PDP Developers Guide

(draft)

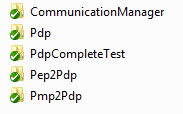
|  |  |  |
| --- | --- | --- |
| Revision history | | |
| Date | Author | Comment |
| 2013-08-10 | Stoimenov | Draft version |
|  |  |  |

Notice:

This guide is intended for developers who want to understand PDP code in order to extend it, fix bugs, or modify it in some other way. Users who want to develop PEP and PMP modules and who want to communicate with PDP should read “PDP User Guide”.

# PDP Architecture

PDP stands for Policy Decision Point. The module is implemented in Java and it provides interfaces which can be used by PEP (Policy Enforcement Point) modules and PMP (Policy Management Point) modules. The code of PDP and related stubs is organized into the following projects:



Maven is used for project configuration. The projects have flat structure. Pdp is the parent project. Its pom.xml (maven descriptor file) resides at the same level in the file system hierarchy as its child projects.

CommunicationManager is part of the PDP and it is responsible for the communication between the PDP and other modules such as PMPs and PEPs.

Pep2Pdp contains a stub which can be used by PEP developers to make communication with PDP.

Pmp2Pdp contains a stub which can be used by PMP developers to communicate with PDP.

The communication between PEP and PDP and PMP and PDP is done via TCP. In addition to java socket programming, Google protocol buffers are used. Google protocol buffers enable an efficient way of serialization (fast and compact). Messages to be exchanged are defined in proto files (see CommunicationManager/src/main/resources/proto). Google protocol buffer tool (freely available from Google) is used to generate Java code in form of classes corresponding to these messages. In addition, there is support for programming languages such as C++ and Python. There is also support for other programming languages by third party vendors.

Figure 1 shows all message types used in PDP:

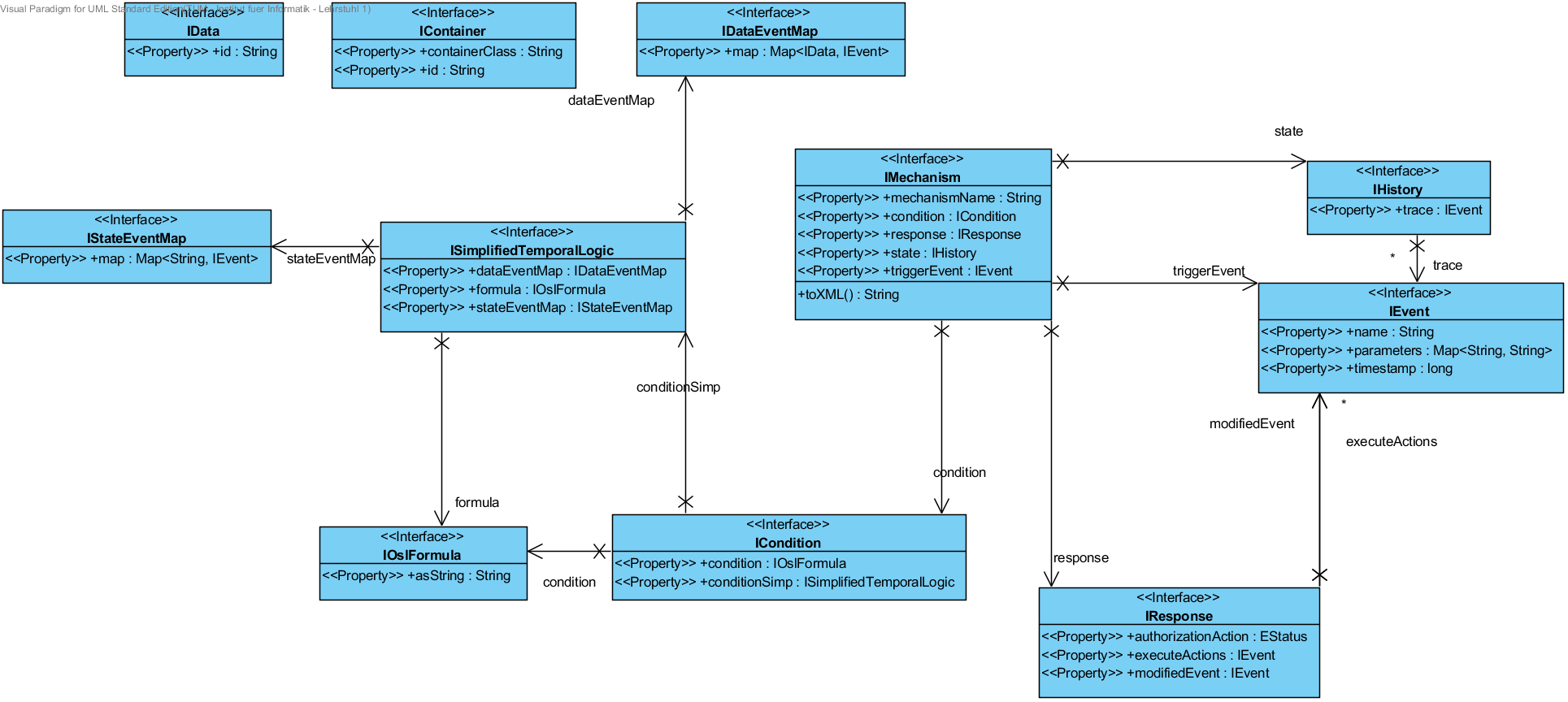


Figure Message Types

Figure 2 represents class diagram of the communication manager. For the sake of simplicity, only the most important classes are shown.

# Scenario: Communication between a PEP and the PDP

Figure 3 shows interaction between a *PEP* client and the *PDP*.

Two threads are created during the *PDP* startup: PepFastServiceHandler and PmpFastServiceHandler. They wait for incoming client connections. Each thread listens to a different port. The port numbers can be specified in the file pdp.properties.

The sequence of method invocation will be explained for the case of communication between a *PEP* client and the *PDP*. Similar scenario applies to the case when a PMP communicates with the PDP.

PEP uses Pep2Pdp stub to establish connection to PDP. Connection parameters are IP address and port number of the PepFastServiceHandler. After establishing the connection, PepFastServiceHandler creates an instance of PepClientConnectionHandler which receives incoming messages from the PEP client. PEP then invokes notifyEvent() method on the stub with an instance of *IEvent* as a parameter. The stub will convert IEvent to corresponding instance of Google Protocol Buffer class GpEvent. The google protocol buffer object is received by the PepClientConnectionHandler. It is than converted to an instance of IEvent. The newly created IEvent is placed in the queue which is part of the RequestHandler class. The PepClientConnectionHandler is paused and it waits for the RequestHandler to process the event. The queue might already contain several requests that will be processed in FIFO order. Once the event is processed, the PepClientConnectionHandler is woken up and it replies to the *PEP* client by sending the response created by the RequestHandler.

RequestHandler ‘s queue elements contain all necessary information to uniquely identify the method invoked on the *PDP* and the thread which is responsible for sending the response to the client.

An example which shows how to communicate with PDP from the perspective of a PEP can be seen in Pep2Pdp/src/test/java/ TestPep2PdpCommunication.

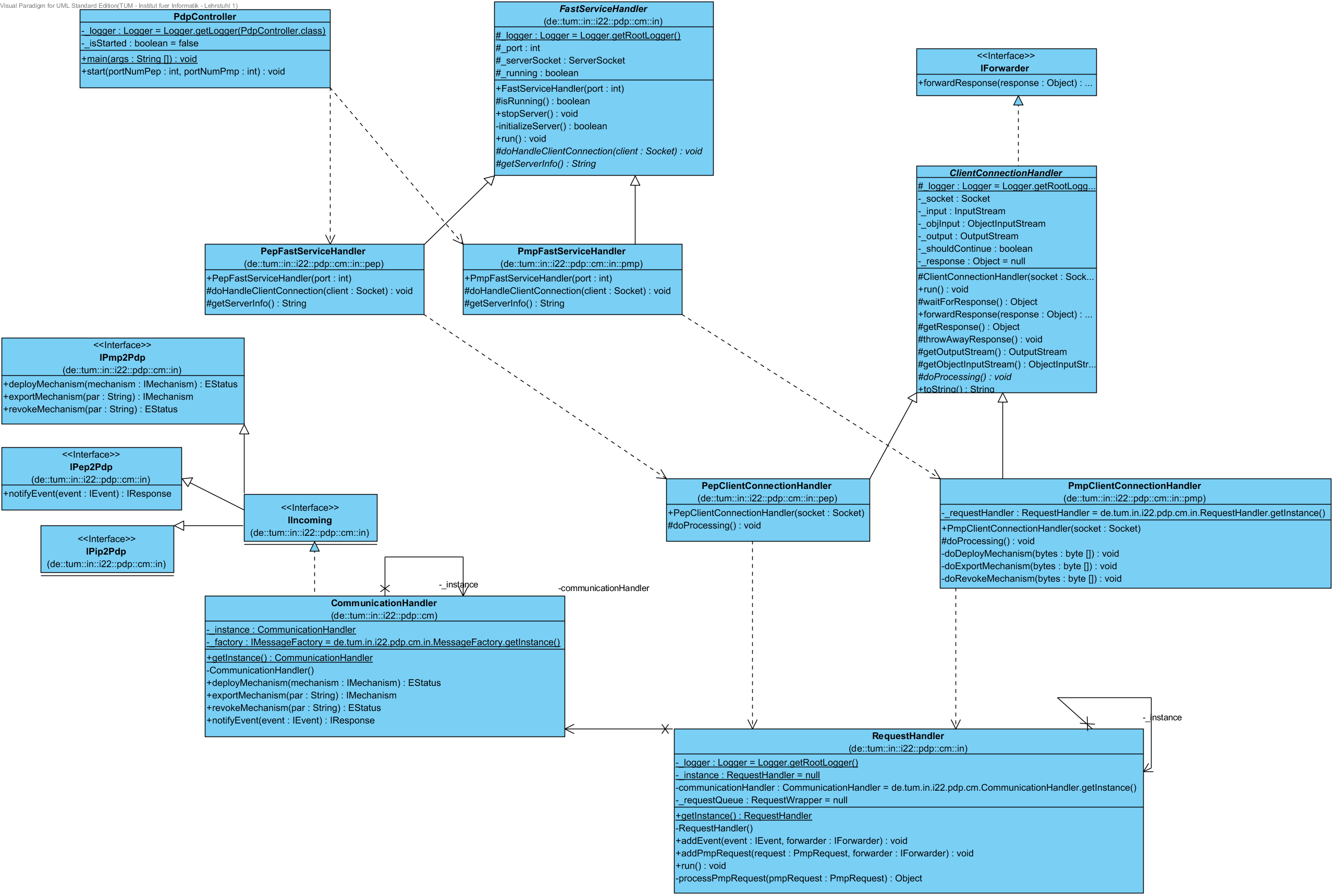


Figure Communication Module Class Diagram

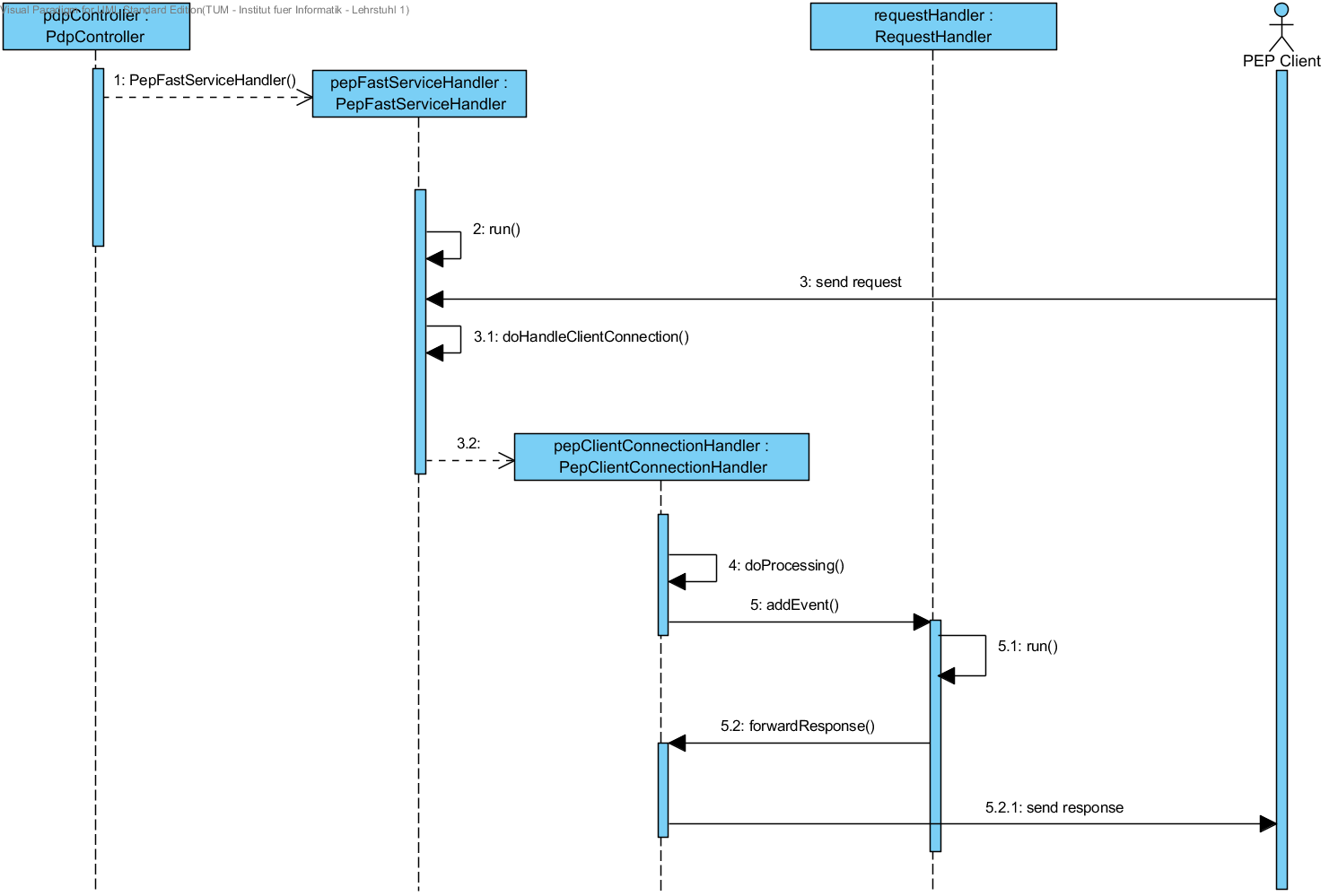


Figure PEP Client - PDP interaction