Graphical user interfaces in the X stack

Zahra Al-Awadai, Anne Brüggemann-Klein,

Christina Grubmüller, Philipp Ulrich

Technical University of Munich

Outline

Introduction: previous work and context, the **X stack**

- XML everywhere: **implementing** and **modeling** web applications
- current focus on graphical user interfaces (**GUIs**)
- paper explores a number of options for GUIs in the X stack
- today: two highlights

Highlight 1 : Multi-client web applications in the X stack

- Philipp Ulrich

Highlight 2 : Modeling & implementation of event-driven systems with statecharts and SCXML

- Christina Grubmüller

Concluding remarks: summary, evaluation, and further work

First, a few words on GUIs and the X stack

Graphical user interfaces in the X stack

Tasks of graphical user interface (GUI), running in a web browser

- presents application data
- offers user interaction
- does internal processing
- interacts with server component

Focus of this work: **explore** XML technologies for GUIs in web applications (survey)

- XHTML with XForms and SVG, Web Components, WebSocket Element for multi-client applications, SCXML
- Highlights from paper: WebSocket Element, SCXML

Requirements

- data are declarative and encoded in XML
- internal processing within GUI is declarative

Multi-client web applications in the X stack

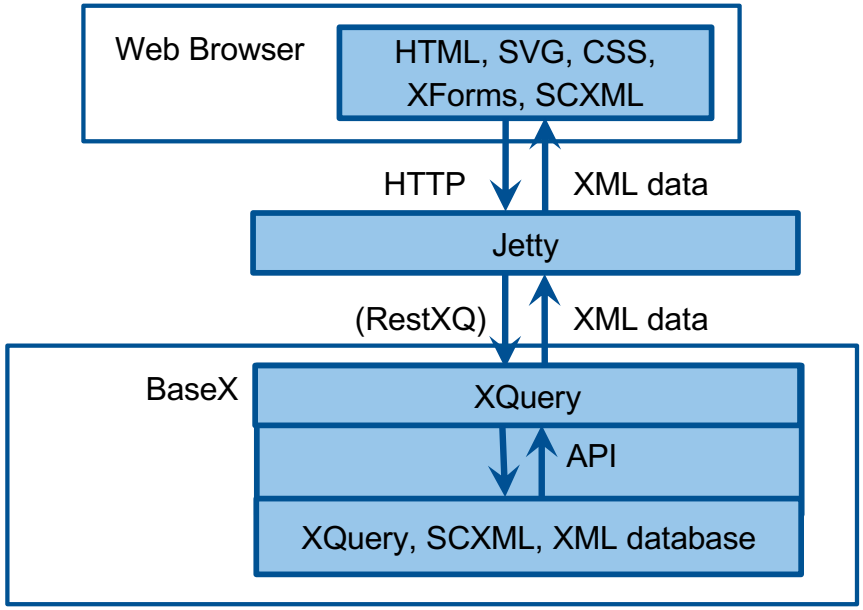
Philipp Ulrich

What are we building: Demo Tic-Tac-Toe

What are multi-client web applications?

Web applications

- Client-server architecture
- The application runs on the server
- The clients use web browsers
- Communication over HTTP



Multi-client web applications

- Multiple clients are communicating through a server with each other
- Requires serverpush → **How?**

Examples

- Games (Blackjack, Tic-Tac-Toe), especially those based on secrets (Poker, Battleship)
- Applications (Chats or collaborative editors)

HTTP - HyperText Transfer Protocol

- Stateless protocol used in web applications
- Defines how messages are constructed and sent
- Utilizes request/response cycles (ping-pong pattern)
  - HTTP requests by the clients get answered with a HTTP response from the server
  - Generally the server cannot send data to the clients without a prior HTTP request
  - Workarounds such as Long- or Busy polling don‘t scale

  → No serverpush possible

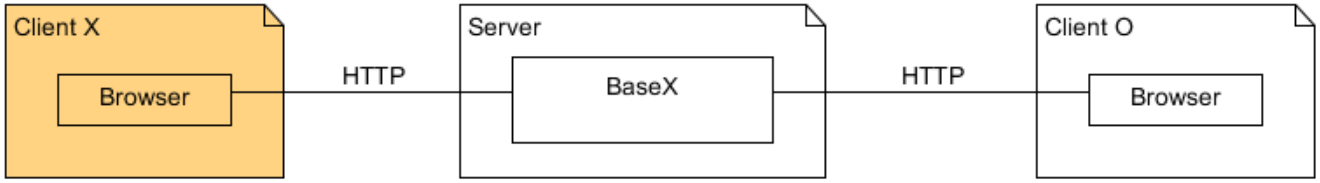
WebSocket protocol

- Enables bidirectional communication between client and server
- Upgrades a regular HTTP connection after a handshake
- Data transmission over a permanent TCP connection
- Less overhead and latency compared to HTTP
- Suitable especially for real-time applications (video, audio chats or multiplayer games)

→ **WebSocket to push data from the server to clients!**

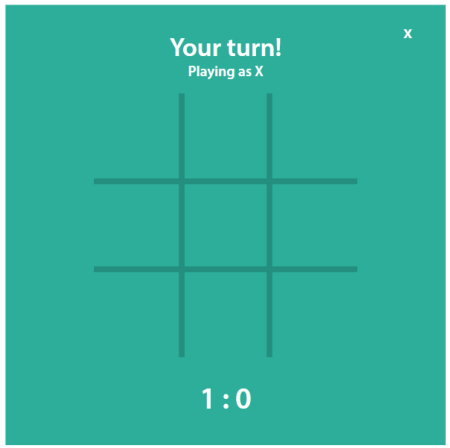
Scenario Tic-Tac-Toe

- Clients on different machines (browsers)
- Server has to push the state of the game to all connected clients → **serverpush**



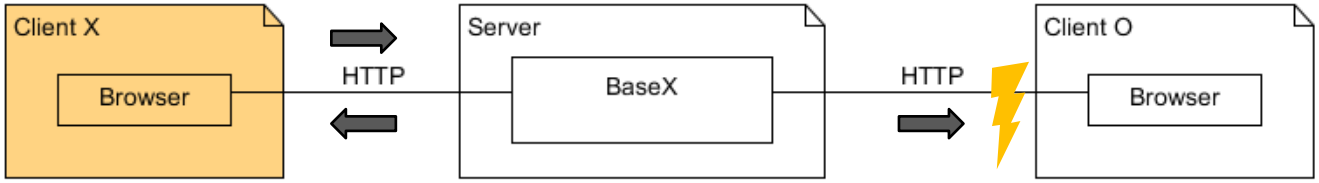
Scenario Tic-Tac-Toe without WebSocket

- Client X wants to set the first X on the board



Client X: State 1

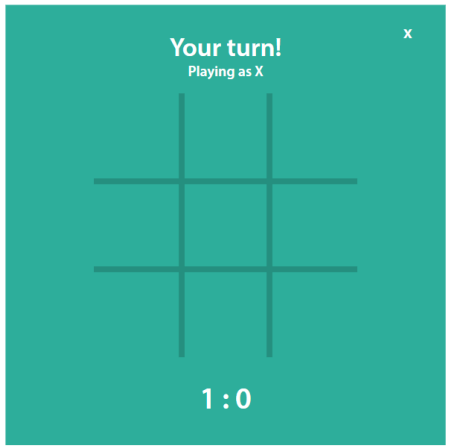
Client X: State 2



→ No serverpush

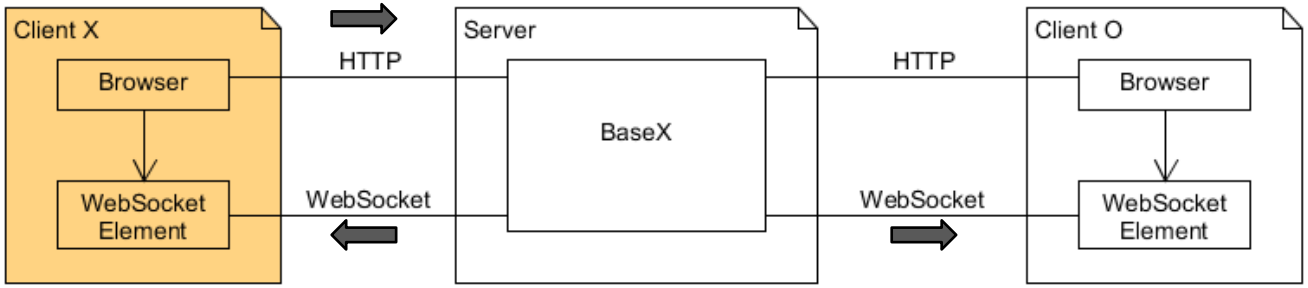
Scenario Tic-Tac-Toe with WebSocket

- Client X wants to set the first X on the board



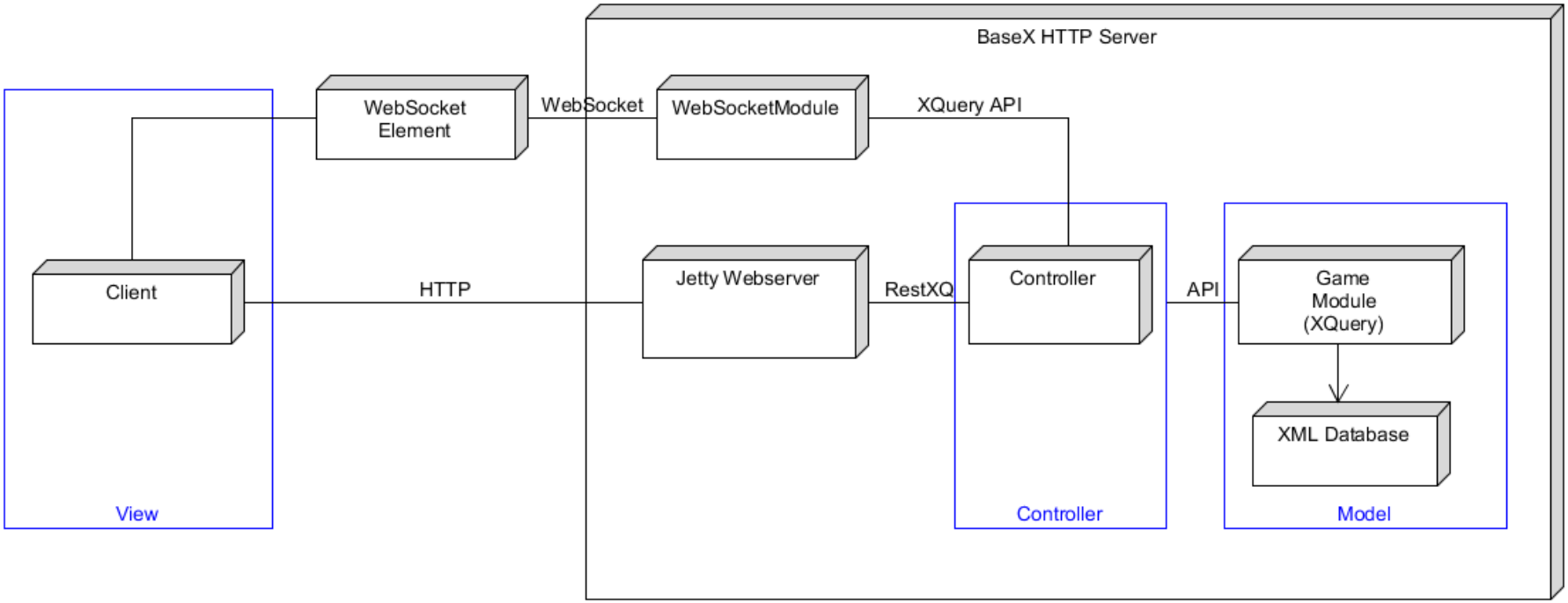
Client X: State 1

Client X: State 2



→ **WebSocket used as response channel**

Architecture



Client side: Websocket Element

A client side WebSocket module

HTML 5 browsers support WebSocket API

Problem

Imperative JavaScript code is necessary to establish the WebSocket connection

- Programming skills required
- Adaption to the web application is necessary
- Code can quickly become complex when multiple connections are established

WebSocket JavaScript Code

```
let url = "ws://localhost:8984/ws/chatWebAPI";
let ws = new WebSocket(url);

ws.onopen = function(event) {
    // Send ping messages regularly to keep the connection alive
    setInterval(function ping() {
        send("ping", "");
    }, 150000);
    // Get the first state of the chat when joining
    let draw_url = url + "/draw";
    $.get(draw_url, function(resp, other) {
        console.log('Get url response: ' + resp);
            if(resp !== null){
                $("#chatWindow").html(resp);
            }
        });
};

ws.onmessage = function(event) {
    $("#chatWindow").html(event.data);
};
```

WebSocket Element

Solution

A declarative Custom Elements HTML element, customizable through parameters, to establish and handle a WebSocket connection

```
<ws-stream id = "myID" url = "ws://localhost:8984/ws"
subscription = "/path" geturl = "/ttt/draw">
Content</ws-stream>
```

- Client side WebSocket interface
- Custom HTML element to build WebSocket connections in a declarative way
- Code for WebSocket handling is hidden in the implementation of the HTML component
- Acts as a container for the received data (= Tic-Tac-Toe game)
- Customizable through the use of parameters

→ On page load, a WebSocket connection is established specified by the parameters

WebSocket Element Code

```
if(this.wsElementParams.url != null){
    let ws = Stomp.client(this.wsElementParams.url, stompParams.protocols);
    ws.debug = function(str) {
        console.log(str + "\n");
      };
    ws.heartbeat.outgoing = stompParams.heartbeatOutgoing;
    ws.heartbeat.incoming = stompParams.heartbeatIncoming;
    ws.reconnect_delay = stompParams.reconnectDelay;

    let stompConnectedCallback = function(){
        console.log("ID: " + streamElement.wsElementParams.id + " - Connected to STOMP server");

        let stompOnMessage = function(message){
            if(streamElement.hasAttribute('xslt')){
                let xsltURL = streamElement.getAttribute('xslt');
                let xslParam = streamElement.getAttribute('xslparam');

                if(xslParam !== undefined){
                    let xslParamsJSON = streamElement.parseXSLTParams(xslParam);
                    console.log("xslParamsJSON: " + JSON.stringify(xslParamsJSON));
                }

                let promise = streamElement.transform(message.body, xsltURL, xslParamsJSON);
                promise.then(function(result) {
                    console.log("Transformed document " + result)
                    $(streamElement.wsElementParams.idString).html(result);
                });
            }

            console.log("ID: " + streamElement.wsElementParams.id + " - MSG: " + message);
            $(streamElement.wsElementParams.idString).html(message.body);
        }
```

Customizing with parameters

```
<ws-stream id = "myID" url = "ws://localhost:8984/ws"
subscription = "/path" geturl = "/ttt/draw">
Content</ws-stream>
```

- **id** – Identification and management of the WebSocket Element
- **url** – Target address to establish the WebSocket connection
- **subscription** – Used to define different channels, client will subscribe to the path
- **geturl** – URL to load a first state into the element via a GET request **[optional]**
- **xslt** – path to a XSL stylesheet for client side transformation **[optional]**
- **xslparam** - Additional parameters for the XSL transformation **[optional]**
- **reconnectDelay** – Delay until a reconnect is attempted after a connection loss **[optional]**

→ **More optional parameters that can be used by more advanced users**

Usage

It is as simple as importing all the necessary dependencies…

```
<script src="/static/tictactoe/JS/jquery-3.2.1.min.js"></script>
<script src="/static/tictactoe/JS/stomp.js"></script>
<script src="/static/tictactoe/JS/ws-element.js"></script>
```

… and declaring the element on your page

```
<ws-stream id = "myID" url = "ws://localhost:8984/ws"
subscription = "/path" geturl = "/ttt/draw">
Content</ws-stream>
```

Conclusions & advantages

- Declarative way of using WebSocket
- Simple usage and customizable through parameters
- A lot shorter than writing imperative JavaScript code to establish a WebSocket connection
- Can be reused across multiple applications
- Acts like a HTML element
  - CSS can be used to style it
  - JavaScript can be used to add or remove it
  - It is recognized as valid HTML element by the browser (Custom Elements specification)

Statecharts in the X stack

Christina Grubmüller

Modeling behavior of event-driven systems

GUI components considered as event-driven systems

- Functionality is triggered by events from user interactions
- Tool for modeling the behavior of event-driven systems: **Statecharts**

Why are Statecharts relevant for XML technologies?

- XML-encoding language for Statecharts has been standardized: **SCXML**
- SCXML supports semantics of Statecharts and defines additional elements

Rise of systems that are able to interpret SCXML-encoded behavior

- In recent studies, Statechart processors based on SCXML-XQ were examined
- We test Apache Commons SCXML: A fully functional SCXML interpreter written in Java

→ Model as the event-driven system: **Proof of concept!**

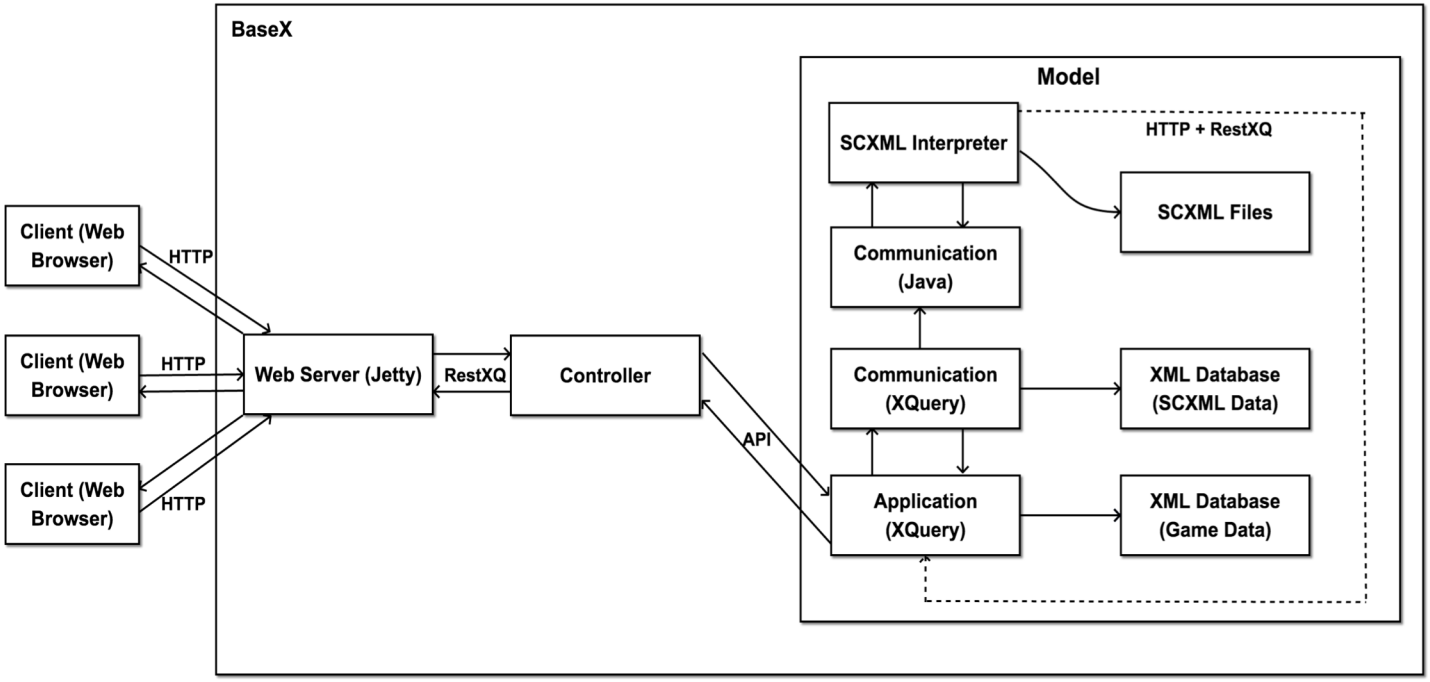
Using Apache Commons SCXML in the X Stack

Model component of a web application as an event-driven system

→ Integration of Java components (interpreter) needs a redesigned architecture

Additional elements in the Model:

- Java bindings of BaseX allow for using external Java classes in XQuery modules
- Communication modules on both sides to manage interpreter instances



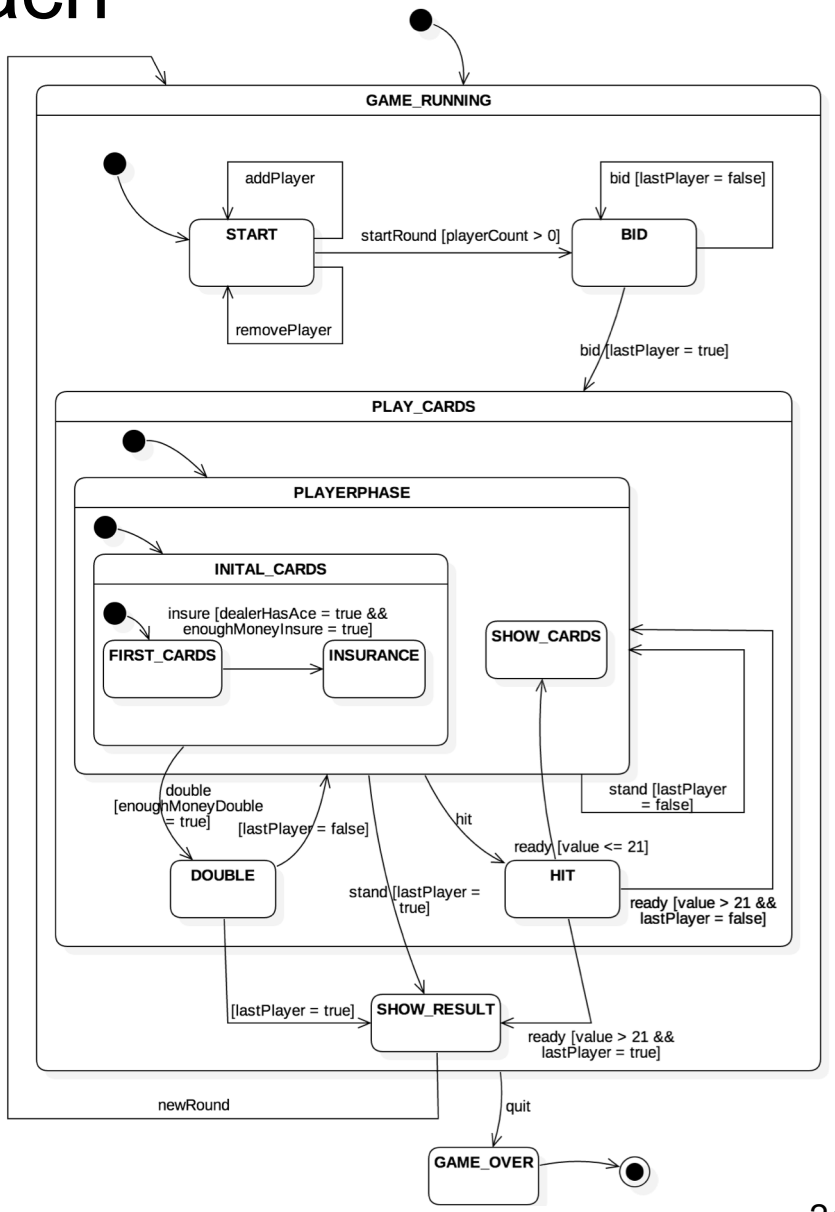
Demonstration of the approach

Implementation of the game Blackjack

- Modeling the behavior with Statecharts
- Encoding Statecharts in SCXML
- Using Apache Commons SCXML as interpreter

Result: Behavior of Model components fully controlled by SCXML interpreter instances!

→ **Demo** of the implementation



Code comparison example: „Hit“ function

Implementation without SCXML vs. Implementation using SCXML interpreter

```
declare
   %updating function bj:hit($gameID as xs:string)
{
    let $variables := Definition of variables here

  return
  (

  if (($nextPlayerBoolean = true() and $nextPlayer != $firstPlayer) or
     (exists($currentPlayer/hasDoubled) and $nextPlayerBoolean = false() and $nextPlayer != 1)) then
     (bj:setNextPlayer($game),
     replace value of node $currentPlayer/HAND/value with $newValue,
     insert node $newCard into $currentPlayer/HAND/CARDS,
     if($currentPlayer/hasDoubled) then (delete node $currentPlayer/hasDoubled) else (),
     db:output(bj:redirect($newURL)))

       else (
            if(($nextPlayerBoolean = true() and $nextPlayer = $firstPlayer) or
            (exists($currentPlayer/hasDoubled) and $nextPlayerBoolean = false() and $nextPlayer = 1)) then
                (replace value of node $currentPlayer/HAND/value with $newValue,
                insert node $newCard into $currentPlayer/HAND/CARDS,
                replace value of node $game/DEALER/isActive with "true",
                replace value of node $game/DEALER/HAND/value with $game/DEALER/HAND/endValue/text(),
                if($currentPlayer/hasDoubled) then (delete node $currentPlayer/hasDoubled) else (),
                db:output(bj:redirect($newURLEnd)))
            else
                (replace value of node $currentPlayer/HAND/value with $newValue,
                insert node $newCard into $currentPlayer/HAND/CARDS,
                db:output(bj:redirect($newURL)))
            )
  )
};
```

```
declare
  %rest:path("blackjack/hit/{$gameID}")
  %rest:POST
  %updating function bj:hit($gameID as xs:string)
{
    let $variables := Definition of variables here

    return
        (
        replace value of node $currentPlayer/Hand/value with $newValue,
        insert node $newCard into $currentPlayer/Hand/Cards,
        bj:sendLocalEvent($gameID, "ready")
        )
};
```

- Lines of code in return statement:

  Without SCXML: **20** vs. With SCXML: **3**

Advantages of the approach

Modeling the behavior of systems using Statecharts

- Clear picture of the behavior of systems under certain conditions
- First definition of functionality needed for implementation

Encoding of Statecharts to SCXML is a straightforward process with defined rules

The approach facilitates a strict **separation of behavior and application logic**

- Implementation of application logic is free of behavioral aspects

→ Application functions only have “one” responsibility

- Systems are easily maintainable and adaptable (change or add functionality)
- Reduction of code complexity

Concluding remarks

Summary

Exploration and review of GUI technologies for use in the X stack

- GUI paradigm: form-based, in the WIMP (Window, Icon, Menu, Pointer) paradigma
- well-known technology: XHTML, XForms, SVG: summarized in the paper
- extending HTML through Web Component technology: applied to WebSocket Element, more detailed explanation in the paper
- supporting multi-client web applications in the X stack through WebSocket
  - server-side support in detail in the paper
  - client-side support in the form of the WebSocket element presented here: declarative extension of HTML through Web Component technology
- declarative support for behaviour of event-driven systems through startecharts
  - encoding of statecharts in SCXML
  - interfacing with Apache statechart processor for direct execution of SCXML statechart for model component of web application → to be adapted to GUIs
- demonstrators for all technologies

Austerity requirement: XML technology only

Because there is a good chance that web applications can be generated from models !

- research work by Zahra Al-Awadai

Because it adds spice to teaching document engineering !

- Opportunity to review and apply principles of software engineering
  - separation of concerns
  - model-driven development
  - declarative approaches / configurations first
- Give context and background to students' experiences with XML in student jobs and database lectures
- Opportunity to create something impressive from scratch, no frameworks

Because it leverages XML competencies for end-user development !

Ongoing and further work

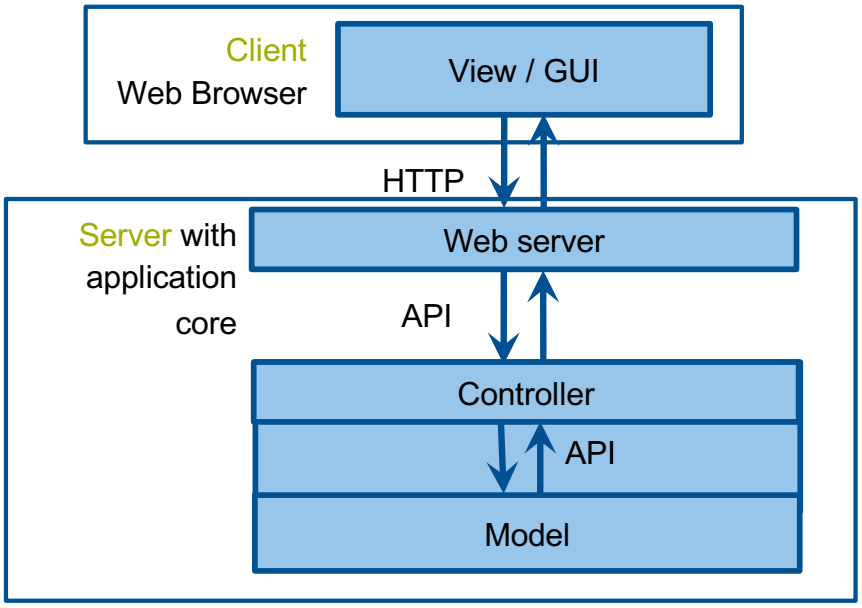
- Websocket Element: fully integrate into teaching of XML technology
- Bring statecharts into the browser
- Rich Internet Applications with XML technology
  - SCXML in the browser
  - Saxon JS
  - non-WIMP interfaces: D3js
- Parallel work with the J stack, based on JSON and JavaScript
  - a principled approach
  - exploring potential for clear architecture and model-driven declarative implementation
  - leverage React as GUI technology

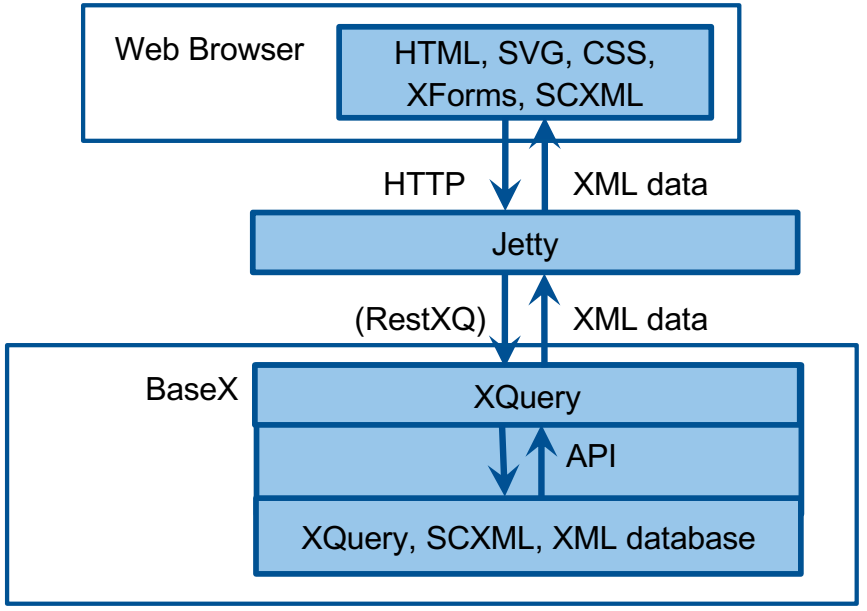
Backup slides

Implementing web applications: XML everywhere

Complete stack for implementing **web** applications with **XML** technology: X stack

- end-to-end encoding of data in XML (**no impedence mismatch**)
- declarative programming
- based on open standards
- standard-conformant, mature, stable implementations across platforms





Modelling web applications: XML is in there, too

Requirement engineering and **modelling** with **XML** technology

- schema languages for XML
  - UML Class Diagrams define data
    → XML Schema and XML instances
- State Chart XML (SCXML)
  - UML State Diagrams define behaviour of event-driven systems, such as GUIs and back components in web applications
    → encoded in SCXML
    → executed with SCXML processors

Server side: BaseX

BaseX WebSocket module

- BaseX supports WebSocket communication and STOMP
- Server side WebSocket interface
- RestXQ „like“ annotation are used to bind WebSocket events to functions
- BaseX implemented a yet to release STOMP WebSocket server which we are using

```
declare
%ws:connect('/')
function test:onConnect() {
    trace("Connection established")
};
```

Connection is established

```
declare
%rest:path("send/{$msg}")
%rest:GET
function test:send($msg){
    let $processedMsg := <message>{$msg}</message>
    return(ws:emit($processedMsg))
};
```

Send a message to all clients