オブジェクト指向開発論

2020年7月18日 土曜

海谷 治彦

目次

パターン その2

- パターンの分類
  - デザイン以外のパターン
- パターンと品質要求
  - セキュリティパターン: 例題として

復習

パターンの分類

- デザインパターンの分類
  - 代表的なパターンを，その範囲と利用目的の二軸で分類している.
- 抽象度に基づく分類
  - アーキテクチャ，要求分析，デザイン，コーディング等それぞれのパターン.
- 品質特性に関する分類
  - セキュリティ，ユーザビリティ等

復習

デザインパターンの分類

		目的		
		生成	構造	振る舞い
範囲	クラス	Factory Method	Adapter	Interpreter Template method
	インスタンス	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Proxy	Chain of responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

抽象度に基づく分類

- アーキテクチャパターン
  - システム全体の構成に関するパターン
  - MVC や REST 等.
- アナリシスパターン
  - 要求分析における分析対象業務の典型的な様式.
- デザインパターン
  - 本講義のメイン
- イデオム
  - コーディング(プログラミング)における典型的なコードの書き方.
  - コーディング規則のことではない.

品質特性

- ソフトウェアや情報システムは，それらが提供する機能に加え，期待される品質特性がある.
  - 例えば，銀行等の入出金機能は，「正確」かつ「高信頼性」(故障が無い)であることが期待される.
  - スマフォ等のソフトでは，例えば電池の持ちが良い等の「パフォーマンス」も期待される.
- これら品質特性は，往々にして，設計やアーキテクチャ決定に大きく依存する.
- パターンというのは，広い意味で，ある品質特性を担保しやすくする設計，実装の定石ともいえる.
  - 前回，紹介したデザインパターンは，「拡張性」という品質に注目しているものが多い.

品質特性とは？

- いくつかの異なる標準規格や，著書等で，いくつかの品質特性のカタログが提示されている.
- しかし，そのほとんどは大きな違いは無い.

- ISO9126
- IEEE830
- FURPS+
- NFR framework

ソフトウェア品質特性標準 – ISO 9126

- **ソフトウェア品質特性**に関する標準
- http://www.cam.hi-ho.ne.jp/adamosute/software-quality/iso9126.htm より

品質特性(quality charastristics)	品質副特性(quality subcharactristics)	主な内容
機能性(fuctionality)	合目的性(suitablity) 正確性(accuracy) 相互運用性(interoperability) 標準適合性(compliance) セキュリティ(security)	目的から求められる必要な機能の実装の度合い
信頼性(reliability)	成熟性(maturity) 障害許容性(fault tolerance) 回復性(recoverability)	機能が正常動作し続ける度合い
使用性	理解性 習得性 運用性	分かりやすさ、使いやすさの度合い
効率性	時間効率性 資源効率性	目的達成のために使用する資源の度合い
保守性	解析性 変更性 安定性 試験性	保守(改訂)作業に必要な努力の度合い
移植性	環境適用性 設置性 規格適合性 置換性	別環境へ移した際そのまま動作する度合い

ソフトウェア品質特性標準 – IEEE 830-1998

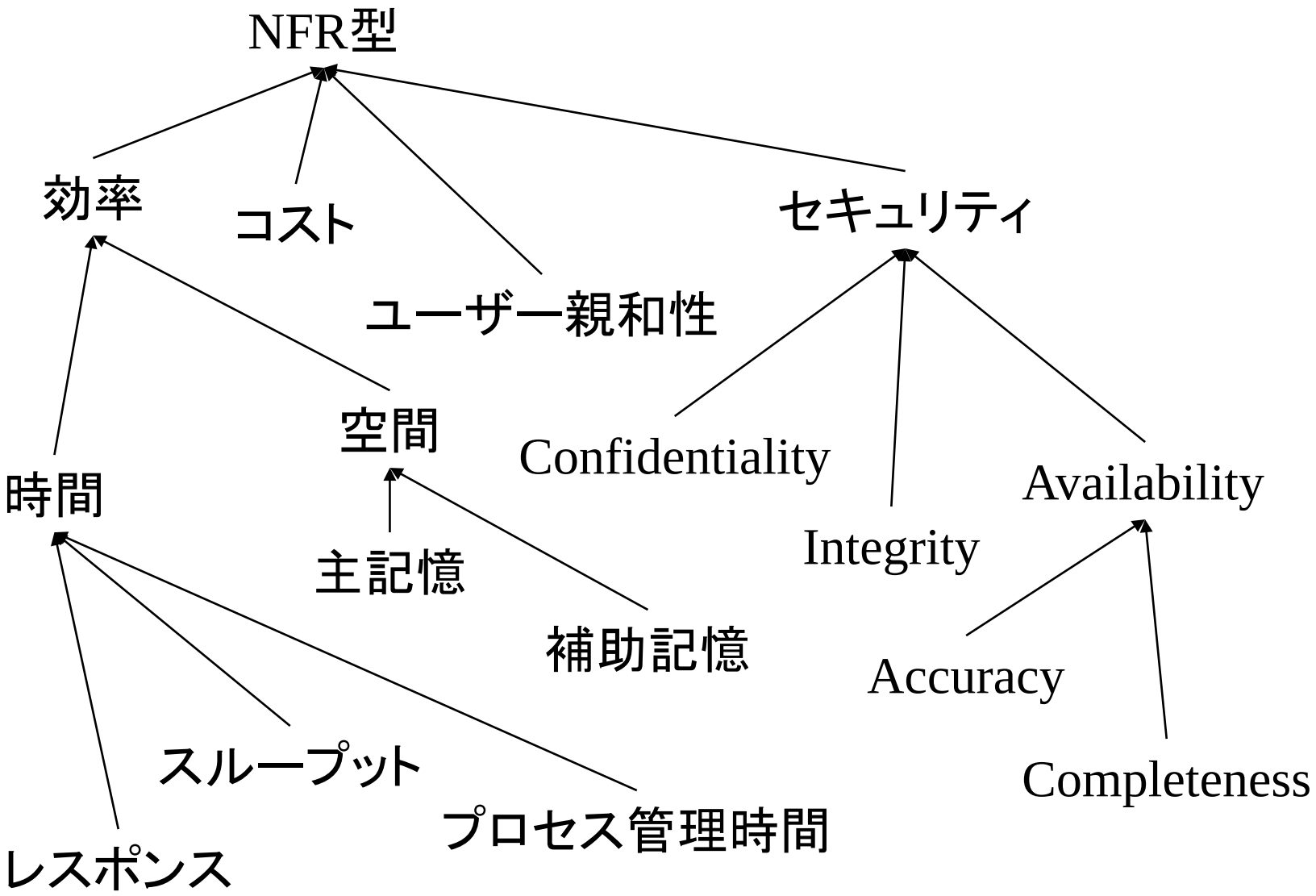
- ソフトウェア要求仕様に対する推奨プラクティス
- 「3.6 Software system attributes」にて、ソフトウェア品質特性を規定
  - 信頼性(Reliability)
  - 可用性(Availability)
  - セキュリティ(Security)
  - 保守性(Maintainability)
  - 移植性(Portability)

分析観点の例: FURPS+モデル

- Grady, “Practical Software Metrics for Project Management and Process Improvement” (Prentice-Hall, 1992)
- http://www.atmarkit.co.jp/farc/rensai/re_mgt02/re_mgt02.html より

機能 (Functionality)	機能	セキュリティ	機能要求
使いやすさ (Usability)	使いやすさ 美しさ	覚えやすさ ユーザー向け文書	機能外要求
信頼性 (Reliability)	故障の頻度／深刻さ 復旧のしやすさ	予測のしやすさ 精度 平均故障間隔	
性能 (Performance)	処理速度 メモリ使用量	スループット 応答時間	
サポートのしやすさ (Supportability)	テスト性 拡張性 適応性 保守性 互換性	構成容易性 保守容易性 インストール容易性 ローカライズ容易性 頑強性	

NFRの型


NFR型
効率
コスト
ユーザー親和性
セキュリティ
時間
空間
主記憶
補助記憶
スループット
レスポンス
プロセス管理時間
Confidentiality
Integrity
Availability
Accuracy
Completeness

セキュリティパターン

- 拡張性以外のパターンの例

- 情報システムやソフトウェアが持つ機能をセキュアにするための設計等の定石である.
- セキュア: Confidentiality, Integrity, Availability (機密が守られる，不正改竄されない，使いたい人がいつでも使える)
- 要求分析，設計，実装，テストそれぞれのセキュリティパターンがあるようだ.

種類

- 特に定まったカタログは無い.
- 研究論文での出現頻度としては以下のようなランキングとなっている. (ソースは2014以前の論文)

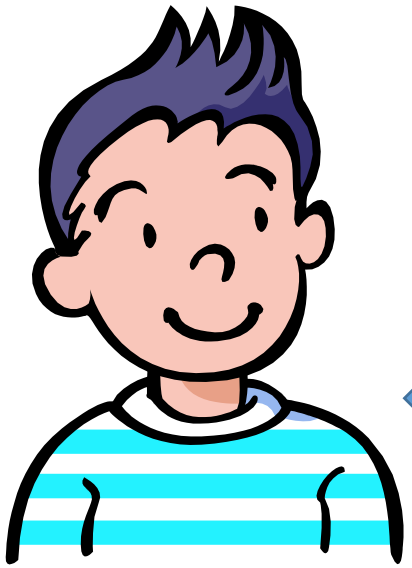
参照している論文数	パターン名
45	rbac
31	authorization
23	access control
20	authentication
18	authenticator
18	check point
16	reference monitor
14	secure logger
13	secure pipe
13	single access point
11	authentication enforcer
10	replicated system
10	abac
10	xacml

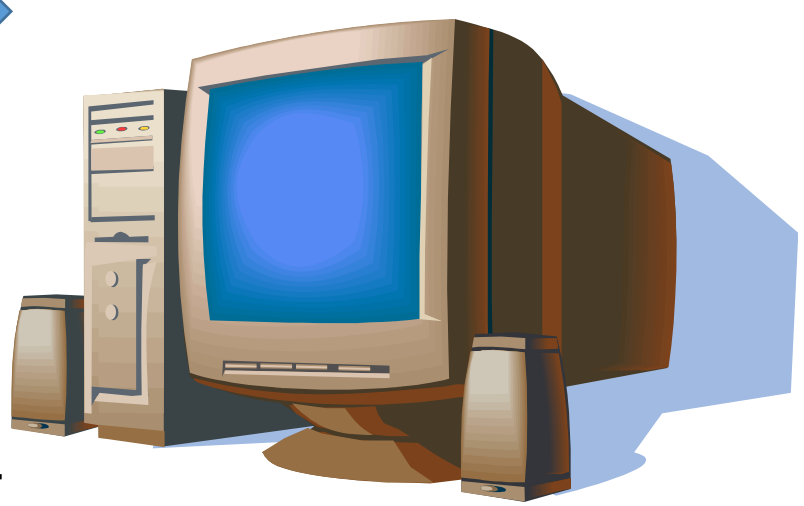
復習
OSより

コンピュータが利用できるまで

2 ホントに〇〇なん？
(Authentication)

1 私は〇〇です.
(Identification)



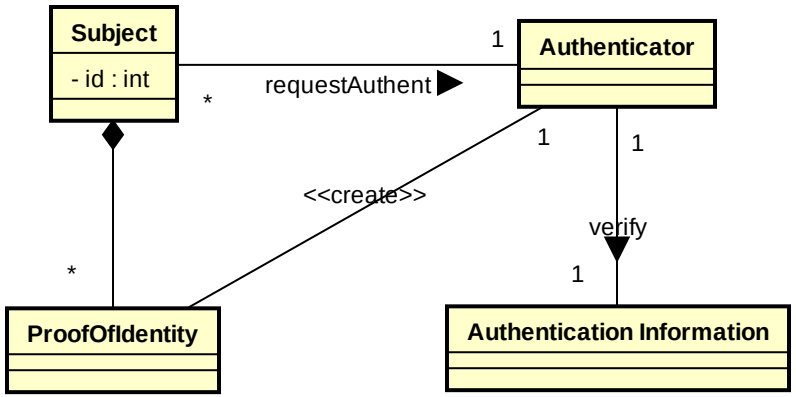


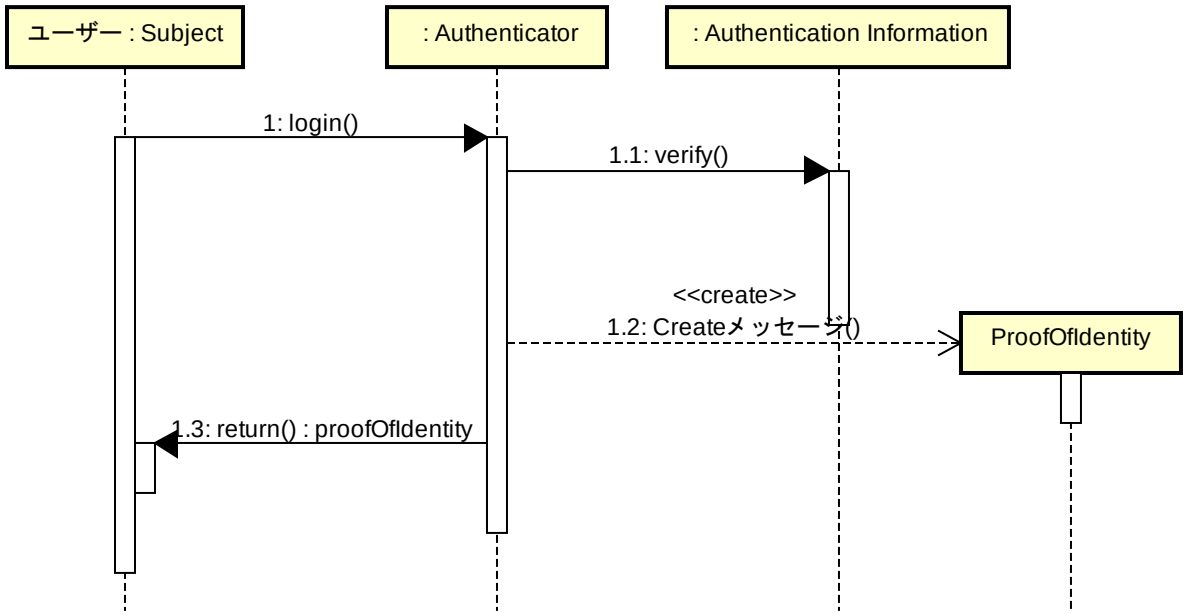
3 〇〇に許可された
操作を受け付けます.
(Authorization)

Authenticator

- ユーザー等がシステムに対して認証を行う場合のパターン.
- 悪意のある者が成り済ますのを防止する基本的な仕組み.
- 基本的に唯一の認証窓口(Authenticator)を提供することで，安全性を担保する.

パターン


Subject
- id : int
Authenticator
1
*
requestAuthent
1
1
<<create>>
verify
*
1
ProofOfIdentity
Authentication Information


ユーザー : Subject
: Authenticator
: Authentication Information
1: login()
1.1: verify()
<<create>>
1.2: Createメッセージ()
ProofOfIdentity
1.3: return() : proofOfIdentity

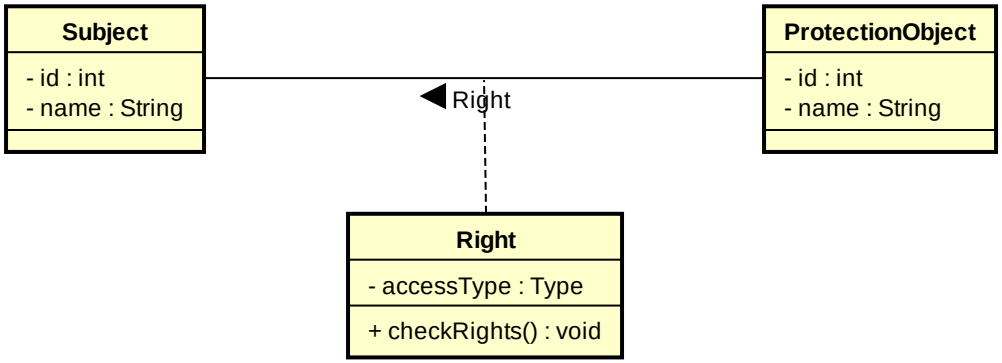
単純な Authorization

- システムの管理する個々の物(ファイル等)を，どのユーザーが，どんな風にアクセスできるかの管理.
- 通常はアクセス行列が使われる.

- Identification and Authentication (I&A)とは別の概念.
- しかし，I&Aを済ましていることを前提とする.

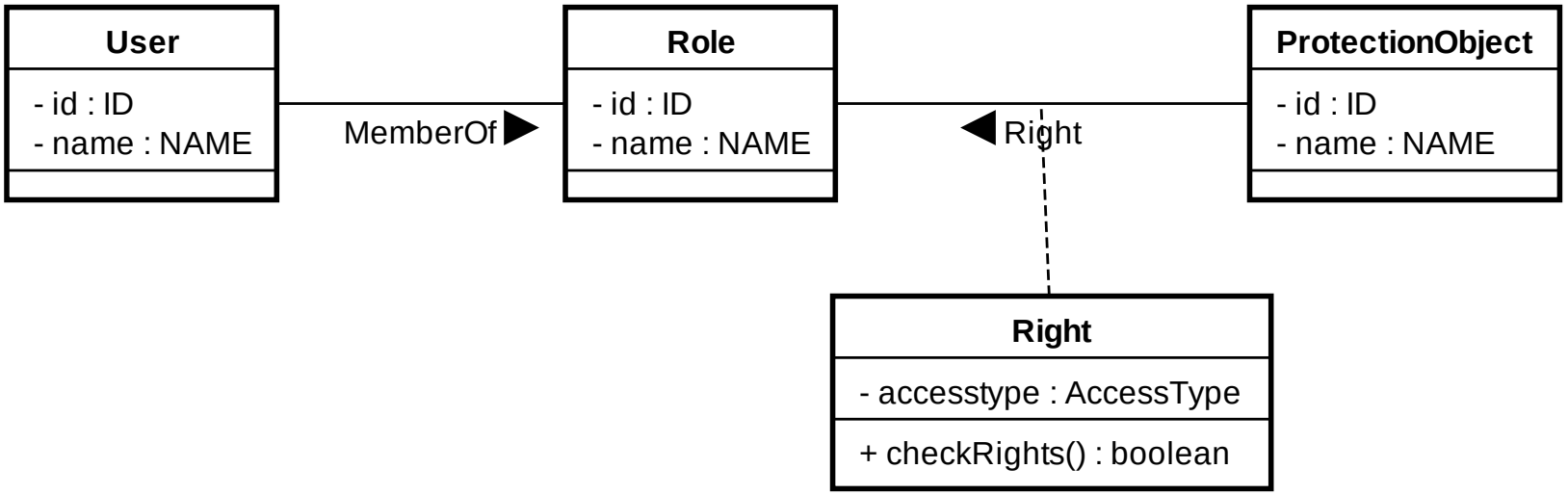
単純なパターン

- 人と物を単純に関連付け，どのようなアクセス許可しているかを関連の属性として付与する.
- アクセス行列をクラス図で書いたもの.



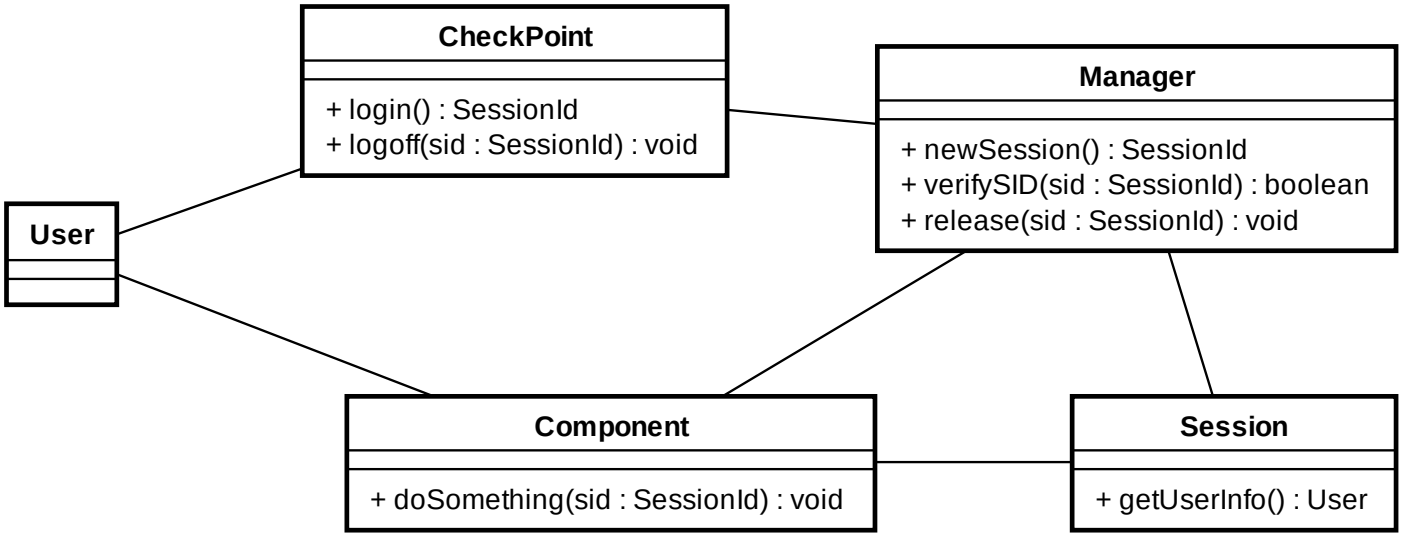
Role based Access Control

- 役割に応じて，対象物に対して，限定的なアクセス型を提供する.
- 人に直接，権限を割り当ててないので，権限集合である役割の再利用や共有が可能.

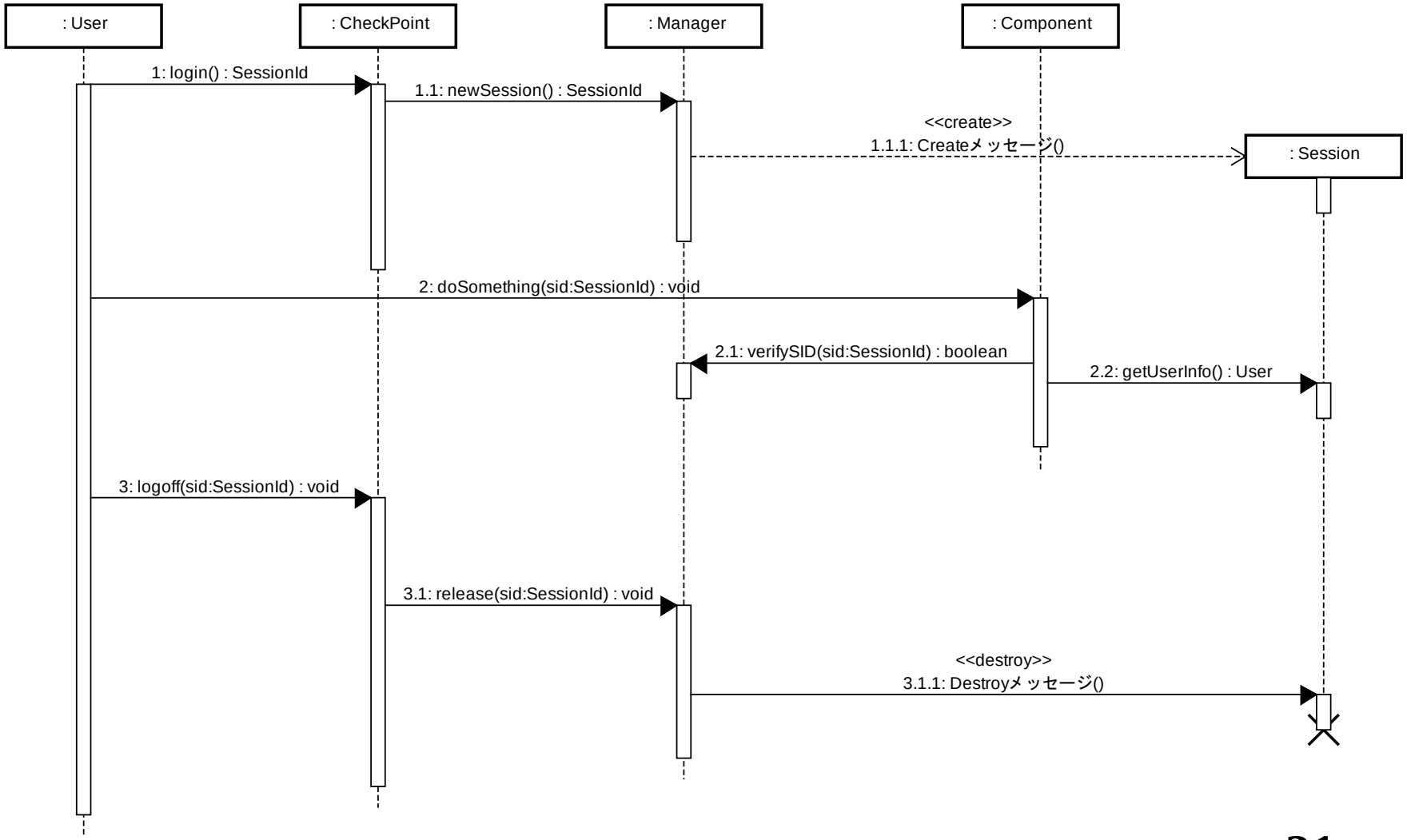


セッションの管理

- 話としては，User が Component をアクセスするだけ.
- しかし，事前にログイン等を行なった User のみに Component のアクセスを限定するため，Session を作成および破棄する機構が組み込まれている.

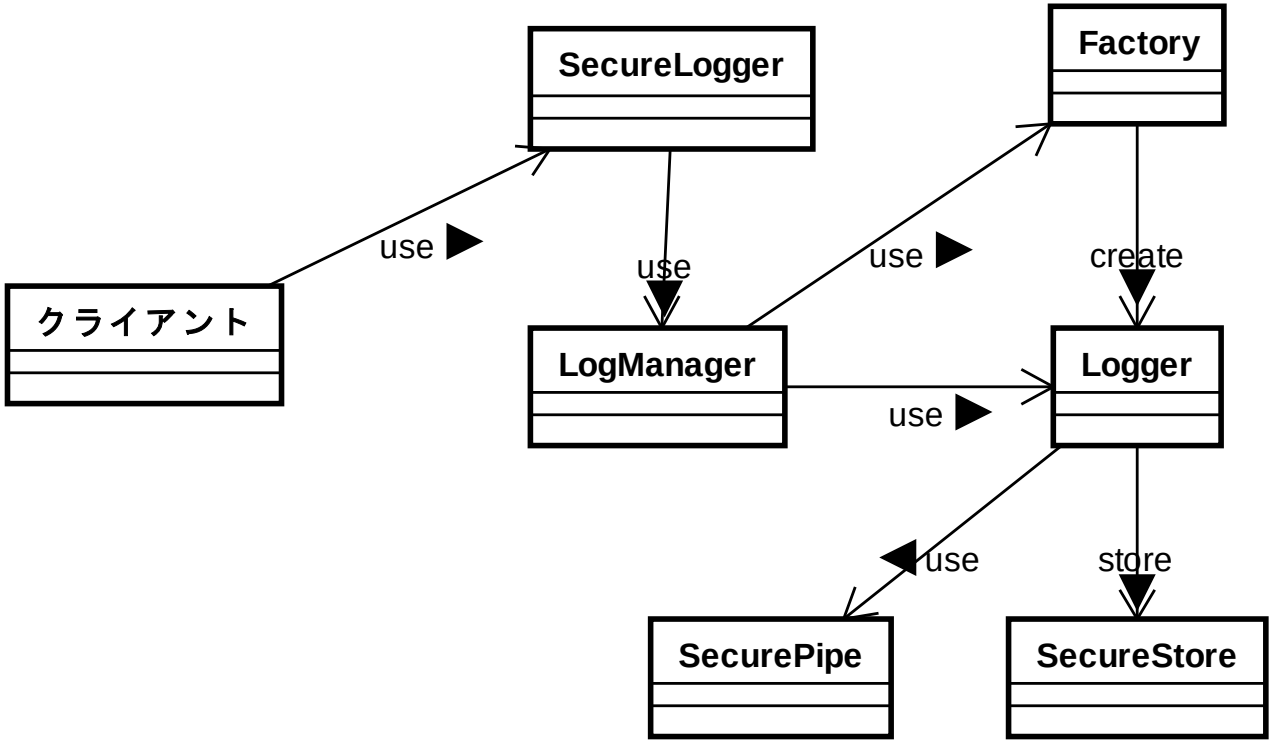


動作シーケンス


: User
: CheckPoint
: Manager
: Component
: Session
1: login() : SessionId
1.1: newSession() : SessionId
<<create>>
1.1.1: Createメッセージ()
2: doSomething(sid:SessionId) : void
2.1: verifySID(sid:SessionId) : boolean
2.2: getUserInfo() : User
3: logoff(sid:SessionId) : void
3.1: release(sid:SessionId) : void
<<destroy>>
3.1.1: Destroyメッセージ()

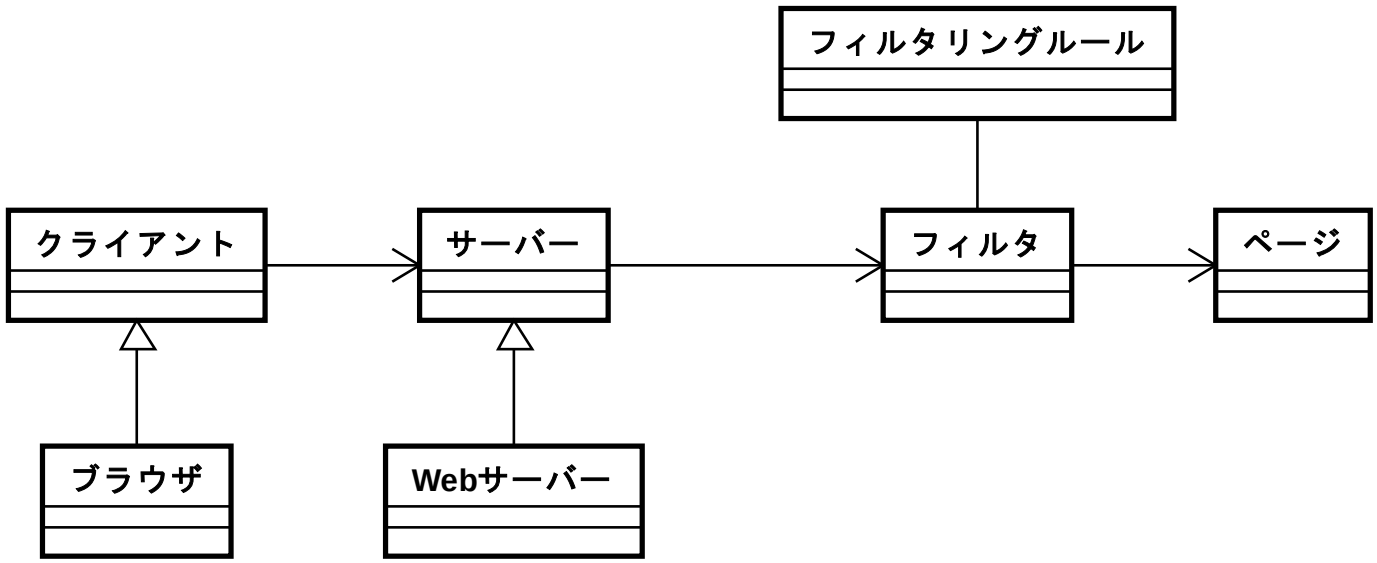
セキュア ロガー

- セキュアに動作記録(ログ)を保持するための設計パターン.



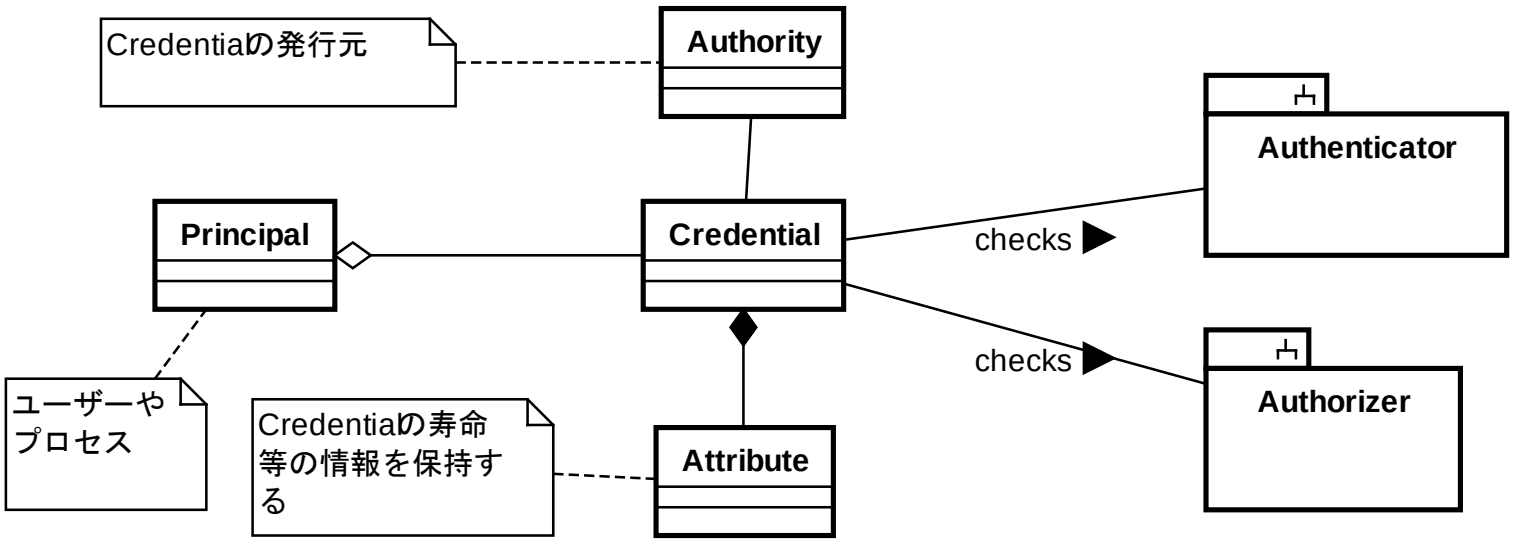
Input Validation

- Web等の任意入力をチェックし，不正なデータアクセス(injectionと呼ばれる)を防止するパターン.
- 実際，多くのライブラリに利用されている.

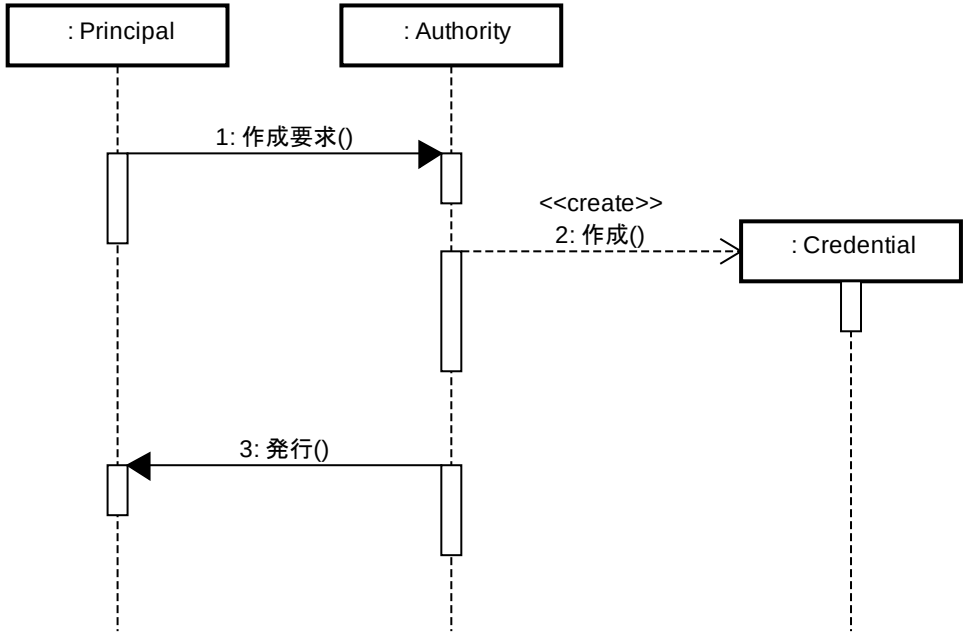


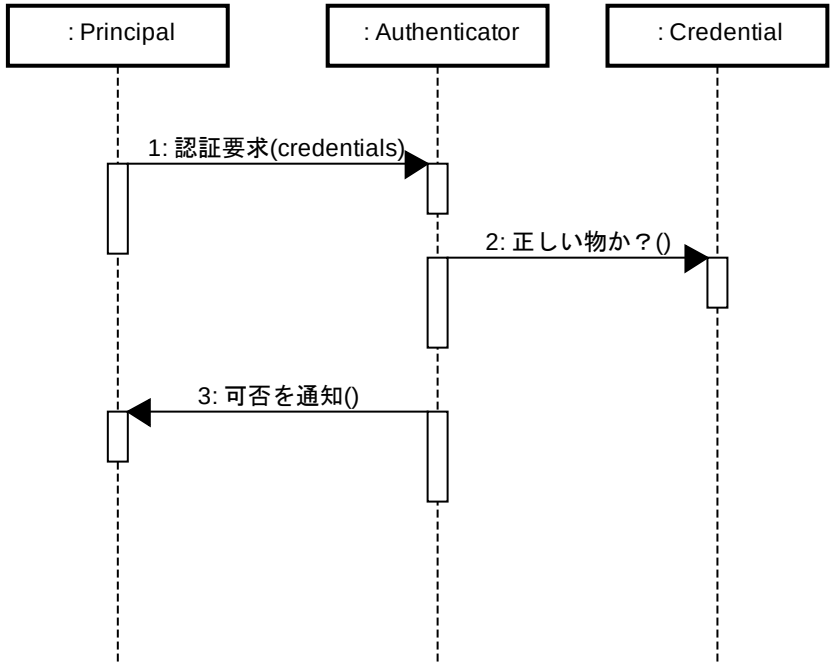
Credential Pattern

- クレデンシャルとは，分散システムにおいて，authenticationとauthorizationの情報を記録する軽便な手段.
- ユーザーの手元のPC等とアクセス権に相当するCredentialを分離し，分散システムでのアクセス管理を平易にする.



発行(左)と認証(右)のシーケンス例



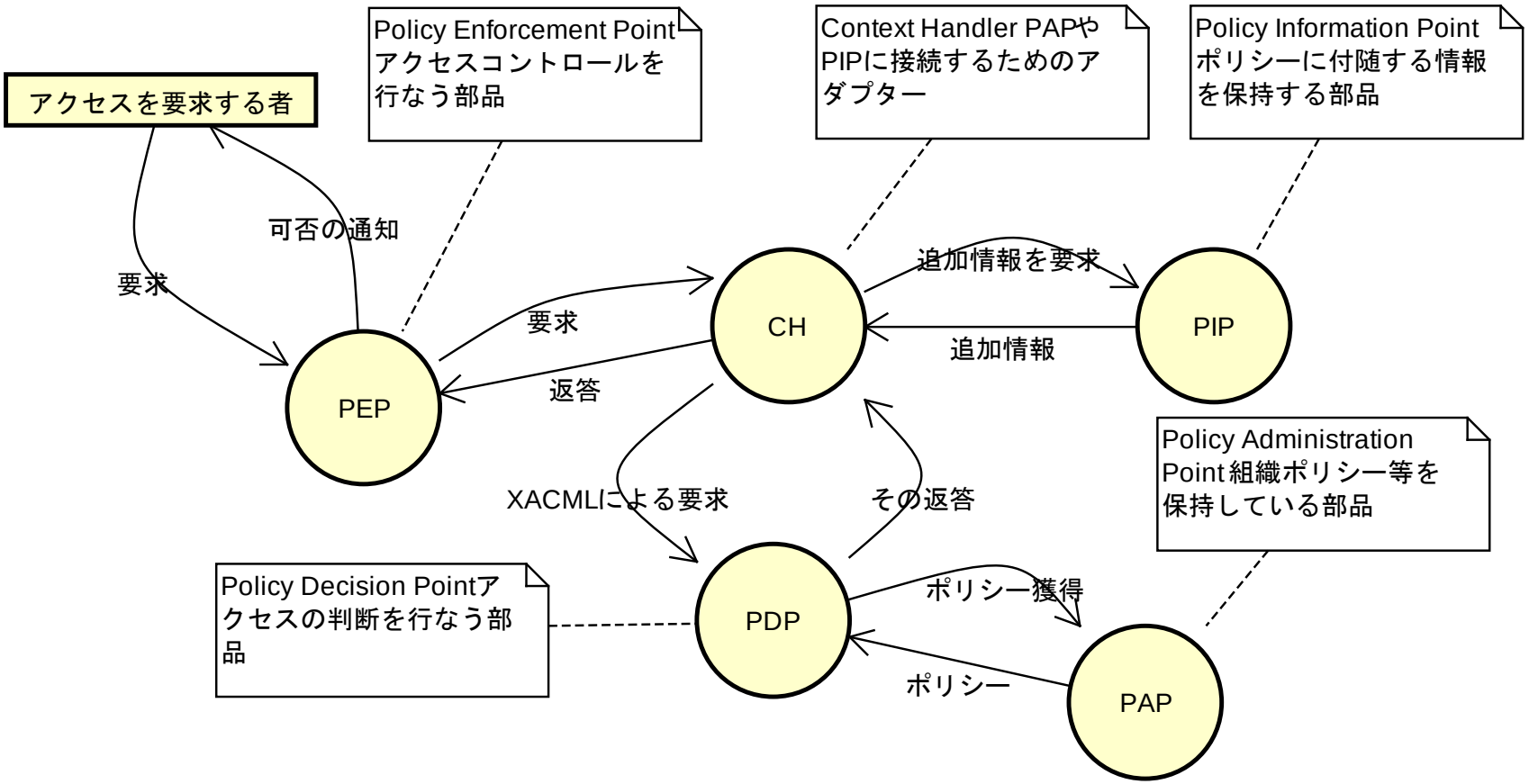


XACML

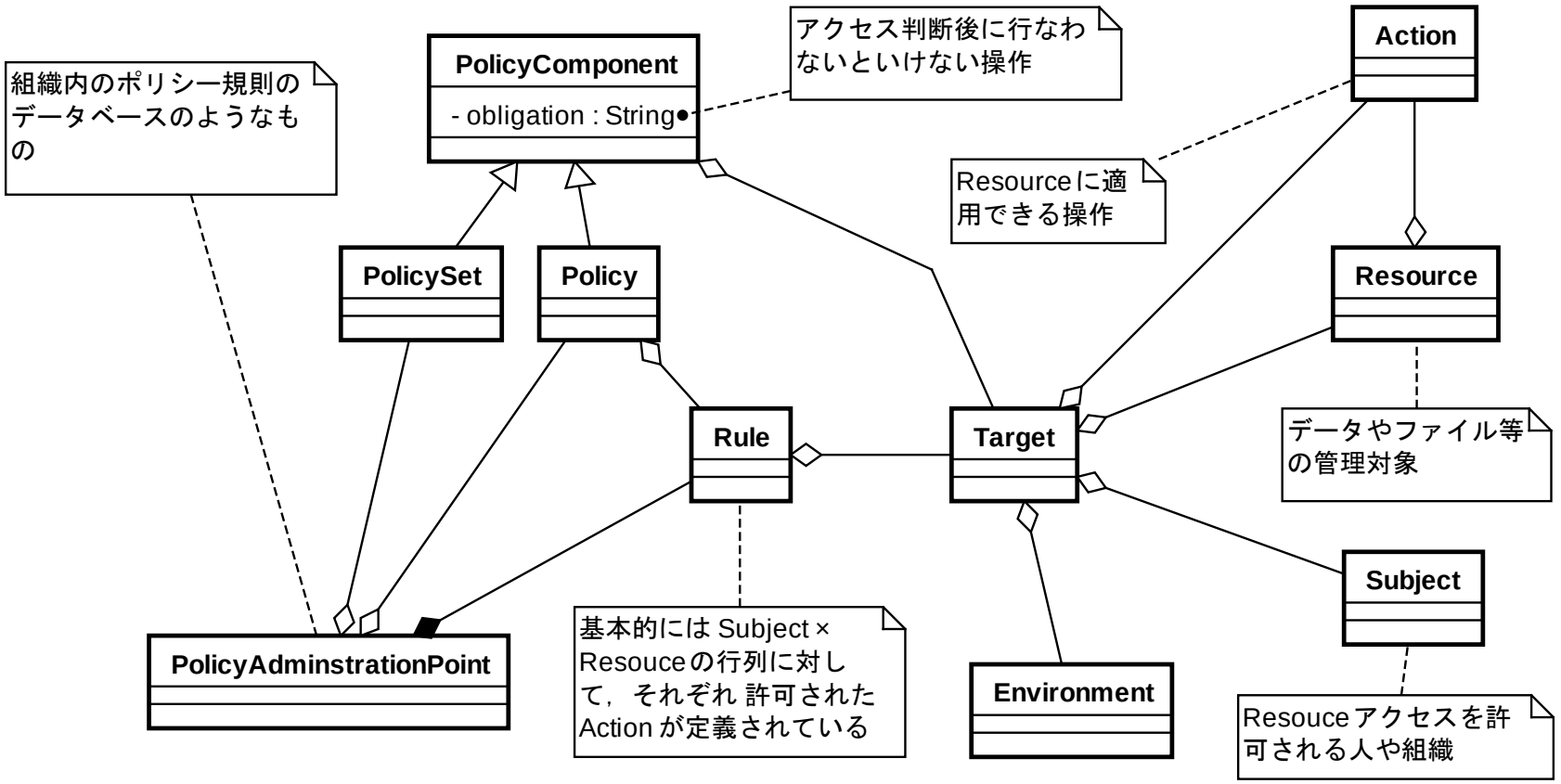
- eXtensible Access Control Markup Language
- XMLでWebサービスのアクセス制御を書く標準.
- XACMLを使うパターンがいくつかある.

- XACML Authorization 組織全体で使える Authorization (認可)規則の定義の書き方.
- 以下のような問題意識でパターンができている.
  - 組織で扱うデータの種類は多様.
  - アクセス制御やポリシー記述もそれぞればらばら.
  - 上記を統一的に扱うための枠組みが必要.

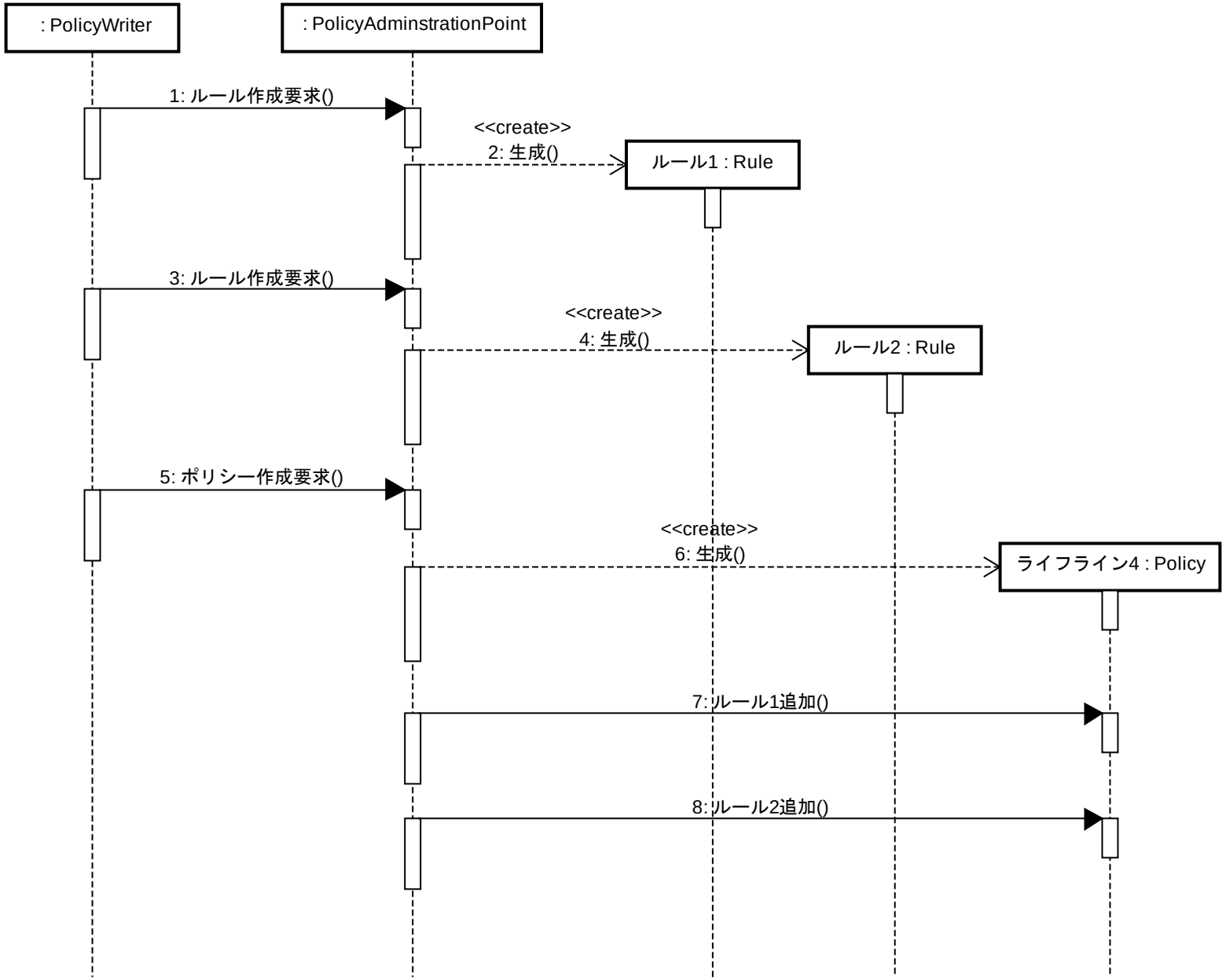
Authorizationの際のデータの流れ



XACMLポリシー記述言語



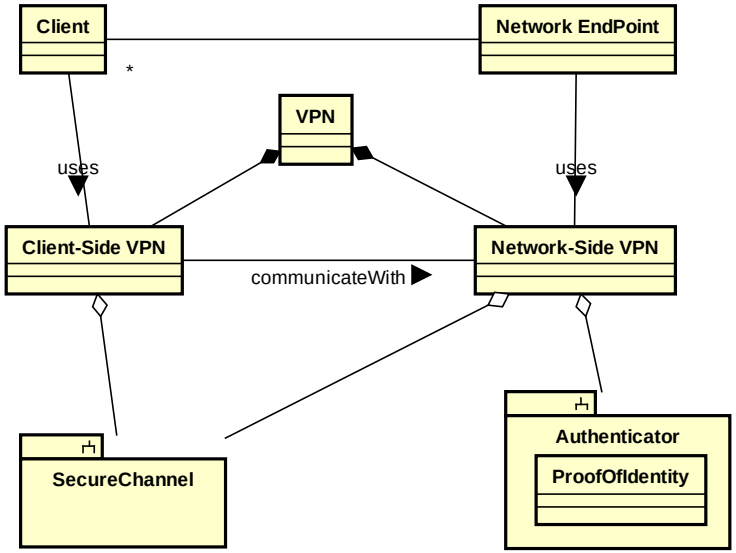
新規ポリシーの作成フロー例

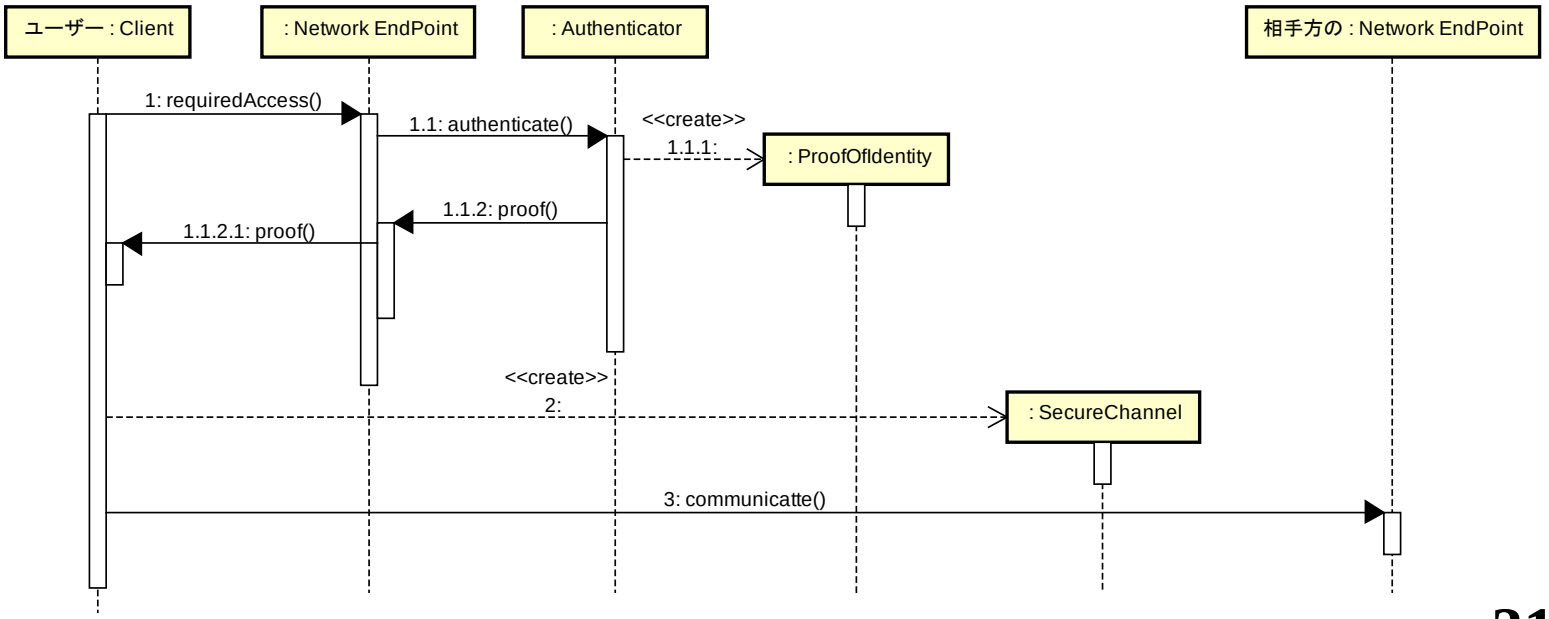


Virtual Private Network (VPN)

- 暗号化された通信路を公共ネットワークに作成し，そのネットワークを用いて異なる場所の機器群でプライベイトなネットワークを構成する方法.
- 世界に複数拠点ある会社等は当然のごとく利用している.

パターン


Client
Network EndPoint
*
VPN
uses
uses
Client-Side VPN
communicateWith
Network-Side VPN
SecureChannel
Authenticator
ProofOfIdentity


ユーザー : Client
: Network EndPoint
: Authenticator
相手方の : Network EndPoint
1: requiredAccess()
1.1: authenticate()
<<create>>
1.1.1:
: ProofOfIdentity
1.1.2: proof()
1.1.2.1: proof()
<<create>>
2:
: SecureChannel
3: communicatte()

セキュリティに対する知識の収集

- パターンでは無いが，以下のようなセキュリティ知識の収集データがある.
- パターン的なものも含まれるが，単純に便利で使える.

- CAPEC 攻撃のパターンを収集
  - Common Attack Pattern Enumeration and Classification
- CVE 脆弱性の例を収集
  - Common Vulnerabilities and Exposures
- OWASP ウェブアプリに対するセキュリティ知識を収集
  - Open Web Application Security Project

CAPECの例

- APIの不正利用

CAPEC-133: Try All Common Switches

Attack Pattern ID: 133
Abstraction: Standard

Status: Draft

Presentation Filter: Basic

Description

An attacker attempts to invoke all common switches and options in the target application for the purpose of discovering weaknesses in the target. For example, in some applications, adding a --debug switch causes debugging information to be displayed, which can sometimes reveal sensitive processing or configuration information to an attacker. This attack differs from other forms of API abuse in that the attacker is blindly attempting to invoke options in the hope that one of them will work rather than specifically targeting a known option. Nonetheless, even if the attacker is familiar with the published options of a targeted application this attack method may still be fruitful as it might discover unpublicized functionality.

Relationships

The table(s) below shows the other attack patterns and high level categories that are related to this attack pattern. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as CanFollow, PeerOf, and CanAlsoBe are defined to show similar attack patterns that the user may want to explore.

Nature	Type	ID	Name
ChildOf	M	113	API Manipulation

Prerequisites

The attacker must be able to control the options or switches sent to the target.

Mitigations

Design: Minimize switch and option functionality to only that necessary for correct function of the command.

Implementation: Remove all debug and testing options from production code.

CVEの例

- Pythonのライブラリにcrlfに関するinjectionの脆弱性があるらしい
- 2019/3登録の新しい情報

CVE-ID	
CVE-2019-9947	Learn more at National Vulnerability Database (NVD) • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

An issue was discovered in urllib2 in Python 2.x through 2.7.16 and urllib in Python 3.x through 3.7.3. CRLF injection is possible if the attacker controls a url parameter, as demonstrated by the first argument to urllib.request.urlopen with \r\n (specifically in the path component of a URL that lacks a ? character) followed by an HTTP header or a Redis command. This is similar to the CVE-2019-9740 query string issue.

References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- CONFIRM:https://security.netapp.com/advisory/ntap-20190404-0004/
- FEDORA:FEDORA-2019-1ffd6b6064
- URL :https://lists.fedoraproject.org/archives/list/package-announce@lists.fedoraproject.org/message

OWASPの例

- いくつかのプロジェクトがあるが攻撃TOP10を文書にまとめてるのが有名.

以上