
ORACLE®

JavaOne™

ORACLE®

WebSocket API.NEXT

Pavel Bucek (pavel.bucek@oracle.com)

Oracle
September 29, 2014

JavaOne
ORACLE

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

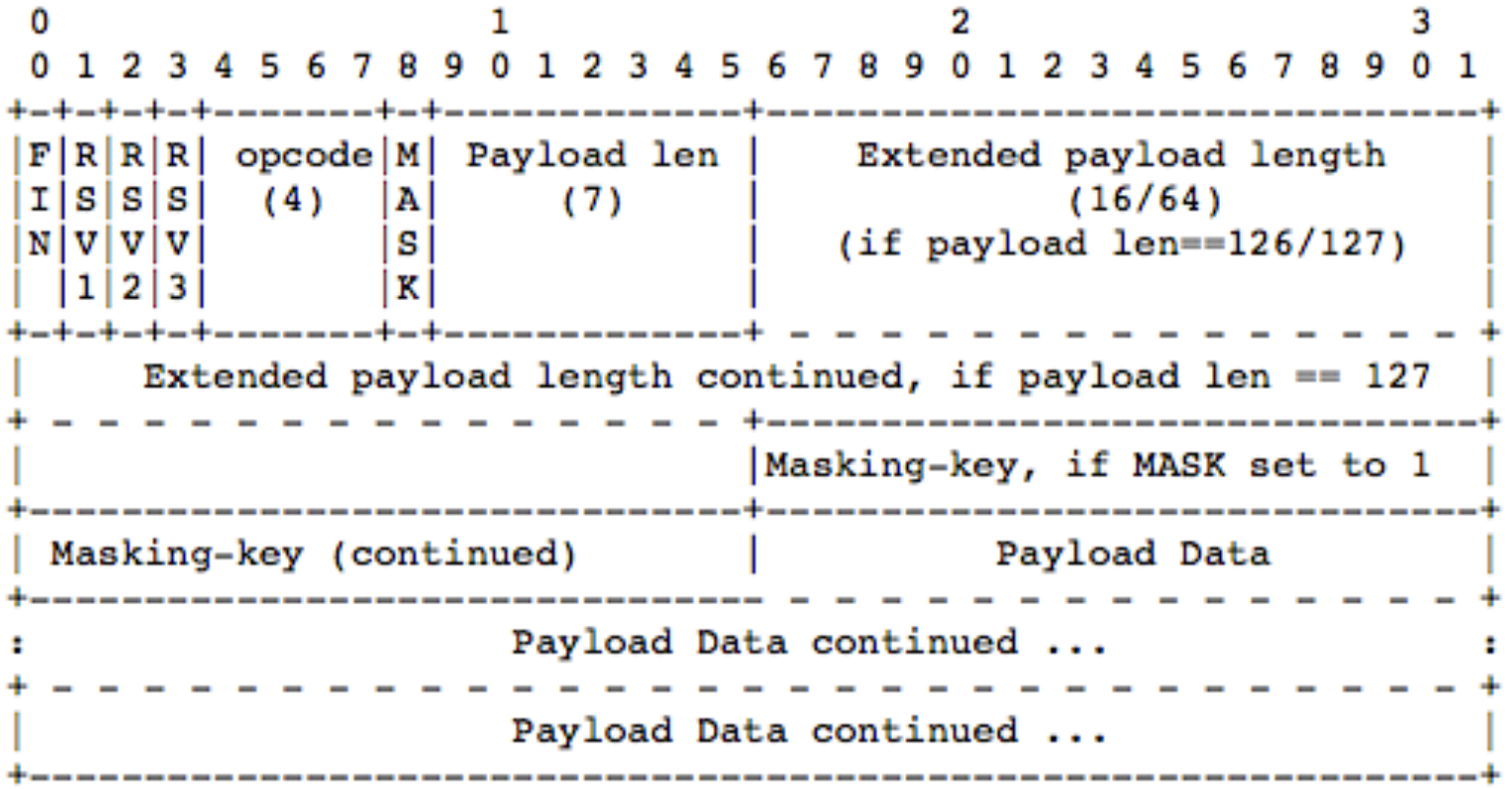
Program Agenda

1. Short intro about JSR 356 – Java API for WebSocket
2. WebSocket API 1.1
3. WebSocket.NEXT
4. CDI, Frame API, Extensions, SubProtocols, Java EE integration, Client API enhancements, smaller issues (broadcast, clustering, security, …)

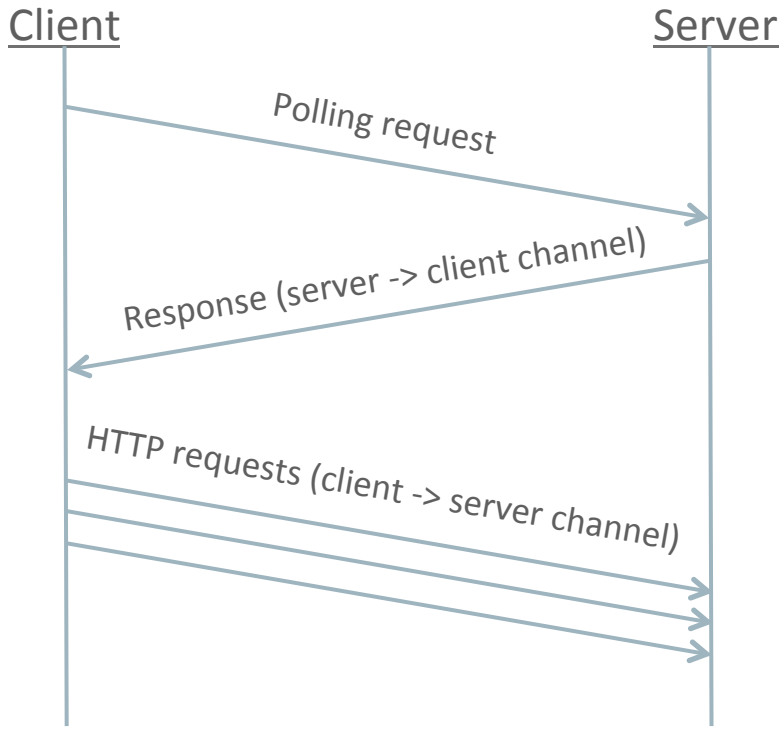
WebSocket protocol

- RFC 6455 (December 2011)
- Two-way communication, replacement for Long-polling
  - Better resource utilization
- Extensions, SubProtocol
- WebSocket handshake uses HTTP, connection is then **UPGRADE**d to “websocket” protocol
  - No HTTP since this point, only WebSocket frame are allowed to be sent or received
  - Connection can be closed using Close frame or just closing underlying TCP connection

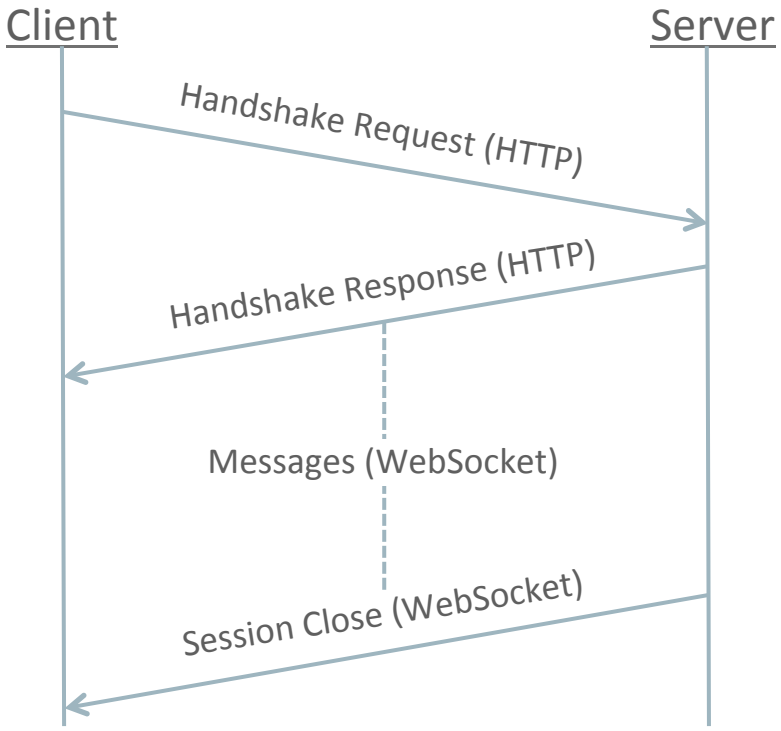
WebSocket protocol - Frame



WebSocket protocol



Long polling



WebSocket

Java API for WebSocket

- JSR 356 – Part of Java EE 7
  - 1.0 (May 2013)
  - 1.1 (August 2014)
- Annotated and programmatic way how to deploy and access WebSocket endpoints
- Event-driven model - @OnOpen, @OnMessage, @OnError, @OnClose
- Encoders/Decoders, Path/Query parameter handling, Handshake headers interceptors, CDI integration, …

Java API for WebSocket – Annotated Endpoint

```java
@ServerEndpoint("/echo")
public class EchoEndpoint {

    @OnOpen
    public void onOpen(Session session) throws IOException {
        session.getBasicRemote().sendText("onOpen");
    }

    @OnMessage
    public void echo(Session session, String message) throws IOException {
        session.getBasicRemote().sendText(message + " (from your server)");
        session.close();
    }

    @OnError
    public void onError(Throwable t) {
        t.printStackTrace();
    }
}
```

Java API for WebSocket – Programmatic Endpoint

```
public class EchoProgrammaticEndpoint extends Endpoint {
    @Override
    public void onOpen(final Session session, EndpointConfig config) {

        session.addMessageHandler(String.class, new MessageHandler.Whole<String>() {
            @Override
            public void onMessage(String message) {
                try {
                    session.getBasicRemote().sendText(message + " (from your server)");
                    session.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }

            }
        });
    }

    @Override
    public void onError(Session session, Throwable thr) {
        thr.printStackTrace();
    }
}
```

WebSocket 1.1

- JSR 356 Maintenance release
  - August 2014
- WEBSOCKET_SPEC-226: Session.addMessageHandler(…) and lambdas
- Impact: JDK8, Nashorn, possibly any other JVM language

WebSocket 1.1

WEBSOCKET_SPEC-226

- `Javax.websocket.Session#addMessageHandler(MessageHandler)`
  - `MessageHandler` contains two sub interfaces, `Whole<T>` and `Partial<T>`
  - WebSocket implementation must know the generic type of added MessageHandler to be able to match appropriate javax.websocket.Decoder
- Solution: add methods which provide the type directly as a parameter
  - `Javax.websocket.Session#addMessageHandler(Class<T>, MessageHandler.Whole<T>)`
  - `Javax.websocket.Session#addMessageHandler(Class<T>, MessageHandler.Partial<T>)`

WebSocket 1.0

```
session.addMessageHandler(
        new MessageHandler.Whole<String>() {
            @Override
            public void onMessage(String message) {
                // process message.
            }
        }
);

// ===== Lambda expression =====

session.addMessageHandler(
        (MessageHandler.Whole<String>) message -> {
            // process message.
        }
);
```

WebSocket 1.1

```
session.addMessageHandler(String.class,
        new MessageHandler.Whole<String>() {
            @Override
            public void onMessage(String message) {
                // process message.
            }
        }
);

// ===== Lambda expression =====

session.addMessageHandler(String.class,
        (message) -> {
            // process message.
        }
);
```

WebSocket 1.1

- Lesson learned: There is no reliable way how to get the generic type (*)
  - Not until generics reification is introduced to the JDK

- (*) during the runtime.

WebSocket.NEXT

- Should be part of Java EE 8
- The scope is not clear yet

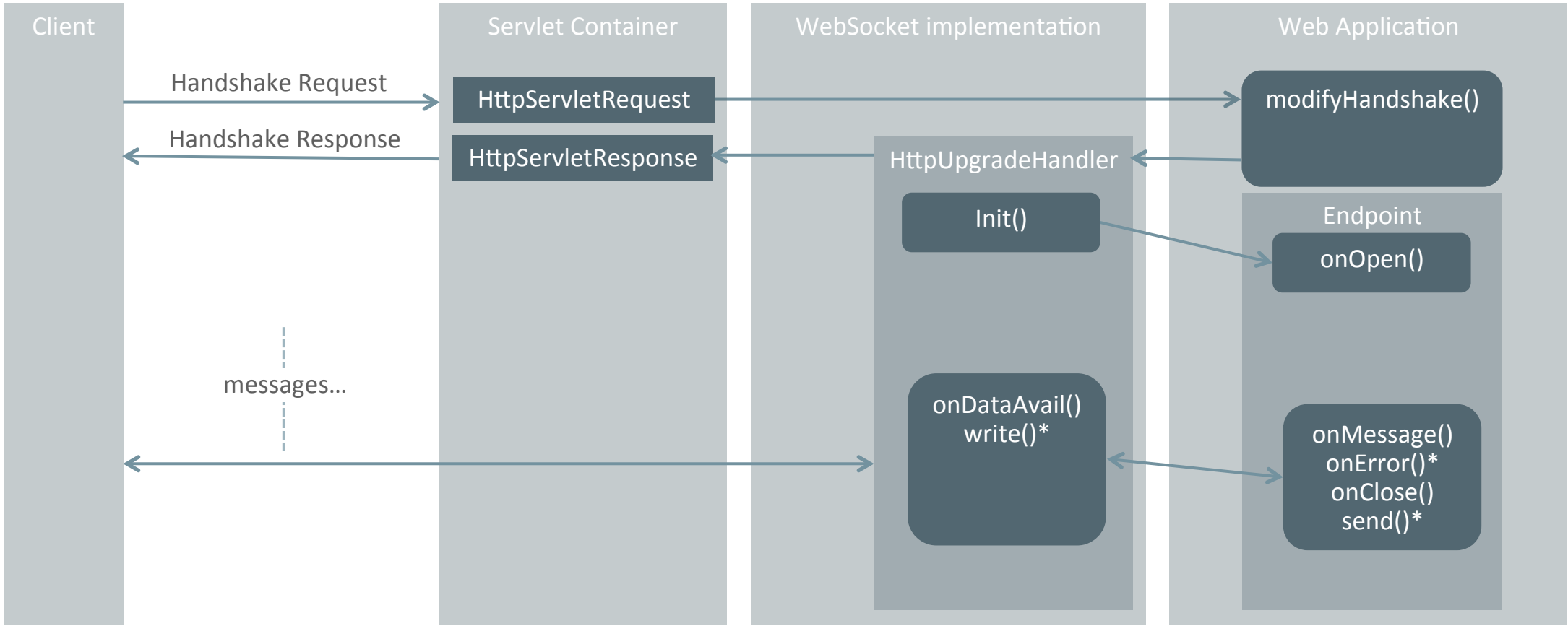
WebSocket.NEXT

CDI integration

- Present in the specification since 1.0
- WebSocket spec depends on Servlet (3.1)
- Issue with CDI scope
  - @RequestScope
  - @SessionScope
  - @ApplicationScope
  - @ConversationScope
- HttpUpgradeHandler (from Servlet 3.1) has CDI scope “undefined”

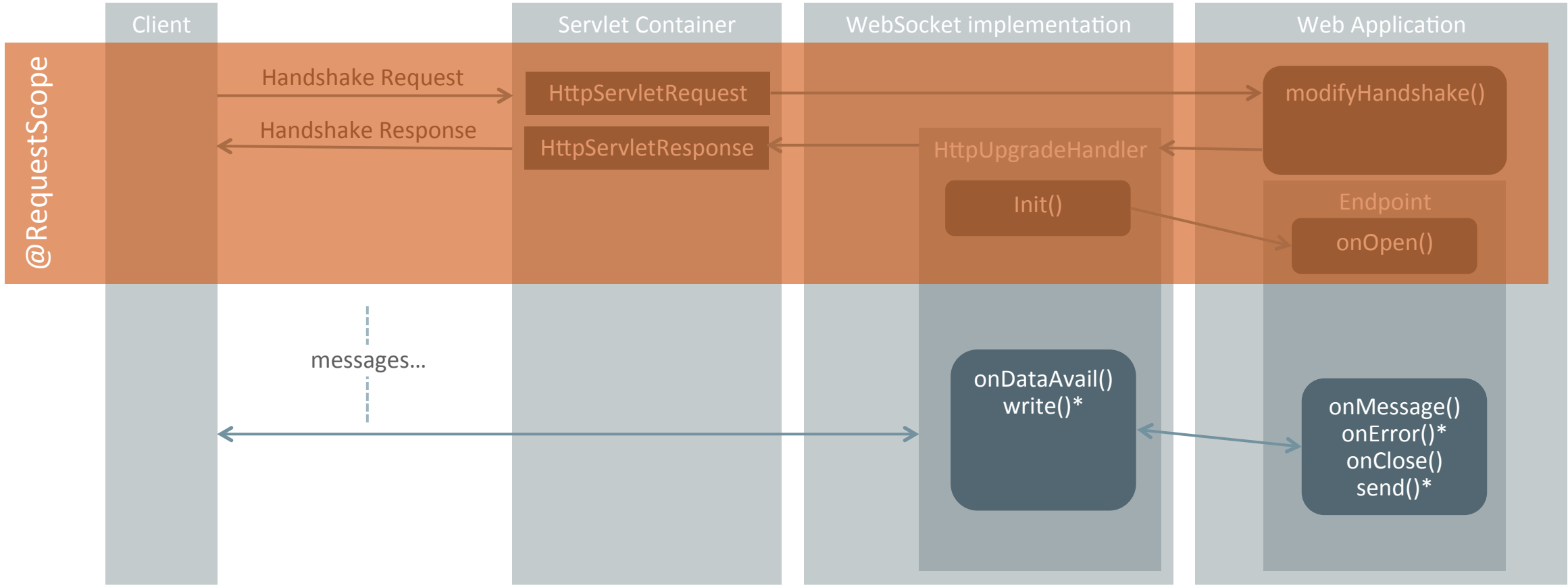
WebSocket.NEXT

CDI integration



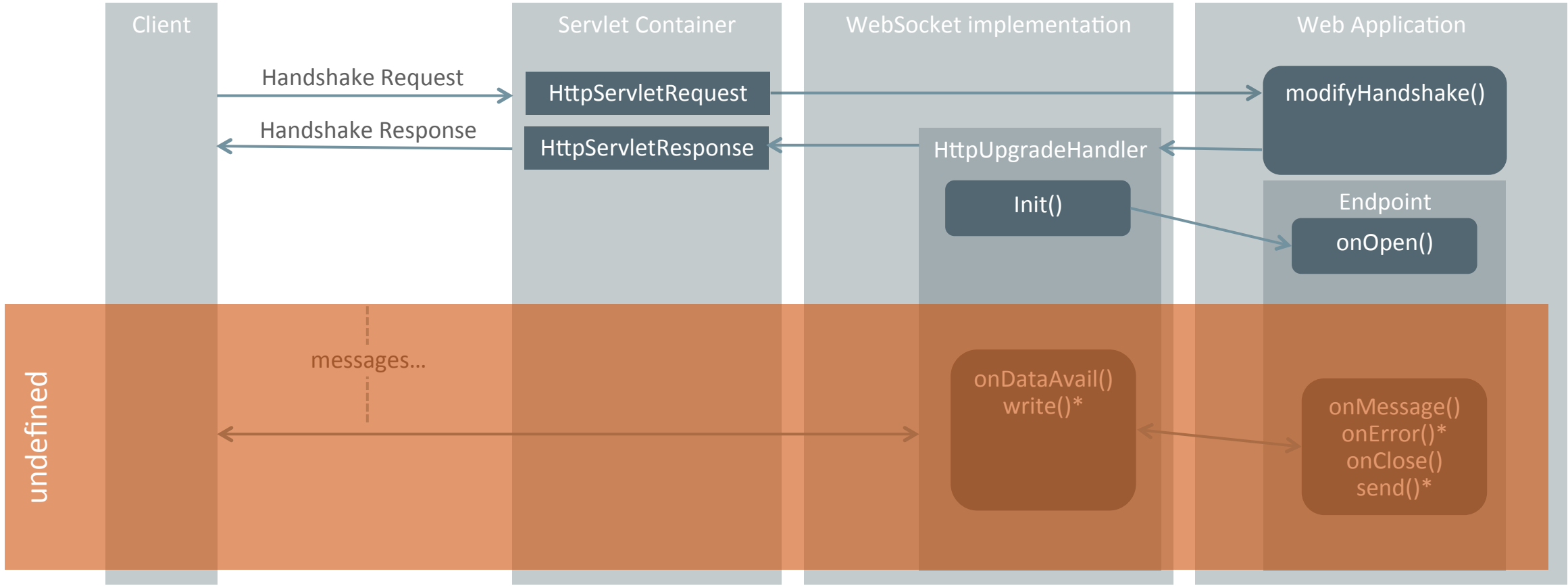
WebSocket.NEXT

CDI integration



WebSocket.NEXT

CDI integration



WebSocket.NEXT

Low level Frame API

- Support for direct handling of WebSocket Frames
  - Direct manipulation with payload or other frame properties
  - Custom encryption, encoding, compression, …
- WebSocket Extension support
  - Currently WebSocket API contains only negotiation-related bits
    - (and not very well executed – not able to negotiate extension parameters, …)
  - Extending current API is possible, but…

WebSocket.NEXT

Low level Frame API – WebSocket Extensions

- Existing WebSocket Extensions proposals:
  - "Compression Extensions for WebSocket" - Submitted to IESG for Publication
  - "A Multiplexing Extension for WebSockets" – expired
- Other option is just state that implementation must/can support some list of extensions
- Common API for Extensions might be hard to craft, since there are no exact rules about what WebSocket Extension might specify.
  - (this extension must be run as first when receiving frame and last when sending, …)

WebSocket.NEXT

Extensions/Message(*) processing

- WEBSOCKET_SPEC-199
  - "Add the ability to define filters and interceptor"
- API for filtering
  - Handshake requests?
    - It may be worth to have better control in handshake process – currently only response headers can be modified, no possibility to deny the request/return HTTP 5xx response
  - WebSocket Frames?
  - Decoded/extracted messages (Frame payload)?

WebSocket.NEXT

SubProtocol support

- Existing WebSocket SubProtocol proposals
  - The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP) – published – RFC 7118
  - An XMPP Sub-protocol for WebSocket – almost done – Submitted for publication
  - The WebSocket Protocol as a Transport for the Message Session Relay Protocol (MSRP)
  - Use of the WebSocket Protocol as a Transport for the Remote Framebuffer Protocol
  - CoAP over WebSockets

WebSocket.NEXT

SubProtocol support

- WebSocket API supports SubProtocols only for negotiation
  - Which might be OK
- It is not clear what everything might be needed for implementing other protocol (again, depends on the subprotocol specification)
- There seem to be common pattern
  - Using WebSocket as a Socket (or just stream) to send/receive messages/frames of the tunneled protocol
  - We could modify existing cases to suite those needs (Streamed MessageHandlers, RemoteEndpoint#getSendStream())

WebSocket.NEXT

Java EE integration

- JAX-P, JAX-B, JSON-P, JSON-B, …
  - Problem with missing metadata
- Bean validation
- Servlet 4.0 (HTTP/2)
- CDI – injectable managed Client/Session/RemoteEndpoint
- CDI – support external events (like CDI event which invokes broadcast)

WebSocket.NEXT

Client API improvements

- Asynchronous connectToServer method
- Security – HTTP Basic/Digest/custom authentication schemes
- Security – SSL related settings, ideally per client connection
  - Might be problematic to agree on single API, could be solved on different level and shared among all Java EE specification which do provide client API
- HTTP Proxy support
  - (at least state that setting Proxy in java-way should be picked up)

WebSocket.NEXT

Smaller issues (in terms of API changes)

- Broadcast support
  - Introduce something better than iterating over Set<Session>
- Security
  - @RolesAllowed, … - fine grain control for on endpoint level
- Clustering
  - Current API is not very friendly to clustered environment
  - Problematic parts: Session#getContainer(), Session#(add|get|del)MessageHandler, Session#getUserProperties, …
  - Enhancements like intercepting messages sent from other nodes, …

WebSocket.NEXT – Questions?

- WebSocket API 1.1.NEXT
- WebSocket-spec: https://java.net/projects/websocket-spec
  - https://java.net/jira/browse/WEBSOCKET_SPEC
- Reference Implementation: Tyrus https://tyrus.java.net
  - users@tyrus.java.net
  - https://java.net/jira/browse/TYRUS
- Pavel Bucek: pavel.bucek@oracle.com

JavaOne™ ORACLE