CommunicationManager

- + CommunicationHandler
- + CommunicationManager
- + FastServiceHandler
- + InputParser
- + JNIHandler
- + WebServiceHandler
- + ICommunication
- + IIncoming
- + IOutgoing
- + IPDP2PEP
- + IPDP2PIP
- + IPDP2PMP
- + IPDP2UCRegistry
- + IPEP2PDP
- + IPIP2PDP
- + IPMP2PDP
- + IUCRegistry2PDP

JNIHandler

WebServiceHandler

FastServiceHandler

«interface»
IPMP2PDP
- + deployMechanism(Mechanism) :Status
- + exportMechanism(string) :Status
- + revokeMechanism(string) :Status

«abstract»
CommunicationHandler
- + evaluatePredicate(string) :boolean
- + execute(Event[0..*]) :Status
- + getContainerForData(Data) :Container[0..*]
- + getDataInContainer(Container) :data : Data[0..*]

InputParser
- + parseEvent(string) :Event
- + parseMechanism(string) :Mechanism

«interface»
IPEP2PDP
- + notifyEvent(Event) :Status

«interface»
IIncoming

«interface»
ICommunication
- + evaluatePredicate(string) :boolean
- + execute(Event[0..*]) :Status
- + getContainerForData(Data) :Container[0..*]
- + getDataInContainer(Container) :Data[0..*]

«interface»
IPIP2PDP

«interface»
IPDP2PMP

«interface»
IOutgoing

CommunicationManager
- - PIPConnection :int
- - PMPConnection :int
- - UCRegistryConnection :int

«interface»
IPDP2PEP
- + execute(Event[0..*]) :Status

«use»

«interface»
IPDP2PIP
- + evaluatePredicate(string) :boolean
- + getContainerForData(Data) :Container[0..*]
- + getDataInContainer(Container) :Data[0..*]

«interface»
IUCRegistry2PDP
- + getStatus() :void

«use»

«interface»
IPDP2UCRegistry
- + registerPDP() :void
- + unregisterPDP() :void

Note: Initial representation is now handled by
IPMP2PIP

PDPManage

- + Arbiter
- + Decisio
- + EventF
- + PDPM
- + Persist
- + PolicyI
- + Simple
- + Simple
- + Transl
- + IPDP

- + depl
- + expo
- + notif
- + revo

«use»

DataTypes

- + Status
- + Condition
- + Container
- + Data

«dataType»
Event

«dataType»
OSLFormula

«dataType»
SimplifiedTemporalLogic

Partial left-margin package list:
- ...onEngineBroker
- ...Preprocessor
- ...anager
- ...enceManager
- ...Preprocessor
- ...eAllowArbiter
- ...eDenyArbiter
- ...ationTable

SimpleAllowArbiter

SimpleDenyArbiter

«abstract»
Arbiter

+ mergeDecisions(Response[0..*]) :Response

DecisionEngineBroker

+ getMechanism(string) :Mechanism
+ instantiateDecisionEngine(Mechanism) :Status
+ listMechanisms() :string[0..*]
+ notifyEvent(Event) :Response
+ removeDecisionEngine(string) :Status

PDPManager

+ evaluatePredicate(String) :boolean
+ getContainerForData(Data) :Container[0..*]
+ getDataInContainer(Container) :Data[0..*]
+ processPersistenceEvent() :void

PersistenceManager

+ loadPDPState() :void
+ persistPDPState() :void

«interface»
IPDP

...oyMechanism(Mechanism) :Status
...rtMechanism(string) :Mechanism
...yEvent(Event) :Response
...keMechanism(string) :Status

EventPreprocessor

+ processEvent(Event) :Event

PolicyPreprocessor

- checkMechanismSanity(Mechanism) :Status
+ processMechanism(Mechanism) :Mechanism
+ revokeMechanism(string) :Status

«persistent»
TranslationTable

- DataEventMap :DataEventMap
- StateEventMap :StateEventMap

+ get() :void
+ iterate() :void
+ put() :void
+ remove() :void

DecisionEngine

+ DecisionEngine
+ ECAMechanism
+ State
+ IDecisionEngine

«persistent»
ECAMechanism

«interface»
IDecisionEngine

+ DecisionEngine(Mechanism) :Status
+ getMechanism() :Mechanism
+ notifyEvent(Event) :Status

DecisionEngine

«persisten...
State

«use»

«t»

+ DataEventMap
+ Event
+ History
+ Mechanism
+ OSLFormula
+ Response
+ SimplifiedTemporalLogic
+ StateEventMap

- eventName :string
- parameters :HashMap

- formula :string

- dataEventMap :DataEventMap
- formula :string
- stateEventMap :StateEventMap

«dataType»
Response

- authorizationAction :Status
- executeActions :Event [0..*]
- modifiedEvent :Event

«dataType»
Mechanism

- condition :Condition
- mechanismName :string
- response :Response
- state :History
- triggerEvent :Event

«dataType»
Condition

- condition :OSLFormula
- conditionSimp :SimplifiedTemporalLogic

«enumeratio...
Status

error1
okay
error2
inhibit
allow
modify

«dataType»
History

- trace :Event [0..*]

«dataType»
DataEventMap

- map :HashMap

«dataType»
StateEventMap

- map :HashMap

«dataType»
Data

- id :string

«dataType»
Container

- class :string
- id :string