XML applications on the web: X stack

Implementation strategies for the Model component in a Model-View-Controller architectural style

Zahra Al-Awadai, Anne Brüggemann-Klein, Michael Conrads,

Andreas Eichner, Marouane Sayih

Technical University of Munich

Why web apps (games) with XML technology?

Because we can !

- XML technologies provide a full stack of modeling languages, implementation languages, and tools for web applications: XML and CSS, XForms, SVG, XHTML(5), UML class diagrams and XML Schema, UML state diagrams and SCXML, XQuery, XSLT, XPath, XLink, XProc …
- End-to-end encoding of data in XML (optional: descriptive Markup)
  → no impedence mismatch
- Same programming languages used across the stack
- Mature, stable, minimal platform dependencies

Why web apps (games) with XML technology?

Because it adds spice to teaching document engineering !

- Opportunity to review and apply principles of software engineering
  (cf Michael's impressive talk about parsing context-free grammars yesterday)
  - separation of concerns
  - model-driven development
  - declarative approaches / configurations first
- Give context and background to students' experiences with XML
  in praktica and database lectures
- Opportunity to create something impressive from scratch, no frameworks

→ demo of recent student project blackjack (Li & Zhang)

Because it leverages XML competencies for end-user development !

Outline

Demo student project Blackjack (Li/Zhang)

Architecture of web applications and the X stack

- Tiers and Model-View-Controller architectural style
- Passive view (thin client, Model-View-View-Model)
- Focus on Model as a software component to be modelled and implemented with XML technology: XQuery functions operating on XML data, executed in XML database system (BaseX)

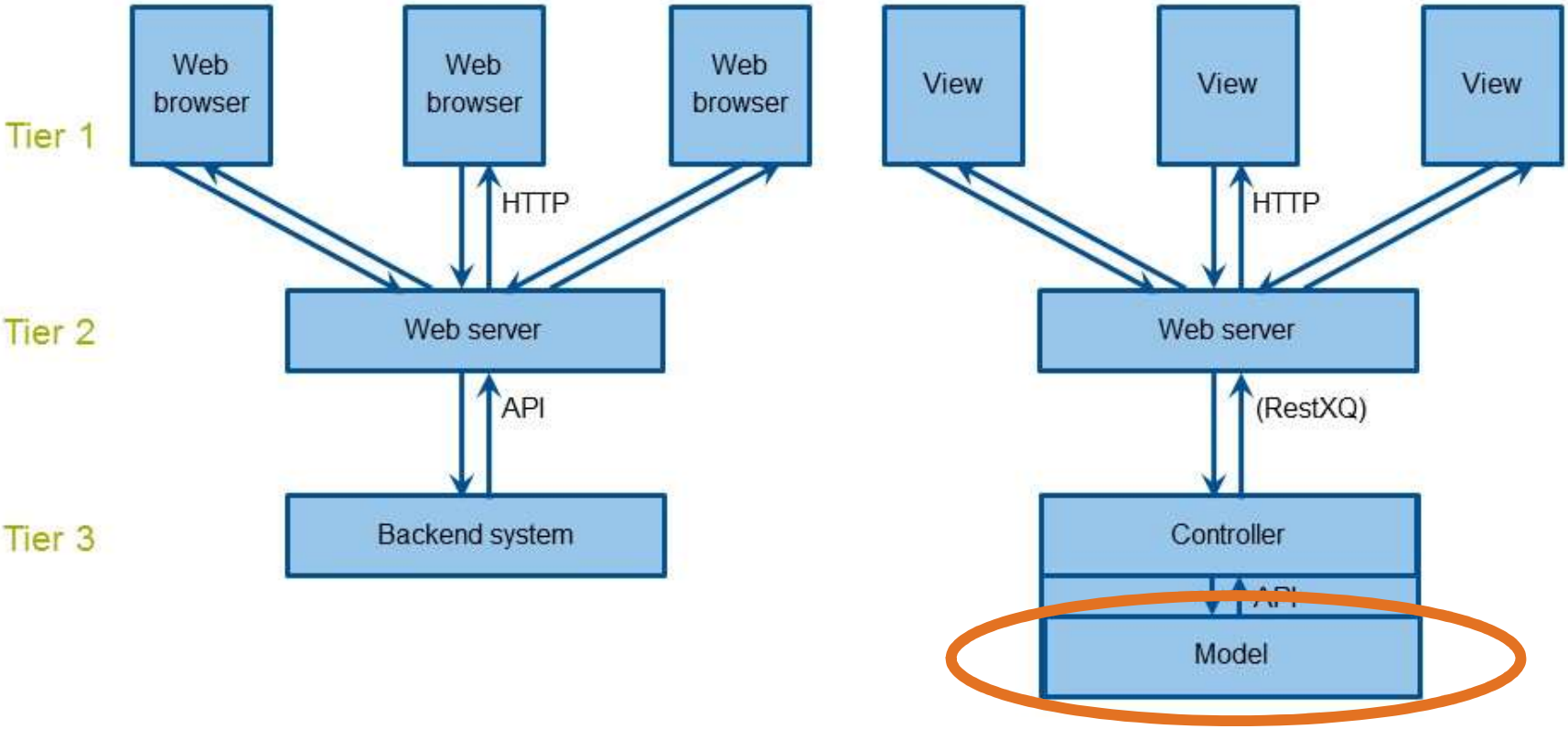
Three contributions

- Functional decomposition in XQuery in view of the XQuery update constraint
- Model-driven development of event-driven systems with UML state diagrams and SCXML
- Server push for multi-player games

Concluding remarks: summary, evaluation, and further work

Architecture of web applications

Passive View, Model-View-View-Model (MVVM), a Model-View-Controller (MVC) variant, mapped to three tiers on the Web



Separation of concerns: keeping the View component independent of the rest of the system

The Model component and the X stack

Responsibility of Model: hold data, compute and provide access

Good practice to make code easier to understand, to maintain and to extend: structure data into collaborating objects that provide access through methods
(encapsulation: be explicit about which data a method operates on)

Live instances of objects (heap)	→ XML collection (BaseX) of objects' states
Object references	→ unique id information on object creation
Methods	→ XQuery functions with XQuery updates
Object methods	→ object id as "self" or "this" parameter
	→ encapsulation

Execution of queries in server-side XML database: BaseX

Functional decomposition for XQuery functions in view of the XQuery update constraint

The XQuery update constraint

Requirement: Methods must be able to change the state of objects in the database

Obvious solution: XQuery Update Facility

- extends XQuery by insert, delete, and change operations on nodes in the database
- used in return clauses of FLOWR expressions
- update constraint: XQuery expressions can either be updating or value-returning

Problem: a method will usually update the state of an object AND return a value

BaseX as a server-side XQuery processor offers some solutions

- HTTP boundary approach: limited but sufficient for simple problems
- EXPath solution: fully compositional

HTTP boundary approach

BaseX supports RestXQ annotations for XQuery functions that map HTTP messages to execution of functions

- execution of XQuery functions that carry RestXQ annotations, can be triggered by HTTP requests (mapping of request information to function parameters)
- return value of such a RestXQ function call is automatically or explicitly wrapped into an HTTP response

A RestXQ function call may mix updates and return values when it wraps them into a db:output call (or when the BaseX option MIXUPDATES is set to true)

Limited solution (but sufficient for blackjack game of Li & Zhang)

- lifting of update constraint at the boundary level of functions that are called via RestXQ
- insufficient for API functions that are called, not triggered via HTTP/RestXQ
- insufficient for inner function calls that occur due to functional decomposition

EXPath solution: split and delegate

BaseX provides function http:send-request that

- accepts an XML-encoded HTTP request as argument
- makes the HTTP request
- returns an XML encoding of the HTTP response

Request and response are encoded as defined by the EXPath industry standard

Wrapping a call to an updating function into a http:send-request call and HTTP message masks the updating nature of the called function → function calls become fullly compositional

How to implement a method that updates the database and returns a value?

- split the method into two, one to update and one to return a value
- delegate calls to each of the new methods to HTTP requests that map to RestXQ annotations of the functions to call

→ demo: advance counter that is stored in a database and displays the new value

Why delegate the call to the non-updating method, too?
→ to circumvent a database locking problem

How does split-and-delegate impact locking?

BaseX treats each query as a transaction

- ACID criteria are guaranteed, esp I for isolation: concurrent access, no dirty reads

Mechanics

- BaseX works with database locks (read locks and write locks)
- A transaction monitor schedules queries for execution so that isolation is guaranteed
- Queries that hold only read locks on databases can operate in parallel;
  a read lock on a database can only be acquired when no write lock is in use.
- Queries can only aquire a write lock on some database when no read lock is in use

Deadlock scenario

- A query reads a database, thus holds a read lock on the database when starting to execute.
- The query issues an HTTP request via http:send-request() that triggers another query via RestXQ; that other query updates the same database, hence needs a write lock. It is now blocked, since the calling query still holds a read lock which it can only release after the called query and then the remainder of its own code have been executed.

A strategy that avoids deadlocks

Recommended practice for queries that delegate database access to other functions via HTTP and RestXQ: also delegate all other access to the same database via HTTP and RestXQ, even if it is only a read access and even if the access occurs after the other delegation.

One pattern that avoids deadlocks:

- Read data from database in method that is called via HTTP / RestXQ.
- Compute new data internally.
- Update data in database in method that is called via HTTP / RestXQ.
- Return the internally computed value

Model-driven development of event-driven systems with UML state diagrams and SCXML

Modeling behaviour of event-driven systems

Model component of a web application and ist sub-components as an event-driven system

- passive component that gets hit by events in the form of functions calls

Proven practice in software engineering: model behaviour of event-driven components with UML state diagrams

- state describes to which events a system reacts
- transition describes to which state a system switches when a specific event occurs and when specific conditions apply
- transition can also trigger function calls or raise internal event

Case study Blackjack: state diagram for each player and for the whole table

- states for player: betting, insuring, playing, waiting
- states for table: setup, playerPhase, dealerPhase, finished

State diagrams and the X stack

SCXML (State Chart XML): an XML application for encoding UML state diagrams in XML

An (imcomplete) SCXML processor written in XQuery by Christoph Schütz (scxml-xq)

- appears to handle complex states and transitions but no activities
- no documentation ☹

An (incomplete) SCXML processor written in XQuery by Andreas Eichner, which has enough functionality to drive a Blackjack application

- no clean separation between SCXML module and the game
- no adequate model of the behaviour of the game (local variables rather than use of API)

→ demo of finite automaton processor

  - finite automata encoded in SCXML and stored in BaseX
  - configurations of current state / automaton stored in BaseX
  - xml application that accepts input symbols encoded as URIs for a configuration and that advances the state of the automaton in the configuration

Server push for multi-payer games

Outlook on multi-player games

HTTP request-response cycle: a client requests a change in the model; the status update is communicated back only to the client who requested the change

What is needed for multi-player games: a connection between client and server through which the server can send messages without a previous client request; that is WebSockets

→ one player requests a change, all players get notified (observer pattern)

Client-side Javascript implementations of WebSockets are part of HTML5, server-side implementations are available in nearly all web servers in many languages, most notable in Node.js.

Michael Conrads implemented two (low-level) solutions for the X stack

- a router component in Node.js
- a BaseX extension that understands a custom WebSockets RestXQ annotation

He also introduced and supported an HTML5 element as an endpoint for data that come through a WebSocket connection; the endpoint can be configured with an XSLT program that transformed incoming XML data (for example, into SVG). How declarative is that, Tommie?

Concluding remarks: summary, evaluation and future work

Summary, evaluation, further student projects

- Reference implementation of the game Mancala as a case study for future students
- Case study Blackjack with focus on state diagrams
  - complete model of behaviour of components with UML State Diagrams
  - system design for Blackjack that incorporates state diagrams and their interpretation
  - requirements for features of state diagrams / functionality of processor that are necessary to drive Blackjack
  - implementation, based on SCXML processor in any programming language
- Case study Blackjack with focus on multi-player functionality
  - architecture that involves proven patterns, for example observer pattern
  - cooperation with BaseX about extension and further WebSocket supports, possibly in the form of a WebSocket XQuery module
  - guidelines for implementions
- Rich Internet Applications with XML technology