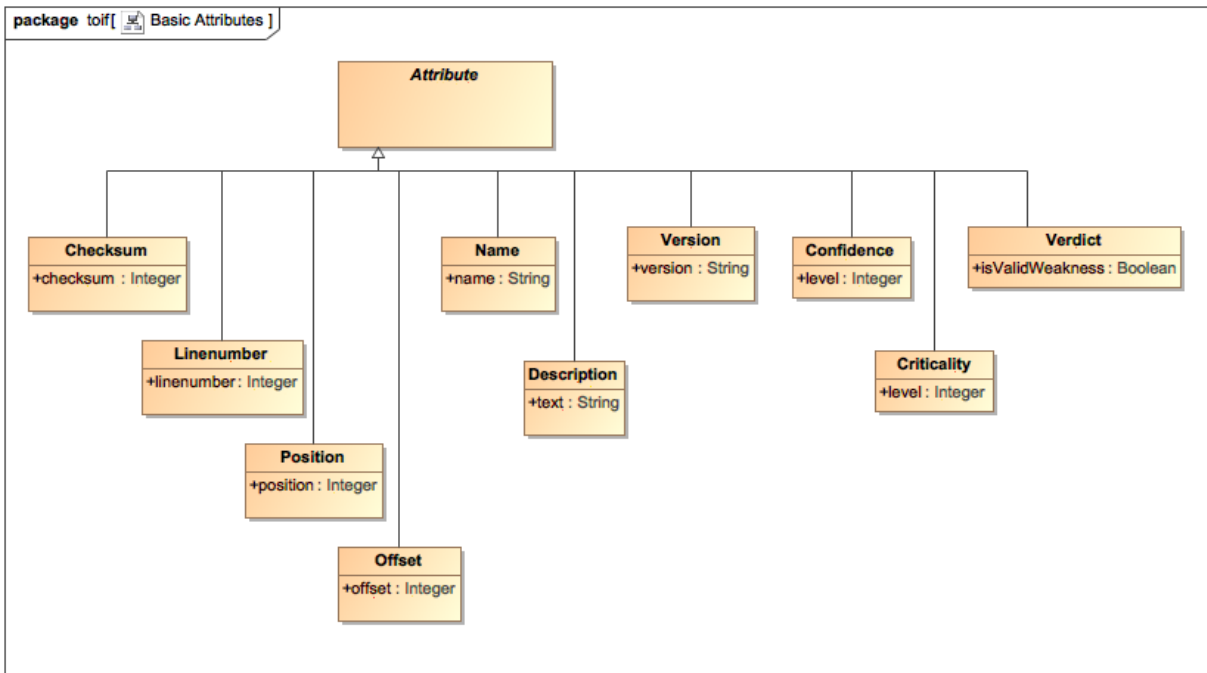


Figure 26. UML class diagram Basic Attributes

10.3.5.1 Offset Class

Superclass

Attribute

Attributes

offset:Integer[1] Offset in a binary image file

10.3.5.2 Checksum Class

Superclass

Attribute

Attributes

checksum:Integer[1] MD5 Checksum of a file

10.3.5.3 Linenumber Class

Superclass

Attribute

Attributes

linenumber:Integer[1] Linenumber in a text File

10.3.5.4 Position Class

Superclass

Attribute

Attributes

position:Integer[1] Position in a line in a text File

10.3.5.5 Name Class

Superclass

Attribute

Attributes

name:String[1] Name of an Entity

10.3.5.6 Version Class

Superclass

Attribute

Attributes

version:String[1] Version of an Thing that may involve multiple versions

10.3.5.7 Description Class

Superclass

Attribute

Attributes

text:String[1] Informal text description

10.3.5.8 Confidence Class

Superclass

Attribute

Attributes

level:Integer[1] Confidence level (0..100) as percent

10.3.5.9 Criticality Class

Superclass

Attribute

Attributes

level:Integer[1] Criticality level (0..100) as percent

10.3.5.10 Verdict Class

Superclass

Attribute

Attributes

isValidWeakness:Boolean[1] True or false

10.3.6 Housekeeping Entities Class Diagram

This section describes the class hierarchy of the housekeeping entities of the TOIF specification.

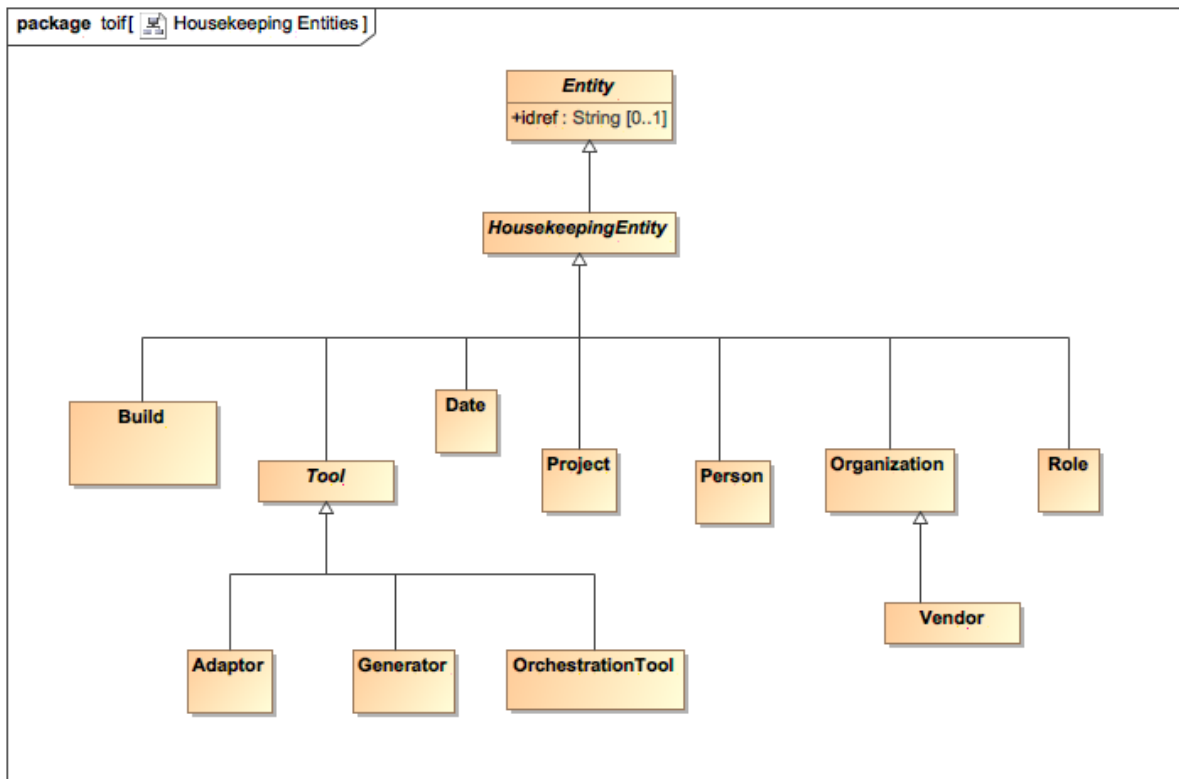


Figure 27. UML class diagram Housekeeping entities

10.3.6.1 HousekeepingEntity Class (abstract)

Superclass

Entity

10.3.7 Housekeeping Facts Class Diagrams

This section describes the class hierarchy of the housekeeping facts of the TOIF specification. Housekeeping facts are grouped into the following three categories: Tool Facts, Build Facts and Project Facts. Each category is represented by own UML diagram. These diagrams introduce 3 abstract superclasses and provide the subclass relationships to the subclasses, defined earlier in Section 10.3.2

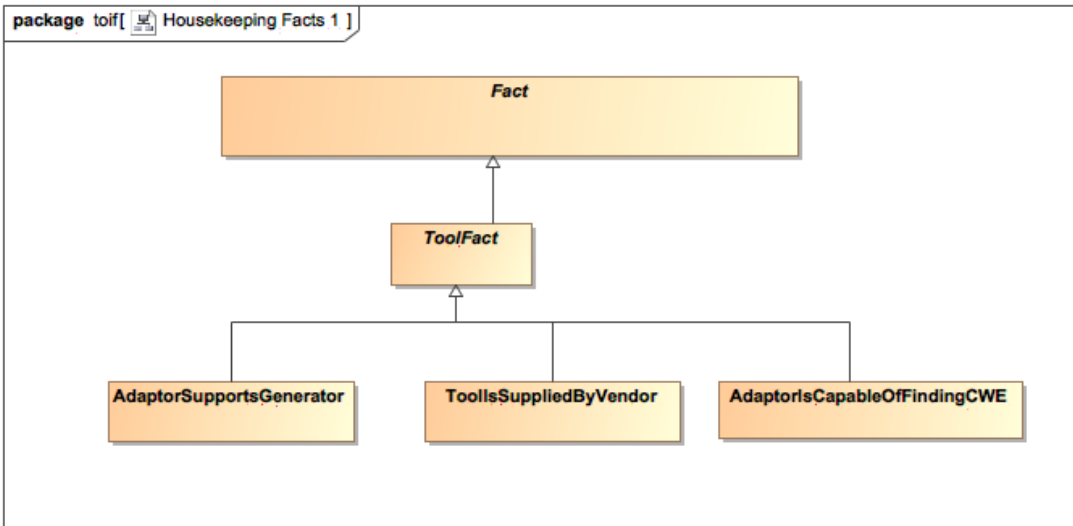


Figure 28. UML class diagram Housekeeping Facts 1

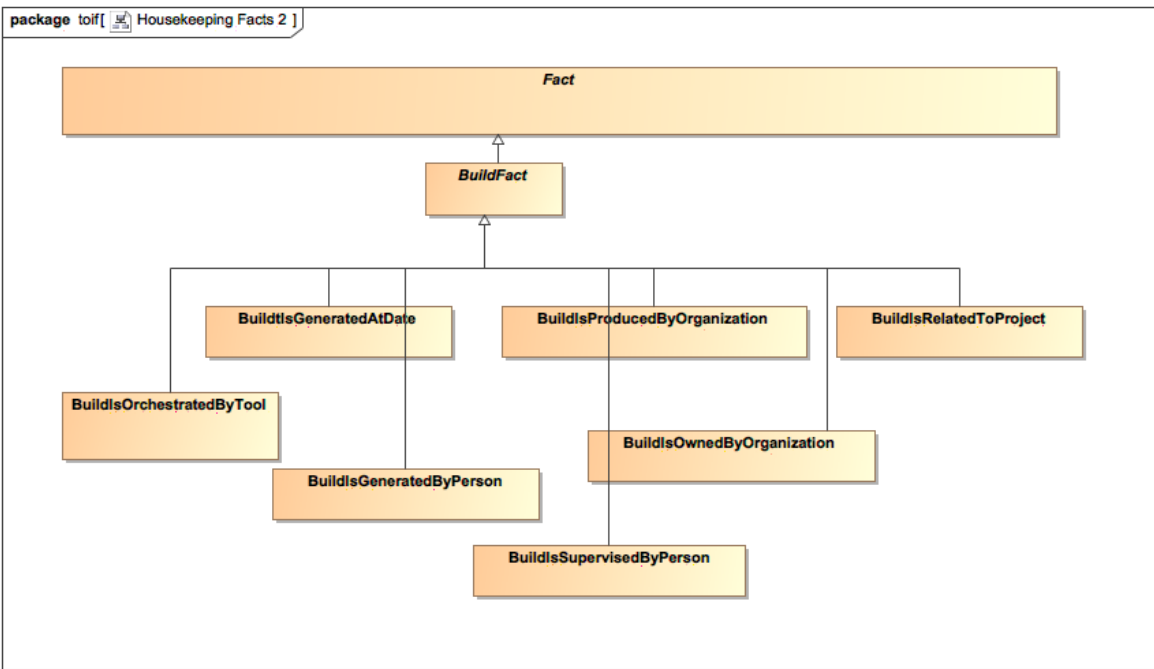


Figure 29. UML class diagram Housekeeping Facts 2

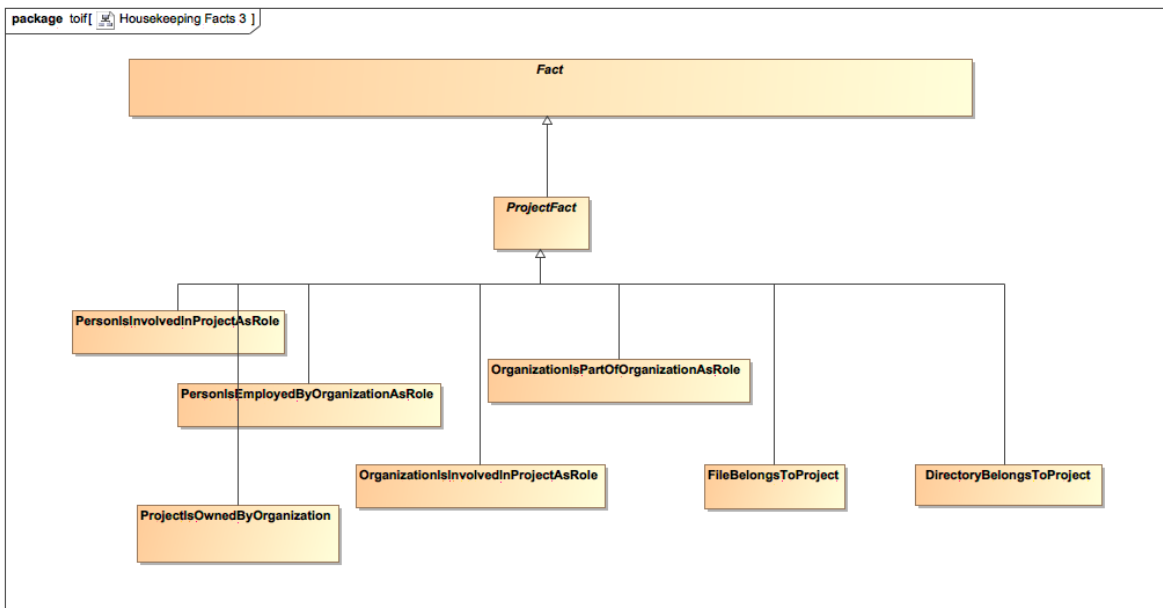


Figure 30. UML class diagram Housekeeping Facts 3

10.3.7.1 ToolFact Class (abstract)

Superclass

Fact

10.3.7.2 BuildFact Class (abstract)

Superclass

Fact

10.3.7.3 ProjectFact Class (abstract)

Superclass

Fact

10.3.8 Housekeeping Attributes Class Diagram

This section continues with the description of the class hierarchy of the attributes of the TOIF specification. More attributes are defined in the Basic Attributes Class Diagram. TOIF does not make distinction between Basic Attributes and Housekeeping Attributes, the two diagrams are split mostly for the presentation purposes.

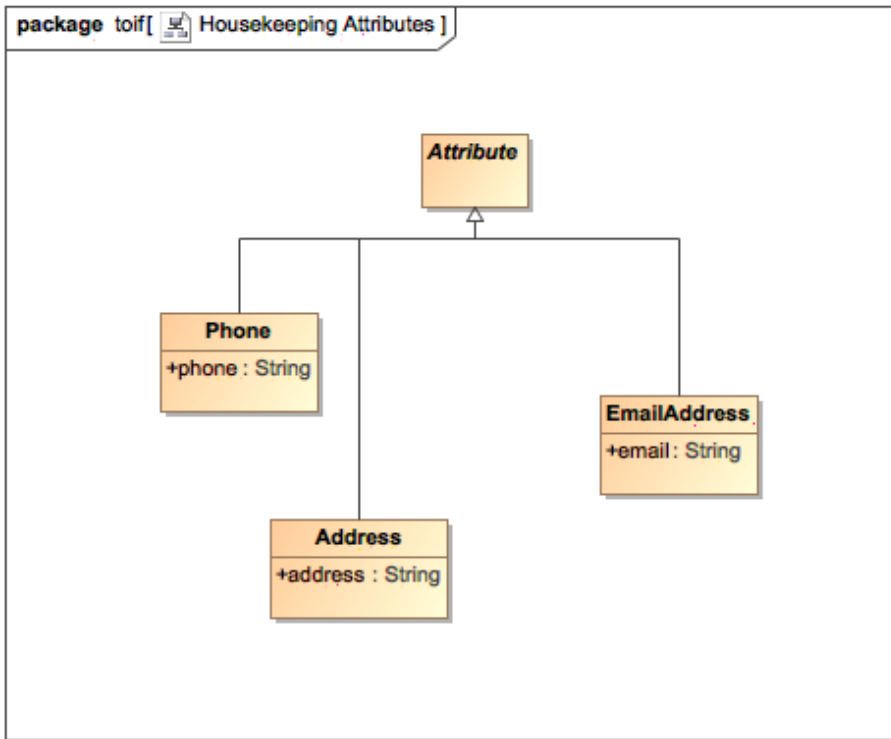


Figure 31. UML class diagram Housekeeping Attributes

10.3.8.1 Phone Class

Superclass

Attribute

Attributes

phone:String[1] Phone number

10.3.8.2 Address Class

Superclass

Attribute

Attributes

address:String[1] Postal Address of some Thing

10.3.8.3 EmailAddress Class

Superclass

Attribute

Attributes

email:String[1] Electornic Mail Address of some Thing

10.4 Evidential Records in TOIF XML

This section presents a single UML diagram that describes evidential records defined in TOIF XML.

10.4.1 EvidentialRecord Class Diagram

This section describes the UML representation of the BuildRecord, CompileRecord and GeneratorRecord records.

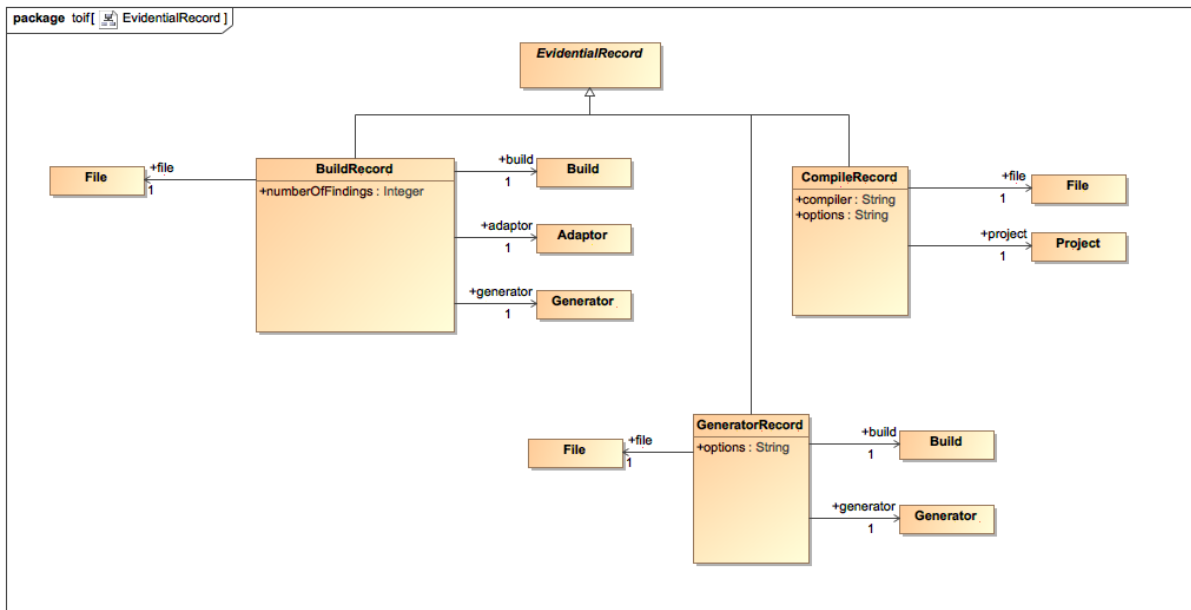


Figure 32 UML class diagram EvidentialRecord

10.4.1.1 BuildRecord Class

Superclass

Record

Attributes

numberOfFindings:Integer[1] Record of the number of findings produced by Adaptor that was processing the output of the Generator processing File in Build. This number can be equal to zero, which indicates the absence of findings, but confirms the fact that File was processed by Generator.

Associations

file:File[1] File of the BuildRecord

adaptor:Adaptor[1] Adaptor of the BuildRecord

generator:Generator[1] Generator of the BuildRecord

build:Build[1] Build of the BuildRecord

Example

```
<fact xmi:type="toif:BuildRecord" file="f10" adaptor="rats-toif-adaptor_1.1"
cwe="CWE-121" generator="rats" build="b10" numberOfFindings=0/>
<fact xmi:type="toif:BuildRecord" file="f11" adaptor="rats-toif-adaptor_1.1"
cwe="CWE-121" generator="rats" build="b10" numberOfFindings=10/>
```

10.4.1.2 CompileRecord Class

Superclass

Record

Attributes

compiler:String[1] Name of the compiler that was used to compile File in Project.

options:String[1] Command line options that were used by the compiler to compile File in Project

Associations

file:File[1] File of the CompileRecord

project:Project[1] Project of the CompileRecord

Example

```
<fact xmi:type="toif:CompileRecord" file="f10" project="pr1" compiler="gcc"
options="-wall -I /usr/include" />
```

10.4.1.3 GeneratorRecord Class

Superclass

Record

Attributes

options:String[1] Command line options that were used by the Generator to process File in Build

Associations

file:File[1] File of the GeneratorRecord

build:Build[1] Build of the GeneratorRecord

generator:Generator[1] Generator of the GeneratorRecord

Example

```
<fact xmi:type="toif:GeneratorRecord" file="f10" build="b10" compiler="gcc"
options="-wall -I /usr/include" />
<fact xmi:type="toif:GeneratorRecord" file="f10" build="b11" compiler="gcc"
options="-wall -I /usr/include" -I /test/sonic-boon/stub />
```