axini

Constructing an **Adapter** for **AMP**

25 January 2023
www.axini.com

Model Based Testing

is an **automated** technical process,

... performed by **automatically** executing/experimenting

... with a **software** system, in a **controlled environment**,

... following prescribed behavior of a **formal model**,

... with the intent of **measuring** one or more **characteristics** or the **quality** of the product

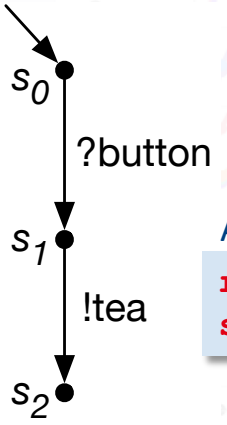
... by demonstrating the **deviation** of the actual status of the product from the required status/**specification**.

*The 'Model' is an **abstract, high-level** and **formal** description of the SUT.*

AMP: Axini Modeling Platform

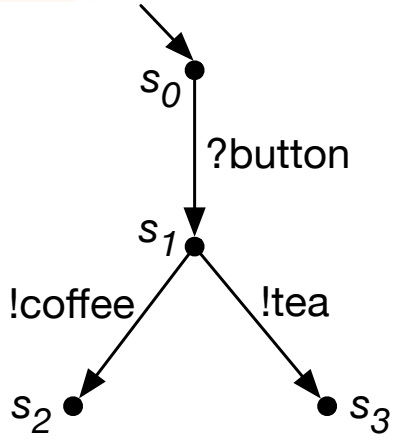
- **AMP**: tool to edit, explore, visualize, test AML **models**
  - **Test-cases** are automatically **generated**.
- Models are written in Axini Modeling Language (**AML**)
  - **behavior**: mapped upon **labeled transition systems (LTSs)**
  - **data**: modern, powerful, static typed 'programming' language

Examples of LTSs



AML

```
receive 'button'
send 'tea'
```

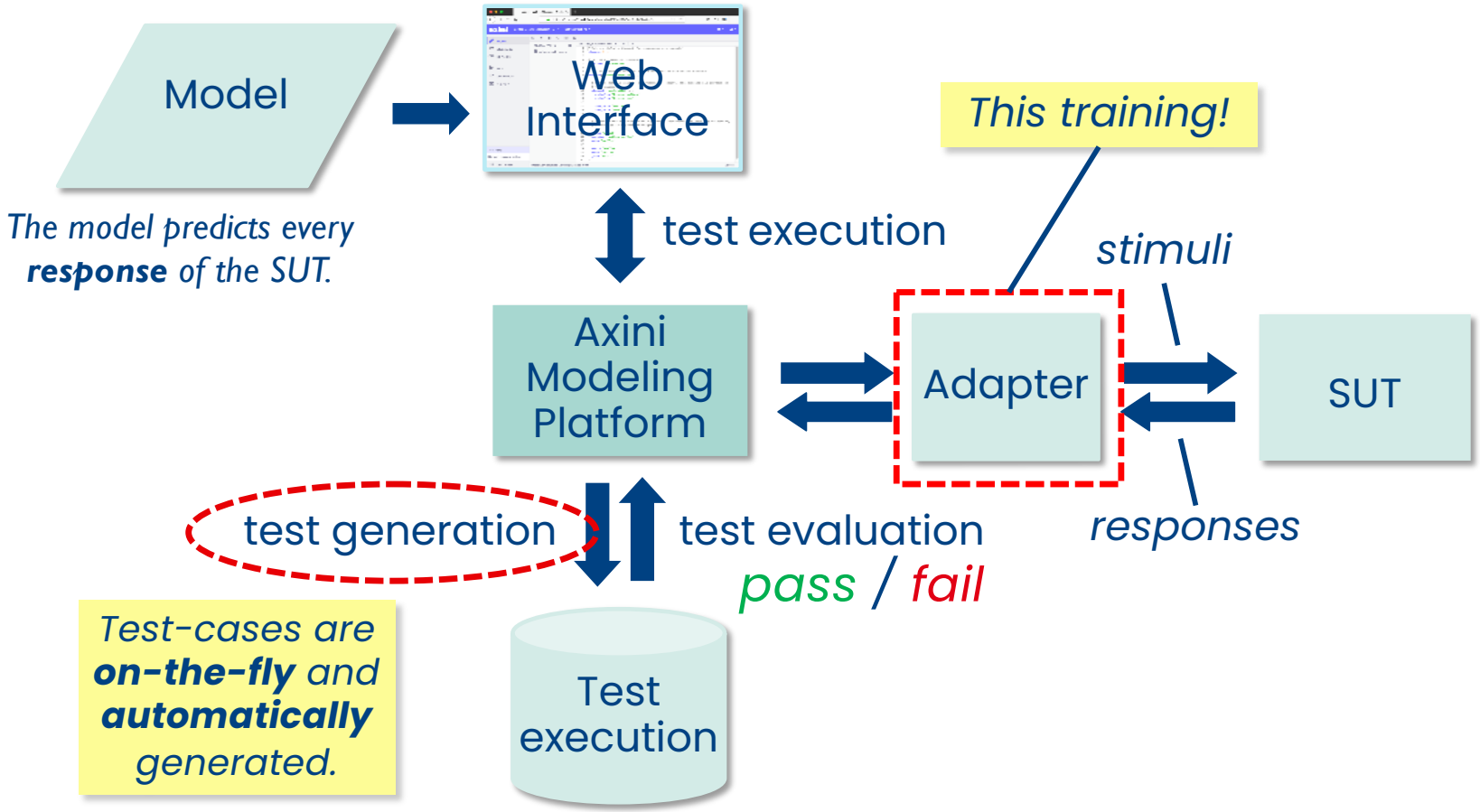


AML

```
receive 'button'
choice {
  o { send 'coffee' }
  o { send 'tea'    }
}
```

? *= input (stimulus)*
! *= output (response)*

AMP architecture



Goals of the adapter training

- understand Axini's **plugin adapter protocol**
- work with **WebSocket** connections
- work with **Protobuf** messages
- understand the **architecture** of a **reference adapter**

Training is intended to **lift** the "**magic**" of writing adapters.

Overview

- Introduction
- Axini's Plugin Adapter Protocol
  - WebSocket
  - Google Protobuf
- Standalone SmartDoor SUT
- Implementation of a plugin-adapter
- Conclusions

*The greater part of the training consists of **developing** (some building blocks for) your own plugin adapter.*

Schedule

Wed 25-Jan

09:30	Welcome
09:40	Introduction to AMP's **Plugin Adapter Protocol**
10:15	— *break*
10:30	[Lab] §1 - First steps with **Protobuf** & **WebSocket**
12:00	— *lunch*
12:45	[Lab] §2 - WebSocket communication with **AMP**
14:30	— *break*
14:45	**Architecture** of Plugin Adapter (Java)
15:15	[Lab] Handler, BrokerConnection & AdapterCore
16:30	Wrap-up
17:00	— *end*

*You will **not** be able to **complete** the adapter; but you should have an idea how to **finish** it.*

Disclaimer

***Rest** of the adapter can be completed **individualy**.*

- **Second time** this adapter training will be given: **beta** material!
  - reference plugin adapter in Java
  - standalone SmartDoor SUT (in Java)
  - laboratory roadmap
  - presentation
- **Not** developed for a **fixed programming language**
  - We only have limited experience with Java, C++, C#, Python, etc.
  - You are the **experts** in your own programming language today!
- **Single day** training
  - Goal: show the **important ingredients** of a plugin adapter.
  - There is definitely **not enough** time to develop a **complete** adapter!

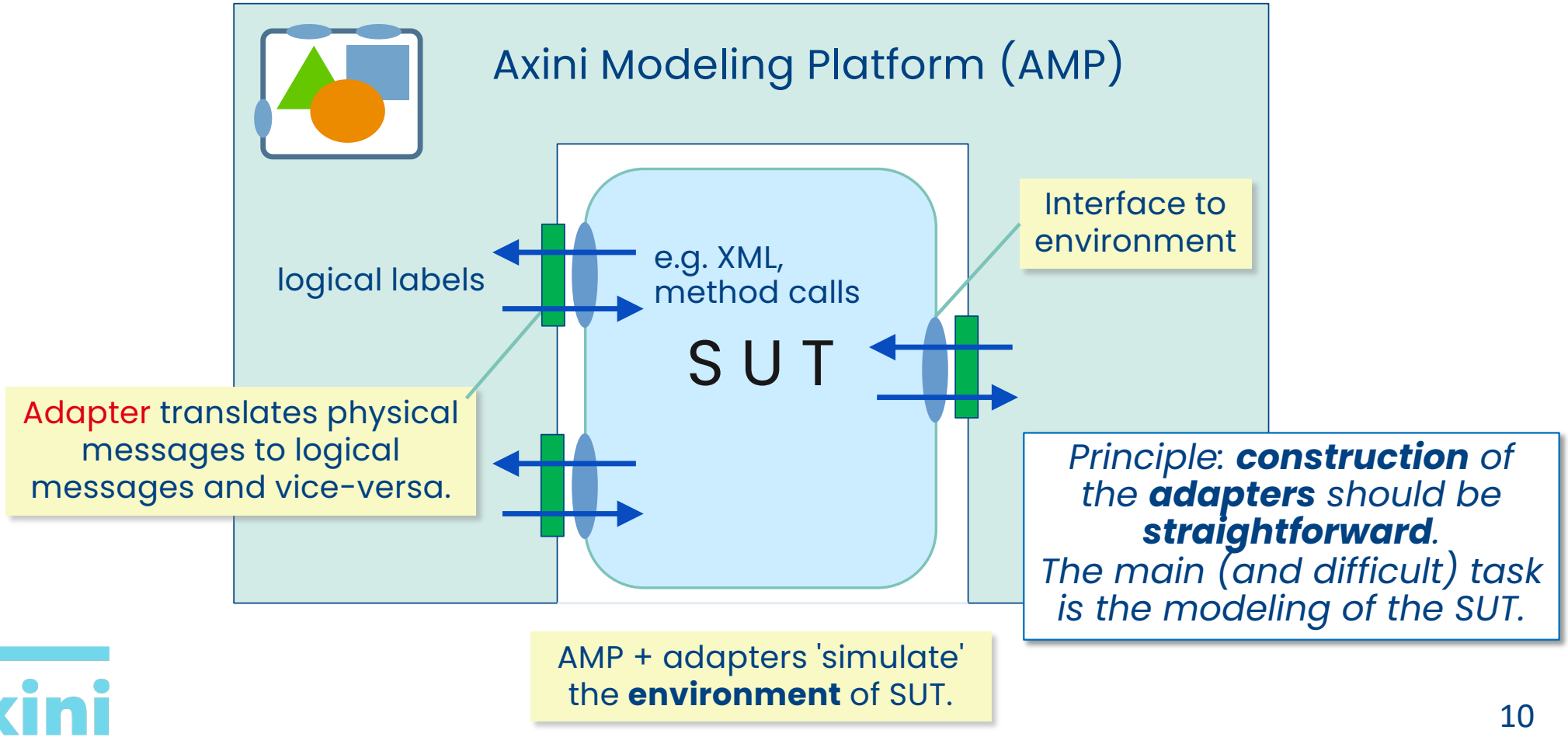
Resources

All resources will prove useful!

- Axini provided:
  - description of the adapter **laboratory**
  - standalone **SmartDoor SUT** (Java program)
  - **.proto** files for description of the **Protobuf** messages
  - **example** of plugin adapter for SmartDoor in **Java** — *Good source for inspiration.*
- Material:
  - **slides** of the presentation
  - Effective AML
    - § 4.5 Adapters: best practices
    - § 5.10 Plugin Adapters

  *Concise description, but with **valuable** tips.*
- Internet:
  - tutorials and examples on **WebSocket** and **ProtoBuf**

AMP: horseshoe around SUT



axini

Adapter is 'only' the adapter

Adapter is 'only' needed to make the MBT-horseshoe work. The real work is the **modeling**.

- Do **not** try to get the adapter **complete**, the first time. Make something that "just" connects AMP and SUT.
  - The **model** will require changes and additions to the adapter.
  - The **interface** is often not yet completely fixed.

If it **works**, it works. No goldplating.

- Try to **re-use** connections to the SUT that are already in place (e.g., for other test purposes).
- **Domain experts** (of the SUT connection) should make the SUT-related parts of the adapter.
  - If this is not possible, be sure to get all the information from the domain experts.

*If it **works**, it works!*

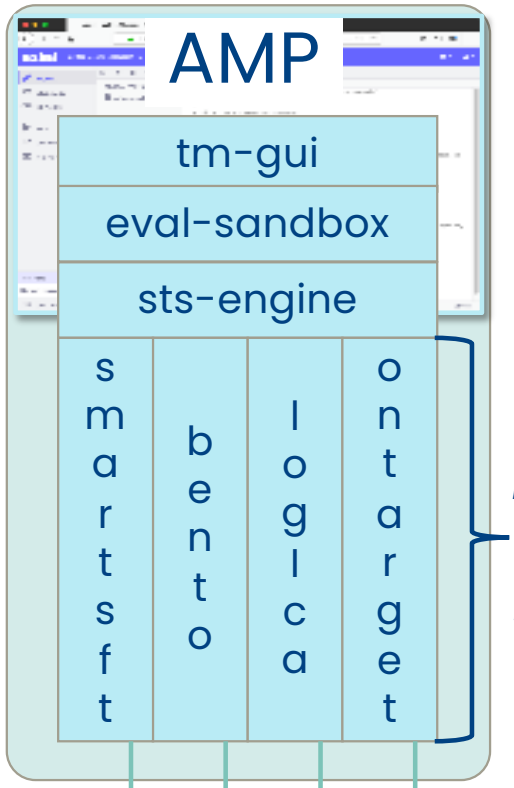
Observation: 'our' own plugin adapters are **not tested** as thoroughly as our production code: **adapters are tested when testing the SUT**.

Important: do **not add** additional **behavior** (state) to the adapter: the SUT is the target of the test (not the adapter).

AMP – Classic Adapters

Advantages:
- all adapters are **up to date**
- adapters can **share** code
- **deploy** is “easy”

Disadvantages:
- **only Axini** can write adapters
- all **adapters** have to be **updated**



Tasks of adapter:
- **Connect** to the SUT (e.g. JMS, SCTP, SSH)
- **Conversion** of AML labels to physical labels

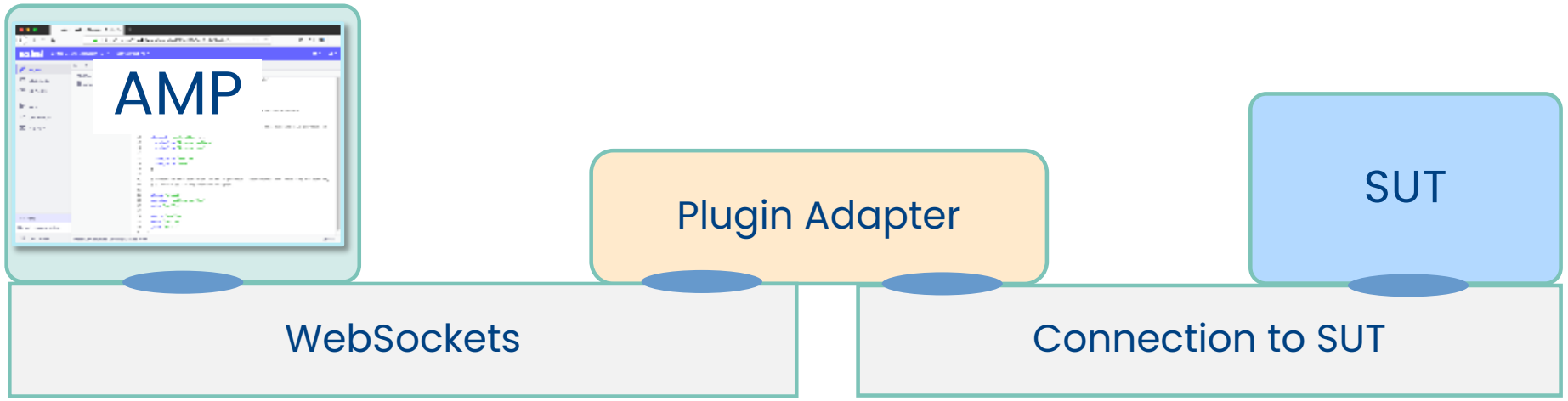
*For each different **SUT** a **special purpose adapter***

SmartSoft SUT | Bento SUT | Logica SUT | OnTarget SUT

AMP – Plugin Adapters

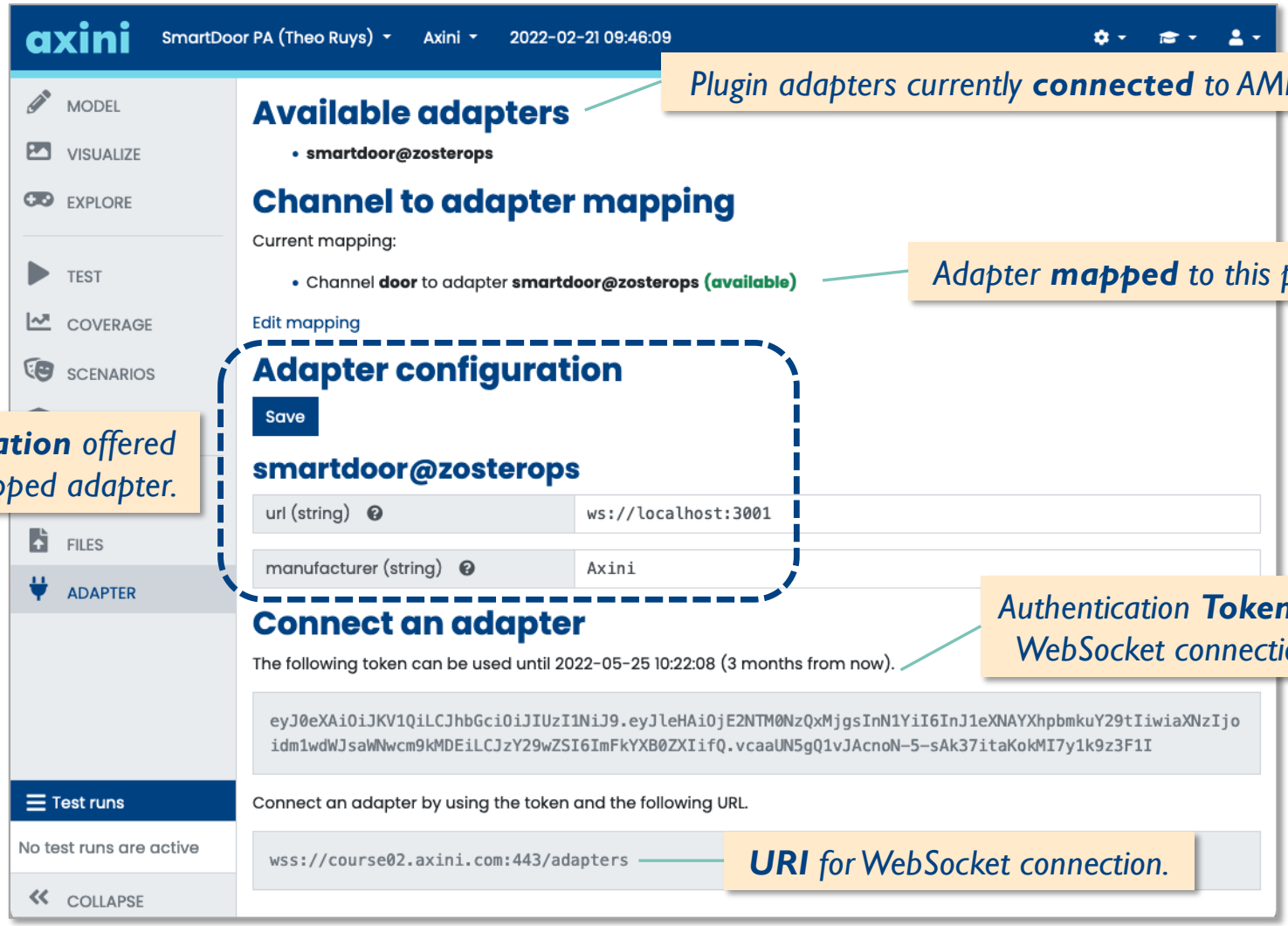
*Objective: **third parties** should be able to develop adapters for AMP.*

- Programming Language **independence**.
- **WebSockets** over HTTP for transport.
- Google **Protobuf** to represent **labels**.
- AMP acts as **server** for incoming adapers.



*Plugin adapter can be run more **closely** to the SUT, which allows for more accurate **timestamps**.*

AMP: adapter page



*Plugin adapters currently **connected** to AMP.*

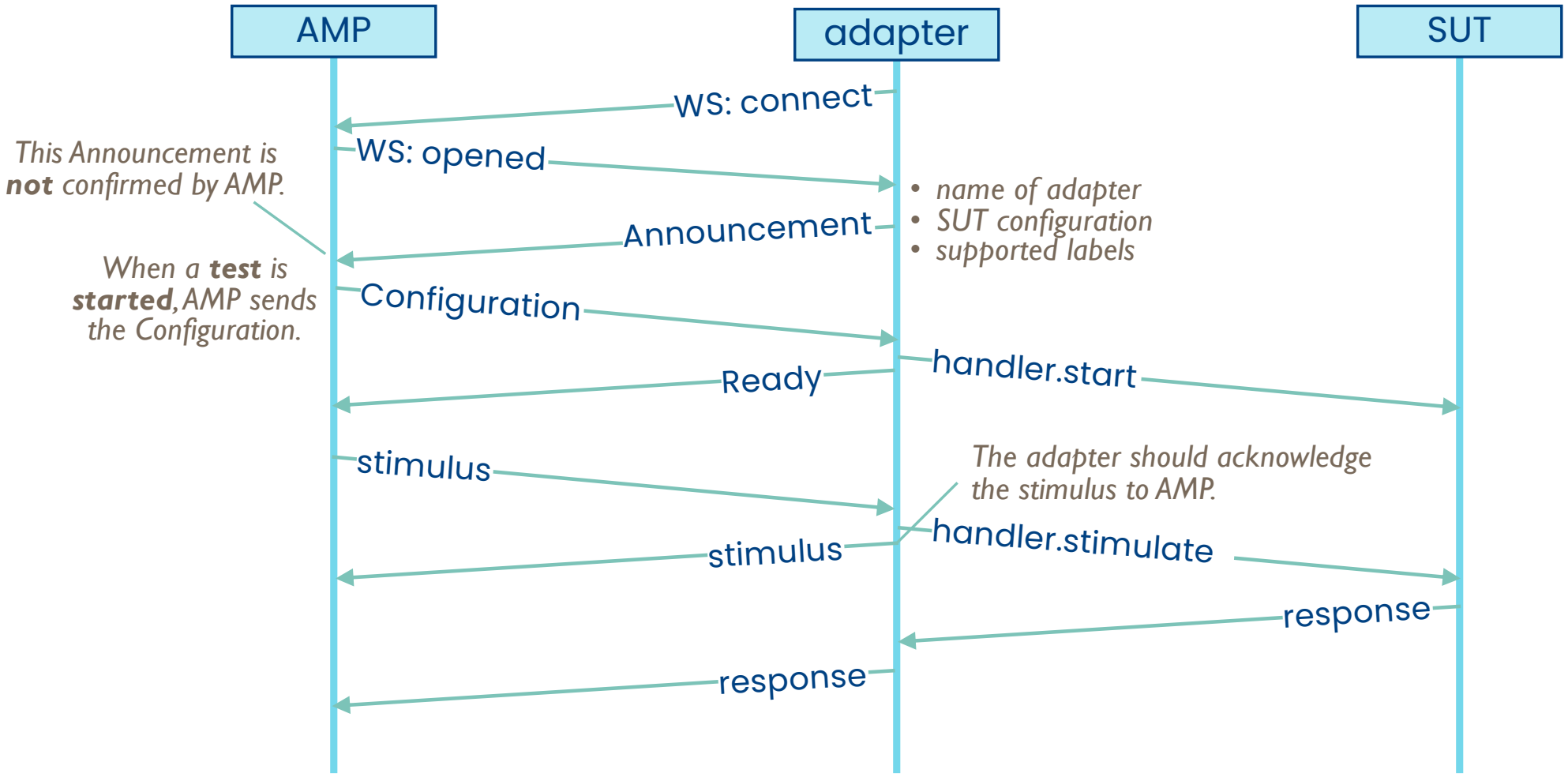
*Adapter **mapped** to this project.*

***Configuration** offered by the mapped adapter.*

*Authentication **Token** for WebSocket connection.*

***URI** for WebSocket connection.*

AMP - Adapter - SUT



Plugin Adapter

Protocol

Only the connection with AMP!

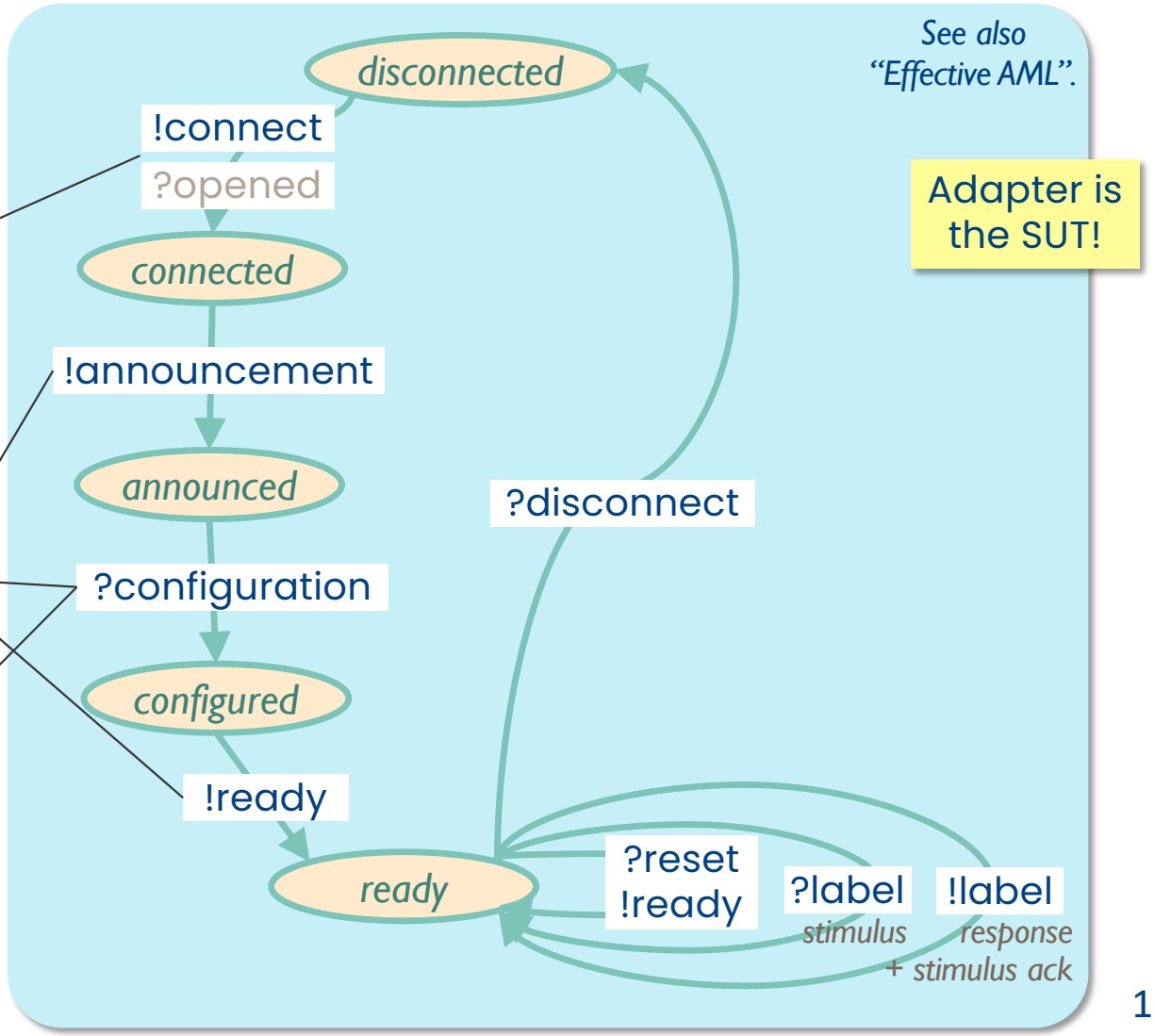
WebSocket

Protobuf messages

*AMP will send the **configuration** when the **test run starts**.*

*There is an **AML model** of the protocol, which has been **verified** with the **SPIN** model checker.*

*State-Transition diagram for the **plugin adapter**.*



WebSocket

specification https://datatracker.ietf.org/doc/html/rfc6455

wikipedia https://en.wikipedia.org/wiki/WebSocket

- **Full-duplex** communication protocol
  - over a single TCP connection, on top of **HTTP**(S)
    - bypasses any firewalls
  - **socket** connection: AMP acts as server, adapter is client
  - exchanging text or binary **messages**
- uniform resource identifier (**URI**): ws & wss
  - `ws://localhost:3001/` *Default URI of standalone SmartDoor SUT*
  - `wss://course02.axini.com:443/adapters` *AMP@course02*
- programming **libraries** abstract from low-level details
  - handshake, connection, SSL, etc. is done by the **library**
  - **call-back** methods to respond to specific **events**

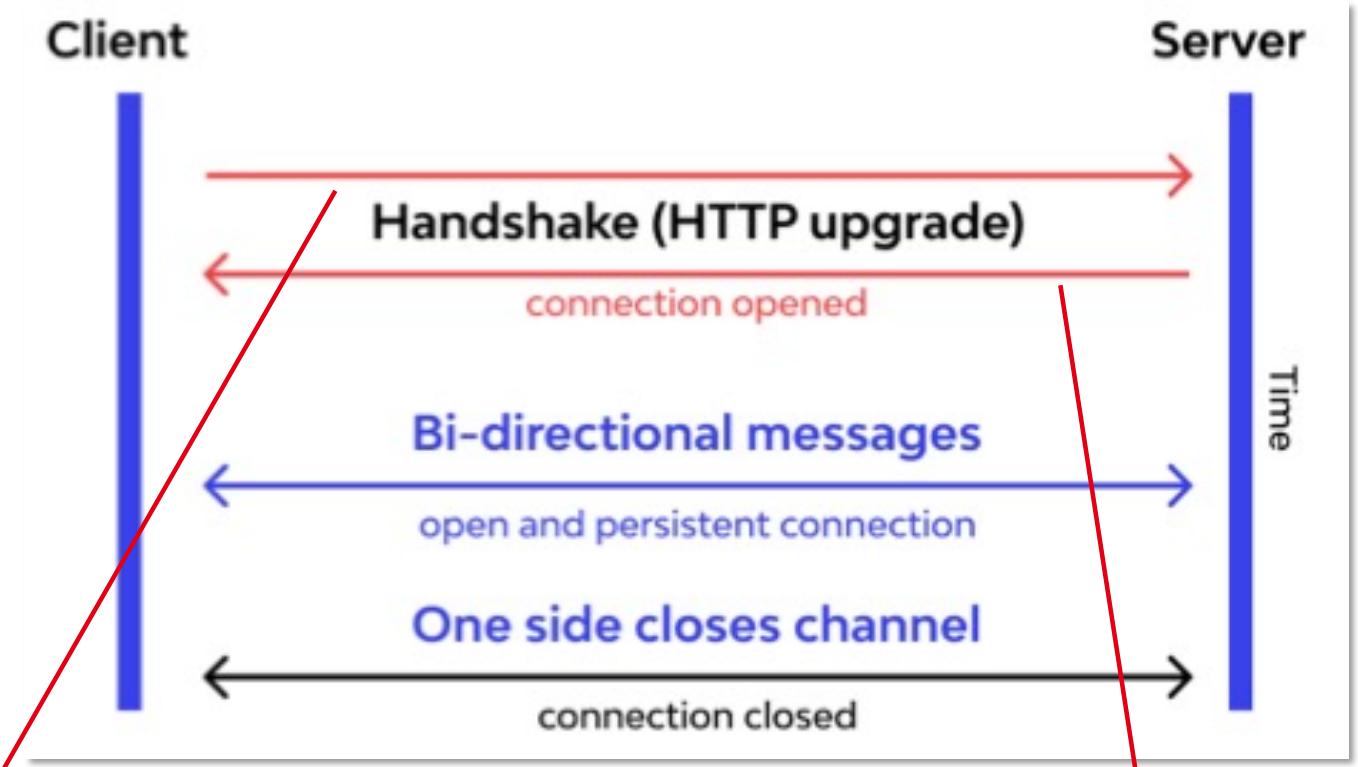
Advantages:
- *real-time communication*
- *efficiency*
- *low latency*

Disadvantage:
- *more complex than HTTP*

*Draft RFCs "**R**equest **F**or **C**omments"*

*Draft RFC 6455 **standardized** WebSockets in 2011. Draft RFC 7692 (2015) introduced **compression** on a per-message basis (DEFLATE algorithm).*

WebSocket Handshake




```
GET ws://example.com:8181/ HTTP/1.1
Host: localhost:8181
Connection: Upgrade
Pragma: no-cache
Cache-Control: no-cache
Upgrade: websocket
Sec-WebSocket-Version: 13
Sec-WebSocket-Key: q4xkcO ...  aSOw==
```

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: fA9dgg ... nyDM=y
```

*After the **handshake** (which upgrades the HTTP connection), a full-duplex **socket** connection is in place.*

Java-WebSocket

https://github.com/TooTallNate/Java-WebSocket

```
public class WebSocketClient {
    public       WebSocketClient(URI serverUri);

    public void  connect();

    public void  onOpen(ServerHandshake handshake);
    public void  onClose(int code, String reason);
    public void  onMessage(ByteBuffer message);
    public void  onMessage(String message);
    public void  onError();

    public void  send(Bytebuffer message);
    public void  send(String message);

    ...
}
```

Address of the WebSocket server.

Connect to the WebSocket server.

EventHandlers

Callback methods when an event has happened on the socket.

Sending a message to the peer.

*The task of creating a separate Thread to receive the messages from the peer is **hidden** from the user of the library.*

```
public void connect() {
    connection = new WebSocketClient(serverUri) {
        public void onOpen(ServerHandshake handshake) {
            logger.info("Connected to AMP: " + getURI());
            adapterCore.onOpen();
        }

        public void onClose(int code, String reason, boolean remote) {
            logger.info("
                "
            adapterCore.o
        }

        public void onMessage(ByteBuffer bytes) {
            adapterCore.handleMessage(bytes);
        }

        public void onError(Exception ex) {
            logger.info("Exception occurred: " + ex);
            ex.printStackTrace();
        }

        public void connect() {
            addHeader("Authorization", "Bearer " + token);
            super.connect();
        }
    };

    connection.connect();
}
```

*SmartDoor adapter in **Java**. The **BrokerConection**'s **`connect`** method registers all callback methods.*

*The **adapterCore** parses and handles the message from AMP.*

AMP's token is added to the header.

Connect to server and handshake.

Protobuf

https://developers.google.com/protocol-buffers

*Excellent **tutorials** for many programming languages.*

- Protocol Buffers (**Protobuf**): serializing structured data
  - language neutral (C++, C#, Go, Java, Python, Ruby, etc.)
  - platform neutral (Windows, macOS, Linux)
  - free, open-source
  - alternative to JSON, XML

*Adapters at TFS do **not** use the latest .proto files.*

- Forwards **compatible**: never change, only add fields.
- Fast processing, **efficient** binary encoding.
- Message are described by a schema: **.proto** file.
  - Source code in target language is **generated** with **protoc**. E.g.
    - create Protobuf objects
    - unpack Protobuf objects

.proto for **Message**

This is the message which is sent over the WebSocket connection, in binary form.

```
message Message {
  message Reset {}
  message Ready {}
  message Error {
    string message = 1;
  }

  oneof type {
    Error error = 1;
    Announcement announcement = 2;
    Configuration configuration = 3;
    Label label = 4;
    Reset reset = 5;
    Ready ready = 6;
  }
}
```

```
message Announcement {
  string name = 1;
  Configuration configuration = 2;
  repeated Label labels = 3;
}
```

Name of adapter (unique).

All labels supported by the adapter.

```
message Configuration {
  message Item {
    string key = 1;
    string description = 2;

    oneof type {
      string string = 3;
      int64 integer = 4;
      float float = 5;
      bool boolean = 6;
    }
  }
  repeated Item items = 1;
}
```

.proto for **Label**

```
message Label {
  enum LabelType {
    STIMULUS = 0;
    RESPONSE = 1;
  }
  LabelType type = 1;

  string label = 2;
  string channel = 3;

  ...

  repeated Parameter parameters = 4;

  uint64 timestamp = 5;
  bytes physical_label = 6;
  uint64 correlation_id = 7;
}
```

Adapter timestamp.

To correlate stimuli.

```
message Parameter {
  message Value {
    ...

    oneof type {
      string string = 1;
      int64  integer = 2;
      double decimal = 3;
      bool   boolean = 4;
      uint64 date = 5; // seconds
      uint64 time = 9; // nanoseconds

      Array array = 6;
      Hash struct = 7;
      Hash hash_value = 8;
    }

    string name = 1;
    Value value = 2;
  }
```

```
message Array {
  repeated Value values = 1;
}

message Hash {
  message Entry {
    Value key = 1;
    Value value = 2;
  }
  repeated Entry entries = 1;
}
```

*Added **after** the initial version of the protocol.*

*The **names** of the attributes are used for the **getters** and **setters** of the objects.*

protoc on label.proto

```
$ protoc --java_out=./.. label.proto
```

Creates ./PluginAdapter/Api/LabelOuterClass.java

[Other programming languages might work slightly different.]

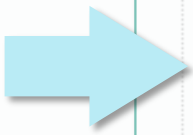
```
message Label {
  enum LabelType {
    STIMULUS = 0;
    RESPONSE = 1;
  }
  LabelType type = 1;

  string label = 2;
  string channel = 3;

  ...

  repeated Parameter parameters = 4;

  uint64 timestamp = 5;
  bytes physical_label = 6;
  uint64 correlation_id = 7;
}
```



LabelOuterClass.java

```
public class Label {
    Label.LabelType                 getType();
    String                          getLabel();
    String                          getChannel();
    List<Label.Parameter>           getParametersList();
    long                            getTimestamp();
    ByteString                      getPhysicalLabel();
    long                            getCorrelationId();
}
```

*For the Label**Builder** there are **setters** (setType, setLabel, etc.) to set the various fields of Label.*

*Each programming back- end (C++, Go, etc.) uses **different approaches** for the unpacking, creation, etc. of messages.*

Protobuf – usage

Java

constructing Protobuf objects

```
Label.Builder builder = Label.newBuilder()
    .setType(LabelType.STIMULUS)
    .setLabel('open')
    .setChannel('door');
Label label = builder.build();
```

*Java uses **builders** to construct Protobuf object.*

encoding: Protobuf to binary

```
private void sendMessage(Message message) {
    byte[] bytes = message.toByteArray();
    brokerConnection.send(bytes);
}
```

accessing fields

```
Messsage message = ...
Label label = message.getLabel();
String name = label.getLabel();
String channel = label.getChannel();
List<Label.Parameter> params =
    label.getParametersList();
```

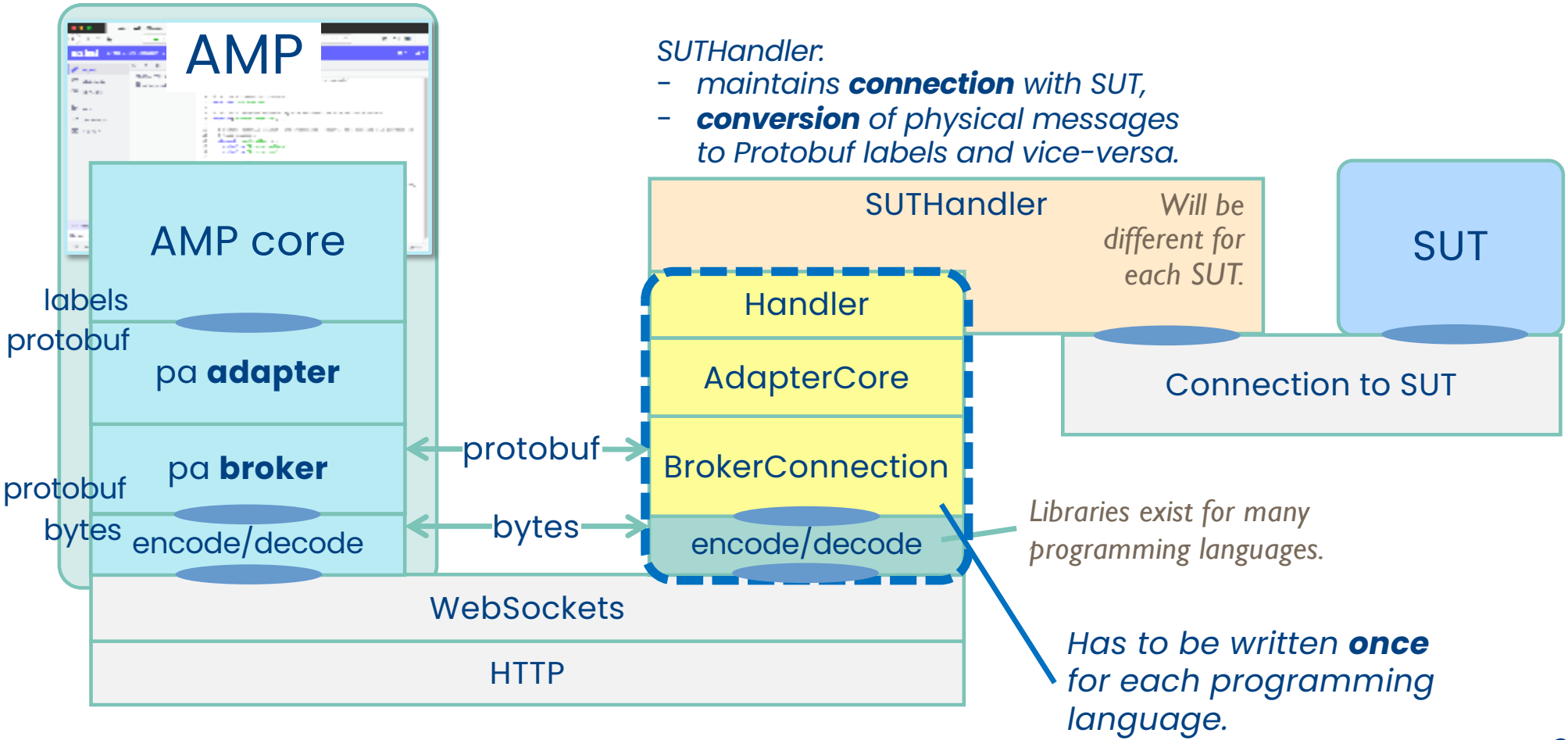
decoding: binary to Protobuf

```
public void handleMessage(ByteBuffer bytes) {
    Message message =
        Message.parseFrom(bytes.array());
    ...
}
```

Class method.

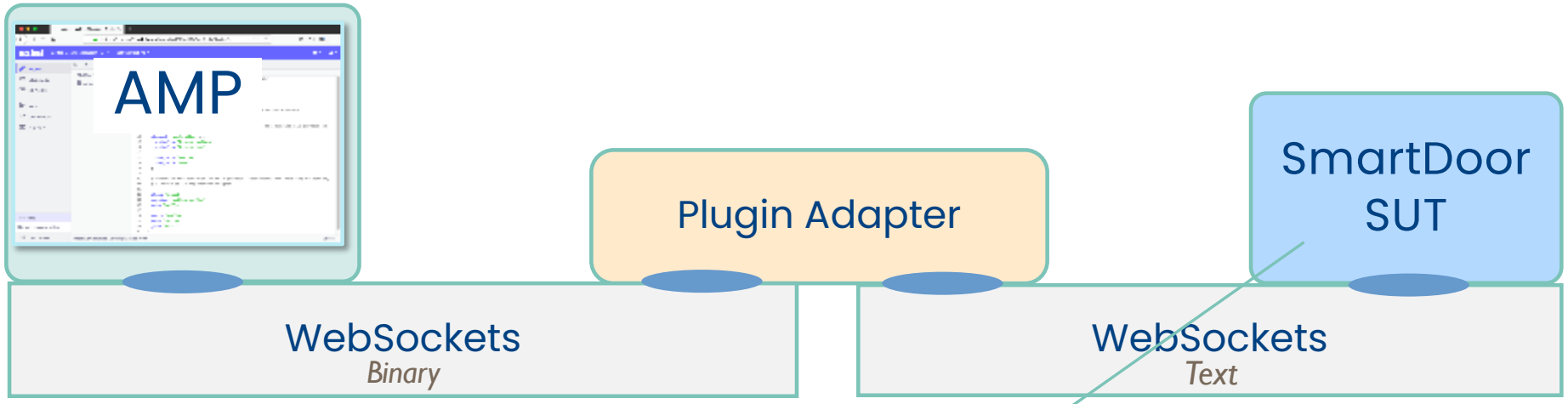
AMP – Plugin Adapters

*Current architecture for Axini's in-house **Ruby** plugin adapters.*



Adapter for SmartDoor SUT

Laboratory Exercise



SmartDoor SUT talks in UPPERCASE!

The SmartDoor SUT is a standalone SUT from the 'SmartDoor' case study.

- commands & responses, e.g. OPEN, OPENED, CLOSE, INVALID_COMMAND
- LOCK with passcode
- different **manufacturers** can be selected: Axini, Besto, Logica, OnTarget, etc.

SmartDoor SUT

- WebSocket **server**
  - No authentication.
  - Only **text** messages.

URI of SUT: `ws://localhost:3001`
But can be configured when started.

- All messages in **UPPERCASE**
  - OPEN CLOSE CLOSED INVALID_COMMAND
- **Parameter** separated using ':' (no spaces)
  - LOCK:1234 UNLOCK:57291
- Extra **RESET** command
  - Resets SmartDoor to **initial state**.
  - optional **manufacturer** parameter
    - RESET:Besto RESET:Axini

After a **RESET** command, the SmartDoor SUT will respond with **RESET_PERFORMED**. This response should (probably) not be sent back to AMP.

The **SmartDoor SUT** is distributed as a (compiled) Java program and an Unix shell script and can be started with:
`./run-smartdoor-sut.sh`

*You can **check** this using the supplied **Java** adapter.*

Testing with AMP

- Terminal: start standalone SmartDoor **SUT**

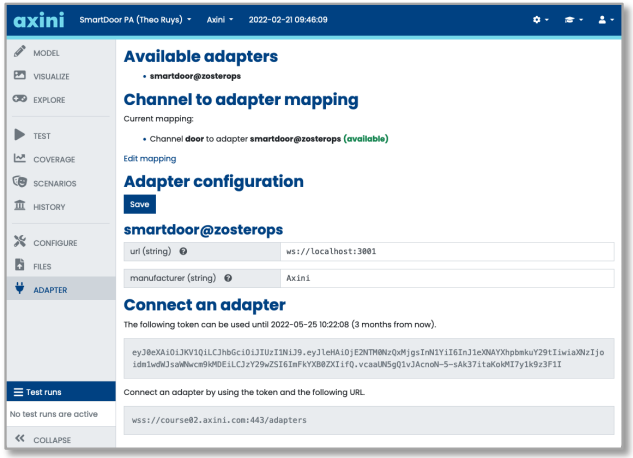
  ```
  $ ./run_smartdoor_sut.sh
  ```

- Terminal: start plugin **adapter** (e.g. Java adapter)

  ```
  $ ./adapter
  ```

  You may have to update AMP's token.

- **AMP** @ course02.axini.com
  - Select "SmartDoor PA (Your Name)" project.
  - Adapter page: attach to your plugin adapter.
  - Start test run.



§1. First Steps

To get a feeling of the two 'new' technologies.

***Building blocks** for actual adapter!*

§ 1.1 Create **Protobuf** messages; use the **protoc** generated source files to create some messages:

- **Label** for stimulus 'open'
- empty **Configuration**
- **Configuration** with single item
- **Message** object containing a Label

§ 1.2 WebSocket connection to SmartDoor SUT.

SmartDoor SUT is also a WebSocket server.

- connect to the SmartDoor SUT
- send a RESET message
- observe the RESET_PERFORMED message

*Secure connection: **avoids problems** with proxies, caches, firewalls.*

Secure WebSocket: **wss**

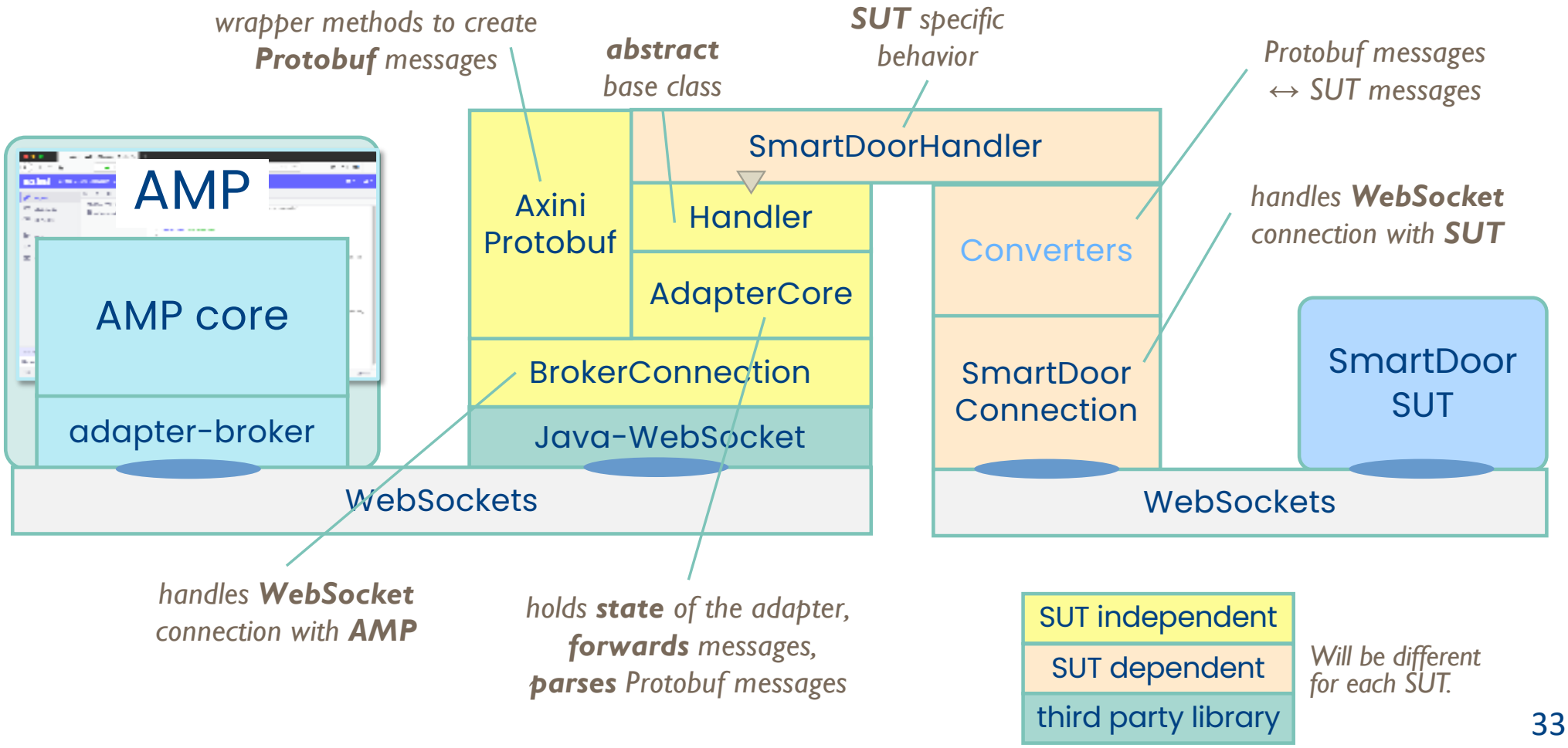
- The **wss** protocol establishes a WebSocket over an **encrypted SSL/TLS** connection (i.e., over **https**).
- Some libraries **automatically** switch to an **encrypted TLS** connection on the URI scheme **"wss://"**.
  - For others (most notably C/C++), you have to **upgrade** the socket connection yourself. Fortunately, many examples exist.
    - *Otherwise, check how libraries of **other** programming languages accomplish this.*

§ 2. WebSocket connection to AMP

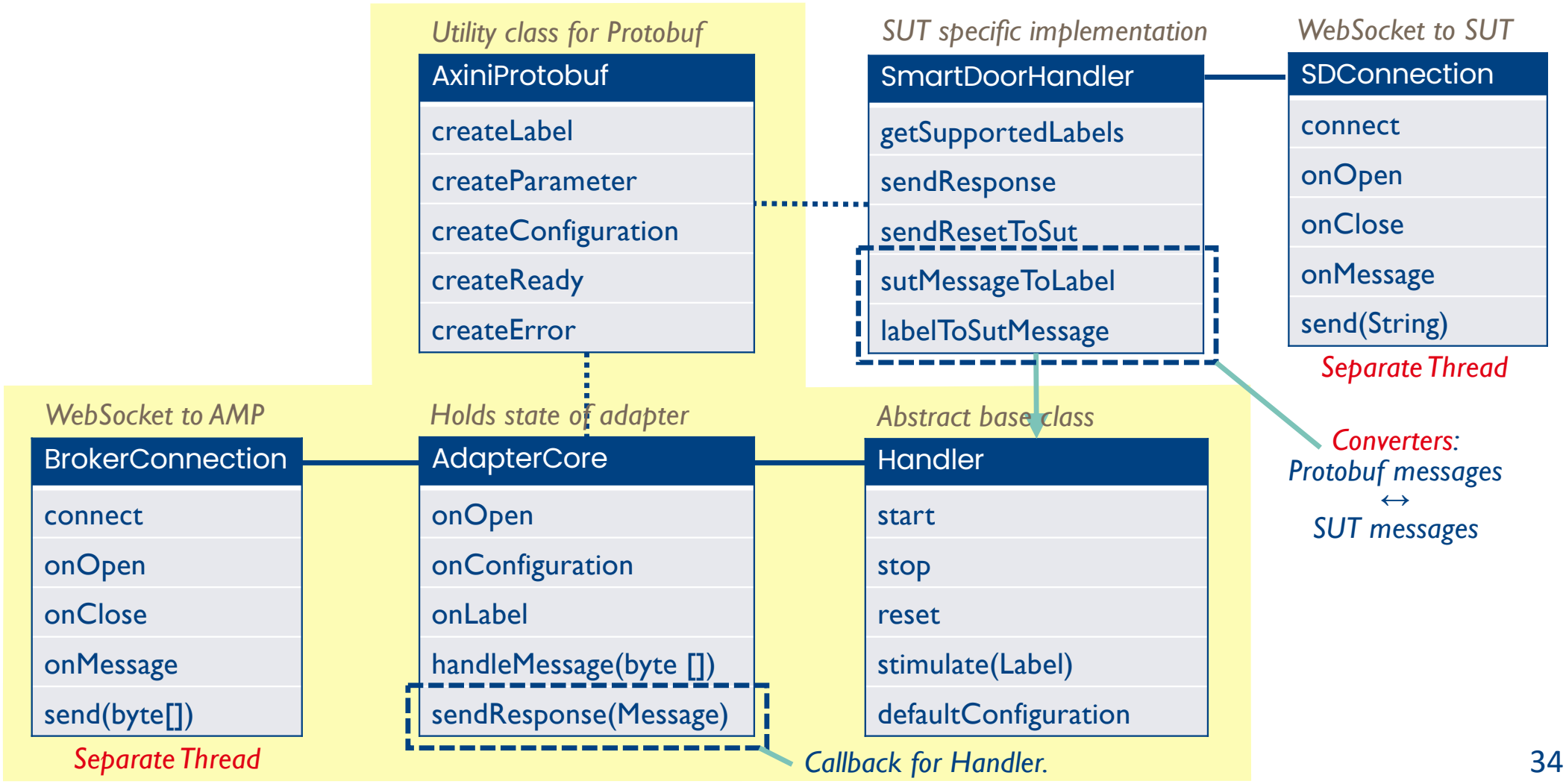
- **Connect** to AMP
  - AMP's adapter-broker is a WebSocket **wss** server.
  - Authorization **token** is required for handshake.
- Send **announcement** to AMP
  - Create a Protobuf **Announcement** Message object
    - name, configuration (empty), labels (empty)
  - Send **Announcement** Message as **binary** message.
- Start test: receive **configuration** from AMP
  - **Observe** the Protobuf message from AMP.
  - Ensure that the message is a **Configuration** message.

AMP – Plugin Adapter for SmartDoor SUT

Java *adapter*



Adapter – Class Diagram

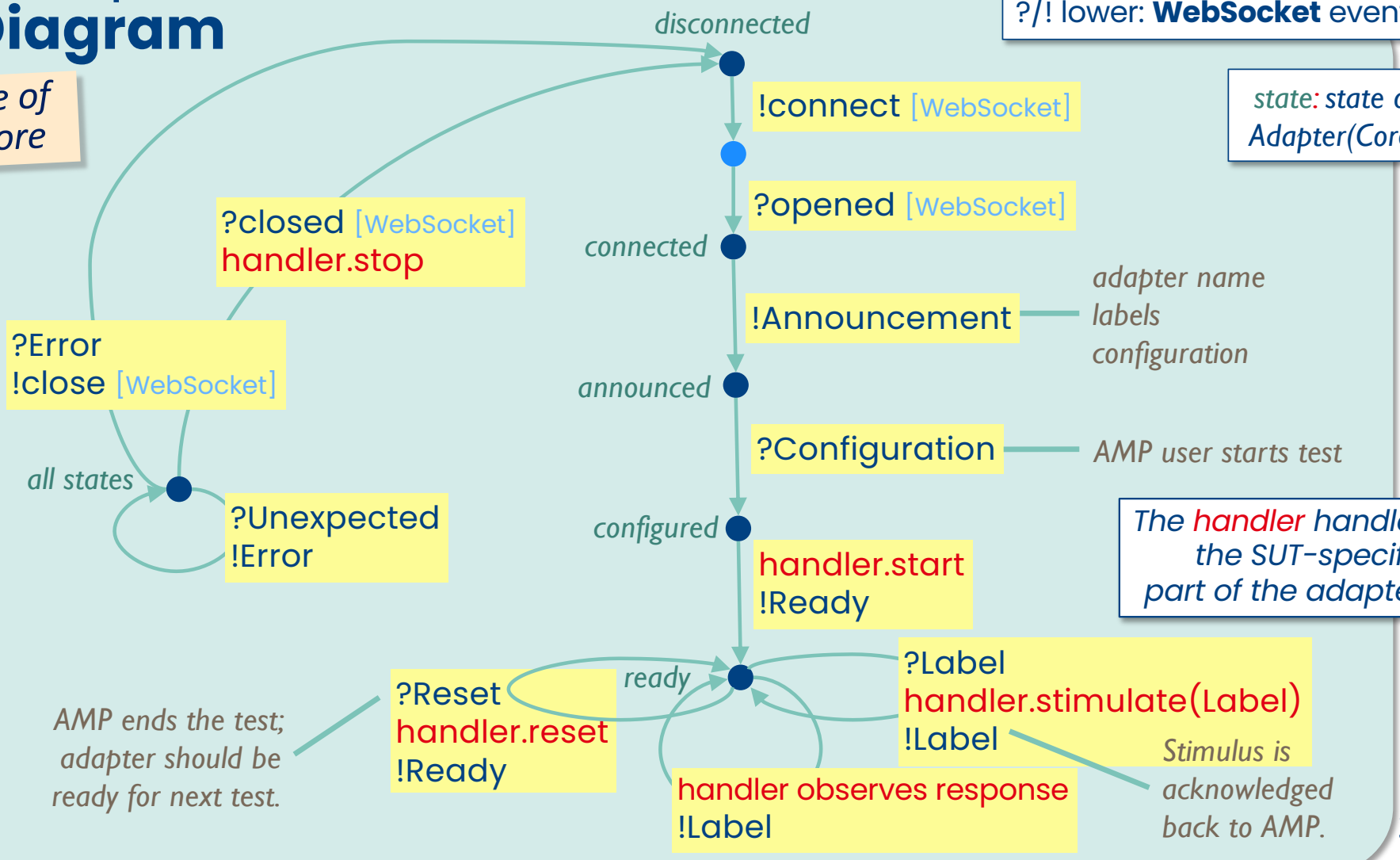


Adapter
State Diagram

= structure of AdapterCore

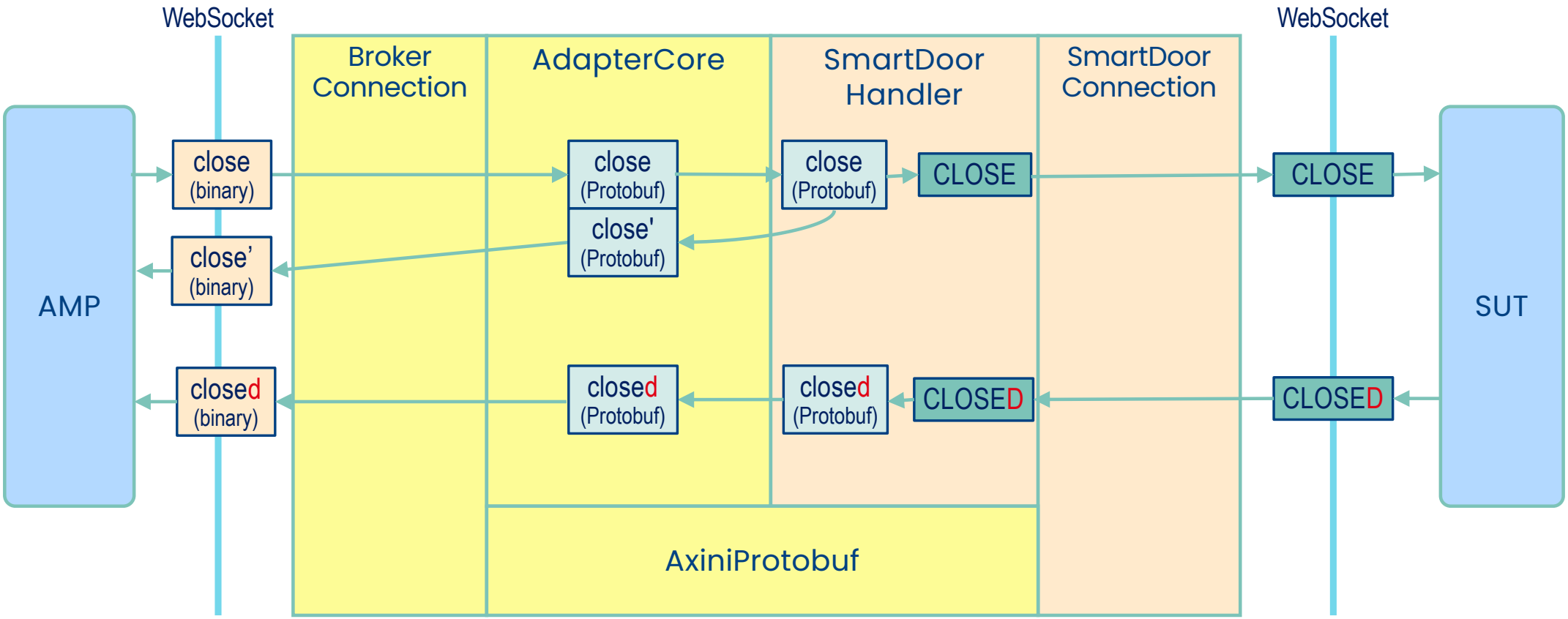
?/! Capitalize: **Protobuf** label
?/! lower: **WebSocket** event

state: state of Adapter(Core)

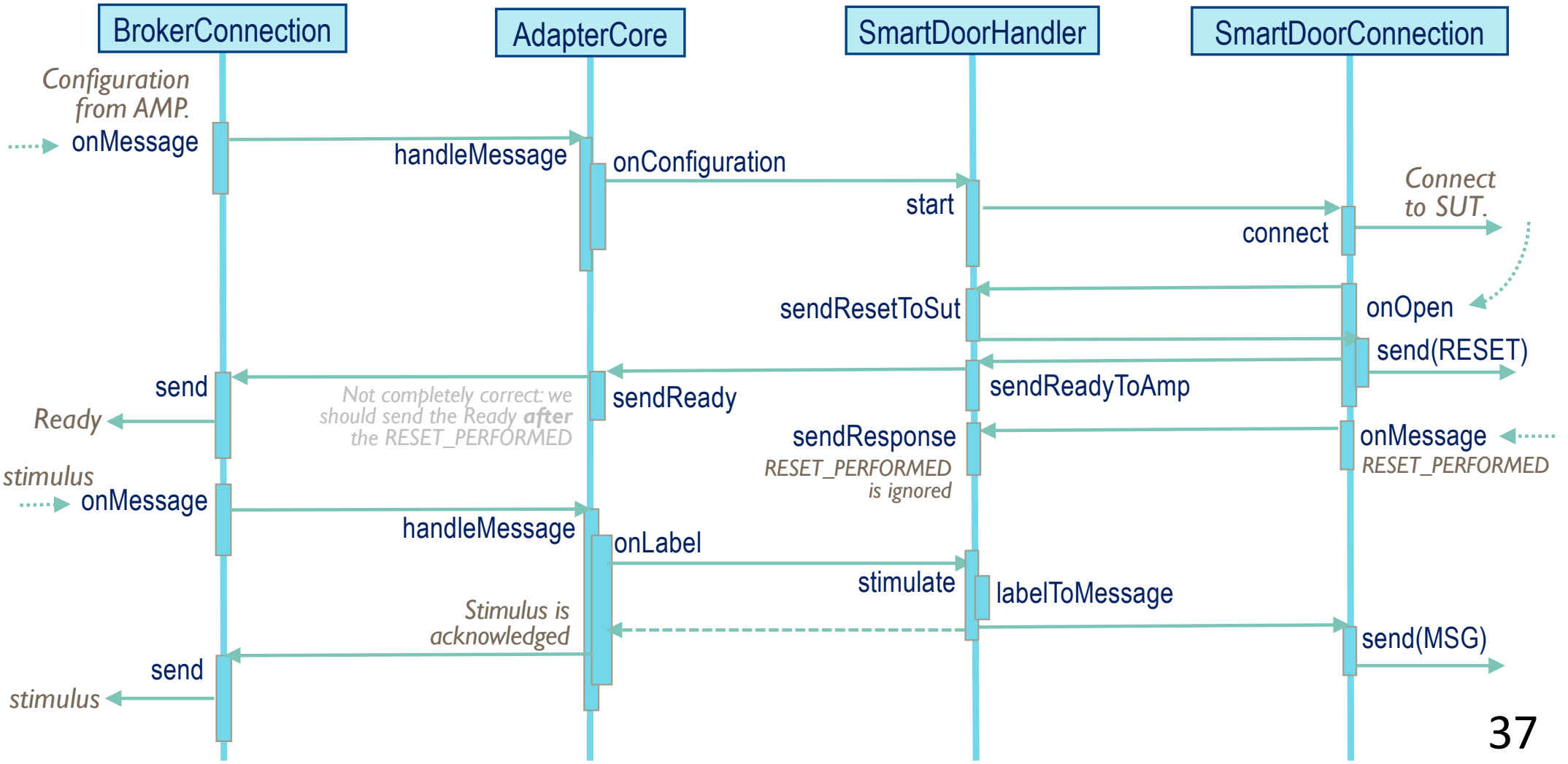


Flow of messages

***close** and **closed** are **labels** of the **SmartDoor**!*



configuration + stimulus



3. Handler, BrokerConnection and AdapterCore

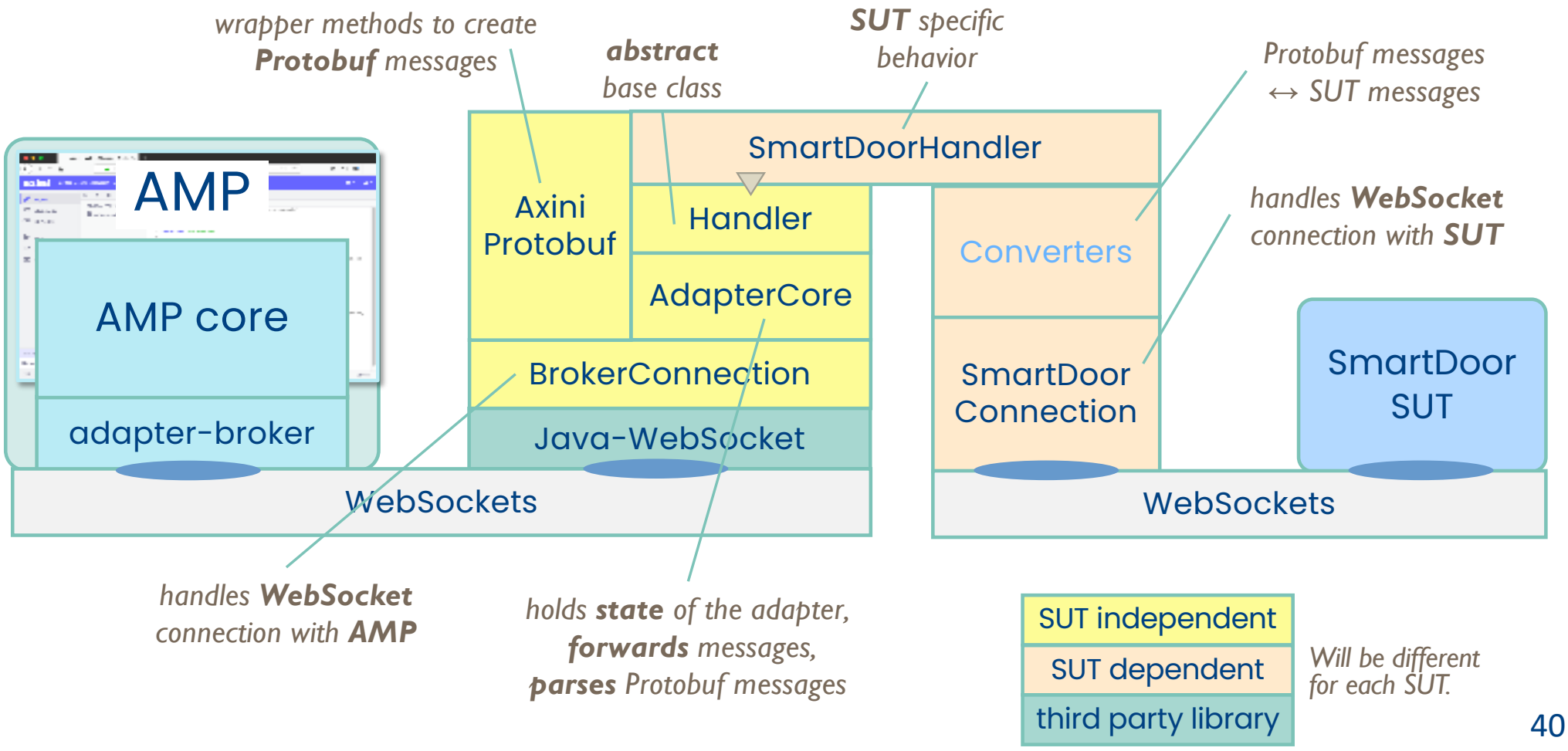
- **Handler**
  - **abstract** base class
  - add **abstract methods** which each SUT-specific SUT should implement: start, stop, reset, stimulate.
- **BrokerConnection**
  - **connect** to AMP: method connect
  - implement **callback** methods: onOpen, onClose, onError, onMessage
- **AdapterCore**
  - implements **state machine**
  - **happy flow**, no error situations (from AMP)
  - calls **abstract methods** of Handler object

- The BrokerConnection knows the AdapterCore.
- The Handler knows the AdapterCore.
- The AdapterCore knows them both.

Wrapping Up

AMP – Plugin Adapter for SmartDoor SUT

Java *adapter*



Configuration of the adapter (or SUT)

- When **starting** the adapter
  - command-line parameters
  - configuration file (JSON, yaml, XML, etc.)
- Using **AMP's Configuration** (for the AMP user)
  - The adapter sends the "default" configuration in the announcement.
  - When test starts, AMP will set the modified configuration.
- Using **virtual** stimuli and responses
  - **not real labels**, will not be passed to the SUT
  - but might change something with the SUT
  - be careful: the adapter now gets **added state**

Adapter **Best Practices** (1/2)

- A single adapter can service **multiple channels**.
  - *Typically, an adapter implements a single interface type (e.g., JMS, SCTP, COM) with the SUT; but on this interface it can offer multiple channels.*
- **Stimuli** should be offered to the SUT in the **same order** as they were selected by AMP.
- **Responses** should be serviced in a **separate thread**.
  - *Do **not use** a single threaded design where the availability of responses is done by **polling**.*
- **Adapters** should **generate** the **AML channel** signatures.
- Adapters should **not** implement **additional queueing** functionality, if this is **not present** at the **SUT**'s interface.

Adapter **Best Practices** (2/2)

See also § 4.5 of "Effective AML".

- Label and channel **names**: match the **specifications**.
- Adapters should **not track state** while testing.
- Adapters should **not check** any **business logic**.
- Try to use **symmetrical converters**
  - support all labels in both directions

*Hard to do for the **TFS** adapters!?*

- **Physical labels** should be **human readable**, if possible.
- Instead of using AMP's configuration, you *could* use **virtual stimuli** to (re)set the SUT.

*Adapters should be as "**thin**" as possible, only:*
- ***manage** the **connection** with the SUT*
- *message/label **conversions***

Closing Remarks

- Try to **complete** your plugin adapter! ;-)
- We hope that the one-day training was **useful**.
- Axini is available for further **help**, of course.
- Remarks, suggestions, etc. to ruys@axini.com.
- In the (near) future, Axini will host a **collection** of example plugin-adapters (GitHub-like).