VOOGA

Xml Specification

Adam Durity
Department of Computer Science
Duke University

Table of Contents

1 General Notes

1.1 Introduction

The Xml documents and schema provided are for your use as a starting point. They are meant to be extended upon as need be. Try to stick to using the schema as a layout, but if an element needs to be added to the schema, add it, but be prudent. Only add something if it's really needed.

1.2 Xml Editor

When working with Xml, you're probably going to want a software tool to design it with. You can use just a plain text editor, but you'll find yourself getting lost in the syntax of namespaces and the like that you'll miss the overall structural picture. An editor with a graphical view will give you a top down look on the Xml you will be editing and writing, allowing you to design better Xml.

As a suggestion for an editor, I recommend the free edition of XmlSpy 2005 which can be found here: http://www.altova.com/support_freexmlspyhome.asp. XmlSpy can be integrated into Eclipse for ease of use.

1.3 Web Resources

I recommend you consider the following resources as needed while working with Xml.

- http://www.w3schools.com/schema/
- http://www.w3schools.com/xml/
- http://apps.gotdotnet.com/xmltools/xsdvalidator/ (for validating any changes you make)
- http://www.w3.org/XML/
- If you can't find what you're looking for on this list, search for it.

1.4 General Types

Several types are used throughout the schema, so these types have been placed at a global scope within the file. The types are described below.

1.4.1 Variable

Many times throughout, you will see that certain elements can contain variable elements which specify the initial values of named variables within the model. For example:

```
<controller ...>
...
<variable name="num_players" value="2" />
</controller>
```

The code above specifies the initial value that the number-of-players variable in the model should have. It does not declare the variable in any way. The java code must already be set to handle each specific variable that can be specified in this way. This is only meant to be a way to specify initial values. Elements which have the capability to contain variables are specified throughout.

1.4.2 Conditional

`conditional` is the base type for the conditional statements that appear within the `level` element, such as `win`, `lose`, and `draw`. This type simply defines an attribute `class` which is the class in which the conditional function appears in order to evaluate a specific condition.

1.4.3 Position, Velocity, and Dimensions

Position, velocity, and dimensions can all be specified with these types as follows:

```
<position x="0" y="0" />
<dimensions height="100" width="100" />
<velocity dx="1" dy="1" />
```

These types will be used throughout the schema to specify the properties of objects as they are placed in the game world.

2 Controller Element

2.1 Controller

The controller takes one required attribute, the controller class. Thus a controller is defined as:

```
<controller class="ControllerClass"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:controller.xsd">
...
</controller>
```

The other attributes, `xmlns:xsi` and `xsi:noNamespaceSchemaLocation`, define that this xml is to be parse with a schema and the location of that schema. The controller tag can contain `variable` elements.

2.2 Players

Within the controller tag the `players` tag must appear and must contain at least one of two elements: `humanplayer` and `computerplayer`.

2.2.1 Human Player

The `humanplayer` tag defines all of the information for the human player of the game. This information includes the controls that a player uses to control the game and any `variable` elements specific to the human player that may need to be set, for instance, health. Also, the `humanplayer` tag has a `class` attribute that can optionally be defined to allow for a class specific to the human player if need be.

```
<humanplayer class="test">
...
</humanplayer>
```

2.2.1.1 Controls

The `controls` set defines a set of `control` elements that can be used to store information about the controls the user has and what events they correspond to, as follows:

```
<controls>
      <control key="a" event="attack" />
      ...
</controls>
```

2.2.2 Computer Player

The `computerplayer` tag defines information about the computer player, if there is one. This element contains information about the AI and any `variable` elements that are relevant to the computer player. Like `humanplayer`, `computerplayer` also has a `class` attribute that can be defined if there is a relevant class to handle computer play specifics other than the AI.

2.2.2.1 AI

The AI element defines the AI by providing a class in the `class` attribute that represents the class that should handle all of the AI for the computer. Also, `ai` can contain `variable` elements for specific AI attributes.

```
<ai class="GameAIClass">
      <variable name="ai_var" value="0" />
</ai>
```

3 Timeline Element

The purpose of the game timeline Xml file is to organize the structure of the game levels and how they proceed from one to the next.

3.1 Levels

The game timeline contains information about all of the levels and the order in which they appear as well as references to their particular Xml files. This is done through the `levels` and `level` elements.

```
<levels num="3">
      <level name="Level 1" file="level1.xml" />
      <level name="Level 2" file="level2.xml" />
      <level name="Level 3" file="level3.xml" />
</levels>
```

3.2 Conditionals

The conditionals define the conditions under which the game is won, lost, or drawn. These elements simply contain a reference to a class which contains all of the logic to determine whether the condition has been met. The `win` and `lose` `conditionals`, as described in 1.4.2, are required, but the `draw` `conditional` is optional. In addition, conditionals can contain `variable` elements.

```
<conditionals>
      <win class="GameWon">
            <variable name="num_success" value="3" />
      </win>
      <lose class="GameLoss" />
      <draw class="GameDraw" />
</conditionals>
```

4 Level Element

4.1 Field

The field defines the entire playing field for the level. In Pac-man this is just what is on the screen, but in a side-scroller, this is the entire level. The `field` element contains the dimensions of the playing field as well as any variable that act on the playing field such as color, sounds, etc.

```
<level name=“” id=“1”>
<field>
<dimensions height=“50” width=“50” />
<variable name=“background-color” value=“#000000” />
<variable name=“music” value=“sound.wav” />
</field>
...
</level>
```

4.2 Frame

The frame defines the visible element of the level. In essence, it is the view, what can be seen by the player at any given time. In the example before, the frame for Pac-man is the whole level, where as in the side-scroller, it is simply the section of the level that the player is currently on. The `frame` element defines the initial position and dimensions of the frame at the start of the level.

4.3 Layout

The layout element defines the starting positions and initial conditions for all objects within the level. These objects are derived from the game objects Xml file. Various properties can be set such as position, dimensions, velocity, and any model-defined variables.

```
<level name=“” id=“1”>
...
      <layout>
            <object name=“obj1” type=“Object”>
                  <position x=“0” y=“0” />
                  <velocity dx=“2” dy=“1” />
            </object>
      </layout>
</level>
```

5 Objects Element

The `objects` element defines all of the types of objects that can be present within a game. The actual instances of these objects are defined and modified within the `level` element's `layout` element. Each object references a Java class which handles all of the properties, action, and interactions with other objects for that object. The `gui` element defines GUI properties for this object for the game to use to display the object. Variables can be defined within the `object` element that acts on all objects within the game, perhaps setting a default for a variable that could be overridden at the scope of the level.

```
<objects>
      <object type=“Object” class=“Object”>
            <dimensions width=“1” height=“1” />
            <gui img=“object.png” />
            <variable name=“parameter” value=“test” />
      </object>
      ...
</objects>
```

6 Conclusion

This is by no means an exhaustive implementation of the Xml game architecture. Rather this is just supposed to give you a starting point from which to grow your own version of the OOGA Xml Specification. When creating your own xml and schemas, keep in mind that xml is simply a way of storing structured data and is not meant for logic of any kind. Good luck!