RAMS - ReArchitecturing MediaScheme

Zarni Htet, Hart Russell, Prashanna Tiwaree
Edited by Samuel A. Rebelsky
3 August 2012

Abstract:
In an effort to attract more diverse students to Grinnell College's introductory CS course, Grinnell's CS faculty has adopted a Mediascripting curriculum that serves to engage a greater audience and helps level the playing field between varying levels of programming experience. To provide a more versatile Mediascripting experience, this paper explores the overhaul of the current Mediascripting application (a locally-developed application called MediaScheme, which interacts with the GNU Image Manipulation Program) with a newer, more adaptable, pedagogical language environment called DrRacket. Because DrRacket is a standalone program, it was interfaced with the GIMP using an interprocess communication tool within the Linux operating system called DBus.

Introduction:

Media Computation, created by Mark Guzdial, (Guzdial, 2003) uses the creation and manipulation of digital media to introduce computing to make programming more interesting to a diverse group of people. Mediascripting is a form of Media Computing that is used in the first introductory Computer Science course in Grinnell College, where students manipulate images in GNU Image Manipulation Program (GIMP) using a functional scripting language, Scheme. It was pioneered in Georgia Institute of Technology and used in their introductory computer science class.

MediaScheme, a locally developed console, was created by Professor Samuel Rebelsky and his research students in 2007 to allow the Mediascripting to take place in Grinnell College. Though MediaScheme has been serving its purpose by providing students and programmers achieve the scripting of the GIMP by using the functional programming language Scheme, this architecture consists of flaws and needs improvement. This paper describes a successful architecture redesign that allows scripting of the GIMP through Scheme but using a more developed commercial platform, DrRacket.

Why DrRacket?

DrRacket is a leading pedagogical language environment that supports "multiple dialects of Lisp and even multiple languages." (The Racket Team, 2012) In addition to supporting multiple languages, DrRacket adds many user-friendly features lacking in the MediaScheme console currently used in CSC-151; some of these user-friendly features include debugging, smarter syntactic highlighting, and more GUI tools (The Racket Team, 2012). Incorporating DrRacket

into the introductory course is important because it will make it easier for those with limited or no programming experience to better understand how to program using a development environment. In order for a communication between Dr.Racket and the GIMP to take place, DBus will be used.

What is DBus?

D-Bus is an open source interprocess communication system that allows messages to be sent and received from one program to another. D-Bus supports two types of bus: system bus and session bus. System bus deals with the hardware side of the computer while the session bus allows the interprocess communication between softwares. It consists of *libdbus*, a library that allows two applications to connect to each other and exchange messages and *dbus-daemon*, message-bus daemon executable which is built on libdbus, that multiple applications can connect to (Love, 2005). In this paper we will discuss on how the DBus API (application programming interface) is created for DrRacket and the Gimp to interface with each other in order for the communication to take place between the two softwares.

Why Gimp ?

The GNU Image Manipulation Program (GIMP) is an image manipulation program similar to Adobe Photoshop but free and open source so anyone with an internet access could download the GIMP and use it. Since GIMP is an open source software, it was chosen for the project because it is easily accessible and one can modify the source code to add new functionalities . The GIMP was also specifically designed to be expandable using plugins, which increases a programmer's options for implementing changes. Finally, GIMP is very user friendly allowing many users to easily utilize the functionality of the program.

Why Scheme?

The choice to use Scheme as the scripting language stems from its ability to do media computation in the GIMP, its unfamiliarity to most students, and its simplicity. Since Scheme is not as widely taught as C++, Python, or Java, those with prior experience in programming do not have an unfair advantage over those without prior experience. Scheme's simplicity also allows students to quickly delve into media computation without first having to learn advanced concepts and syntax found in some other functional languages.

Overview of Client and Server Implementation:

Background

Gnome is a desktop environment and graphical user interface that runs on top of a computer operating system (The GNOME Project, 2011).

The **Gio** library of Gnome provides the libraries (or procedures) to connect to DBus. **GVariant** datatype which can hold multiple data types and is compatible with DBus is extensively used in our implementation (The GNOME Project, 2011)..
Inside Racket C API is one way to build additional libraries for Racket interpreter (The Racket Team, 2012).
Scheme_Object is a struct that holds multiple data types used via Racket C API (The Racket Team, 2012).
A **proxy object** on DBus is created to represent a remote object, which in our case, is the server. Invoking procedures on the proxy object calls the methods on the actual object (server) and DBus has automatic handling of sending and receiving data between proxy and the server (Pennington, Havoc., Wheeler, David., Palmieri, John., & Walters Colin.).
A **tuple** is an immutable list.

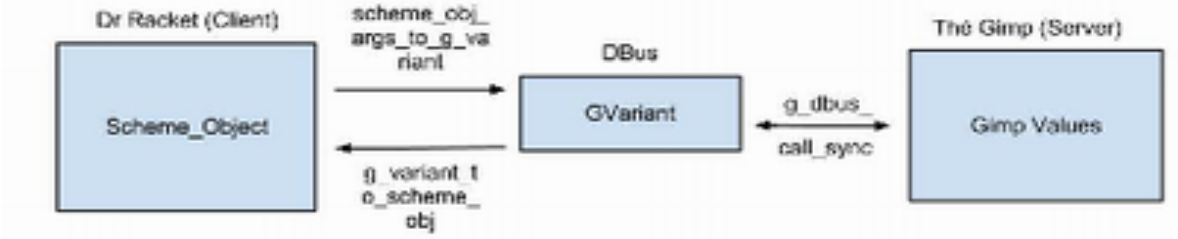

Figure 1 : Client-DBus-Server Overview

Client

Our client API implementation will bridge the Racket interface (client) with the Gimp or any other scriptable graphics application (server) via DBus. The proxy object of the server along with its list of interfaces and procedures is created on DBus. Thus, the Scheme_Object arguments we input through the client interface are wrapped into GVariant data type to render it usable by the server procedures. The server function call will do its work on its respective server. If it is the Gimp, it may draw a picture, add a layer or change the color. If the function call returns a value or values to the client, those values will be converted from GVariant to Scheme_Object for display on the Racket.

Server
The DBUS takes all calls from the client and makes them available to the server. The server implements all calls made by the client. In order to understand the calls made by the client, however, a list of the available procedure calls must be made and registered on the DBUS.

Implementation:

Client-side

Extending Racket
We use Inside Racket C API to integrate our client Racket API for DBus into DrRacket namespace where all the standard global procedures and syntax of Racket are loaded (The Racket Team, 2012). We put all the procedures of our library into a Racket module which is added onto the namespace environment.

The Core
There are three core procedures for our Client API. They are the initializing procedure, the get-proxy-object procedure and the get-method procedure. The initializing procedure sets up the type system that we need for using Gio library. The get-proxy-object procedure gives us the path to the server application we want to call methods from. It has three input parameters of service name, object path and interface name.

The service name specifies the name of the application (server) we are connecting to. The object path links us to the particular application object that we would like to manipulate. For instance, we can set the object path to manipulate image-id 1 in Gimp or image-id 2. The interface path provides us a collection of methods that we can use to manipulate the application object we specified in the object path.

The get-proxy-object procedure is designed to support multiple proxy objects. For instance, from the same racket client interface, we can call functions from Gimp, Inkscape or any other application whose proxy is listed on DBus.

The call-method procedure takes in three parameters: the proxy object, the name of the procedure we are calling from the proxy and a list of arguments. First, the procedure checks

whether the proxy object exists or not. If it does not, an error message will appear. If the proxy object does exist, scheme_obj_args_to_gvariant and g_variant_to_scheme_obj,the two primary helper procedures, will establish the communication between the client and the DBus.

The scheme_object_args_to_g_variant is a wrapper procedure that converts the scheme_object data type that the user inputs (via the client) to g_variant data type to be put on the DBus. The wrapper procedure checks the data type of the scheme_object such as integer, double, string and tuple. It then converts each data type into a g_variant data type.

The g_variant_to_scheme_obj is also a wrapper procedure that converts the g_variant data type that the DBus receives from the server to scheme_object data type which is the output displayed in the client (Drracket). The wrapper procedure has one parameter where it takes in a g_variant value and then checks the data type of value as integer, string, double or tuple. It then converts the g_variant data type to the scheme_object data type returned correspondingly in the client.

Examples

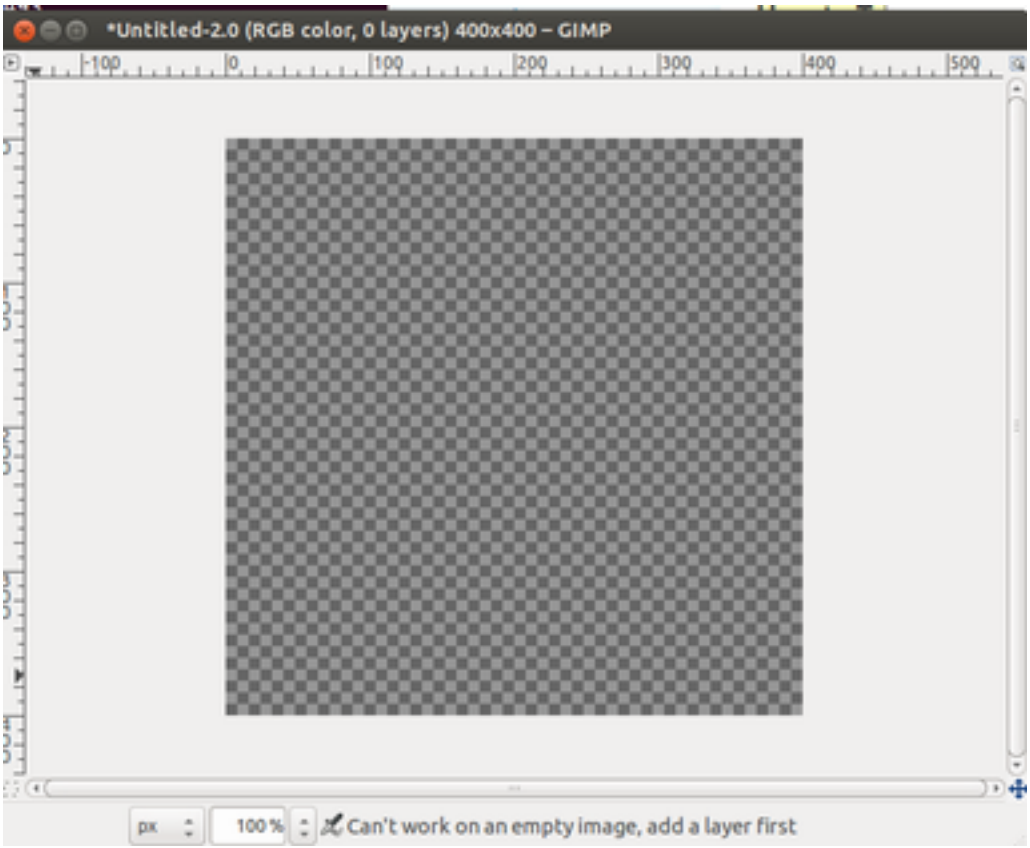

Figure 3 and 4: The figures above is an example of how the client (Drracket) interacts with the server (GIMP). It is first necessary to load the shared object file which is done by typing in the command “rdbu.so” where rdbus.so is the name of the shared object file. It is then necessary to get the proxy object from the dbus in which case we need to call the proxy object that represents the GIMP server and the rdbus-get-object is used to do that. As you can notice the rdbus-call-method takes in the service path, object path, and the interface path. It is then important to use rdbus-call-method to call the different methods in the server. In this example, the image_new procedure implemented in the server takes in two parameters, the image width and height and then displays the image of that size.

Server:

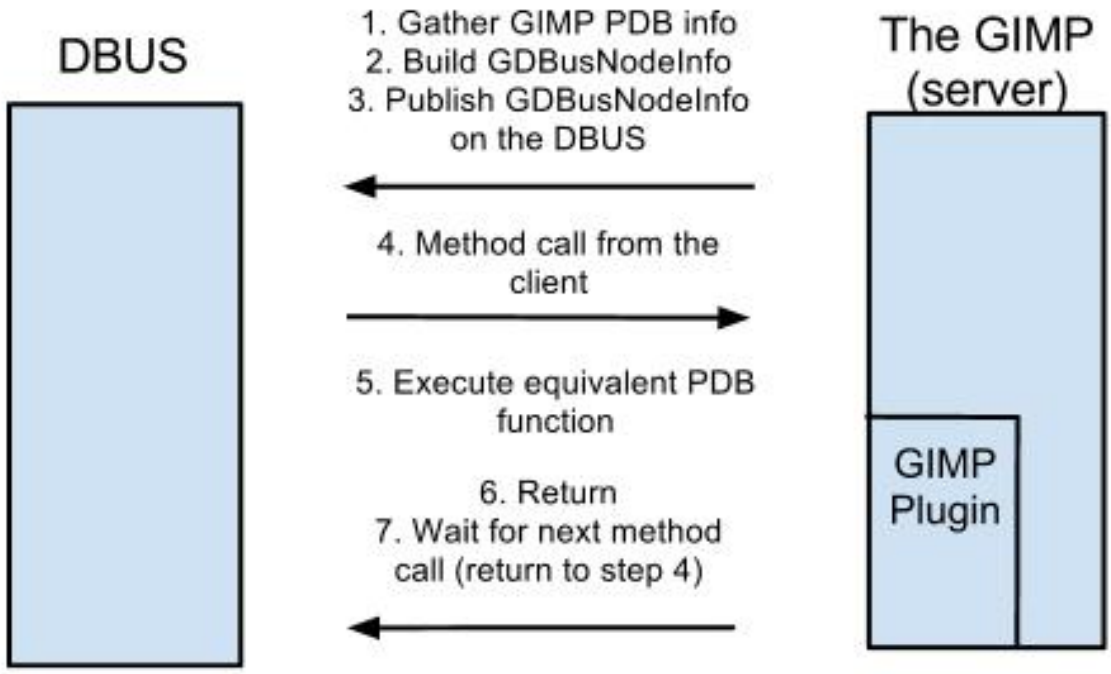

Figure 2

One of the end goals of this project is to produce a working server that can handle and execute all method calls from the client. Because the introductory CS curriculum already utilizes the GIMP as the medium for mediascripting, it is necessary to create a GIMP plugin that uses a DBus library (Gio) to create a server.

A plugin is a user written C program that extends the functionality of an application. In this case, we are telling the GIMP to act as a server by taking commands from the DBus and use its own Procedural Database (PDB), a set of procedures available to those who extend the GIMP, to execute appropriate commands.

In order for the GIMP to accurately interact with the DBus, many procedures must be coded into the GIMP plugin. To start, the available list of usable functions in the GIMP must be published on the DBus, so the client may call those functions. To query the GIMP database for all of the GIMP functions we call gimp_procedural_db_query to list the names of all available functions and call pdb_print_info on those names to get and store the parameters of those functions.

Using the listed function names and parameters, we then create a GDBusNodeInfo node that contains all of the GIMP PDB functions. Using the GDBusNodeInfo, the GIMP plugin then registers itself on the DBus as a server using (_dbus_gconnection_register_object). Now, the GIMP plugin is registered on the DBus as a server with all of the PDB functions available to call.

At this point, the client may call any of the registered functions on the DBus. If the correct function is called with the correct amount and type of parameters, the GIMP may retrieve and execute that function with handle_method_call and gimp_run_procedure2 () respectively. The GIMP may then return values back to the DBus and therefore back to the client using g_dbus_method_invocation_return_value.

As of writing this, the server has not been completed. So far, we have created a primitive server that registers only a few GIMP functions on the DBus. The functions, including new image and new layer, can be successfully called from the client and return values back to the DBus.

Conclusion

Mediascripting serves to engage a greater audience and helps level the playing field between varying levels of programming experience. In Grinnell College, Mediascripting was taught using the locally developed MediaScheme console which allowed the scripting of the GIMP through the Scheme programming language. Though serving its purpose since 2007, MediaScheme had many rooms for improvement. To provide a more versatile Mediascripting environment, a commercial and more adaptable pedagogical language environment called DrRacket was interfaced with the GIMP to allow communication between DrRacket and the GIMP. This communication allows the users to script images in GIMP using the Scheme programming language in the DrRacket interface.

The success of the Re-Architecturing Mediascheme project will in the near future replace the MediaScheme console, which has continuously been used in the CSC-151, the first introductory computer class class, since 2007. For this project to replace the MediaScheme console, the server side implementation needs to be fully completed. As of now, there are only few procedures from the GIMP procedural database that can be called from Dr.Racket. Once every GIMP procedure is able to be called from Dr. Racket so the users can script images in GIMP, this project will be able to replace MediaScheme and be a part of the new CSC-151 curriculum that would use Dr. Racket to script images in GIMP.

Works Cited

Guzdial, M (2003). A media computation course for non-major. ACM SIGCSE Bulletin, 35(3), pp. 104-108.

Love, Robert. (2005). *Get on the D-Bus.* Linux Journal. Online resource, available at http://www.linuxjournal.com/article/7744.

Rebelsky, S. and Davis, J. (1998). *Reformulating Media Computation with Functional Programming, Scripting, and Design Principles*. National Science Foundation grant proposal.

The GNOME Project (2011). *GIO Reference Manual, Introduction.* Retrieved from http://developer.gnome.org/gio/stable/ch01.html

The GNOME Project (2011). *GLib Reference Manual, GVariant.* Retrieved from http://developer.gnome.org/glib/2.32/glib-GVariant.html#glib-GVariant.description

The Racket Team (2012). *Inside: Racket C API.* Retrieved from http://docs.racket-lang.org/inside/index.html

The Racket Team (2012). *14 Structures.* Retrieved from http://docs.racket-lang.org/inside/Structures.html

The Racket Team (2012). *4 Namespaces and Modules.* Retrieved from http://docs.racket-lang.org/inside/im_env.html#%28cpp._scheme_finish_primitive_module%29

Pennington, Havoc., Wheeler, David., Palmieri, John., & Walters Colin. *D-Bus Tutorial.* Retrieved from http://dbus.freedesktop.org/doc/dbus-tutorial.html