GameX — Event-Based Programming with XML Technology

Version 2.0 (2014 April 20)

Marouane Sayih, Technische Universität München <sayih@in.tum.de>

Martin Kuhn, Technische Universität München <martin.kuhn@in.tum.de>

Anne Brüggemann-Klein, Technische Universität München <brueggem@in.tum.de>

Abstract

GameX, a student project at Technische Universität München, is a “serious” browser game that is intended to further systemic thinking in players. Technically, GameX is nearly exclusively implemented with XML technology, which makes the game essentially platform independent. The challenge is to implement a quintessentially event-driven, reactive system with XML technology. We present design and implementation of GameX, relating in particular our strategies and insights for putting the event-driven programming paradigm into practice on an implementation platform of XML technology.

Table of Contents

Introduction

Over recent years, browser games, and particularly “serious”, educational games, have become popular among Web applications. As with all Web applications, players can start playing without having to install specific software, just by loading a Web page over the Internet into a Web browser.

Our research group Engineering Publishing Technology (EPT) at Technische Universität München leverages XML technology for Web applications [BS08] [BMS14] [BRS12], for a number of reasons: First, implementing Web applications with XML technology makes them platform independent. Second, XML technology lends itself to a development method that involves domain experts in all phases of the development process, with the potential to build functional and usable systems.

Finally, XML technology is well-suited for generative, model-driven approaches that can be adapted to changing requirements. Browser games provide us with new challenges. One of them is graphics. Another one, and that is the one we are focusing on in this paper, is the fact that browser games are quintessential event-driven, reactive systems, raising the question how to put the event-driven programming paradigm into practice on an implementation platform of XML technology.

We demonstrate our approach with a case stude, a browser game named GameX. In the work for his Master Thesis [K14], one of this paper's authors, Martin Kuhn, has designed and implemented GameX; the thesis work was supervised by his two co-authors in this paper. GameX belongs the the class of “serious” games and is intended to further systemic thinking in players. GameX is nearly exclusively implemented with XML technology (XML, XForms, SVG, XSLT, XQuery and XProc).

The remainder of the paper is organized as follows: In the section called “The concept of GameX”, we introduce GameX as a game. In the section called “The GameX architecture”, we present the architecture of GameX. In the section called “Event processing in the client”, we deal with layers of event processing in the browser. In the section called “Communication between XML technologies”, we survey methods of communication between XML technologies. Finally, in the section called “Conclusions and future works”, we draw some conclusions, discuss limitations and raise ideas for future work.

The concept of GameX

GameX presents players with a map of towns and fields. Players manage their own towns and the surrounding fileds, thus determining the growth rate of a town's population and the revenue that a town generates for them. Players can construct a number of buildings in their towns such as industrial plants, schools, recreational buildings or environmental facilites that affect a number of parameters such as industrial value and satisfaction of population and, ultimately, the revenue (gold) that a town generates. Players can communicate with and fight against each other, to form strategic coalitions or to conquer each other's towns. GameX challenges players to understand interdependencies and consequences of their actions, thus furthering their systemic thinking skills.

In the remainder of this section, we describe the user interface, the playing options and the underlying mathematical model of GameX.

The GameX graphical user interface

At the core of the GameX graphical user interface (GUI) is a two-dimensional partial view of the world that is divided into tiles. Users can choose one of two views: The World Map (Figure 1, “Screenshot of World Map”) and the Local Map (Figure 2, “Screenshot of Local Map”). Tiles on the map are either towns or fields. Towns can contain several types of building: Industry (sawmill, blacksmith or market hall), Entertainment (inn, bathhouse or residence), Knowledge (library, town hall or laboratory) or Environment (town park, filter or sewerage). Fields can be classified into into seven categories: water, meadow, forest, fishery, wheat, lumberjack and mine.

The GameX GUI is built from the following components: graphical elements, information elements and interaction elements. Information elements and interaction elements are similar in both views, World Map and Local Map:

- Information elements: Jobs list, amount of gold and information about the owner of the towns when the mouse pointer moves over them.

- Interaction elements: A player can click on a tile, can select items of the menu on the top left corner of the screen (news, building table, settings) or of the menu in the top center of the screen (selected town, change view, terraforming), or can use the windrose to navigate the map.

The graphical elements depend on the zoom level and, hence, are different for the two views, World Map and Local Map.

Graphical elements of World Map: Sea (blue), Free town (white), Occupied town (black), Forest (brown), Mine (gray), Field (yellow) and Fishery (dark blue).

Graphical elements of Local Map: fields, building within towns, towns with infobox (name and owner of the town)

Figure 1. Screenshot of World Map

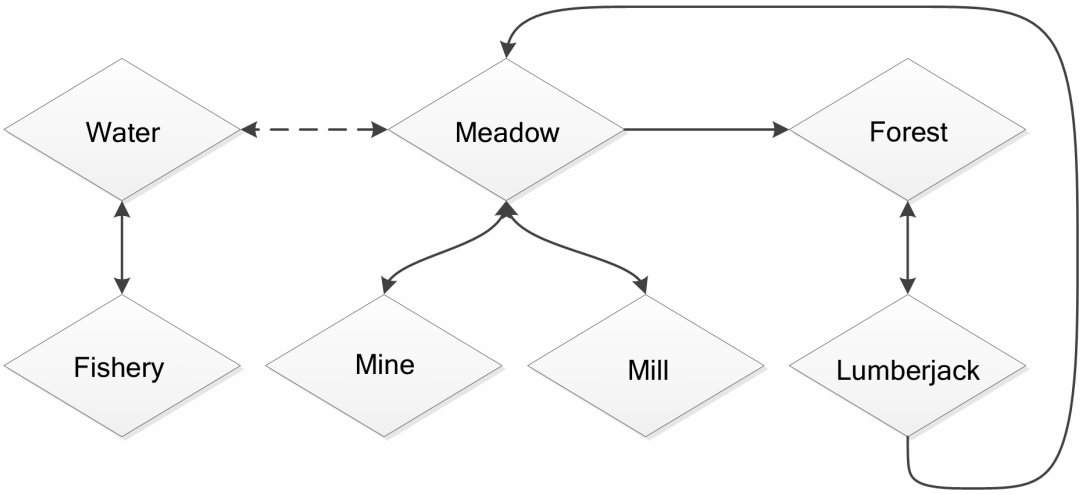
Figure 2. Screenshot of Local Map

Mechanisms and mathematical model

The game option are based on three mechanisms: building, conversion of fields and interaction between players. Construction of buildings is one of the basic mechanisms; it is only possible if the player has got enough gold and the town has space for the building. Additionally, specific types of fields must be in the vicinity of the building that is about to be constructed (for example, a mine field has to be near the location of a smithy). Hence, players need to transform fields as a prerequisite for erecting new buildings. Figure 3, “The conversion options for fields” illustrates the various options to

convert fields from one type to the other. Finally, the third mechanism allows players to interact with each other by exchanging messages or by going to war to conquer towns.

Figure 3. The conversion options for fields



The concept of GameX is based on rounds. Each round occurs at discrete point in time. The game state is updated at every round in which the values of the following parameters are calculated:

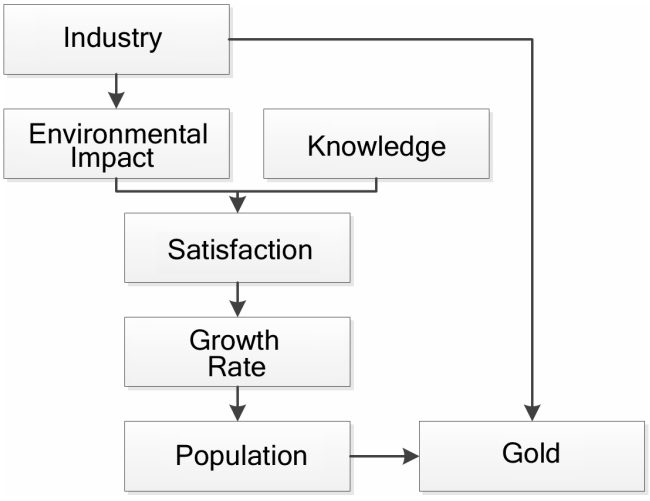
- Industry, Environmental Impact, Knowledge, Population, Satisfaction, Growth Rate, Gold.

The impact of these parameters on each other is described as follows and as shown in Figure 4, “The interdependencies of parameters”:

- The Environmental Impact parameter is affected by the Industry parameter.
- The Satisfaction parameter is influenced by the Environmental Impact and the Knowledge parameters.
- The Satisfacion parameter decides the Growth Rate parameter, which in turn determines the population parameter.
- Finally, the Gold parameter depends on the population and the industry parameters.

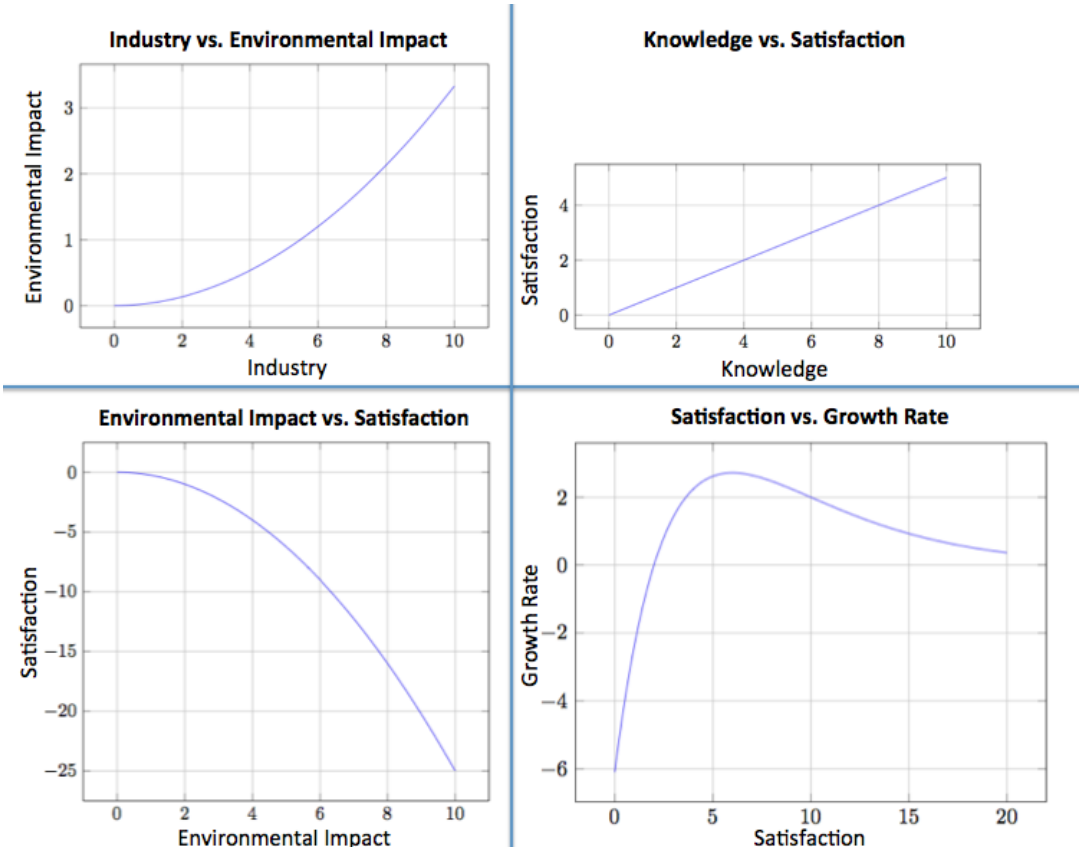
Figure 4, “The interdependencies of parameters” demonstrates how parameters influence each other.

Figure 4. The interdependencies of parameters



We have experimented with nonlinear formulas to model parameter dependencies to avoid trivial game scenarios, as illustrated in the following figure:

Figure 5. Modeling parameter dependencies



The GameX architecture

In this section, we present the architecture of GameX. First, we briefly outline the development process. Then we describe the components of GameX and discuss how the model-view-controller architecture maps to these components.

Development process model: V-Model

In the development of GameX, it is important to consider all requirements necessary for software development and specially the usability as well as the adaptability and extensibility. For this reason our decision is to use the V-Model.

The V-Model is one of the important software development life cycle models. We can distinguish the four stages of the V-Model based on the level of abstraction. The Requirements Analysis is the first phase of the development cycle where the game was analyzed and specified at the beginning of the development. In the next two stages (i.e. Architecture Design and Module Design), the system is being abstracted as a model consisting of modules. The lowest level of abstraction is the source code layer where the modules are implemented. Regarding the testing phase, we used three levels of testing: unit, system and acceptance testing. Figure 6, “Development process model” illustrates the different stages used in the development model of GameX.

To better enable extensibility and maintenance, modifications of earlier phases are still possible and taken into consideration. because these modifications are only changes in the configuration parameters or they are extensions which are already enabled by design or they concern extensions that are enabled by the design of GameX. We provide in GameX the possibility to change and modify parameters of individual elements of the game. For this purpose, the game elements such as buildings, fields or units are dynamically integrated. To illustrate this, we consider the example of element Building. Both cost and construction time are level dependent, and thus, a base value of these parameters are

specified in attributes. To calculate the actual value in a specific level, the base value is multiplied by the level multiplier. Furthermore, the effects of constructing a building on town parameters are stored as absolute values in element attributes.

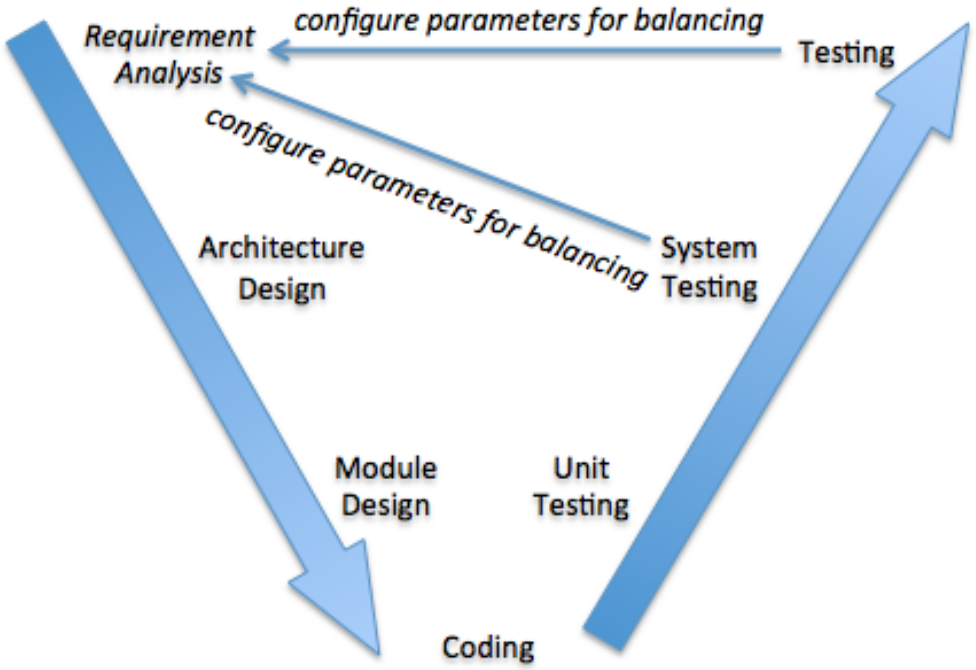
In case of an air filter the element looks like the following:

```
<filter time="60" timeperlevel="1"
           cost="100" costperlevel="1.2"
           industry="0" environment="11"
           amusement="0" knowledge="0">

           <name>air filter</name>

           <description>
           An air filter is a system used to enhance the quality
           of air released from industrial and commercial processes
           by collecting particles from air and to neutralize
           the environmental impact of industry.
           </description>
</filter>
```

Figure 6. Development process model



Components

We employ a modular approach based on separation of concerns to define the components used by the project. We use the standard three-tiers architecture for web applications consisting of a client, a server and a backend database combined with an additional module for user management.

Client

The client represents the graphical view of the game and offers interaction methods. The user interface is designed to support several tasks, including navigating around the map, opening the game settings

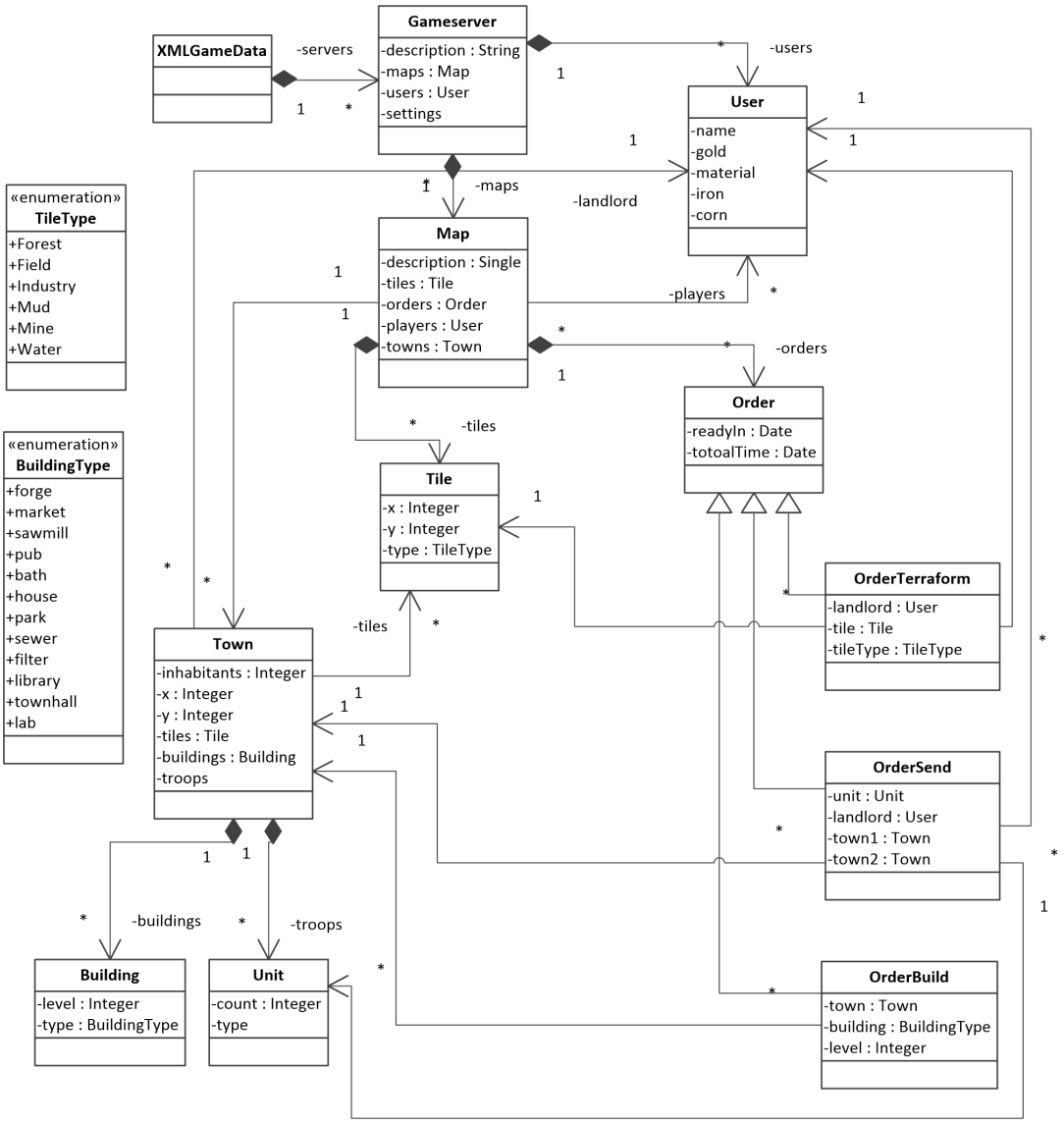
and features that represent the town building progress. We use the XHTML and XForms Standards to implement the GUI components such as navigation menu and the SVG to describe graphical elements. The CSS is used in formatting.

Database

The database is designed to support several tasks, such as the management of user data, user settings, jobs, position and state of towns. In the development of the proof of concept system, the eXist-db technologies were used. The eXist-db offers several advantages such as the platform independence and primarily the compatibility with the XQuery and XSLT standards.

As shown in Figure 7, “The structure of the database”, the structure of the database can be described as follows: Multiple servers can exist. Players are registered and can participate in rounds of different maps. The map provides an overview of all contained objects, such as towns, tiles, armies and orders which can be divided into three classes: OrderTerraform converts a field to a different type. OrderSend sends troops from one place to another and OrderBuild gives an order to build with reference to a town not a player, so the building orders will persist after transfer.

Figure 7. The structure of the database



In previous work [BST07][PB09][BRS12], we have shown how the constraints of a UML class diagram can be transformed into XML Schema; we use the same technique here.

Server

The server provides several modules in order to enable access to the database. It is also responsible for performing game changes according to user actions. We use Cron in the updates synchronization and XProc summarizing transformation steps.

User management

The user management provides several tasks such as registration of new users, modifying or deleting accounts and the rights assignments. eXist-db always has an internal user administration, which can be expanded using XForms in order to meet the requirements of GameX.

Model-View-Controller architecture

The client component holds the view, displaying information and offering controls for user interaction. It also passes these raw interface events on to the controller. The controller lives partly on the client and partly on the server. The client part of the controller transforms the raw interface events into higher-level domain events, selects appropriate handlers, computes their input data and then delegates the actual handling to its server part in the form of HTTP requests. The server part of the controller executes handlers, querying and updating the database as needed, and returns new client data to the cient part of the controller in the form of HTTP responses. The client part of the controller, in turn, initiates a refresh of the view. Hence, GameX is basically a one-page Web application with a thin client.

Event processing in the client

The GameX client component runs in a Web browser and, conceptually, consists of an HTML page with embedded SVG graphics for the map and XForms components for menues and event processing. The client component runs the GameX user interface and a part of the controller (see above).

We divide event processing in the client into three layers: First, the physical or sensor event layer; second, the domain event layer; third, the domain action layer.

The physical event layer is part of the user interface. Physical events mostly originate from user interaction with the game. Those come in two forms: Interaction via XForms controls or via raising DOM events by clicking tiles on a map. Another type of physical event originates from a timer, who fires periodically to initiate updates in the progress bars for current jobs.

Our main idea for the client part of the controller is that it should be run by the XForms processor in the browser, controlled by a mapping from higher-level events, that make sense at the domain level, to higher-level actions, that also make sense at the domain level. We call this the domain action layer.

This entails that at least those physical events that are not already XForms events must be transformed into XForms events, which will be of custom type. We are using two mechanisms for this domain event layer, that both involve a tiny bit of JavaScript:

First, in the case of the timer, we are setting up periodic firing of an event with the JavaScript function `setTimeout()`. For the firing of the event itself, we use the JavaScript function `XsltForms_xmlevents.dispatch()` that is provided by the XForms processor XSLTForms. XForms 2.0 is expected to provide a timer component that can replace this bit of JavaScript; hence, in the future the solution can be solely based on XML technology and will be independent on the XForms processor.

Second, in the case of DOM events that are fired by clicking on a map, a JavaScript handler transforms the DOM event into an XForms event by calling `XsltForms_xmlevents.dispatch()`, as explained in the previous case. Why did we resort to DOM events in the first place to handle clicks on the map? The SVG code for a map copies tiles from a repository and geometrically transforms them to their correct coordinates. Such a transformation can only concern graphics, not XForms controls. So we looked for a method to position XForms controls on top of the tiles and have them raise XForms

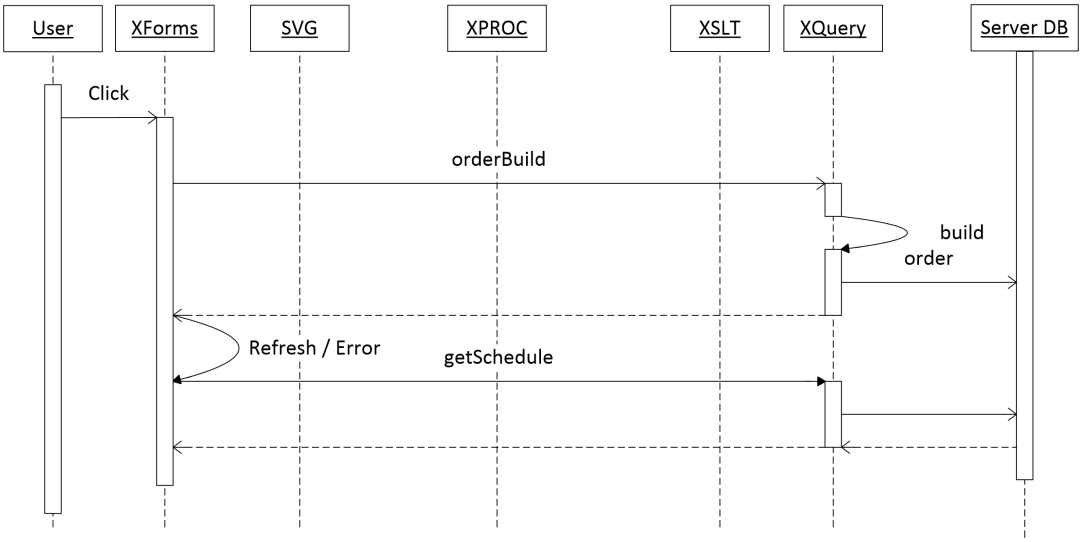
events. Conceptually, we could have used XSLT processing in the browser with SAXON-CE, but in practice we could not sort out conflicts in event handling between the two simultaneously active JavaScript programs for XSLT and XForms.

Communication between XML technologies

In this section, we introduce the several communication methods between XML technologies.

At first, we present an example use case “building order” using sequence diagram that illustrates the interaction between several XML-Technologies, as shown in Figure 8, “Use case “Build Order””. A building order is started, when the player clicks the button “build”. The XForms prepares an order element and transfers it to the XQuery order_build.xquery and then a query is called, which checks the order and stores it in the database. If the order completed successfully without error, then XForms loads the job list.

Figure 8. Use case “Build Order”



As we see from the presented use case, the order management is divided into 3 categories: order placement, order processing and order representation.

Order placement:

If it is ordered to construct a building, to send troops or to convert a field to a different type, the XForms-Trigger sends a POST request to the corresponding XQuery with its related job parameters. Then a function of the Module gameProcedures is called. This module contains all functions that are important for the management of game procedures. Within the called function, the order data is checked for correctness and if there are unfulfilled conditions (e.g. insufficient Gold), it returns an error which is displayed as a message using XForms. If no error occurs, an order element is added in the database which contains the appropriate information depending on the job/order type. In case of a building order, the following information is stored (order type, building type, required construction time, remaining time, the building level, the field-ID).

An example of this building element is as follows:

```
<order type=“build“ color=“209“>
    <readyIn>PT0H10M</readyIn>
    <totalTime>PT2H20M</totalTime>
    <tID>3750</tID>
    <building>market</building>
    <level>5</level>
</order>
```

In this element, a market hall of level 5 will be built in the town with the ID 3750. The construction takes a total of 2 hours and 20 minutes and it lasts in 10 minutes. The TID is used as both field ID and town ID, as towns get the ID of the field on which they are built.

Order processing:

The next step is processing of orders. The server updates remaining times of all orders every minute. If the total time of the order is reached, the corresponding action is executed (e.g. send troops, building order etc.).

The eXist-db contains besides a database and a user management modules a scheduler module which used to organize timming among asynchronous jobs. The following example describes a user job that uses the attributes delay and period which specify the time of the first run and time period between runs respectively. In this case, every 60 seconds the XQuery .../serverUpdate.xquery runs.

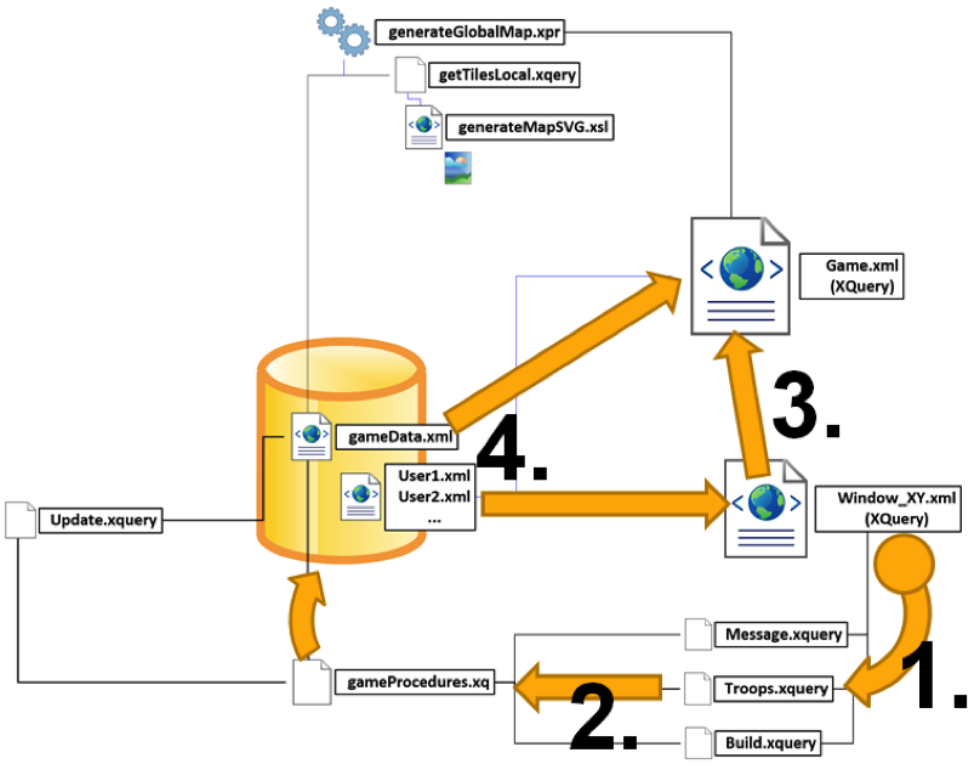
```
<job type="user"
        class="/db/apps/gameserver/server/serverUpdate.xquery"
        period="60000" delay="60000"/>
```

Order representation:

When there are modifications in an order, they are forwarded using XForms to the relevant places in the GUI. This is also realized sometimes using Events which are taken place by XForms internal communications. In addition, the forwarding of the modification from the update function to the user can be considered as indirect commuication.

Figure 9, "Communication between modules" illustrates the structure of the GameX-Project and the used communication methods.

Figure 9. Communication between modules



We distinguish between the following communication methods:

POST request

The POST method is a standard request type in HTML and can be used in XHTML and XForms as well. As an example, the method of POST-request is used in (step 1, Figure 9, “Communication between modules”). We use only the URL and parameters in the Post-request. The following code example explains how an external document (XQuery-file, named order_build.xquery) is requested using XForms.

```
<xf:submission id=“sbuild“ method=“post“ action=“server/order_build.xquery“
       replace=“instance“ instance=“iresponse“>
```

XQuery internal function call

Modules within the XQuery language can be integrated and used across files. In case building order example, individual parameters are extracted in the first XQuery from the XML data and then they are transferred to a function of the module.

XForms events

The built in events in XForms are useful, in communication with Javascript or when separate XForms modules communicate together. Events can be exchanged between different XForms elements. This is also possible if they are in different instances (step 3, Figure 9, “Communication between modules”). To trigger an XForms event, use the following Dispatch:

```
<xf:dispatch ev:event=“xforms-submit-done“
       name=“refreshscheduleevent“ targetid="master"/>
```

Indirect communication

Indirect communication is particularly interesting in that the values are not directly transferred, but they are stored in the database, and then the changes will occur.

In the example build order, the order has been transferred, processed and saved in the database successfully. In case of error, a string is returned containing the error message (step 4, Figure 9, “Communication between modules”). This message is stored and displayed in the instance i-response. If the transmission is successful, the s-refresh-data triggers the loading of the database.

If no error occurs, the XForms loads its database, and represent the new orders directly to the order list. The XForms submission error handling is as follows:

```
<xf:submission id=“sbuild“ method=“post“
       action=“server/order_build.xquery“ replace=“instance“
       instance=“iresponse“>
       <xf:message ev:event=“xformssubmitdone“
       if=“not(instance(’iresponse’)//msg = ’’)“><xf:output
       value=“instance(’iresponse’)//msg“/></xf:message>
       <xf:messageev:event=“xformssubmiterror“>Fehler beim
       Submit! (<xf:output
       value=“event(’responsestatuscode’)“/>)</xf:message>
       <xf:send ev:event=“xformssubmitdone“
       submission=“srefreshdata“/>
</xf:submission>
```

Conclusions and future works

Conclusion: Minimal use of non-XML technology (JavaScript in the browser). This will no longer be necessary once XForms processors and XSLT processors can technically coexist in browsers and XForms processors support new XForms features such as the timer.

Conclusion: Minimal dependency on XForms processor (handling of subforms).

Limitation: We have addressed some timing aspects, but no hard real-time requirements.

Further work: Explore modeling of event-driven reactive systems: Harel's state charts are a classical tool, which are currently given an XML representation by W3C under the heading “State Chart XML”. This opens the vista of representing and executing models with XML technology. We intend to explore this further, in lieu of Abstract State Machines (ASMs). Thank you to James Fuller for the hint!

Bibliography

Further references: XForms, XForms processors, XML databases, State Chart XML, Domain-Driven Design.

[BS08] Anne Brüggemann-Klein, Lorenz Singer *Hypertext Links and Relationships in XML Databases*, Balisage 2008, 2008. *available from http://www.balisage.net/Proceedings/vol1/html/Bruggemann-Klein01/BalisageVol1-Bruggemann-Klein01.html. doi:10.4242/BalisageVol1.Bruggemann-Klein01.*

[BMS14] Anne Brüggemann-Klein, Mustapha Maalej, Marouane Sayih *XML Schema Identity Constraints Revisited*, XMLPrague 2014, 2014. *available from http://www.xmlprague.cz/sessions2014/#xsd.*

[BRS12] Brüggemann-Klein, Anne, Jose Tomas Robles Hahn and Marouane Sayih *Leveraging XML Technology for Web Applications*, Balisage 2012, 2012. *available from http://www.balisage.net/Proceedings/vol8/html/Bruggemann-Klein01/BalisageVol8-Bruggemann-Klein01.html. doi:10.4242/BalisageVol8.Bruggemann-Klein01.*

[BST07] A. Brüggemann-Klein, Th. Schöpf, K. Toni *Principles, Patterns and Procedures of XML Schema Design — Reporting from the XBlog Project*, Extreme Markup Languages 2007. *available from http://conferences.idealliance.org/extreme/.*

[K14] Martin Kuhn, *Lerning Systemic Thinking: Design and Implementation of a Browser Game based on XML Technology*, Master Thesis, TU München, 2014.

[PB09] Pagano, Dennis, and Anne Brüggemann-Klein *Engineering Document Applications — From UML Models to XML Schemas*, Balisage 2009, 2009. *available from http://www.balisage.net/Proceedings/. doi:10.4242/BalisageVol3.Bruggemann-Klein01.*