GingaFrEvo & GingaRAP

Evolução do Middleware Ginga para Múltiplas Plataformas (Componentização)
&
Ferramentas para Desenvolvimento e Distribuição de Aplicações Declarativas

Relatório Técnico RT- 25

Manual Operacional do Componente de Interação Multimodal

PROGRAMA CTIC

REDE NACIONAL DE ENSINO E PESQUISA

Apoio MCT

HISTÓRICO

Data	Versão	Descrição	Autor
29/01/10	0.1	Elaboração Inicial do Documento	Diogo Pedrosa e José Augusto Martins
16/06/10	0.2	Versão quase final do documento, ainda falta descrição da parte do input mode; Acréscimo do Apêndice 1.	Diogo Pedrosa e José Augusto Martins
30/06/10	0.3	Descrição do input mode; melhora na introdução	Caio Viel e Diogo Pedrosa
17/11/10	0.9	Acréscimo da explicação sobre a associação do id do evento multimodal a um evento de tecla na seção Protocolo de interação multimodal; Melhoria na seção "Inicialização do Componente de Interação Multimodal", para atualizar as modificações que devem ser feitas no componentes já existentes, incluindo o uso de parâmetros da main para habilitar aplicações de teste; Acréscimo de instruções para aplicações Lua e acréscimo do Apêndice 2; Acréscimo da tabela com descrição e caminho das aplicações de teste.	Diogo Pedrosa
18/11/10	1.0	Revisão final do documento	Erick Melo

ÍNDICE

1 Introdução

Este componente tem por objetivo oferecer a aplicações Ginga a possibilidade de receber dados multimodais de entrada (áudio, vídeo, imagem, tinta eletrônica, dados de acelerômetro, texto, voz) provenientes de múltiplos dispositivos, tais como, celulares, PDAs, netbooks, tablet PCs, notebooks e desktops.

Um componente que processe a interação do usuário através de dispositivos de entrada outros que o controle remoto convencional permite a construção de aplicações inovadoras. Por um lado, a possibilidade de interação via modalidades variadas, como por exemplo voz e gestos, ampliam o acesso à plataforma de usuários necessidades especiais. Por exemplo, um usuário com dificuldades motoras que dificultam que ele utilize um controle remoto convencional pode interagir via comandos de voz. Outro exemplo de melhoria de acessibilidade é o de um usuário tetraplégico, que pode utilizar gestos com sua cabeça que para interagir com a plataforma.

O uso de interação multimodal permite ainda a construção de aplicações inovadoras que no âmbito do paradigma ***Watch-and-Comment (WaC)***. Nesse paradigma, a interação multimodal do usuário é capturada e registrada, para ser associada ao programa apresentado na plataforma – associação essa que pode tomar a forma de anotações e comentários ou, ainda edição realizadas do lado do cliente.

Com o paradigma **WaC**, dispositivos da rede doméstica do usuário passar a fazer parte da gama de recursos integrados à plataforma de modo transparente e ubíquo. Por exemplo, um dispositivo que grava áudio pode permitir o envio de comandos ou comentários de voz.

1.1 Objetivos

Este documento descreve o manual operacional de uso do Componente para Captura de Interação Multimodal. É descrita a API de uso do componente além da estruturação do código-fonte e instruções para compilação, instalação e utilização.

1.2 Escopo

Este documento está associado ao projeto **GingaRAP**. O projeto é divido em diversos subprojetos. O componente especificado neste documento enquadra-se no subprojeto **Ginga-WAC**, atendendo aos objetivos de incorporação de serviços para suporte à interação multimodal.

1.3 Estrutura do Documento

O documento está estruturado da seguinte forma: a Seção 2 apresentado um tutorial de instalação do componente; a Seção 3 apresenta detalhes técnicos do componente: diagrama de classes e descrição da API do componente; a Seção 3 ainda apresenta exemplos de uso do componente.

2 Instalação

2.1 Conteúdo do componente

O Componente de Interação Multimodal possui arquivos de cabeçalho e de código fonte C++ e arquivos de projeto para compilação do módulo. A localização dos arquivos é mostrada na Tabela 1.

Tipo de arquivo	Diretório
Arquivos de cabeçalho	gingacc-mmi/include/
Arquivos de código fonte	gingacc-mmi/src/
Arquivos de projeto	gingacc-mmi

Tabela 1: Localização dos arquivos do Componente de Interação Multimodal.

Aplicações de teste também foram desenvolvidas para permitir a validação do componente. A localização dos arquivos é mostrada na Tabela 2.

Aplicação	Diretório
Aplicação de linha de comando, escrita em C, para enviar eventos multimodais completos, porém fixos, para o Ginga.	commandline-mm-event-gen-app
Aplicação para iPad, que envia eventos multimodais para o Ginga contendo apenas uma string de texto.	ipad-text-event-gen-app
Principal aplicação de teste, escrita em C++, que utiliza todas as funcionalidades providas pelo componente.	p2p-test-app
Aplicação em NCLua que recebe eventos multimodais contendo strings de texto.	lua-text-event-receiver-app

Tabela 2: Localização dos arquivos das aplicações de teste.

2.2 Requisitos do Sistema

O Componente de Interação Multimodal requer:

Software	URL
gingacc-system	http://svn.softwarepublico.gov.br/trac/ginga/browser/gingacc-cpp*
telemidia-util	http://svn.softwarepublico.gov.br/trac/ginga/browser/telemidia-util-cpp**
gingacc-player	http://svn.softwarepublico.gov.br/trac/ginga/browser/gingacc-cpp***
Avahi	http://www.avahi.org
D-Bus	http://www.freedesktop.org/wiki/Software/dbus
Logger (Lince Util)	http://lince.dc.ufscar.br/redmine/attachments/161/LinceUtil-0.1.tar.gz****
Xerces	http://xerces.apache.org/xerces-c*

Tabela 3: Requisitos do Sistema

* Já disponibilizados na máquina virtual 0.11.2 (http://www.ncl.org.br/ferramentas/fedora-fc7-ginga-i386.zip).

** Já disponibilizados na máquina virtual 0.11.2, mas é necessário realizar as alterações descritas no Apêndice 1.

*** Já disponibilizados na máquina virtual 0.11.2, mas é necessário realizar as alterações descritas no Apêndice 2.

****É necessário ter autorização de acesso.

A instalação dos componentes Avahi e D-Bus, quando utilizando a máquina virtual ginga versão 0.11.2, pode ser feita pelo gerenciador de pacotes yum:

```
yum install dbus-devel
yum install avahi-devel
yum install avahi-compat-libdns_sd-devel
```

Após a instalação é necessário inicializar os componentes:

```
dbus-daemon --system
/etc/init.d/avahi-daemon start
```

OBS.: O primeiro comando irá gerar uma saída com um aviso sobre a falta do usuário gdm. Isso não é um erro (*Unknown username "gdm" in message bus configuration file*).

O aviso do segundo comando deve ser um **"[OK]"**.

2.3 Compilação e Instalação

O Componente de Interação Multimodal pode ser obtido no seguinte endereço:

URL
http://lince.dc.ufscar.br/redmine/projects/ginga-ctic/

O módulo pode ser compilado e instalado através dos seguintes passos:

```
cd gingacc-cpp/gingacc-mmi
./autogen.sh
make
sudo make install
```

3 Componente de Interação Multimodal

Este componente desempenha duas funções principais: gerencia eventos de tecla e eventos multimodais e gerencia modos de entrada. Os eventos multimodais podem ser recebidos através da rede e por isso um protocolo para o recebimento das entradas multimodais foi especificado, o qual é explicado na próxima subseção.

3.1 Protocolo de interação multimodal

Um exemplo de interação multimodal é mostrado na Tabela 3:

```
<?xml version="1.0"?>
<multimodal xmlns="http://agua.intermidia.icmc.usp.br/intermidia"
    xmlns:inkml="http://www.w3.org/2003/InkML"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="multimodal.xsd"
    id="idDoEventoDeTeste">

    <head>
          <device id="DEADBEEF-DEAF-BABA-FEED-BABE00000006"       model="iPhone 3GS"/>
          <user id="59616261-6461-6261-4E50-472050325033"/>
          <timestamp begin="2010-05-19T09:30:10.5" end="2010-05-19T09:30:17.8"/>
    </head>
    <body>
          <value id="nomeDaApp">
                PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4NCg==
          </value>
          <value id="projeto">
                PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4NCg==
          </value>

          <text>Texto de teste 1</text>
          <text>Texto de teste 2</text>

          <inkml:ink>
                <inkml:trace>
                      10 0, 9 14, 8 28, 7 42, 6 56, 6 70, 8 84, 8 98, 8 112, 9 126, 10 140,
                      13 154, 14 168, 17 182, 18 188, 23 174, 30 160, 38 147, 49 135,
                      58 124, 72 121, 77 135, 80 149, 82 163, 84 177, 87 191, 93 205
                </inkml:trace>
                <inkml:trace>
                      130 155, 144 159, 158 160, 170 154, 179 143, 179 129, 166 125,
                      152 128, 140 136, 131 149, 126 163, 124 177, 128 190, 137 200,
                      150 208, 163 210, 178 208, 192 201, 205 192, 214 180
                </inkml:trace>
                <inkml:trace>
                      227 50, 226 64, 225 78, 227 92, 228 106, 228 120, 229 134,
                      230 148, 234 162, 235 176, 238 190, 241 204
                </inkml:trace>
                <inkml:trace>
                      282 45, 281 59, 284 73, 285 87, 287 101, 288 115, 290 129,
                      291 143, 294 157, 294 171, 294 185, 296 199, 300 213
                </inkml:trace>
                <inkml:trace>
                      366 130, 359 143, 354 157, 349 171, 352 185, 359 197,
                      371 204, 385 205, 398 202, 408 191, 413 177, 413 163,
                      405 150, 392 143, 378 141, 365 150
                </inkml:trace>
          </inkml:ink>

          <accel xValue="3" yValue="-2" zValue="1"/>

          <voice>aqui entra um voicexml</voice>

          <binary filename="file1.txt" mimetype="text/plain">
                PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4NCg==
```

```
56          </binary>
57          <binary filename="file2.txt" mimetype="text/plain">
58                PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4NCg==
59          </binary>
60    </body>
61</multimodal>
```

Tabela 4: Exemplo de Interação multimodal

Na Tabela 3, as linhas 02 a 06 mostram o cabeçalho do arquivo xml. A linha 02 declara o elemento raiz (multimodal) e o *namespace* ao qual esse elemento pertence. As linhas 03 e 04 mostram os outros *namespaces* incluídos na criação desse documento de interação; note que ambos trazem consigo um prefixo para evitar conflito de nomes. A linha 05 define o XML Schema utilizado para validar o documento e a linha 06 define o id do evento no formato de uma string. Esse id pode conter um valor qualquer, mas caso seja um dos valores especificados para o atributo key de um elemento simpleCondition da norma Ginga-NCL, um evento regular de tecla também é disparado. Isso faz com que o componente permita também que diferentes dispositivos, como um celular, possam ser usados para mudar de canal ou aumentar volume, por exemplo.

As linhas 08 a 12 definem o cabeçalho do protocolo de interação multimodal. Nesse cabeçalho devem estar contidos o ID do dispositivo no formato UUID e o modelo do dispositivo no formato de uma string (linha 09), o ID do usuário também no formato UUID (linha 10) e um *timestamp* que define o começo e o fim da interação no formato dateTime.

As linhas 13 a 60 definem o *payload* do documento, esse *payload* deve conter todos os dados da interação devidamente discriminados entre os tipos possíveis. Os tipos de dados que podem ser inseridos são:

- **Mapa de chave-valor** – como nos exemplos das linhas 14 a 16 e 17 a 19, onde o atributo ID é do tipo string e corresponde à chave enquanto o conteúdo da tag é o valor, em qualquer codificação. Esse tipo de dado possibilita uma maior flexibilidade, pois permite que aplicações façam uso de tipos não previstos na API, de acordo com sua necessidade. O programador deve apenas ter consciência que esse tipo é genérico e os dados podem estar sendo utilizados por outras aplicações, portanto apesar do evento entregar esses dados para sua aplicação, não necessariamente esses dados pertencem à mesma.
- **Strings de texto** – como nas linhas 21 e 22, úteis para que textos já prontos nos dispositivos possam ser enviados para aplicações, como por exemplo, em uma aplicação de chat.
- **Dados de tinta eletrônica** – como mostrado nas linhas 24 a 48 que são lidas pelo parser da inkmllib[1] resultantes da interação do usuário com um dispositivo sensível ao toque ou munido de uma caneta eletrônica, por exemplo. Como o tipo ink está sendo lido por um parser de terceiros é necessário que todas as tags ink contenham um prefixo como pode ser visto na linha 03.
- **Dados de acelerômetro** – compostos por 3 strings – atributos xValue, yValue e zValue – com os valores dos eixos x, y e z respectivamente (linha 50).
- **Dados de voz** – podem ser resultantes de sistemas de reconhecimento de fala ou servirem de entrada para sistemas sintetizadores de voz, como mostrado na linha 52. Atualmente o planejamento é para que esses dados sejam codificados no padrão voiceXML[2] mas ainda não está implementado nesta versão.

---

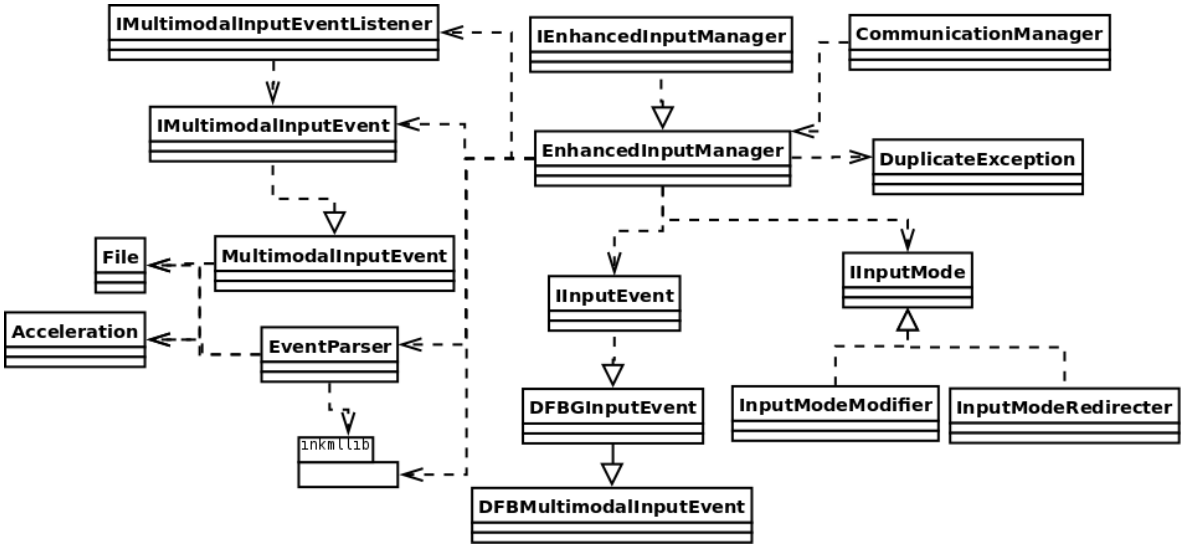
[1] disponível em: http://sourceforge.net/apps/trac/inkmltk/wiki/InkMLLib última visita: 06/2010

[2] disponível em: http://www.w3.org/TR/voicexml21/ última visita: 06/2010

- **Dados binários** – compreendem áudio, video e imagem e arquivos em geral que devem conter o nome e o *mime type do* arquivo especificados como atributos e o valor do dado deve estar codificado em base64Binary localizado como o elemento da tag como mostrado nas linhas 54 a 56 e 57 a 59.

3.2 Diagrama de Classes

O diagrama de classes seguinte apresenta de forma simplificada a estrutura do Componente de Interação Multimodal.



A classe *EnhancedInputManager* tem o objetivo de substituir a classe *br::pucrio::telemidial::ginga::core::system::io::InputManager*. Ela implementa a interface *IEnhancedInputManager* e é responsável por manter listas de observadores interessados em receber eventos, tanto multimodais quanto de tecla, e notificá-los da ocorrência de algum desses eventos.

As classes *InputModeModifier* e *InputModeRedirecter* implementam a interface *IInputMode*. Os desenvolvedores podem utilizar essas classes para criar modos de interação diferenciados. Através desses modos é possível multiplexar várias funcionalidades em uma mesma tecla do controle remoto, uma para cada modo de interação existente. Os modos feitos através de herança da classe *InputModeRedirecter* irão redirecionar as interações recebidas para algum módulo especificado pelo desenvolver. Já os modos feitos com herança da classe *InputModeModifier* irão apenas modificar os parâmetros da interação realizada, para depois utilizar o mecanismo de envio de eventos existente na classe *EnhancedInputManager* para despachar os eventos para seus destinatários.

A classe *CommunicationManager* é responsável por oferecer um serviço ZeroConf[3] de recebimento de um evento multimodal no formato xml explicado na seção anterior. O gerenciador de comunicação executa em uma *thread* separada, que é inicializada assim que a instância do EnhancedInputManager é criada. Essa *thread* permanece num loop infinito que realiza as seguintes tarefas: 1) espera a conexão de um dispositivo no *socket* criado por ela; 2) recebe os dados transmitidos pelo dispositivo, os quais consistem de um evento *xml* como descrito na Seção

[3] http://www.zeroconf.org/

3.1 e os armazena numa string; e 3) passa a string recebida para o gerenciador de eventos, para que este realize as ações apropriadas.

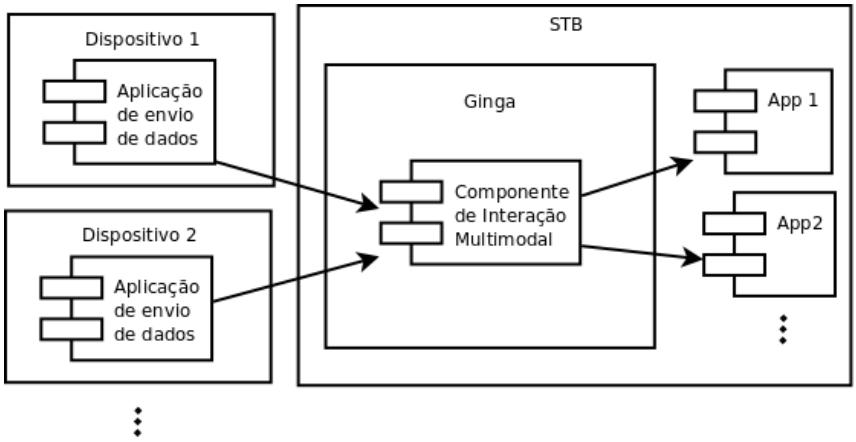
A classe *EventParser* é responsável por realizar o parser de um evento multimodal no formato xml definido pelo protocolo de interação multimodal. Ela é acionada pelo EnhancedInputManager e retorna um objeto do tipo *MultimodalInputEvent*.

A classe *MultimodalInputEvent* representa um evento multimodal e implementa a interface *IMultimodalInputEvent*. Os dados do eventos são armazenados em diversos atributos, já que o evento pode conter diferentes tipos de dados, e todos possuem métodos get e set associados. O id do evento é armazenado numa string. Textos são armazenados num vetor de string. O conjunto de chaves/valores é armazenado num map. Por enquanto esse mapa está armazenando também as informações contidas no cabeçalho do protocolo. Os dados de tinta eletrônica são armazenados num objeto da classe *Ink* pertencentes ao namespace *inkmllib*. Os dados de acelerômetro são armazenados num objeto da classe *Acceleration*. Dados de voz ainda não estão sendo armazenados. Os arquivos de imagem, áudio e vídeo, são armazenados num vetor de objetos da classe *File*.

3.3 Utilização

O componente foi desenvolvido utilizando a versão 0.11.2 da implementação de referência.

A utilização desse componente é realizada em 3 etapas. A primeira etapa é a execução do componente no Ginga, a segunda é a geração de eventos multimodais por parte de dispositivos externos e a terceira é o recebimento de eventos por parte das aplicações interessadas.



3.3.1 Inicialização do Componente de Interação Multimodal

Para utilizar esse componente é necessário alterar arquivos de 3 componentes da implementação de referência. Para facilitar o uso, esses 3 componentes são disponibilizados junto com o CMI. Abaixo segue uma breve descrição das alterações necessárias:

- Este componente deve ser colocado em execução pelo *PresentationEngineManager*, utilizando a classe ComponentManager. Para isso, deve-se incluir o arquivo de cabeçalho mmi/IEnhancedInputManager.h e utilizar o *namespace* br::ufscar::lince::ginga::core::mmi. O componente começa a gerenciar eventos assim que uma instância da classe IEnhacendInputManger é obtida pela primeira vez. Este componente deve ser usado no

lugar do InputManager, e não em conjunto com ele. No arquivo PresentationEngineManager.h, deve-se mudar:

```
//#include "system/io/IInputManager.h"

#include "mmi/IEnhancedInputManager.h"
using namespace ::br::ufscar::lince::ginga::core::mmi;
...
//                static IInputManager* im;

                  static IEnhancedInputManager* im;
```

- No arquivo PresentationEngineManager.cpp, deve-se mudar:

```
//#include "../../gingacc-system/include/io/InputManager.h"

#include "../../gingacc-mmi/include/EnhancedInputManager.h"
...
//       IInputManager* PresentationEngineManager::im       = NULL;

         IEnhancedInputManager* PresentationEngineManager::im = NULL;
...
// im = ((InputManagerCreator*)(cm->getObject("InputManager")))();
eim = ((EnhancedInputManagerCreator*)(cm->getObject("InputManager")))();
```

- main.cpp – Acrescentar código que faz com que o fluxo trave até que o método *unlockConditionSatisfied* do gerenciador de eventos seja chamado. Isso permite sair do ginga recebendo a mensagem MAIN ALL DONE e, ao mesmo tempo, permite que o ginga permaneça em execução mesmo se não existir uma aplicação NCL em execução e se não estiver ocorrendo o recebimento de um TS. Permite também, por exemplo, que a aplicação de teste do componente P2P permaneça em execução esperando pelo recebimento de algum arquivo. A chamada ao método *unlockConditionSatisfied* está sendo feita no método *EnhancedInputManager::run()*, quando a tecla F10 é pressionada.

```
#include "../../gingacc-eim/include/EnhancedInputManager.h"
...
#include "mmi/IEnhancedInputManager.h"

using namespace ::br::ufscar::lince::ginga::core::mmi;
...
         bool enableMultimodalTestApp = false;

         bool enableP2PTestApp = false;

...
         cout << "    --enable-multimodal-test-app Enable the execution of the";

         cout << " application that tests the gingacc-mmi component." << endl;

         cout << "    --enable-p2p-test-app Enable the execution of the application";

         cout << " that tests the gingacc-p2p component." << endl;
...
                  } else if (strcmp(argv[i], "--enable-multimodal-test-app") == 0) {

                           enableMultimodalTestApp = true;

                  } else if (strcmp(argv[i], "--enable-p2p-test-app") == 0) {
```

```
                        enableP2PTestApp = true;
...
                if (enableP2PTestApp) {

                        cout << "Iniciando P2PTestApp" << endl;

                        P2PTest *p2pTest = P2PTest::getInstance();

                }



                if (enableMultimodalTestApp) {

                        MultimodalTest *mmt = MultimodalTest::getInstance();

                        mmt->registerListener();

                }
...
IEnhancedInputManager* eim;
#if HAVE_COMPSUPPORT
        eim = ((EnhancedInputManagerCreator*)(cm->getObject("InputManager")))();
#else
        im = EnhancedInputManager::getInstance();
#endif
        eim->waitForUnlockCondition();
```

- gincacc-cpp/gingacc-cm/files/componentDescription.xml – Comentar a linha:

```
<symbol        object="InputManager"        creator="createInputManager"        destroyer="destroyInputManager"
interface="IInputManager"/>
```

e acrescentar o trecho:

```
<!-- MMI -->
  <component package="gingacc-mmi" name="libgingaccmmi.so" version="0.11.2" type="dynamic">
    <dependency componentName="libgingaccsystemthread.so" version="0.11.2"/>
    <dependency componentName="libgingaccsystemiocodemap.so" version="0.11.2"/>
    <dependency componentName="libgingaccsystemio.so" version="0.11.2"/>
    <dependency componentName="libgingaccsystemiodfb.so" version="0.11.2"/>
    <dependency componentName="libxerces-c.so" version="2.1.27"/>
    <symbol object="EnhancedInputManager" creator="createEnhancedInputManager"
destroyer="destroyEnhancedInputManager" interface="IEnhancedInputManager"/>
    <location type="local" uri="/usr/local/lib/ginga/"/>
    <repository uri="http://www.telemidia.puc-rio.br/~marcio/components/vm/"/>
  </component>
```

3.3.2 Aplicação de envio de dados multimodais

Qualquer dispositivo que suportar o protocolo **ZeroConf** pode servir fonte de eventos multimodais para aplicações Ginga. Duas aplicações simples foram desenvolvidas como prova de conceito.

A primeira é uma aplicação de linha de comando que roda em Linux. Ela possui o código necessário para buscar o serviço ZeroConf provido pelo CommunicationManager utilizando a biblioteca Avahi, conectar-se ao socket do CommunicationManager e enviar o conteúdo de um

xml chamado eventoMultimodal.xml e localizado na mesma pasta em que a aplicação está executando. Ela permanece em um laço infinito enviando o mesmo conteúdo a cada vez que a tecla enter é pressionada.

A segunda aplicação foi desenvolvida para iPad, na plataforma Mac. Ela mostra um botão no canto superior direito que permite a busca por serviços ZeroConf. O usuário deve escolher nessa lista o serviço provido pelo CommunicationManager e utilizar o teclado virtual para escrever em um campo de texto disponibilizado na interface. Depois disso, o usuário deve apertar o botão enviar para que a aplicação gere um xml contendo um evento multimodal simples, que possui apenas o texto digitado, conecte-se ao socket do CommunicationManager e envie o gerado. Podem ser enviados quantos eventos o usuário desejar.

3.3.3 Aplicações multimodais

A versão atual do componente permite que aplicações residentes escritas em C++ recebam eventos multimodais. Para permitir um uso mais amplo do componente, APIs em Java e Lua também devem ser implementadas. A API Lua começou a ser desenvolvida e já permite o recebimento de strings.

Aplicações C++

As aplicações multimodais devem implementar a interface *IMultimodalInputEventListener*, registrar um listener de um evento multimodal junto ao *EnhancedInputManager* e tratar os eventos recebidos dentro do método *userEventReceived*, como mostrado no exemplo abaixo.

```
#include "mmi/IMultimodalInputEventListener.h"
#include "mmi/IMultimodalInputEvent.h"
#include "mmi/EnhancedInputManager.h"
using namespace ::br::ufscar::lince::ginga::core::mmi;
#include <mmi/Acceleration.h>
#include <mmi/Ink.h>
using namespace ::br::ufscar::lince::ginga::core::mmi::inkmllib;
...
EnhancedInputManager* eim = EnhancedInputManager::getInstance();
eim->addMultimodalInputEventListener(this);
...
bool MultimodalTest::userMultimodalEventReceived(
                IMultimodalInputEvent* ev) {

        LoggerUtil_info(logger, "userMultimodalEventReceived("
                        "IMultimodalInputEvent* ev)");

        LoggerUtil_info(logger, "ID = " << ev->getId());

        map<string, void*>* values = ev->getMultimodalValuesMap();
        for (map<string, void*>::iterator i = values->begin();
                        i != values->end(); i++) {
                char* value = (char*)(i->second);
                LoggerUtil_info(logger, "(" << i->first << ", " << value << ")");
        }

        vector<string>* strs = ev->getStrings();
        for (vector<string>::iterator j = strs->begin();
                        j != strs->end(); j++) {
                LoggerUtil_info(logger, *j);
        }
```

```
        Ink *ink = ev->getInk();
        for (int t = 0; t < ink->vectTrace->size(); t++) {
                Trace trace = ink->vectTrace->at(t);

                cout << "Dados do trace " << t << ": ";

                int limite;
                if (trace.vectX->size() > trace.vectY->size()) {
                        limite = trace.vectX->size();
                } else {
                        limite = trace.vectY->size();
                }

                for (int i = 0; i < limite; i++) {
                        long x = trace.vectX->at(i);
                        long y = trace.vectY->at(i);
                        cout << x << " " << y << ", ";
                }

                cout << endl;

        }

        vector<File*>* binaries = ev->getBinaries();
        for (vector<File*>::iterator k = binaries->begin();
                        k != binaries->end(); k++) {
                LoggerUtil_info(logger, "Nome: " << (*k)->getName() <<
                                " Mimetype: " << (*k)->getMimetype() <<
                                " Payload: " << (*k)->getPayload());
        }

        Acceleration *accel = ev->getAcceleration();
        LoggerUtil_info(logger, "Aceleração: xValue: " << accel->getXValue() <<
                                                " yValue: " << accel->getYValue() <<
                                                " zValue: " << accel->getZValue());

        return false;
}
```

Aplicações Lua

Para que aplicações em Lua recebam eventos multimodais, elas precisam apenas registrar um tratador de eventos, o que já é feito pela maioria das aplicações, como mostrado no exemplo abaixo:

```
event.register(handler)
```

No tratador de evento, a aplicação deve checar se o evento recebido é do tipo multimodal, e em caso positivo, acessar os dados multimodais contidos, como mostrado abaixo:

```
function handler (evt)
        if evt.class == 'multimodal' then
                if (evt.stringCount >= '1') then
                        doSomething(evt.string1)
                end
        end
end
```

4 Responsáveis pelo Documento

Nome: Cesar Augusto Camillo Teixeira

E-mail: cesar@dc.ufscar.br

Telefone: (0xx16) 3351-8614

Instituição: Laboratório para Inovação em Computação e Engenharia da Universidade Federal de São Carlos (Lince-UFSCar)

Nome: Maria da Graça Campos Pimentel

E-mail: mgp@icmc.usp.br

Telefone: (0xx16) 3373-9657

Instituição: Laboratório Intermidia do Instituto de Computação e Matemática computacional da Universidade de São Carlos (Intermidia - ICMC/USP)

Nome: Erick Lazaro Melo

E-mail: erick_melo@dc.ufscar.br

Telefone: (0xx16) 3351-8614

Instituição: Laboratório para Inovação em Computação e Engenharia da Universidade Federal de São Carlos (Lince-UFSCar)

5 Apêndice 1 – Alterações no gingacc-system

Este apêndice descreve como alterar o componente gingacc-system para que ofereça as funcionalidades necessárias para o componente gingacc-mmi.

- Acrescentar o seguinte código no arquivo gingacc-system/src/io/interface/input/CodeMap.cpp:

```
int CodeMap::addCode(string codeStr) {
	if (hasCode(codeStr)) {
		return KEY_NULL;
	}
	int code = nextCode++;
	(*valueMap)[code] = codeStr;
	(*keyMap)[codeStr] = code;
	return code;
}

bool CodeMap::removeCode(int code) {
	if (!hasCode(code)) {
		return false;
	}
	string codeStr = getValue(code);
	if (!hasCode(codeStr)) {
		return false;
	}
	if (valueMap->erase(code) != 0 && keyMap->erase(codeStr) != 0) {
		return true;
	}
	return false;
}

bool CodeMap::removeCode(string codeStr) {
	if (!hasCode(codeStr)) {
		return false;
	}
	int code = getCode(codeStr);
	if (!hasCode(code)) {
		return false;
	}
	if (valueMap->erase(code) != 0 && keyMap->erase(codeStr) != 0) {
		return true;
	}
	return false;
}

bool CodeMap::hasCode(int code) {
	if (valueMap->count(code) == 0) {
		return false;
	}
	return true;
}

bool CodeMap::hasCode(string codeStr) {
	if (keyMap->count(codeStr) == 0) {
		return false;
	}
	return true;
```

```
}
```

- Acrescentar o seguinte código no arquivo gingacc-system/include/io/interface/input/CodeMap.h:

```
public:
        //static const int KEY_MODE_CHANGE;
        int addCode(string codeStr);
        bool removeCode(int code);
        bool removeCode(string codeStr);
        bool hasCode(int code);
        bool hasCode(string codeStr);
private:
        int nextCode;
```

Executar os seguintes comandos:

```
cd /usr/src/gingaNcl/gingacc-cpp/gingacc-system
chmod +x autogen.sh
./autogen.sh
make
make install
```

6 Apêndice 2 – Alterações no gingacc-player

Este apêndice descreve como alterar o componente gingacc-player da implementação de referência para que aplicações em Lua pudessem passar a receber eventos multimodais.

- include/LuaPlayer.h -

Trocar:

```
#include "system/io/IInputManager.h"
```

Por:

```
#include "mmi/IEnhancedInputManager.h"

using namespace ::br::ufscar::lince::ginga::core::mmi;
```

Trocar:

```
public Player, public Thread, public io::IInputEventListener {
```

Por:

```
public Player, public Thread, public io::IInputEventListener,

        public IMultimodalInputEventListener {
```

Trocar:

```
IInputManager* im;
```

Por:

```
IEnhancedInputManager* im;
```

Acrescentar:

```
bool multimodalUserEventReceived (IMultimodalInputEvent* evt);
```

- src/application/imperative/lua/LuaPlayer.cpp -

No construtor, trocar:

```
// this->im = ((InputManagerCreator*)(cm->getObject("InputManager")))();
```

por:

```
this->im = ((EnhancedInputManagerCreator*)(cm->getObject("InputManager")))();
```

Ainda no construtor, acrescentar:

```
this->im->addMultimodalInputEventListener(this);
```

Acrescentar o método:

```
bool LuaPlayer::multimodalUserEventReceived (IMultimodalInputEvent* evt) {
        this->lock();

        if (this->isHandler) {
                map<string,string> lua_evt;
                lua_evt["class"] = "multimodal";
                lua_evt["id"] = evt->getId();
```

```
                vector<string>* strings = evt->getStrings();
                stringstream temp;
                temp << strings->size();
                lua_evt["stringCount"] = temp.str();

                if (strings->size() >= 1) {
                        lua_evt["string1"] = strings->at(0);
                }

                ext_postHash(this->L, lua_evt);
        }

        this->unlock();
        return true;
}
```

Executar os seguintes comandos:

```
cd /usr/src/gingaNcl/gingacc-cpp/gingacc-player
chmod +x autogen.sh
./autogen.sh
make
make install
```