ATHUS: A Generic Framework for Game Development on Ginga Middleware

Ricardo M. C. Segundo               Julio César F. da Silva               Tatiana Aires Tavares

Federal University of Paraíba, Digital Video Application Lab (LAVID), Brazil

Abstract

Games are ancient tools of entertainment for humans, but also serve as tools for your intellectual growth. Games are also a way to overcome our limits, since they essentially have the character of a challenge. A game should give the player the ability to solve problems generated by the dynamics of the game. Thus we have a simple definition of play: the game is an activity of problem solving that provides fun. Due to its large capacity for games, it is important to explore the capabilities of games for the DTV, for that we present a propose of a generic framework for developing games that aims to facilitate the development of games for the Ginga, and other studies on games for DTV.

Keywords: Games, Digital Television, Framework, Ginga.

1. Introduction

The interactivity in Digital TV (DTV) can be defined as the result of convergence between television and new digital technologies [Pessoa, 2001]. According to [Schell, 2002], the component that is responsible for communication between the user and the OS to a device onboard, such as Set-Top Box (STB), is known as middleware. The Brazilian middleware, named Ginga, has been undergoing constant changes in its architecture, due to the natural process of consolidation of the national standard. It is basically divided into two subsystems: Ginga-NCL and Ginga-J [ABNT 15606-1, 2010].

With the incorporation of two subsystems to the specification of middleware Ginga in the Brazil, it creates a doubt the programmers about which one to use or what the most appropriate for the deployment of a specific application. This doubt exists because the languages have different characteristics and are directed to different domains.

This paper’s aims to present a generic game development framework for the SBTVD-T, using Ginga-NCL and Ginga-J. The framework presents an application model and a set of classes that simplifies game development for the Digital TV.

2. Related Work

2.1 Tuga

The middleware TUGA [Ferreira and Souza, 2009] aims to answer the technological requirements needed for the game industry, that can now exploit the market for digital games in Interactive Digital TV. This middleware supports graphic system, sound and controls into the Ginga. More than these systems, it is necessary layer of software that can actually meet the new requirements and thus brought the development of games, for such the Tuga offers a layer of GBF.Tv framework that provides features such as: manager sprites, fonts, characters, among others [Ferreira and Souza, 2009].

2.2 Ginga Game

Ginga Game [Barboza e Gonzales, 2009] is a Digital TV game development framework. Its goal is to provide a structure that makes it easier to develop games for the Digital TV and make this task more similar to the development for personal computers. The purpose of the creation of a software framework for game development is to avoid that the common tasks be implemented gain every time a new game is produced. [Valente 2005] presents a similar approach, but focuses on the reuse of software components for computer game development. Ginga Game provides an application model that automatically performs several recurring tasks concerning game development, such as resources loading and component management, and allows the developer to focus on its game specific code. This approach is similar to the models provided y XNA [Microsoft 2009] and Unity 3D [Unity 2009], for instance. These tools provide a complete game structure and the developer must only write the code that defines the behavior of its ame components and add them to the game scenes.
3. GINGA

ISDTV-T is currently being adopted as Brazilian's official system for Terrestrial Digital TV [ABNT 15606-1, 2010]. The applications to be executed over Ginga are classified into two categories depending on the way they are written. Procedural applications are those written using the Java language and declarative applications are those written using the NCL language. Ginga application execution environments are similarly classified into two categories depending upon whether they process declarative or procedural applications, and are called Ginga-NCL and Ginga-J, respectively.

Ginga-NCL is a logical subsystem of the Ginga middleware which is responsible for processing and presenting NCL documents. In addition, Ginga-J is the middleware's subsystem in charge of defining all the Java Application Program Interfaces (APIs), content and data formats, besides protocols up to the application level.

3.1 Ginga-NCL

Ginga-NCL is the declarative environment of Ginga middleware responsible that is for running NCL applications.

Applications developed in this language have the characteristic of being focused on the direct manipulation of hypermedia documents, synchronized in time and space in various media objects.

With the Ginga-NCL is possible to write applications in XHTML, which may or may not use CSS, which is a language in which properties are declared and values of styling for HTML elements. It is also possible to implement applications with scripting techniques using ECMA Script or Lua. Also, it’s can call code in Lua by NCL.

As NCL is a declarative language, the programmer does not specify any details of the functionality of the application, witch is how they will be executed. What is determined is what will be done and what will be used. This means that the behavior of the application does not change according to their state, there is no decision to be made at runtime.

3.2 Ginga-J

Ginga-J is the subsystem of the Ginga providing the infrastructure needed to run Java applications, the paradigm of object orientation. In the first specification of the Ginga-J was given support to applications that were developed using the Java TV API, JMF, DAVIC, and DVB. However, some of them are not free. With this problem, the Forum SBTVD decided it would be necessary to change the specifications of the subsystem. Therefore, Sun Microsystems gave the specifications of a newly created API called Java DTV for Ginga-J, royalty-free [Montez and Becker, 2005]. The Java DTV API, according to [Fernandes et al, 2004], is an extension of the Java platform that enables the production of content for interactive television. The main goal of Java DTV is to enable the development of portables interactive applications, which are independent of network technology diffusion [Montez and Becker, 2005].

This means that Java allows the development of DTV applications for digital TV with high level of abstraction, similar to the standard Java platform. The development of applications for set-top boxes via the Ginga-J is done identically to applications made for Desktop with Java SE, at the level of paradigm, the code structure and abstraction, but on top of a virtual machine reduced because set-top boxes have less processing power and storage.

4. ATHUS

ATHUS is a generic framework that provides support to programmers for developing games to Brazilian Digital Television System (SBTVD). It acts at a high level as an API for Ginga-J developers and as modules for Ginga-NCL ones. As we can see in Figure 1, there are two versions of implementation of the ATHUS, one based on Java for use on Ginga-J and another based on Lua for use on Ginga-Ncl subsystem of Ginga.

![Figura 1: ATHUS on Ginga Middleware](image)

The two versions are based on the same architecture, but considering that the two subsystems are independent form each other, the best approach was to build two versions of the framework, based on Figure 2.
As showed in Figure 2, we built many modules that provides basic support to game development, as we are going to see on session 5 where we shows games developed with this modules. So in next we describe the main functionalities of the modules that were ahead developed.

4.1 Engine

The engine module acts as interface to the access of the TV, that is, it simplifies all the call necessary to print a image on the TV, or to take information from this one, like the resolution that is being used on the TV.

Its main functions are:

- **buffer(obj):** this function receives a GameObject (described later) as parameter and stores the images to be displayed in memory. This way we can buffer all images and only then print then all at once at screen, what constitutes a double buffering technique that is used to avoid flickering on screen;
- **getCanvasSize():** function used to get the size of the screen of the TV in pixels. This is useful to adjust the game resolution to the TV resolution;
- **clean():** function that cleans the screen of the TV and the buffer;
- **update():** after all images are buffered, this function is called to print the image in the buffer on the screen;
- **timer(time, func):** an auxiliary function that is used as a timer. It receives a function and a time. The function is called after the time passed ends;

4.2 Menu

Nowadays all games have some elements in common, like a menu. In this we can find information about the game like controls, the authors and even start the game. That’s why the menu module was built, to simplify the building of menus out and in the game.

Its main functions are:

- **draw():** this function prints all the options that can be select by the player on screen, with a pre-selected option Highlighted;
- **addOp(source, tipo, action, x, y):** it is used to add a option to a menu. This option can be a action, another menu or a picture. The source parameter is used to indicates the button or image of the option, tipo indicates the kind of the option, action is what happens if the option is a action and x y indicates the position of the option on screen;
- **removeOp():** removes a option from a menu. It is useful to make dynamic menus during th game;
- **action(type):** acts as a listener on the menu. The parameter type is read by the function is used to change the options on menu, or to activate the sub-menu;

Another point in the menu module, it can have other menus as submenus.

4.3 Main

The main module in fact isn’t a framework module, it is where the logic of the game is implemented. It also acts as a connector for all the other modules, as we can see in figure 2, because only with the game logic we can know what a specific game is going to use or not.

The unique point of this module is that is necessary to implement two methods: one responsible for updating all the game and one for drawing all images in the buffer. This is better explained in session 4.7.

4.4 GameObject
This module is used to represent active objects in the game, like players, NPCs, chests and items. Other objects like scenarios will be represent by tiles and figures modules.

With the use of the GameObject module we can create many objects that are going to have similar behavior, what makes easier to display then in the screen through the engine module.

This module is constituted essentially by attributes and attribution functions, which we present now:

- **alive**: this attribute is used as a flag to indicate if that object is going to be draw on the screen or if it is there waiting for some event and must not be drawn now;
- **center**: stores the position of the center of the object. Can be used to calculate collisions.
- **scale**: used to store the scale of the object. The engine module gets this information and draws the image with the indicated scale. This can be used to simulate depth in many games, like a music game as GuitarHero;
- **ai**: references the AI module. It was not built yet, but when it does this will enable the GameObject to update itself without external calls from the main;
- **position**: this attribute has the X and Y axes positions of the object. This information is used by the engine to display it correctly and by the main module to update the game;
- **rotation**: used like the escala attribute, but is used as rotation base. Can be used to make animation effects in the game;
- **sprite**: references the sprite module that wil be detailed next;
- **speed**: represents the speed of the object. It stores both X and Y axes speed that is useful for objects that can move in both axes at same time;
- **collided**: indicates if the object suffered collision during the update phase of the game;

Although these attributes supports most of the games attributes, other attributes can be created making use of inheritance in this module, which is permitted in Lua and Java.

**4.5 Sprite**

This module has shown being very important for all the games created recently with the framework. It is used to control the animations that use the sprite technique, that are sets of data, in our case images, which define a particular object or character in a game. For example, to a avatar we may have a sprite that contains the vertical and horizontal positions it in the world and the direction to where it is facing, and each direction contain many images that showed one after other in certainly frequencies give the gamers the impression that the person in the game is moving. An example of this technique is shown in session five with the RoboWalker Game.

This module has some attributes, a constructor and many functions. The most important attributes will be described with the constructor, and after we describe the main functions.

**Constructor:**

- `new (path , tw , th , fw , fh )`: the constructor receives five parameters:
  - `path`: it is used to indicate the path to the image file that contains the sprites that are going to be stored in memory;
  - `tw`: indicates the width size of the full image;
  - `th`: indicates the height size of the full image;
  - `fw`: indicates the width size of the a frame of the sprite;
  - `fh`: indicates the height size of the a frame of the sprite;

**Functions:**

- `getCanvas()`: this function returns the actual frame of the sprite;
- `setEstado(novoEstado)`: with this function we set a new sequence of frames of the sprite. This means, if we are in a sequence that shows the animation of a person moving right, calling this function we can change the sequence of frames for the moving left one;
- `nextFrame()`: the call of this function updates the actual frame, bringing in the next frame of sequence of sprites;

**4.6 CG/Audio**

As these modules are similar we are going to describe then together.

The main functionality of these modules is to generate automatically all NCL code necessary to play music and audios. This is necessary because we can’t play these kind of media directly from the Lua medias. So it’s necessary to call from the Lua media the NCL file to play the videos and audios. That’s why is necessary to generate automatically the NCL codes, otherwise the programmer will have to create all the NCL by himself, what would demand extra time and effort. Unfortunately the NCL Virtual STB does not support this kind of operation, what makes the actual CG/Audio incomplete.
The implemented methods are simple, and acts like commands of a player like start, stop, pause, unpause and others, so we are not going to detail these methods.

4.7 Anim

The anim module is responsible for the maintenance of game’s frame and update speeds, in another words FPS (frames per second) and UPS (updates per second) taxes.

FPS is a measure of how many frames of data is used to display moving video. Each frame is a static image, displaying images in quick succession creates the illusion of movement. The more frames per second (FPS), the smoother the motion appears. A good FPS, described by Davison (2005) is bounded below by the human eye and the critical flicker frequency (CFF) (Peter, 2009), which is the rate at which a flickering light appears to be continuous. UPS is the measure of game updates taxes. Game updates are events that change the attributes in the game, like positions, AI, camera and others.

The maintenance of these taxes are important to avoid flickering gameplay and inconstant update of the game, what can compromise the experience of the player. With this objective the module maintains a constant loop that maintains a constant call rate of the player. With this objective the module maintains a game, what can compromise the experience of the player. Avoid flickering gameplay and inconstant update of the camera and others.

The main function are:

- start(update,draw): the function receives two parameters: update and draw which are functions implemented in the main module. Their references are stored in local variables, so they can be called all the time. After this, this function calls the run function that is responsible for the game loop;
- stop(): the call of this function will result in stopping the game loop;
- run(): the principal function of this module, is here where we have the control of the games taxes (FPS and UPS). This control is made by controlling the time that takes to draw the images on screen, on buffer and the errors of timing of some methods responsible for making the program stop for some time to maintain the desired FPS. If the drawing takes longer than planned the algorithm stores the excessive time in a error variable. If this times gets too big it’ll cause the algorithm to skip some frames, so it can synchronize again. If the opposite happens the algorithm will make the loop to sleep a little more making it synchronize again.

4.8 Collision

Collisions happens when two objects in a game occupies in a moment the same position. Examples of that are: a car moving into a wall, a arrow hitting a character, a bullet hitting a target. All these events will result in collision, and after that is confirmed the game acts in different ways. But if a collision is not detected the car can pass throw the wall and make a shortcut or a character doesn’t lose life with the arrow. That why collisions are important and where implemented.

This version of the framework presents only one form of collision detection, which is by the box collision technique. This method performs four checks and if all then are true, the collision will have occurred. The checks are (considerate two objects: image1 an image2):

- The position x of one image1 plus the width of this image is beyond the x position of the image2?
- The position x of the image2 plus the width of the this image exceeded the x position of image1?
- The y position of image1 plus the height of image1 is beyond the y position of image2?
- The position y of the image2 plus the height of the image2 is beyond the position y image1?

This is a simple method, but it covers most kinds of collision, mainly in 2D aspects.

The function that implements this algorithm receives two parameters that are both GameObjects. As we saw the in the game objects we have all information that we need to calculate if they are going to collide or not.

4.9 Control

The control module takes care of all aspects related to the remote control, which in the digital television is the main human interaction device.

Inside the module there is a listener that awaits any button in remote control that the user presses. Once a button is pressed the listener updates a table that contains a field for each button. These fields are: pressed and released. As the name suggests pressed informs if that button was pressed and released is the opposite. The listener then updates only the field relative to the pressed or released button.

During the update phase of the game loop main module that has a reference for the controle one calls the getState function that returns the actual state of the buttons. The call of this function also results in cleaning all values to false, indicating nothing happened. This happens because in the next update
fase the main can take another shot of the control status and see if something new happened to a specific button, because it has the past and the present of the buttons.

4.10 NCL

The NCL module represents all NCL code generated in the game. It presents code to initialize the Lua main module and to give it the control of the remote control. More code can be generated in this module, as we saw with the CG/Audio modules, that automatically generate all the code necessary to play audio and videos.

The framework then requires no knowledge of NCL language from the programmers, then it is necessary only knowledge in Lua that is a language more normal for all in this area.

5. USING ATHUS FRAMEWORK

During the development of this work we created some games in a way that we could validate the aspects of the framework. Some of them uses modules not yet finished, in a way that the modules were not described before, but will be presented here.

5.1 Asteroids

Based on the Ataris’s classic game, Asteroids was developed in order to validate the possibility of use of primitives components like lines, triangles and points supported by the NCLua primitive modules, to develop a game.

As we used only lines, points and triangles this game is being the base of the primitivas module that will incorporate these and other elements.

The developed game can be seen in Figure 3.

5.2 GingaCraft

GingaCraft is a game based on the classic videogame Pong. The main difference between this game and the original pong is the fact that the GingaCraft has the possibility of moving that ship on two axis (X, Y) while the original could only be moved in the Y axis.

Compared to the Asteroids game the GingaCraft brought the possibility of use of images instead only primitives, giving origin to the GameObject module. Other point is that for the first time we could put two players playing at the same time one against other, using different devices like keyboard, mouse and the remote control.

Also this game was the first with entertainment aspects, like score, better graphics and a small storyline behind it.

The GingaCraft game can be seen on Figure 4.

5.3 WalkerRobot

WalkerRobot was the first game that incorporated many modules of the architecture, making it the first try of using the framework to develop a game. Here we used the GameObject, sprite, anim and controle modules that were described before.

A module unfinished, but initialized with this game was the tile one. Tile Mapping is the technique of using tiles, which consists of forming two-dimensional images from small blocks, tiles. An image formed by this technique tends to have several parts that are repeated; therefore it would be wasteful to allocate memory for the same part repeatedly. Tiles are also smaller blocks of image structures that store various information about the image to be drawn, as the type block that it represents, if it is an obstacle or not, if it has animation, among others.

A module unfinished, but initialized with this game was the tile one. Tile Mapping is the technique of using tiles, which consists of forming two-dimensional images from small blocks, tiles. An image formed by this technique tends to have several parts that are repeated; therefore it would be wasteful to allocate memory for the same part repeatedly. Tiles are also smaller blocks of image structures that store various information about the image to be drawn, as the type block that it represents, if it is an obstacle or not, if it has animation, among others.

In this game we could analyze a considerable advance on graphics, because the techniques we used like tiles and sprites, are enabling a more robust gameplay than the Ginga-Craft. By seeing the
execution of the game WalkerRobot, we can see that the animation of the character from the Sprites, the construction of the environment, even using only one tile, and movement of ground already resemble Super Nintendo, GBA and MegaDrive games which cannot be impressive at first glance, but as the possibilities and limits of game development for digital TV are still being found, we can consider a great leap of a Pong game style to something with animation of a character and scenario, using a few kilobytes.

The Figure 5 shows the result of the game, that uses the character Robo from the ChronoTrigger game as character for validation of the sprite module. In Figures 6 and 7 we have the tile used for scenario and the sprites sequence respectively.

![WalkerRobot Game](image1)

**Figura 5: WalkerRobot Game**

![Tile used](image2)

**Figura 6: Tile used**

![Sprites sequence](image3)

**Figura 7: Sprites sequency**

5.4 GingaHero

GingaHero is a musical game based on big commercial successes as GuitarHero and RockBand. The choice of this game to validate the feasibility of SBTVD games comes from his great popularity with gamers of all ages as well as the TV, and being a style of play present in all platforms, from the latest generation of consoles the cell, turns Ginga a possible platform for game development.

GingaHero was the first game to use all the modules of the framework, and also some others were developed to help in the logic of the game like:

- **Musica(Music):** this module represents the music of the game, containing the accords that are going to be shown on interface and information like band, author, number of notes and others. This way it is ease to change the music of the game to any other;
- **Score:** basically calculates the points of the player. It is based on GuitarHero score: the more notes in sequence you hit correctly, more points you make. Many games uses different ways of scoring, so we are still studying the best form to make an auxiliary module of scoring for the framework;

The Figures 8 and 9 shows the GingaHero interfaces.

![GingaHero Menu](image4)

**Figura 8: GingaHero Menu**

![GingaHero ingame](image5)

**Figura 9: GingaHero ingame**

6. CONCLUSION
The development of games for the Digital TV, using the Ginga middleware, either within the procedural environment (Ginga-J) or the declarative environment (Ginga-NCL) is possible and the appeal the games have may help popularize the interactive content on SBTVD-T.

Software development auxiliary tools has great importance and the creation of frameworks that allows greater code reuse and reduces the needing to rewrite code for common tasks may help to make the process quicker and more intuitive.

With the specification of ATHUS and the implementation of the ATHUS-J and ATHUS-NCL, it’s expected to make the process of creating games for Ginga simpler, providing an environment that abstract the execution platform and allows the developer to focus only on the game’s logic. Through its structure, ATHUS proposes an environment that allows a high rate of software component reuse, reducing the creation time for new games, as previously created components can be reused in new projects.

Future versions of this framework could add other modules where not built yet, but are planned like modules to support artificial intelligence (AI), tiles, world manipulation, network support and others.

References


Barboza and e Gonzales. SBGames 2009 "Ginga Game : A Framework for Game Development for the Interactive Digital Television".