RCCola: Remote Controlled Cola

Takashi Yamamiya

This material is based upon work supported in part by the National Science Foundation under Grant No. 0639876. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

VPRI Memo M-2009-005

Viewpoints Research Institute, 1209 Grand Central Avenue, Glendale, CA 91201 t: (818) 332-3001 f: (818) 244-9761

RCCola: Remote Controlled Cola

Takashi Yamamiya

takashi@vpri.org

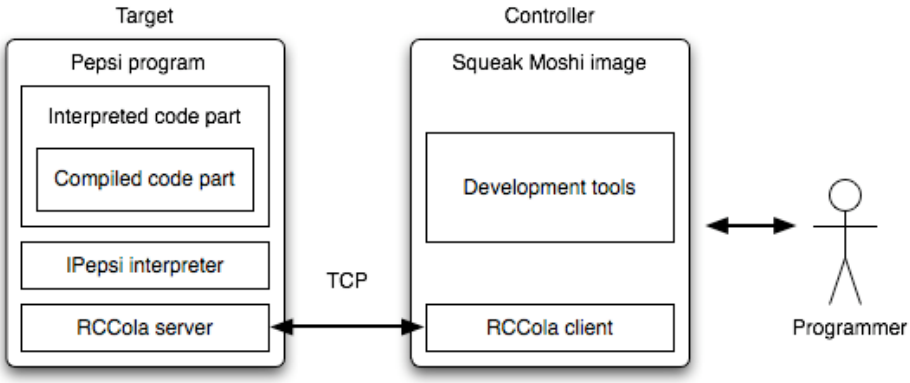
2009-04-16

Introduction

This memo describes a remote programming environment for COLA/Pepsi system. Because Pepsi language is still under development, it has only primitive programming tool like eval loop shell, text editor, and printf debugging. Remote Controlled Cola, RCCola is designed to improve this situation.

One of notable advantages of Smalltalk is its dynamic programming environment. Unlike other programming languages have strong distinction between programming and runtime, Smalltalk's tools allow a programmer to manipulate runtime objects during whole programming process. RCCola provides COLA/Pepsi with similar rich programming environment as Smalltalk through Squeak's flexible Morphic user interface framework.

Design

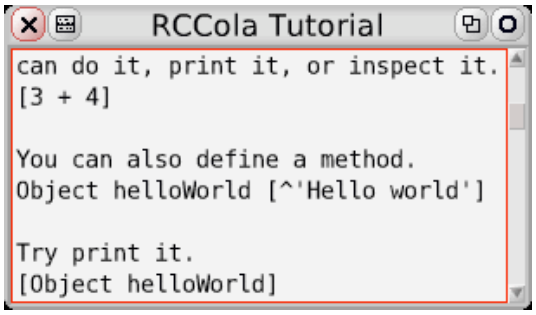


RCCola is based on client server model. A target is a Pepsi program with RCCola server component. A controller is a RCCola client component in Squeak. A Pepsi program is a hybrid of compiled code and interpreted code. A compiled code part is compiled by Pepsi's idc compiler from Pepsi source code to machine code. An interpreted code part is driven by a byte code interpreter. Both compiled code part and interpreted part accept same syntax. You can browse, edit, and inspect the target program in runtime through the controller.

When a target program starts running, the RCCola server component begins listening a TCP connection. The server component provides information about global variables and source code of the target. Through the connection, a RCCola client component observes the internal state of the target, modify the state, and even change running program. If server component is requested to modify a method in the program, the new method is replaced to original one and mark it as interpreted in rest of the session. The source code is stored in the client side, and it is used for further development.

Tools

Workspace



A workspace is the primary tool of RCCola. You can evaluate any Pepsi expression to the RCCola server by the workspace, but there are subtle differences compare to Squeak's workspace.

Pepsi expression

```
[3 + 4]                          " Immediate code "
Object helloWold [^'Hello world'] " Method definition "
```

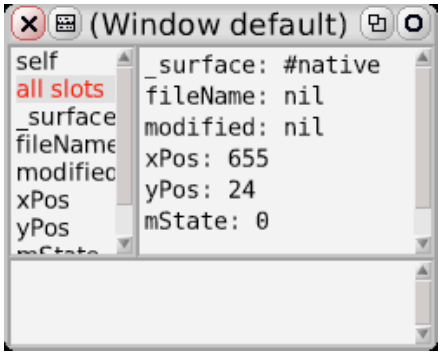
Immediate executable source code and method definition is described in Pepsi syntax.

Meta expression

```
@ RCColaInspector open: 'Window default'.
```

An expression begins with @ is executed by Squeak in stead of IPepsi. This is useful if you want to execute RCCola tool itself in the workspace.

Inspector



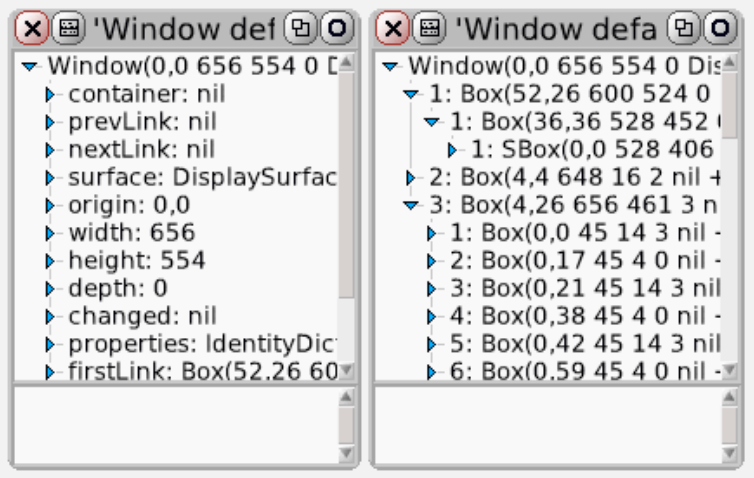

```
@ RCColaInspector open: 'Window default'. "Open RCColaInspector"
```

A RCColaInspector shows internal state of the Pepsi program. An object shown by the inspector must be referred through a global variable in the runtime environment. Because RCCola client doesn't keep object identity, a value is retrieved by a global variable each time when it is necessary. In other words, inspector actually works as watcher. For example, when if you inspect `xPos` slot of `surface` slot at an inspector showing `Window default`, RCCola client keeps request this expression.

```
((Window default) slotValueAt: 'surface') slotValueAt: 'xPos'
```

And `xPos` is not just a number at all, but it is a value bound with a certain context. `slotValueAt:` is a Pepsi method which gets a named slot value. It defined as RCCOLA high level API.

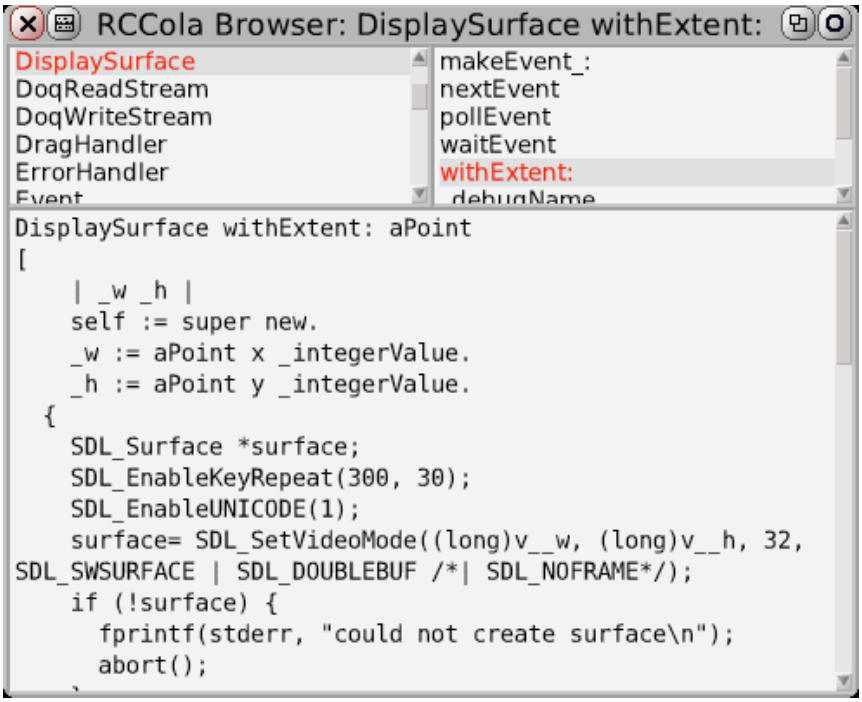
Explorer




```
@ RCColaExplorer open: 'window defined'. "Open RCColaExplorer"
@ QuicheExplorer open: 'Window default'. "Open QuicheExplorer"
```

RCCola provides two kinds of explorer. `RCColaExplorer` (the left window) is a normal explorer as same as Squeak. And `QuicheExplorer` (the right window) is a special version which is optimized for Quiche graphics framework. `QuicheExplorer` shows only parent / children relationship instead of all slots.

Browser




```
@ RCColaBrowser open: 'Window'
```

You can browse and edit source code of running Pepsi program by `RCColaBrowser`. Currently, RCCola server has introspection function to get the source code and it can dynamically accept new code, but the server doesn't save source code text. Saving source code is responsibility of RCCola client. When a user saves code, RCCola client send the code to the server and saved it into the local cache dictionary. When the browser is selected same method, cached version of source code is shown.

RCCola client API

RCCola client consists of three layers. Tool layer provides development tools, hight level API provides basic object introspection facilities, and low level API deal with RCCola network protocol. This section describes high level and low level API.

High level API

High level API provides functions necessary for development tools. Every RCCola tools talk to the server through the high level API. In the API, only `String` is used as input value, and `String` or collection of `String` is used as return value. There is not type mapping between Squeak class and Pepsi type other than that.

`RCCola >> slotKeysFor: expr`
: Return a collection of slot names of the expression.

`RCCola >> slotValueFor: expr at: keyName`
: Return a value at the slot

`RCCola >> slotValueFor: expr at: keyName put: aString`
: Set a value at the slot

`RCCola >> longPrintStringFor: expr`
: Return a long description of the expression.

`RCCola >> types`
: Answer a collection of all type names in the system.

`RCCola >> typeFor: expr`

Return a type name of the expression

`RCCola >> selectorsIn: typeName`

Return a collection of all selector names in the class.

`RCCola >> getSourceFor: selectorName in: typeName`

Return a source code string of this type and method.

`RCCola >> eval: expr`

Evaluate any expression. CR <-> LF conversion is applied.

Low level API

`RCCola >> rpc: aString`

Send a string to the server, and return the result. It may trigger an error signal based on the return value. This is a simple mechanism to convert Pepsi error to Squeak error.

Network Protocol

RCCola uses simple client server protocol. It assumes that the end of line character is `LF`. The protocol is explained by evaluating `"[3 + 4]"` program.

Request (Client to Server)

```
[3 + 4];;
```

pepsiExpression `;;`

Request is a Pepsi expression terminated with two semicolons.

Response (Server to Client)

```
#(ok '7')
```

`#(`[`ok`|`error`] *pepsiString* `)` *endOfLine*

Response is either OK or error. If it is OK, it follows result expression as a Pepsi string expression. In case of error, the string includes back trace information. Entire message is surrounded by parentheses like Smalltalk array, and terminated with `LF`.

Getting started

Contents

RCCola server is a part of babysteps source tree, and RCCola client is included in the Squeak Moshi image. The babysteps is a branch of idst source tree. It includes Quiche graphics framework and IPepsi interpreter. RCCola is built on top of IPepsi and Quiche. Some important files used in RCCola are listed.

`function/examples/quiche/rccola.st`

The source code of main executable file

`function/examples/socket/RShell.st`

RCCola network protocol

`function/jolt3/IShell.st`

Public interface of IPepsi interpreter

`examples/quiche/ExQuiche.st`

Additional module to let Quiche work smoothly with RCCola

`function/examples/quiche/paintTest.st`

An example file

Build

On the babysteps source tree, this command sequence builds all necessary module of RCCola. It is tested with Windows VISTA and Mac OSX 10.5.7.

```
make
cd function/jolt3
make
cd ../examples/quiche
make rccola
```

Start a session

When you invoke `rccola` command without arguments, RCCola server starts and listens on the connection at port 1972. Or you can specify a start up code as the argument. The start up code is run by IPepsi interpreter.

```
$ ./rccola paintTest.st
listening port 1972 ...
```

In Moshi image, you can open RCCola tools. `RCColaWorkspace` is the most useful entry point

```
RCColaWorkspace openTutorial
```

How to link your program with RCCola

`rccola.st` is the simplest use case to use RCCola with Quiche graphics framework. If you want to link other program against RCCola server module, you need to specify these options:

```
-I../socket ../socket/RShell.o -I../../jolt3 ../../jolt3/ipepsi.a
```

with idst command line, and insert some lines end of your program if necessary.

```
{ import: RShell }      " RCCola library "
{ import: ExQuiche }    " Quiche extension for RCCola "
{ include <SDL/SDL.h> } " Required by Quiche "

" If you have an event loop, you need to listen the server port in the loop. "
DisplaySurface waitEvent
[
    | event |
    [RShell wait: 0.                        " This listens the server port for 0 seconds. "
     event := self pollEvent] whileFalse: [ OS sleep: 0.020 ].
    ^event.
]

[
    RShell listen: 1972.                    " Set the listen port to 1972. "
    SymCode processArguments: OS arguments. " Pass the debug options to IPepsi interpreter. "
    RShell mainLoop.                        " Begin RCCola session. "
]
```

Future work

While RCCola had been designed for Pepsi language with IPepsi interpreter, tools in RCCola client and the architecture can be usable for other programming language which doesn't have rich development environment by itself. Based on RCCola client API and Morphic graphics framework, you could make rich tools quickly for new languages by a lot of reusable code available on Squeak.

One of next goals of RCCola is to establish a common network protocol which you can use it regardless of target language. Current implementation of RCCola network protocol is based on Pepsi syntax, but language neutral syntax like S-Expression or XML is desirable to support other languages.

Debugger support is another goal. Squeak's great productivity is come from integrated tools of debugger, browser, and inspector. RCCola debugger will provide same functionality to new language. To support debugger, it is necessary to design a remote debugging protocol for RCCola.