The Dataflow Library and the Arts

Stjepan Rajko
Arizona State University

NSF IGERT trainee,
Arts, Media and Engineering Program

Ph. D. student, Computer Science

M.F.A. student, Dance

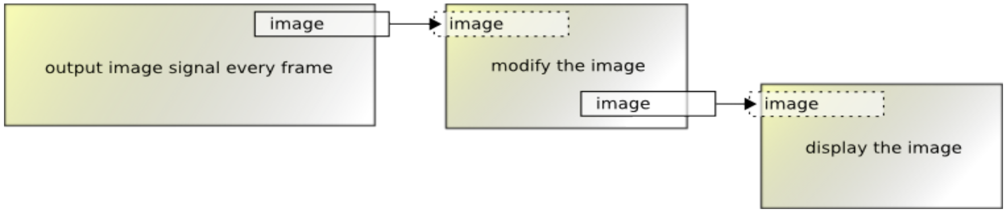
stjepan.rajko@gmail.com http://dancinghacker.com/

Presentation Outline

- The Dataflow library
  - Motivation
  - Overview
  - Future Directions
- The Arts
  - Art based interactive installations
  - AMELiA and a pattern recognition library developed with the interactive installations
  - Rehearsal Assistant – a media tool for performing arts

Dataflow

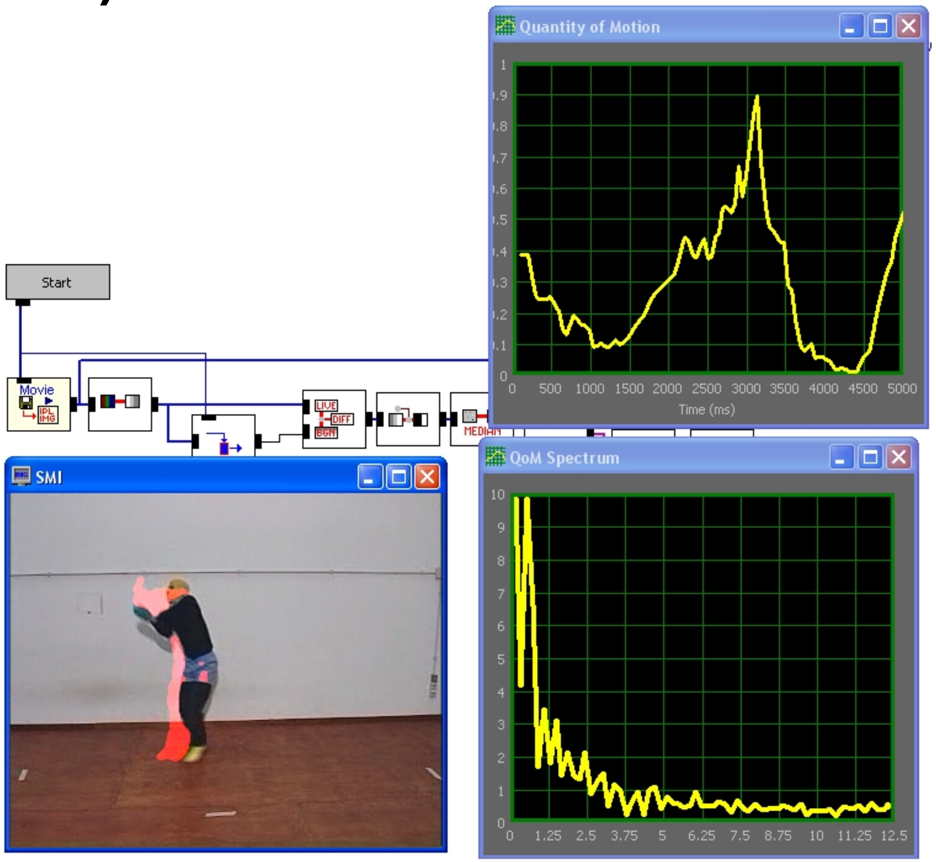
- models a program as a directed graph of the data flowing between operations (wikipedia)
- Example:



- My interest comes from experience with various visual programming environments

Visual Dataflow Environments

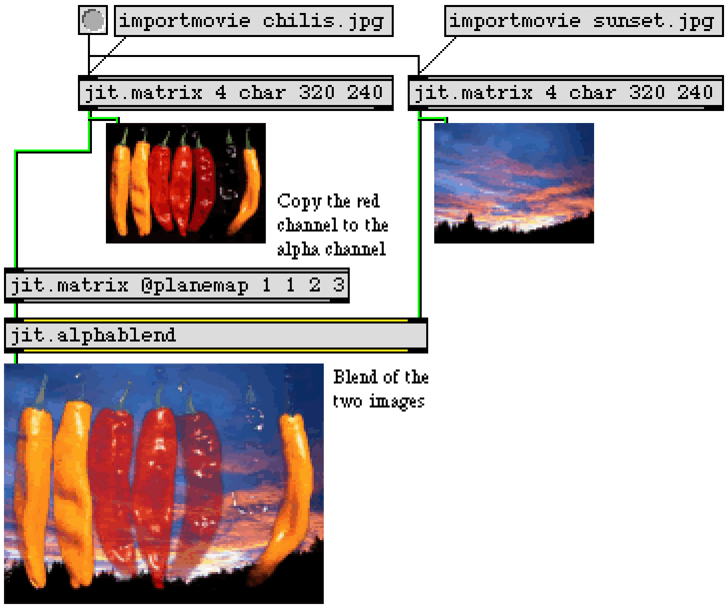
- EyesWeb (real-time multimodal distributed interactive applications)



EyesWeb © University of Genoa

Visual Dataflow Environments

- Max / MSP (interactive graphical programming environment for music, audio, and media)



Max / MSP © Cycling74

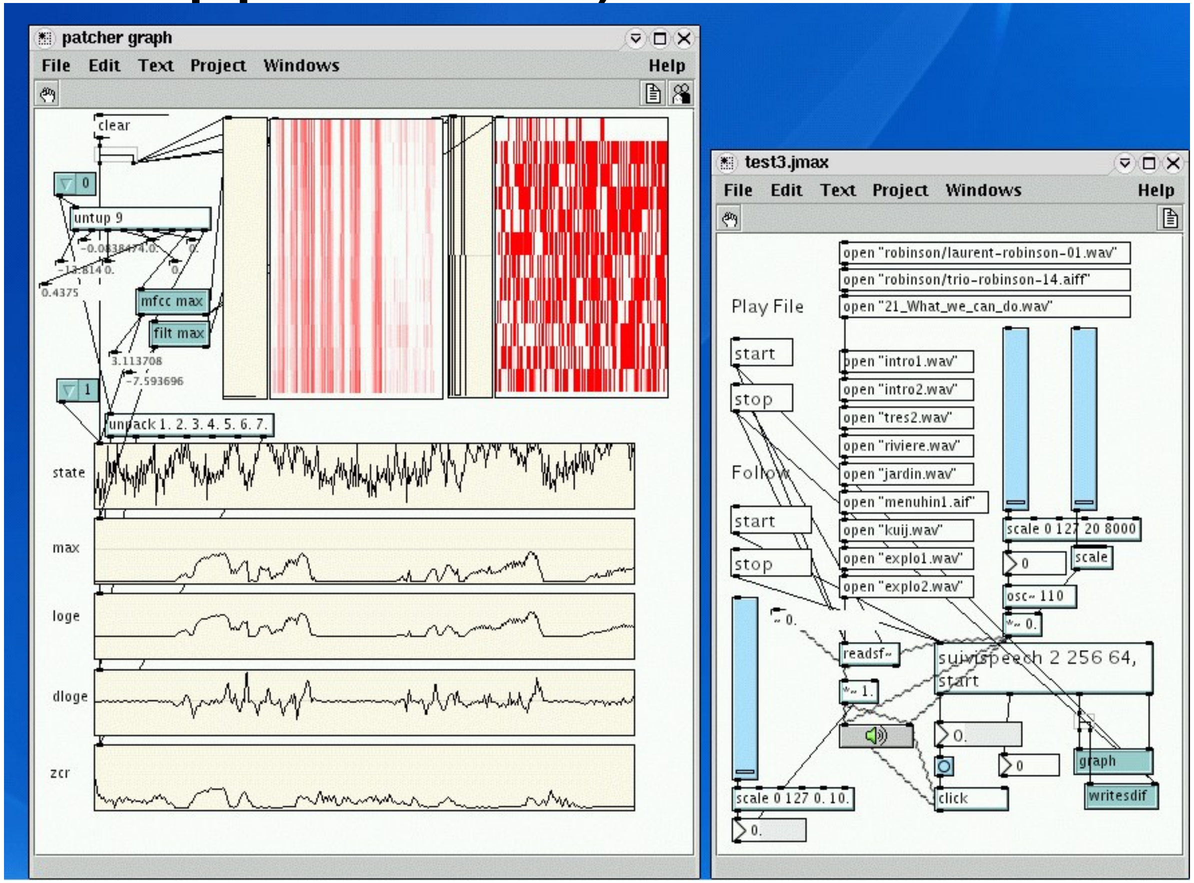
Visual Dataflow Environments

- Pure Data (creation of interactive computer music and multimedia works)

Pure Data © Miller Puckette

Visual Dataflow Environments

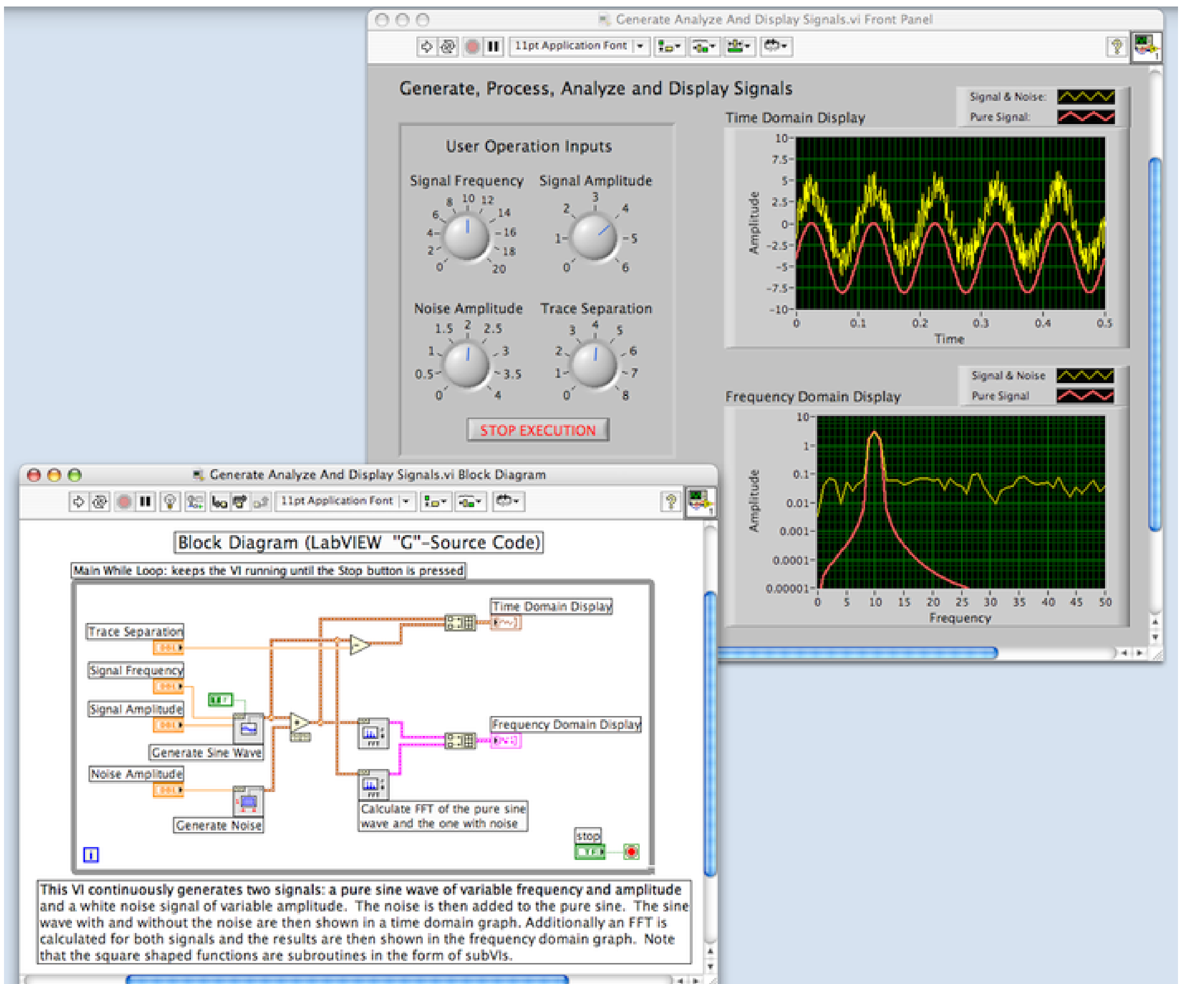
- jMax (building interactive real-time music and multimedia applications)



jMax © Ircam-Centre Pompidou

Visual Dataflow Environments

- LabVIEW (commonly used for data acquisition, instrument control, and industrial automation)



LabVIEW © National Instruments

Advantages of dataflow programming

- not exclusive of other paradigms
- promotes some good programming practices
- makes development and maintenance very intuitive
- can be divided between threads, processors, or computers more easily
- lends itself well to visual programming environments

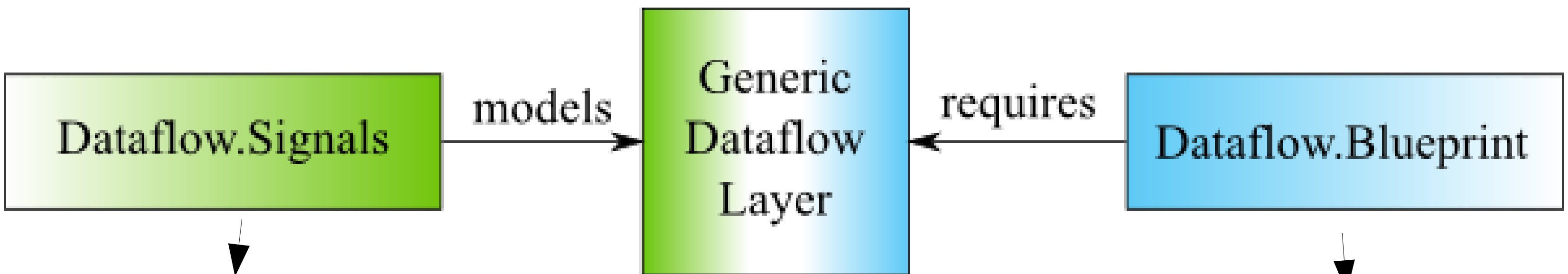
The Dataflow library

Original intention: let's make another dataflow framework using Boost.Signals

Google Summer of Code (thanks Google, thanks Boost!)

Let's make a generic dataflow library

Dataflow library overview



number of implemented
components using Boost.Signals
as a data transport mechanism

providing runtime reflection and network
modeling using the Boost Graph Library

centered around concepts, adaptable to
various dataflow frameworks and data transport
mechanisms

- Fundamental concepts:
  - Port
  - Component

Dataflow.Signals

- Uses function calls to transport data with connections stored using Boost.Signals
- Offers
  - A number of useful general-purpose components, and building blocks for implementing new components.
  - Various free functions and operators for connecting and using components.

Dataflow.Signals example 1

```cpp
#include <boost/dataflow/signals/support.hpp>
#include <iostream>
#include <string>

void consumer_function(const std::string &data)
{
      std::cout << data << std::endl;
}

void signal_function_example()
{
      boost::signal<void(const std::string &)> producer;
      boost::function<void(const std::string &)> consumer(consumer_function);

      connect(producer, consumer); // make a connection between the two.
      producer("Hello World"); // signal goes to the function.
}
```

Dataflow.Signals example 2

```cpp
#include <boost/dataflow/utility/bind_mem_fn.hpp>

class consumer_class
{
public:
    void consumer_mem_fn(const std::string &data)
    {
        std::cout << data << std::endl;
    }
};

void signal_mem_fn_example()
{
    using boost::dataflow::utility::bind_mem_fn;

    boost::signal<void(const std::string &)> producer;
    consumer_class consumer;

    // make a connection between the producer and consumer.
    connect(producer, bind_mem_fn(&consumer_class::consumer_mem_fn, consumer));
    producer("Hello World"); // signal goes to the member function.
}
```

Dataflow.Signals example 3

```cpp
class producer_component
    : public signals::filter<producer_component, void(const std::string &)>
{
public:
    void invoke()
    {
        // out is the default output signal.
        out("Hello World");
    }
};

class consumer_component
    : public signals::consumer<consumer_component>
{
public:
    // This is our signal consumer.
    void operator()(const std::string &data)
    {
        std::cout << data << std::endl;
    }
};

void component_component_example()
{
    producer_component producer;
    consumer_component consumer;

    // Because we inherited from filter/consumer, connecting is easy.
    connect(producer, consumer);
    producer.invoke(); // producer sends "Hello World" to consumer.
};
```

Dataflow.Signals example 4

```
#include <boost/dataflow/signals/connection/operators.hpp>

class filter_component
    : public signals::filter<filter_component, void(const std::string &)>
{
public:
    // This is our signal consumer.  It will also produce a signal when called.
    void operator()(const std::string &data)
    {
        out(data + "!");
    }
};

void component_component_component_example()
{
    producer_component producer;
    filter_component filter;
    consumer_component consumer;

    // The following is equivalent to:
    // connect(producer, filter);
    // connect(filter, consumer);
    producer >>= filter >>= consumer;
    // producer sends "Hello World" to filter, filter sends "Hello World!".
    producer.invoke();
}
```

Dataflow.Signals example 5

```
// instantiate all of the components we need
signals::storage<void ()> banger;
signals::storage<void (float)> floater(2.5f);
signals::storage<void (float)> collector(0.0f);

// ---Connect the dataflow network ----------------------------
//
// ,--------.  void()
// | banger | -------.
// `--------'        |
//                   |
//                   V
//            ,-(send_slot)-.  void(float)   ,-----------.
//            |   floater    | -------------> | collector |
//            `-------------'                `-----------'
//
// -----------------------------------------------------------

banger >>= floater.send_slot();
floater >>= collector;

// signal from banger is will invoke floater.send(), which causes
// floater to output 2.5
banger();
BOOST_CHECK_EQUAL(floater.at<0>(), 2.5f);
BOOST_CHECK_EQUAL(collector.at<0>(), 2.5f);

floater.close();
floater(1.5f); // change the value in floater
invoke(floater); // we can also signal floater directly
BOOST_CHECK_EQUAL(collector.at<0>(), 1.5f);
```

More examples with library

- Implementing distributed dataflow applications using Dataflow.Signals and Boost.Asio

```
// ---Connect the dataflow network on one computer------
//
//     ,---------.     ,---------.
//     |  input  | --> |  sender  --- -- - (socket)
//     `---------'     `---------'
//
// -----------------------------------------------------

// ---Connect the dataflow network on other computer -----------
//
//                     ,---------.     ,---------.     ,---------.
// (socket) - -- ---  receiver | --> |  proc    | --> |   out   |
//                     `---------'     `---------'     `---------'
//
// ------------------------------------------------------------
```

More examples with library

- An image processing network using Dataflow.Signals and Boost.GIL

```
// ---Connect the dataflow network -----------------------------------------
//
//                     ,---------.
//                     | control | -----------------------+
//                     `---------'                        |
//                          |                             |
//                          v                             v
//    ,-------.     ,-----------.                   ,-------.
//    | timer | --> | generator | -+--------------> 0       |
//    `-------'     `-----------'  |                |  mux  | -> | display |
//                                 |  ,---------.   |       |    `---------'
//                                 +->| filter |--> 1       |
//                                    `---------'   `-------'
//
// -------------------------------------------------------------------------
```

More examples from the documentation

- Pull-based networks
- Disconnecting
- Multiple slots of different signatures
- Multiple inputs of the same signature
- Multiple outputs
- Implementing new components using filter and consumer classes

Dataflow.Signals components

Component	Use
consumer	Base class for your own input-only components.
filter	Base class for your own input/output or output-only components.
storage	Stores signal arguments.
counter	Counts the number of signals passing through.
junction	Convenient when multiple producers need to be connected to the same set of consumers. Also has gate functionality.
multiplexer	Allows selection of which of the input ports is forwarded
mutex	Provides mutexing on incoming signals for multithreaded environments
condition	Signals a threading condition whenever a signal is received
function	Allows any Boost.Function object to be applied to a passing signal
chain	Chains a number of components together into a new component
socket_sender and socket_receiver	Allow a signal dataflow network to straddle a network socket

Component Bases - instantiator

- Instantiator instantiates an object while a signal is passing
- Example – mutex instantiates a scoped_lock on a mutex:

```
template<typename Signature,
    typename OutSignal=SIGNAL_NETWORK_DEFAULT_OUT,
    typename SignalArgs=typename default_signal_args<Signature>::type
>
class mutex : public
    instantiator<
        mutex<Signature, OutSignal, SignalArgs>,
        boost::mutex, boost::mutex::scoped_lock, Signature, OutSignal, SignalArgs>
{
};
```

Component Bases - applicator

- Applicator applies a function object to a member whenever a signal passes
- Example – counter increments an int:

```
template<
    typename Signature,
    typename OutSignal=SIGNAL_NETWORK_DEFAULT_OUT,
    typename T=int,
    typename SignalArgs=typename default_signal_args<Signature>::type
 >
class counter : public applicator<
    counter<Signature, OutSignal, T, SignalArgs>,
    T, detail::postincrement<T>, Signature, OutSignal, SignalArgs>
{
public:
    /** Initializes the internal counter to 0. */
    counter()
    {   reset(); }
    /** Sets the internal counter to 0.    */
    void reset()
    {   counter::member = 0; }
    /** \return The internal signal counter.    */
    T count() const
    {   return counter::member; }
};
```

Component Bases - conditional

- Conditional only forwards a signal when a condition evaluates to true
- Example – junction forwards signal when open:

```
template<
    typename Signature,
    typename OutSignal=SIGNAL_NETWORK_DEFAULT_OUT,
    typename SignalArgs=typename default_signal_args<Signature>::type
>
class junction
    : public conditional<
        junction<Signature, OutSignal, SignalArgs>,
        volatile bool, detail::identity<bool>, Signature, OutSignal, SignalArgs>
{
public:

    /** Initializes the junction to be enabled.    */
    junction(bool opened=true)
    {
        junction::member=opened;
    }
    /** Enables the junction (signals will be forwarded).    */
    void open() {junction::member = true;}
    /**  Disables the junction (signals will not be forwarded).    */
    void close() {junction::member = false;}
};
```

Generic Component Bases

- modifier
  - Can modify a passing signal (e.g., signals::function)
- conditional_modifier
  - Can modify a passing signal, it is passed on optionally (e.g., signals::storage)

The Dataflow library

Generic Dataflow Layer

Generic Dataflow Layer concepts

- Port
  - ComplementedPort
  - VectorPort
  - KeyedPort
- Component
- Operations
  - BinaryOperable (Connectable, Extractable...)
  - UnaryOperable (AllDisconnectable)
  - ComponentOperable (Invocable)

Dataflow.Signals as an example

- Ports
  - boost::signal (call producer)
  - boost::function (call consumer)
  - Connectable if of the same signature
- Components
  - signals::consumer (base class)
  - signals::filter (base class)
  - ...

How boost::signal becomes a ComplementedPort

- We declare a PortTraits type – boost::dataflow::signals::producer

```
template<typename T>
struct producer
    : public complemented_port_traits<ports::producer, boost::function<T>, tag>
{
    typedef T signature_type;
};
```

- We then register boost::signal with the PortTraits

```
template<typename Signature, typename Combiner, typename Group, typename GroupCompare>
struct register_traits<boost::signal<Signature, Combiner, Group, GroupCompare>, signals::tag >
{
    typedef signals::producer<Signature> type;
};
```

How boost::function becomes a Port

- We declare a PortTraits type – boost::dataflow::signals::consumer

```
template<typename T>
struct consumer
    : public port_traits<ports::consumer, tag>
{
    typedef T signature_type;
};
```

- We then register boost::signal with the PortTraits

```
template<typename Signature>
struct register_traits<boost::function<Signature>, signals::tag >
{
    typedef signals::consumer<Signature> type;
};
```

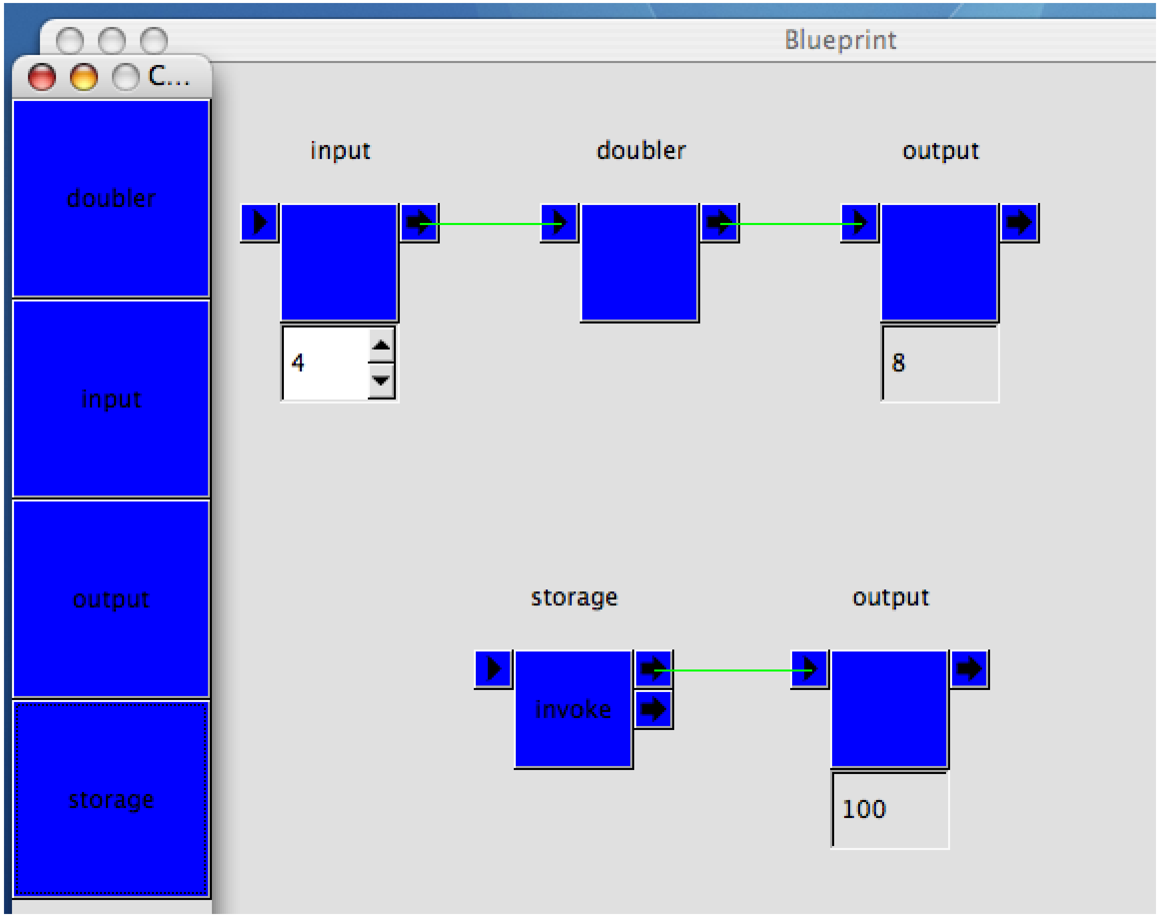
How they become Connectable

```
template<typename T>
struct binary_operation_impl<signals::producer<T>, signals::consumer<T>, operations::connect>
{
    typedef boost::signals::connection result_type;

    template<typename Producer, typename Consumer>
    result_type operator()(Producer &producer, Consumer &consumer)
    {
        return producer.connect(consumer);
    }
};
```

- They can now be treated in a generic fashion through the Dataflow library
- Other things offered at the generic level:
  - Disconnecting, Extracting, Invoking
  - Support for function objects as KeyedPorts

What are the benefits of generic Dataflow?

- Generic algorithms / applications... like a visual programming environment!



Dataflow.Blueprint

- Very much a work in progress
- Offers run-time reflection of components
- Offers run-time polymorphic classes with all Dataflow functionality (e.g., useful if components are loaded from a factory / bank / plugin)
- Stores the dataflow network blueprint in a BGL graph, which can then be analyzed
- ... see example in documentation

Future Directions

- Expanding the blueprint layer
- Improving GUI based dataflow network construction
- Support for more layers
- Supporting a pin-based approach, as proposed by Tobias Schwinger

No. 2: The Arts

The Arts

- Art settings provide a lot of very interesting computational challenges
- Outline
  - Art based interactive installations
  - AMELiA – a pattern recognition library developed with the interactive installations
  - Rehearsal Assistant – a media tool for performing arts

Art based interactive installations

- Installation art uses sculptural materials and other media to modify the way a particular space is experienced (wikipedia)
- At Arts, Media and Engineering (AME), we construct environments that allow users to:
  - Explore their movement potential and creativity
  - Manipulate sound using movement
  - Learn concepts (physics, sustainability)
  - Communicate
  - Collaborate
- Art brings crucial knowledge to the construction

Example 1: Handjabber

- Handjabber is an interactive audio installation that draws from understanding of how people use nonverbal language to communicate with each other.
- The installation takes advantage of motion capture technology and is meant for two people at a time.
- ... show video

Example 2: The "Enactive" environment

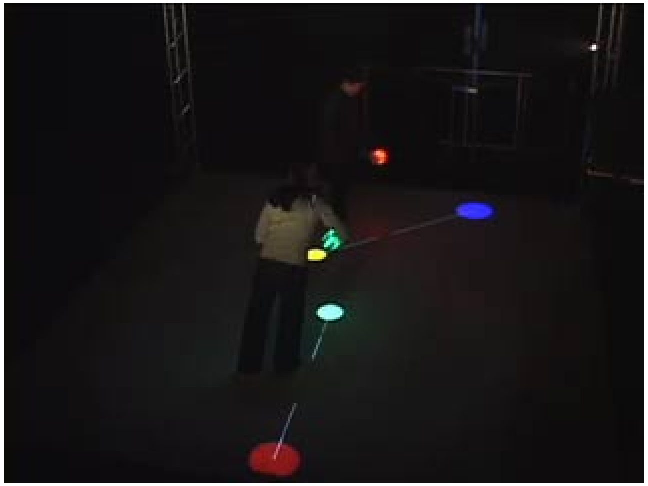
- This environment analyzes certain spatial qualities of movement, and provides audio feedback reflecting the analysis
- The audio feedback can increase the user's awareness of their movement.
- ... show video

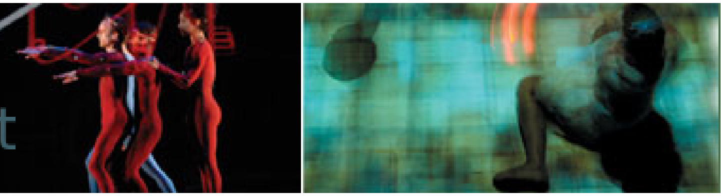
Example 3: Biofeedback for stroke rehabilitation

- The environment provides a purposeful, engaging, visual and auditory scene in which patients can practice functional therapeutic reaching and grasping tasks, while receiving different types of simultaneous feedback indicating measures of both performance and results.
- ... show video

Example 4: the SMALLab environment

- Interactivity using tangible objects
- Visual and audio feedback
- A modular design environment, many interactive scenarios
  - Mediated complexity / sustainability (experiential environments modeling complex problems)
  - Talking Circle (communication environment inspired by Native American traditions)
  - ...
- ... show video





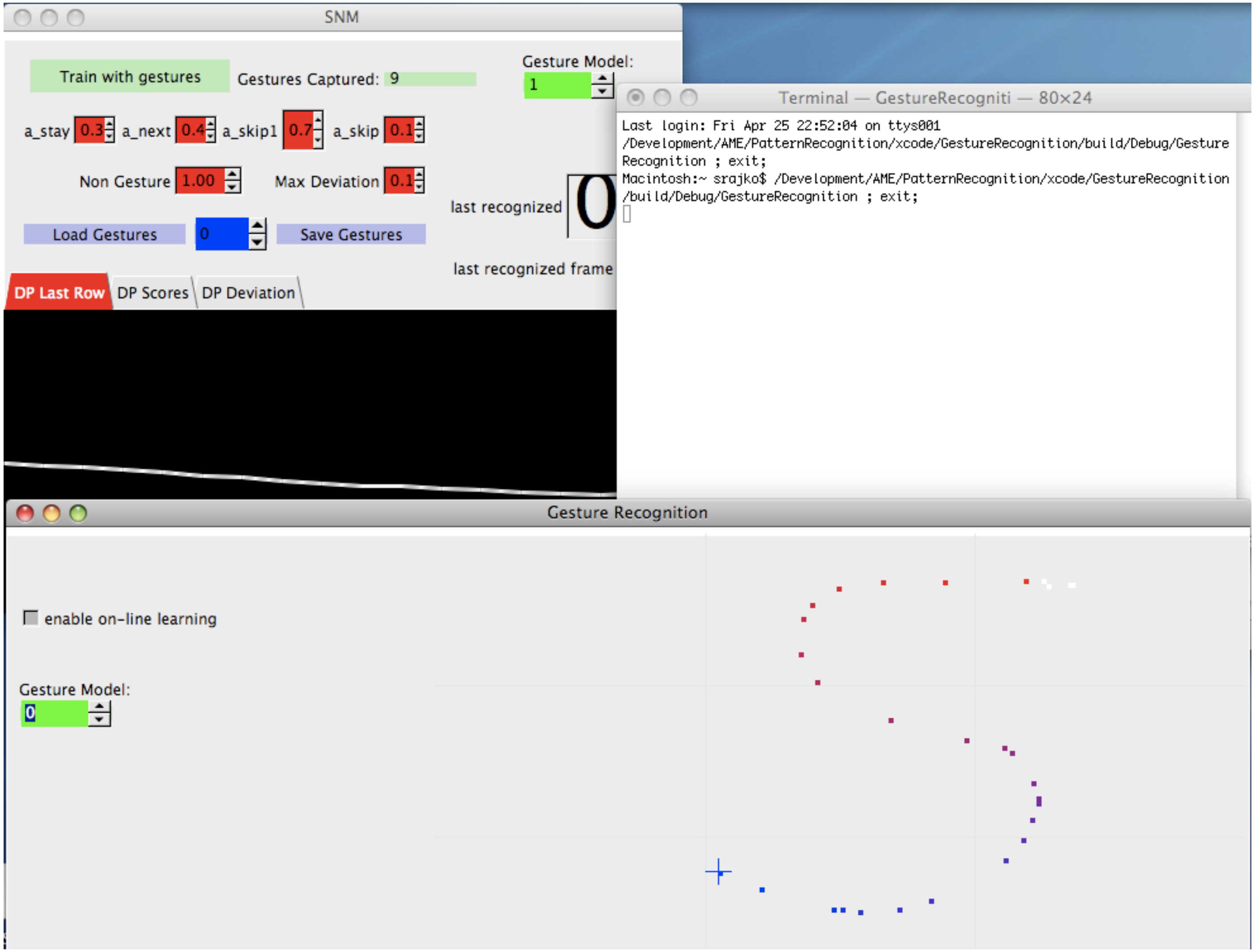
- AME recently allowed release of software under open source licenses
- Moving code from proprietary codebase to AMELiA (currently GPL-ed)
- First major library moved: A Patterns library
  - Hidden Markov model gesture recognition
  - Reduced parameter models for reduced training requirements / better run-time performance

AME Patterns library

- a library for pattern analysis and recognition
- generic in the sense that it can be applied to patterns in different domains / modalities.
- to use the library in a particular domain, the library user simply needs to provide appropriate models of concepts used by the library.
- Currently focused on gesture recognition
  - Body gestures sensed using motion capture data
  - Gestures of tangible objects (e.g., ball) in space
  - Gestures of a mouse in a plane

Mouse Gesture Recognition Example

- ... run example



Using the Patterns library

- To recognize gestures, we need a ModelState:

```
class normal_model_state
{
public:
    typedef double observation_type;

    normal_model_state(double min_st_dev)
        : m_min_st_dev(min_st_dev)
    {}
    double operator()(double a)
    {
        return pdf(m_distribution,a);
    }
    template<typename Range>
    void train_with_examples(const Range &examples)
    {
        if (examples.size()==0)
            return;
        double mean=ame::range::mean(examples);
        double st_dev=ame::range::standard_deviation(examples, mean);

        m_distribution = boost::math::normal_distribution<>(mean, (std::max)(st_dev, m_min_st_dev));
    }
    const boost::math::normal_distribution<> distribution() const
    {   return m_distribution; }
private:
    double m_min_st_dev;
    boost::math::normal_distribution<> m_distribution;
};
```

We can now train gestures

- All it takes is one or more examples of each gesture

```
patterns::gesture_set_recognition<gesture_model_type> gsr(0.1, true);

using namespace boost::assign;

std::vector<std::vector<double> > training;

training.push_back(std::vector<double>());
training.back() += 1, 2, 3, 4, 5, 6;
gsr.add_gesture_with_examples(training, 0.2, 0.7, 0.1, 6, 10, 0, patterns::normal_model_state(1));

training.clear();
training.push_back(std::vector<double>());
training.back() += 6, 5, 4, 3, 2, 1;
gsr.add_gesture_with_examples(training, 0.2, 0.7, 0.1, 6, 10, 0, patterns::normal_model_state(1));
```

... and recognize gestures

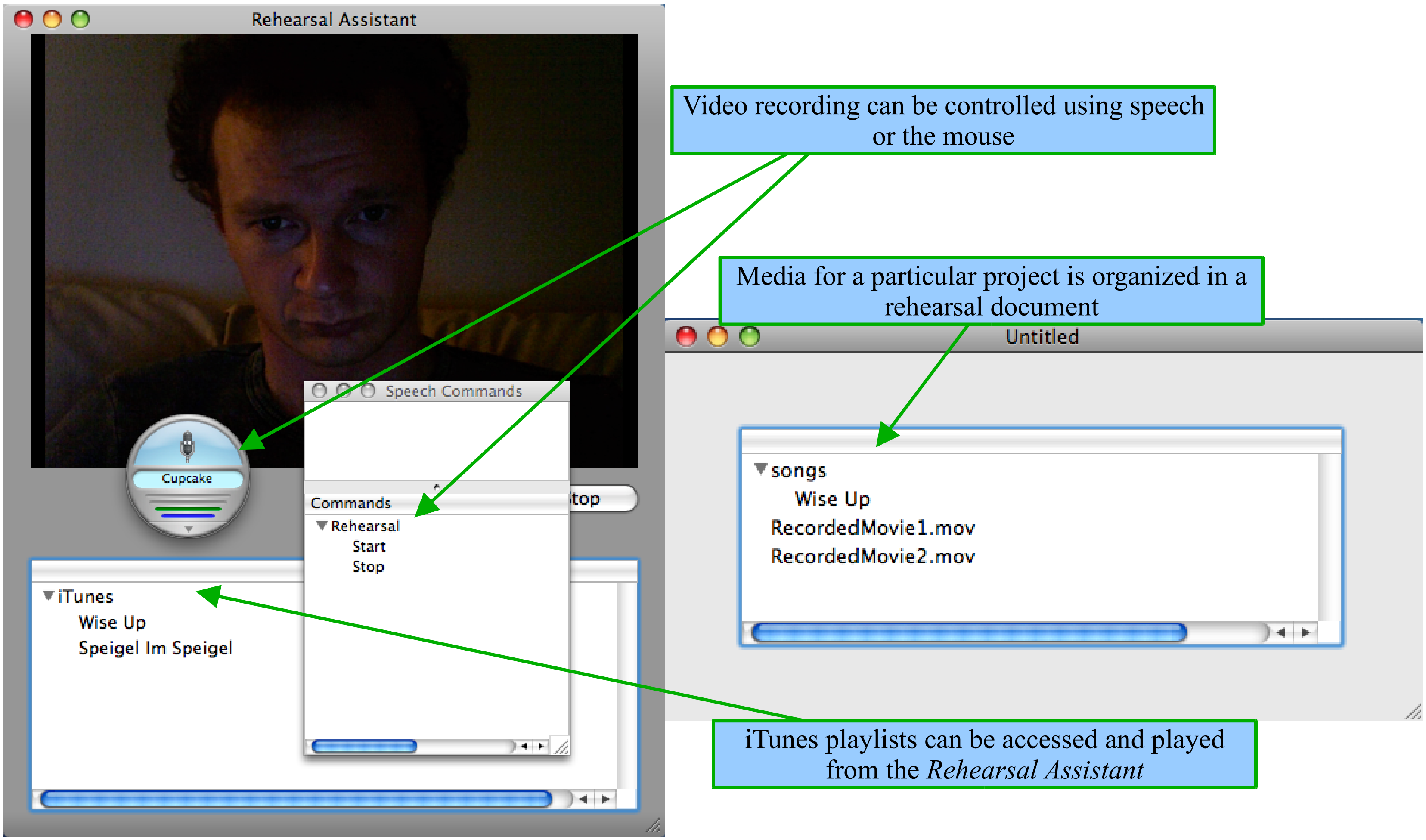
- ... run example

```
double input;
for(;;)
{
    std::cout << "enter next observation (enter a negative number to quit): " << std::endl;
    std::cin >> input;
    if (input<0)
        break;

    std::cout << "matches gesture: " << gsr.match(input) << std::endl;

    gsr.get_match(match);
    std::cout << "match elements (observation, state): ";
    BOOST_FOREACH(gesture_model_type::match_element_type &element, match)
    {
        std::cout << "(" << element.first << ", " << element.second << ") ";
    }
    std::cout << std::endl;
}
```

Rehearsal Assistant

- In very early prototype phase
- a software tool which helps performance artists conduct rehearsals by addressing media-related needs

Prototype


Rehearsal Assistant
Video recording can be controlled using speech or the mouse
Speech Commands
Cupcake
Commands
Rehearsal
Start
Stop
iTunes
Wise Up
Speigel Im Speigel
Media for a particular project is organized in a rehearsal document
Untitled
songs
Wise Up
RecordedMovie1.mov
RecordedMovie2.mov
iTunes playlists can be accessed and played from the Rehearsal Assistant

Audio Features

- preparation of audio/music tracks before rehearsal (track selection, cuing presets for relevant parts, tempo adjustment)
- playback during the rehearsal

Video

- recording of rehearsals via a built-in or external camera
- playback of rehearsal videos
- streamlined sharing of videos through online repositories

Annotation

- verbal annotations made by the rehearsal director can be recorded and synchronized with the corresponding sections of the rehearsal video recording
- annotations can be played back while replaying the video (additionally, a list of annotations is compiled and can be used to jump to specific parts of the recording)
- integration with MetaVidWiki for adding and viewing of text annotations through a web based interface, synchronized with the uploaded video recordings (work to ease the uploading of video to MetaVidWiki was accepted as a Google Summer of Code 2008 project)

Thank You!!!

- Questions?

stjepan.rajko@gmail.com