Human Centered Multimedia
Institute of Computer Science

Social Signal Interpretation Tutorial

Johannes Wagner <wagner@openssi.net>

(last update: 15.04.14)

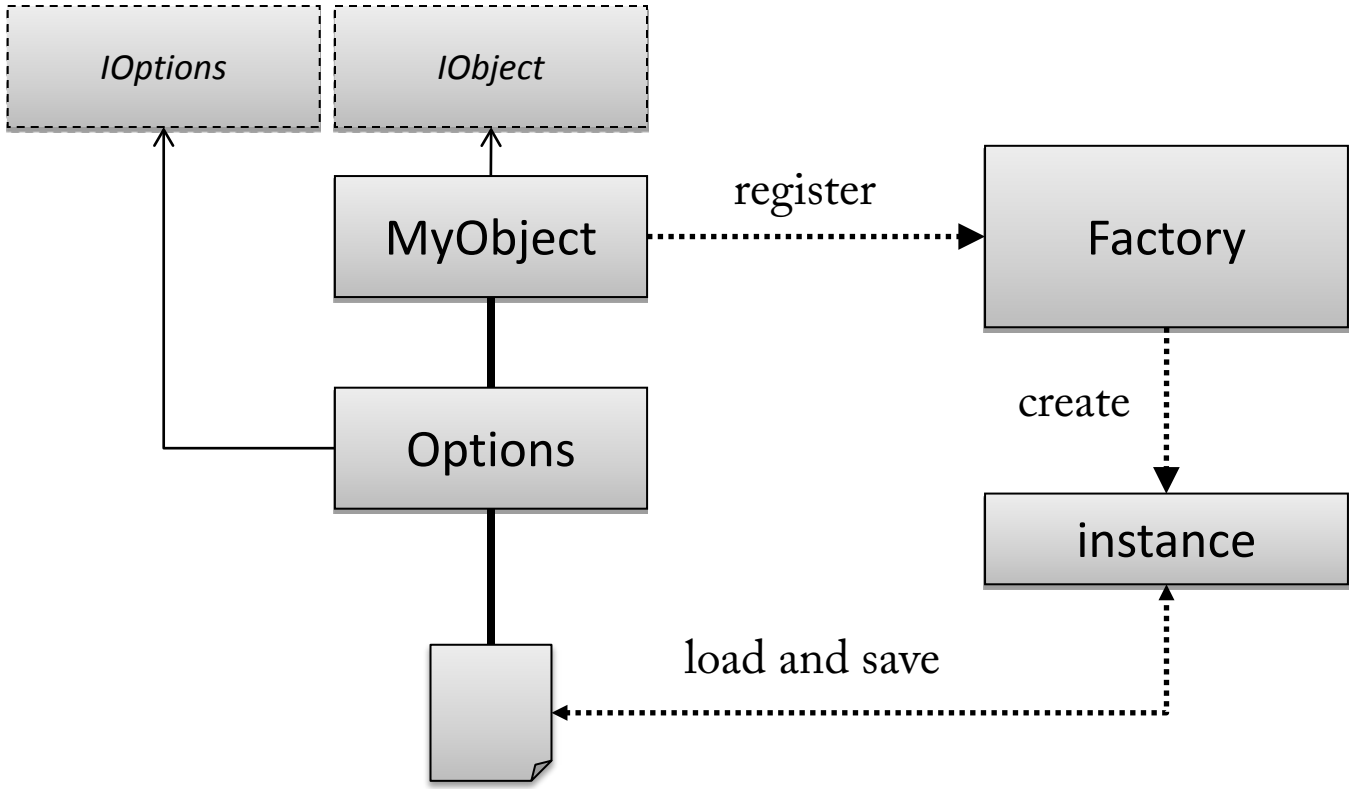
http://openssi.net

- A comprehensive tutorial that describes architecture and features of the Social Signal Interpretation (SSI) framework
- For reasons of clarity and comprehensibility the following slides contain mostly code snippets; full source code examples are found here
- Running source code examples requires Microsoft Visual Studio 2010 and the SSI core framework available for free download here
- For further questions and corrections please contact the author

Social Signal Interpretation

OBJECTS

Object Management



Factory

```
class IObject: {

  typedef IObject * (*create_fptr_t) (const ssi_char_t *file);

  virtual IOptions *getOptions () = 0;
  virtual const ssi_char_t *getName () = 0;
  virtual const ssi_char_t *getInfo () = 0;
  virtual ssi_object_t getType () { return SSI_OBJECT; };
  virtual void setLogLevel (ssi_size_t level) {};

};

class Factory {
...
  static bool Register (const ssi_char_t *name,
                        IObject::create_fptr_t create_fptr);
  static IObject *Factory::Create (const ssi_char_t *name,
        const ssi_char_t *file = 0,
        bool auto_free = true);
...
};
```

Options

```
class IOptions {

  virtual bool addOption (const char *name,
    void *ptr,
    ssi_size_t num,
    ssi_type_t type,
    const ssi_char_t *help) = 0;
  virtual bool getOptionValue (const char *name, void *ptr) = 0;
  virtual bool setOptionValue (const char *name, void *ptr) = 0;
  virtual ssi_option_t *getOption (const char *name) = 0;
  virtual ssi_option_t *getOption (ssi_size_t index) = 0;
  virtual ssi_size_t getSize () = 0;
};

class OptionList : public IOptions {
  ...
  static bool LoadXML (const ssi_char_t *filename, IOptions &list)
  static bool SaveXML (const ssi_char_t *filename, IOptions &list);
  ...
};
```

Object Example

```
class MyObject : public IObject {
public:

  class Options : public OptionList {
  public:
     Options () : toggle (false) {
       string[0] = '\0';
       addOption ("toogle", &toggle, 1, SSI_BOOL, "i'm a toggle");
       addOption ("string", string, SSI_MAX_CHAR, SSI_CHAR, "i'm a string");
     }
     void setString (const ssi_char_t *string) {
        ssi_strcpy (this->string, string);
     }
     bool toggle;
     ssi_char_t string[SSI_MAX_CHAR];
    };
...
```

Object Example

```
...
  static const ssi_char_t *GetCreateName () { return "myobject"; };
  static IObject *Create(const ssi_char_t *file) { return new MyObject (file); };
  ~MyObject ();

  Options *getOptions () { return &_options; };
  const ssi_char_t *getName () { return GetCreateName (); };
  const ssi_char_t *getInfo () { return "just a sample object"; };

  virtual void print ();

protected:

  MyObject (const ssi_char_t *file = 0);
  ssi_char_t *_file;
  Options _options;
  static char ssi_log_name[];
};
...
```

Object Example

```
char MyObject::ssi_log_name[] = "myobject__";

  MyObject::MyObject (const ssi_char_t *file)
  : _file (0) {
    if (file) {
       if (!OptionList::LoadXML (file, _options)) {
          OptionList::SaveXML (file, _options);
           }
       _file = ssi_strcpy (file);
    }
  }

  MyObject::~MyObject () {
    if (_file) {
      OptionList::SaveXML (_file, _options);
      delete[] _file;
    }
  }
  ...
```

Object Example

```
...
void MyObject::print () {

  ssi_msg (SSI_LOG_LEVEL_BASIC, "calling print()..");
  ssi_print ("string=%s\n", _options.string);
  if (!_options.toggle) {
     ssi_wrn ("toggle is off");
  }
}

int main () {

  Factory::Register (MyObject::GetCreateName (), MyObject::Create);

  MyObject *o = ssi_factory_create (MyObject, "object", true);
  o->getOptions ()->setString ("hello world");
  o->getOptions ()->toggle = true;
  o->print ();
  o->getOptions ()->print (ssiout);

  Factory::Clear ();
};
```

Social Signal Interpretation

STRINGS

Strings

- 99 % of the time represented as an array of chars (`ssi_char_t`)
- In some cases you may want to use `String` class for convenience,

  e.g. `String str = String ("hello") + String (" ") + String ("world");`
- Global strings are managed by Factory:

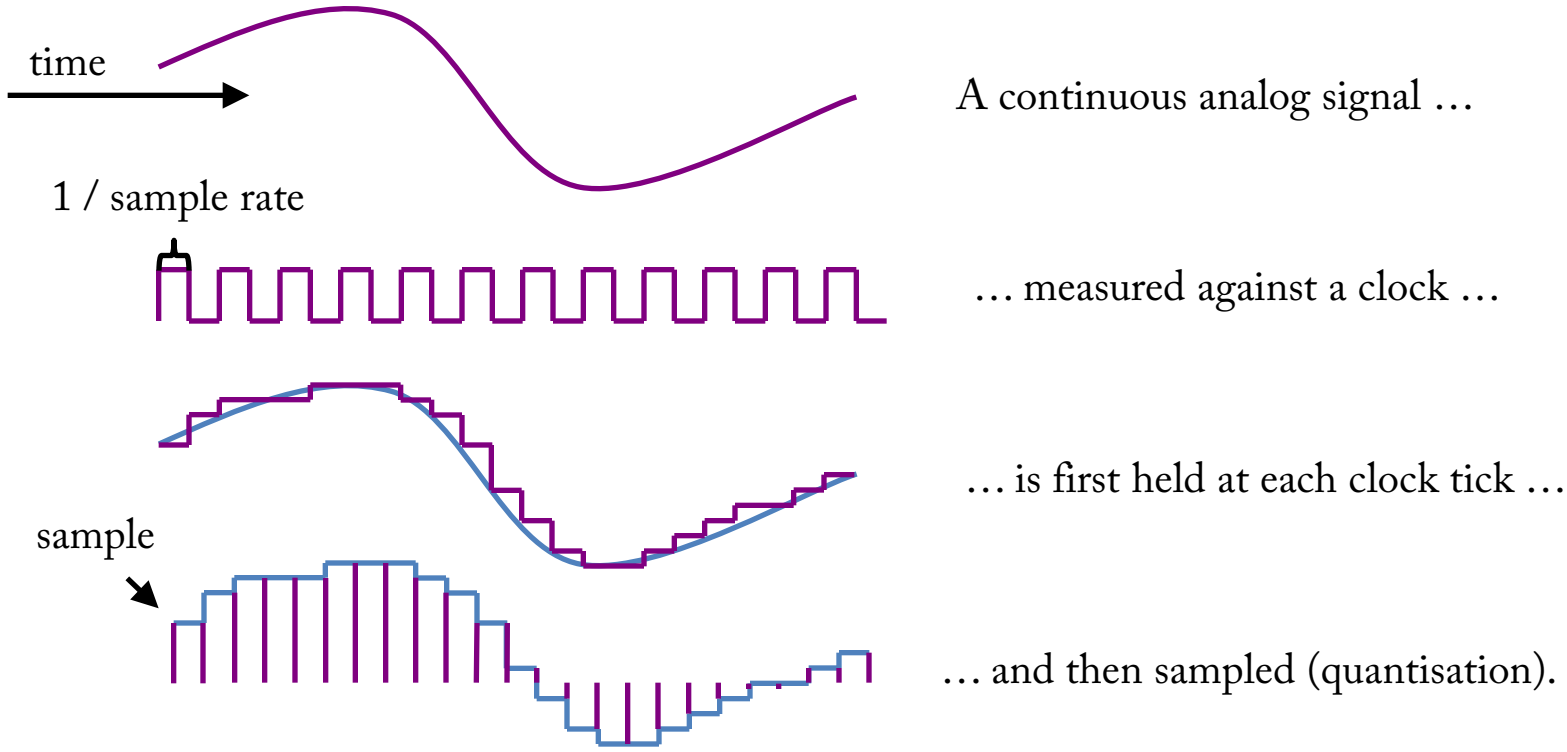
```
ssi_size_t sid = Factory::AddString ("a new string");
ssi_size_t sid = Factory::GetStringId ("a new string");
const ssi_char_t *str = Factory::GetString (sid);
```

Social Signal Interpretation

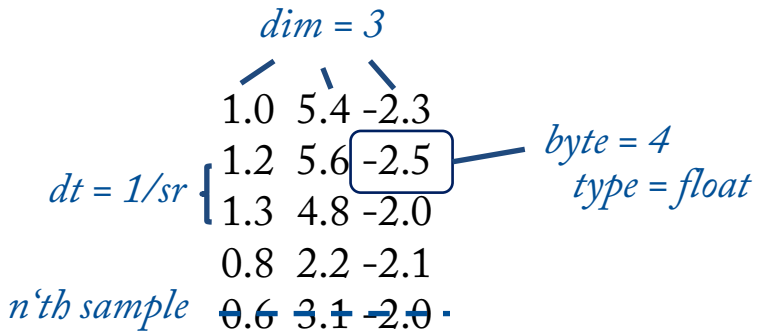
STREAMS

Digital Signals

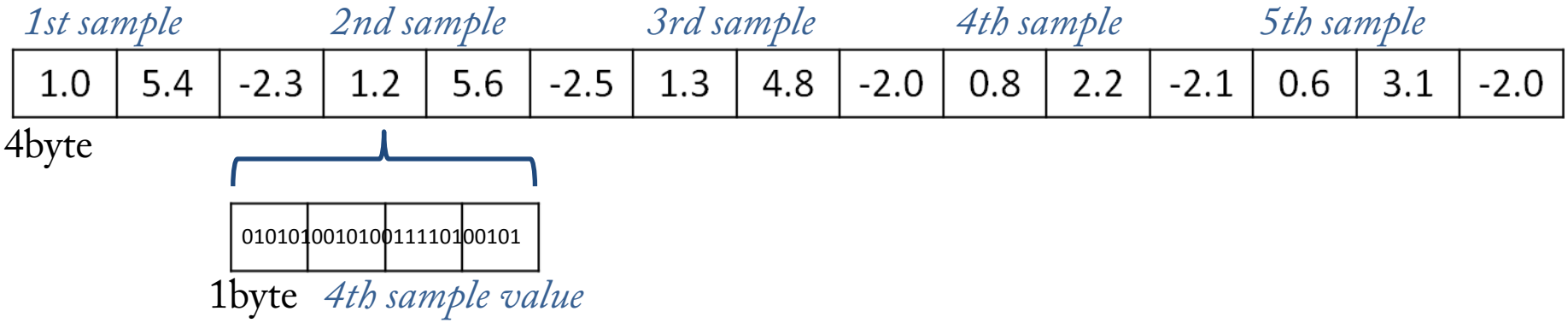
- Converting analog to digital signal



Stream



- Streams are characterized by:
  - sample rate in Hz (sr)
  - sample dimension (dim)
  - bytes per sample (byte)
  - sample type (type)
- Memory required for 1s data: (sr * dim * byte) bytes
  e.g. stereo audio in cd quality: (44100 * 2 * 2) bytes
- Samples are stored interleaved, i.e. values of first sample, followed by values of second sample, and so on:



Stream Struct

```
struct ssi_stream_t {
    ssi_size_t num; // number of used samples
    ssi_size_t num_real; // maximal number of samples
    ssi_size_t dim; // stream dimension
    ssi_size_t byte; // size in bytes of a single sample value
    ssi_size_t tot; // num * dim * byte
    ssi_size_t tot_real; // num_real * dim * byte
    ssi_byte_t *ptr; // pointer to the data
    ssi_time_t sr; // sample rate in Hz
    ssi_time_t time; // time stamp in seconds
    ssi_type_t type; // data type
};
```

Pre-defined sample types:

```
    SSI_UNDEF = 0, SSI_CHAR = 1, SSI_UCHAR = 2, SSI_SHORT = 3, SSI_USHORT = 4,
    SSI_INT = 5, SSI_UINT = 6, SSI_LONG = 7, SSI_ULONG = 8, SSI_FLOAT = 9,
    SSI_DOUBLE = 10, SSI_LDOUBLE = 11, SSI_STRUCT = 12, SSI_IMAGE = 13, SSI_BOOL = 14
```

Create Stream

```
ssi_stream_t s;

ssi_time_t len = 1.0;
ssi_time_t sr = 10.0;
ssi_size_t dim = 3;
ssi_size_t byte = sizeof (float);
ssi_type_t type = SSI_FLOAT;

ssi_size_t num = ssi_cast (ssi_size_t, len * sr);
ssi_stream_init (s, num, dim, byte, type, sr);

float *ptr = ssi_pcast (float, s.ptr);
for (ssi_size_t nsamp = 0; nsamp < s.num; nsamp++) {
    for (ssi_size_t ndim = 0; ndim < s.dim; ndim++) {
      *ptr++ = ssi_cast (float, ssi_random ());
    }
}

/// do something with s ///

ssi_stream_destroy (s);
```

In/Output Stream

```
// output to stdout
File *console = File::CreateAndOpen (File::ASCII, File::WRITE, 0);
console->writeLine ("writing on the console...");
console->setType (s.type);
console->write (s.ptr, s.dim, s.dim * s.num);

// write to and read from file
FileTools::WriteStreamFile (File::ASCII, "data", s);
FileTools::ReadStreamFile ("data", s);

// continuous output
FileStreamOut file_out;
file_out.open (s, "data", File::BINARY);
file_out.write (s, true);
file_out.write (s, true);
file_out.write (s, true);
file_out.close ();
```

Social Signal Interpretation

THREADING

Thread Class

The thread libary let you execute your code in separate threads and offers tools to synchronize them (Mutex, Event, Condition, …)

```
Thread (bool single_execution = false,
  ssi_size_t timeout_in_ms = 10000);

void start (); // called by user to start/stop thread
void stop ();  // in single execution stop is automatically called
void setName (const ssi_char_t *name); // set thread name
virtual void enter () {}; // called before thread is created
virtual void run () = 0;  // continuously called during execution
                          // called once in case of single execution
virtual void flush () {}; // called when thread has terminated

Mutex ();
void acquire (); // acquire mutex
void release (); // release mutex

Lock (Mutex &mutex); // acquires mutex in constructor
                     // and releases mutex in destructor
```

Thread Example

```
class MyThread : public Thread {
public:

  MyThread (const ssi_char_t *msg,
    ssi_size_t sleep_in_ms,
    bool single_execution);
  ~MyThread ();

  void run ();
  void enter ();
  void flush ();

protected:
  ssi_char_t *_msg;
  ssi_size_t _sleep_in_ms;
  static int _counter;
  static Mutex _mutex;
};

int MyThread::_counter = 0;
Mutex MyThread::_mutex;
```

```
MyThread::MyThread (ssi_char_t *msg,
  ssi_size_t sleep_in_ms,
  bool single_execution)
  : Thread (single_execution),
  _sleep_in_ms (sleep_in_ms) {
  _msg = ssi_strcpy (msg);
  setName (_msg);
}

MyThread::~MyThread () {
  delete[] _msg;
}

void MyThread::run () {
  sleep_ms (_sleep_in_ms);
  {
    Lock lock (_mutex);
    ssi_print ("%d: %s\n",
++_counter, _msg);
  }
}
```

Thread Example

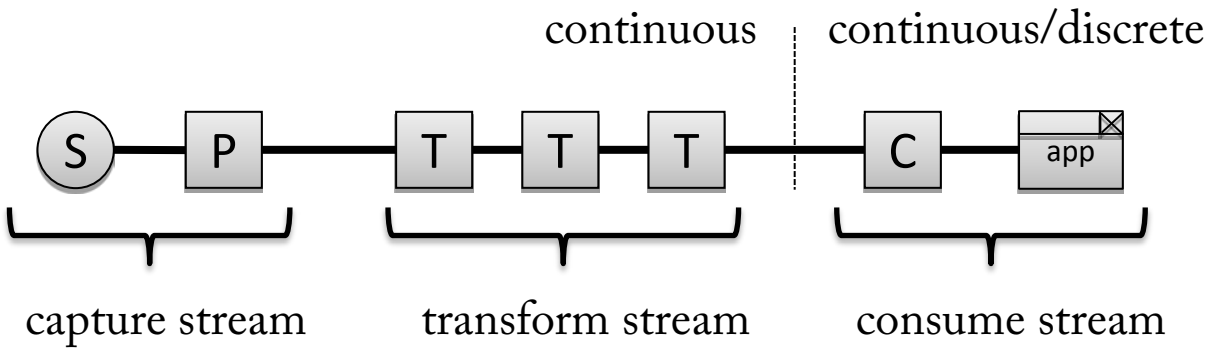
```
void main () {

  MyThread single_t ("single", 1000, true);
  MyThread multi_t_1 ("ping", 500, false);
  MyThread multi_t_2 ("pong", 300, false);

  single_t.start ();
  multi_t_1.start ();
  multi_t_2.start ();

  ssi_print ("\nPress enter to stop!\n");
  getchar ();

  multi_t_1.stop ();
  multi_t_2.stop ();

}
```

```
> 1: pong
> 2: ping
> 3: pong
> 4: pong
> 5: single
> flush single
> 6: ping
> 7: pong
> 8: ping
> 9: pong
> 10: pong
> 11: ping
> 12: pong
```

Social Signal Interpretation

PIPELINES

Processing pipeline



S — Sensor, captures sensor stream

P — Provider, feeds stream into pipeline

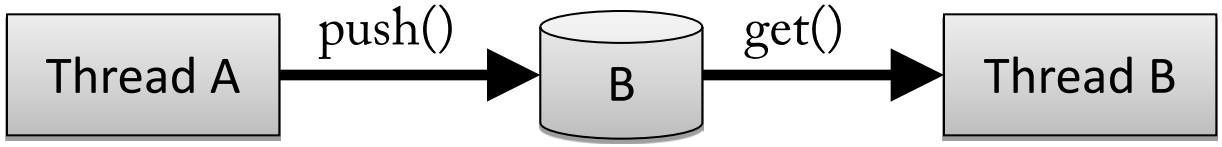
T — Transformer, applies transformation to stream

C — Consumer, fetches stream from pipeline

app — Application, responds to stream

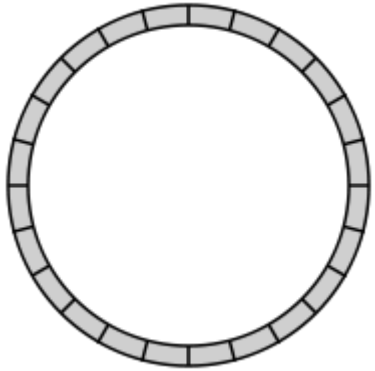
Buffering

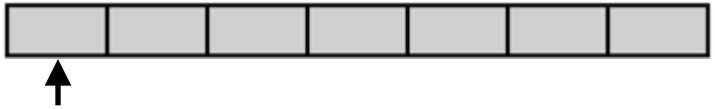
- In some situations it becomes necessary to buffer sensor data before using it, e.g. to make past data blocks available to an application or to share the same data between several applications
- Solution: allocate a region of memory to temporarily hold data while it is being moved from one place to another
- Problem: if several threads share same buffer we need to synchronize access (e.g. in a consumer-producer situation)



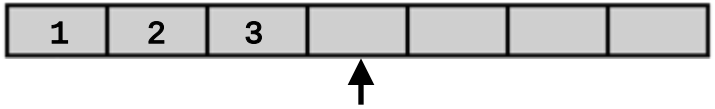
Ring Buffer

- A ring buffer is a data structure that uses a single, fixed-size buffer as if it were connected end-to-end
- Advantage: elements need not be shuffled around when a portion of the buffer is used
- A circular buffer first starts empty pointing to the first element:

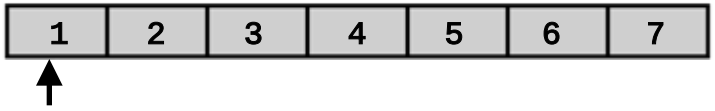


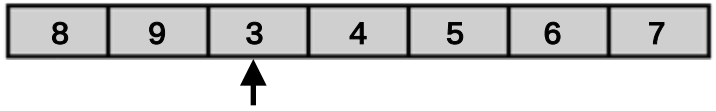


- New elements are appended at the position of the pointer and the pointer is moved accordingly:



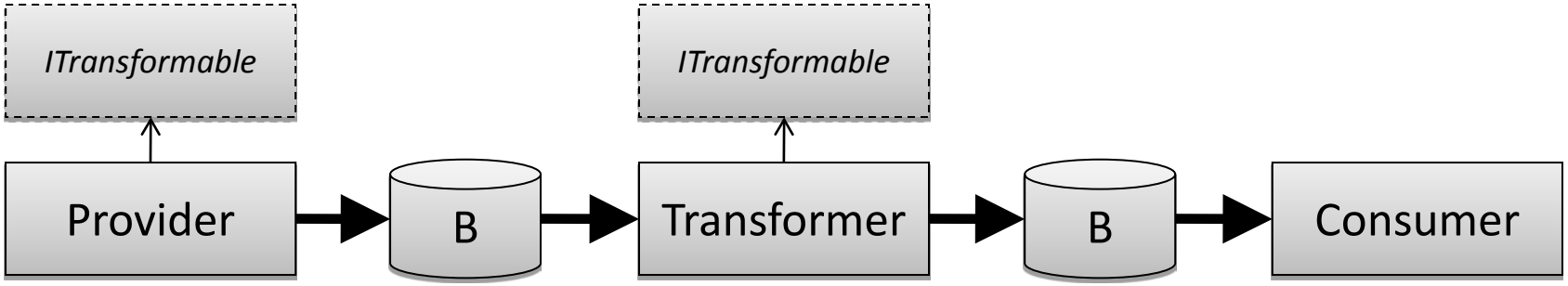
- Once the end is reached the pointer is again moved to the first position and old elements are overwritten:





TheFramework Class

- Manages buffer and solves thread access
- Provider: puts data it into one buffer
- Transformer: reads data from one (or more) buffer, manipulates it and writes result back to one buffer
- Consumer: fetches data from one (or more) buffer



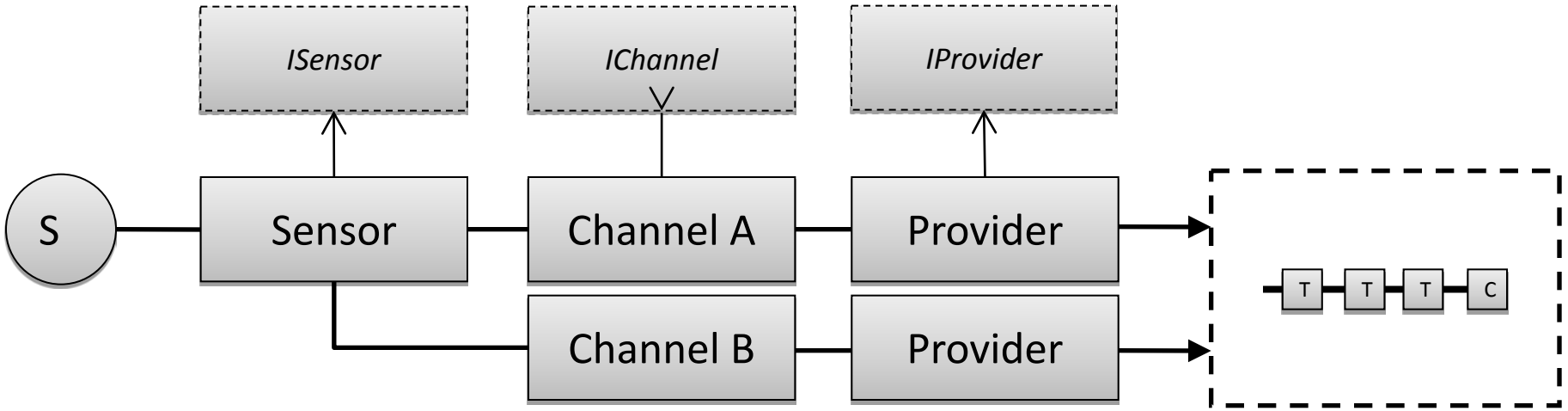
Run Pipeline

```
// get instance
ITheFramework *frame = Factory::GetFramwork ();

// add components
ITransformable *p = frame->AddProvider (...);
ITransformable *t = frame->AddTransformer (p, ...);
frame->AddConsumer (p, ...);
frame->AddConsumer (t, ...);

// run pipeline
frame->Start ();
getchar ();
frame->Stop ();
// clean up
frame->Clear ();
```

Social Signal Interpretation

SENSOR

Sensor




```
MySensor *sensor = ssi_create (MySensor, "sensor", true);
sensor->getOptions ()-> ...
ITransformable *sensor_p = frame->AddProvider (sensor, NAME);
frame->AddSensor (sensor);
```

Interfaces

```
class IRunnable {
    virtual ~IRunnable () {};
    virtual bool start () = 0;
    virtual bool stop () = 0;
};
class ISensor : public IObject, public IRunnable {
  virtual ssi_size_t getChannelSize () = 0;
  virtual IChannel *getChannel (ssi_size_t index) = 0;
  virtual bool setProvider (const ssi_char_t *name, IProvider *provider)= 0;
  virtual bool connect () = 0;
  virtual bool disconnect () = 0;
};
class IChannel {
  virtual const ssi_char_t *getName () = 0;
  virtual const ssi_char_t *getInfo () = 0;
  virtual ssi_stream_t getStream () = 0;
};
class IProvider : public IObject, public IComponent {
  virtual void init (IChannel *channel) = 0;
  virtual void provide (ssi_byte_t *data, ssi_size_t sample_number) = 0;
};
```

Sensor Example

```
#define MYSENSOR_PROVIDER_NAME "cursor"
#define MYSENSOR_SAMPLE_TYPE ssi_real_t

class MySensor : public ISensor, public Thread {
public:

  class MyChannel : public IChannel {
    friend class MySensor;
    public:
      MyChannel () {
      ssi_stream_init (stream,0,2,sizeof(MYSENSOR_SAMPLE_TYPE),SSI_REAL,0);
      }
      ~MyChannel () {
        ssi_stream_destroy (stream);
      }
      const ssi_char_t *getName () { return MYSENSOR_PROVIDER_NAME; };
      const ssi_char_t *getInfo () { return "mouse cursor"; };
      ssi_stream_t getStream () { return stream; };
    protected:
      ssi_stream_t stream;
  };
...
```

Sensor Example

```
...
public:

  class Options : public OptionList {
  public:
    Options ()
    : sr (25.0) {
      addOption ("sr", &sr, 1, SSI_DOUBLE, "sample rate in Hz");
    };
    ssi_time_t sr;
  };

  static const ssi_char_t *GetCreateName () { return "mysensor"; };
  static IObject *Create (const ssi_char_t *file) {
    return new MySensor (file); };
  ~MySensor ();

  Options *getOptions () { return &_options; };
  const ssi_char_t *getName () { return GetCreateName (); };
  const ssi_char_t *getInfo () { return "captures mouse"; };
...
```

Sensor Example

```
...
  ssi_size_t getChannelSize () { return 1; };
  IChannel *getChannel (ssi_size_t index) { return &_channel; };
  bool setProvider (const ssi_char_t *name, IProvider *provider);

  bool connect ();
  bool start () { return Thread::start (); };
  bool stop () { return Thread::stop (); };
  void run ();
  bool disconnect ();

protected:
  MySensor (const ssi_char_t *file = 0);
  Options _options;
  MyChannel _channel;
  IProvider *_provider;
  float _max_x, _max_y;
  Timer *_timer;
};
...
```

Sensor Example

```
...
bool MySensor::setProvider (const ssi_char_t *n,
  IProvider *p) {
  _provider = p;
  _channel.stream.sr = _options.sr;
  _provider->init (&_channel);
  return true;
}

bool MySensor::connect () {
  RECT rect;
  HWND desktop = ::GetDesktopWindow ();
  ::GetWindowRect (desktop, &rect);
  _max_x = ssi_cast (float, rect.right);
  _max_y = ssi_cast (float, rect.bottom);
  _timer = new Timer (1.0/_options.sr);
  return true;
}
...
```

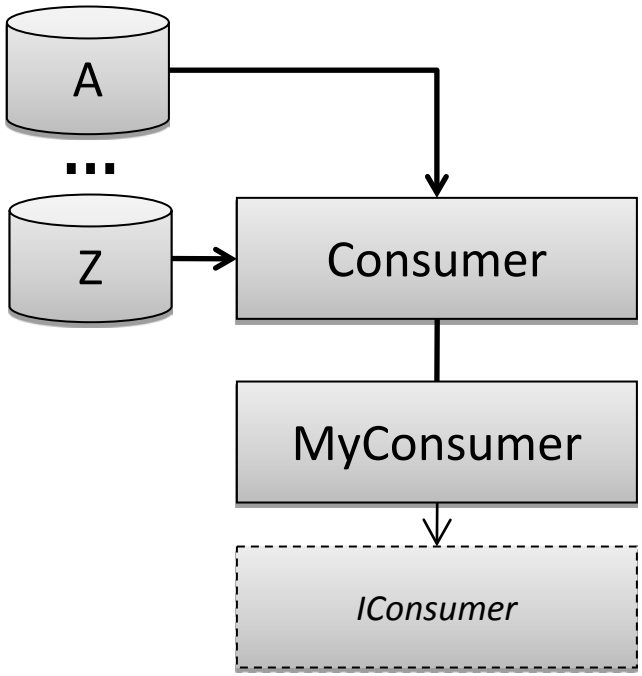
Sensor Example

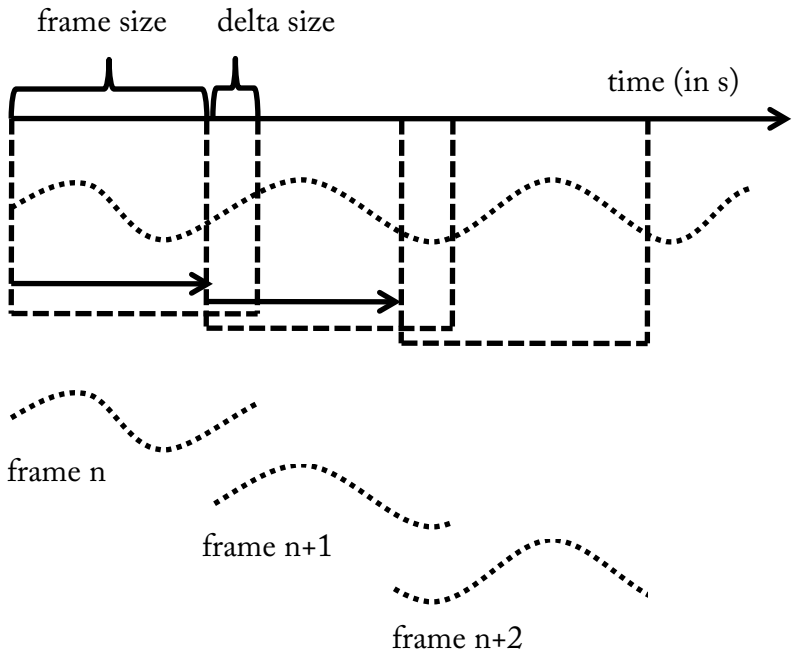
```
...
void MySensor::run () {

  POINT point;
  float cursor[2];

  ::GetCursorPos (&point);
  cursor[0] = point.x / _max_x;
  cursor[1] = point.y / _max_y;
  _provider->provide (ssi_pcast (ssi_byte_t, cursor), 1);

  _timer->wait ();
  return true;
}

bool MySensor::disconnect () {
  delete _timer; _timer = 0;
  return true;
}
```

Social Signal Interpretation

CONSUMER

Consumer






```
consume_enter ()
Loop:
    consume ()
consume_flush ()
```

IConsumer

```
class IConsumer : public IObject {

  enum STATUS {NO_TRIGGER = 0, COMPLETED, CONTINUED};
  struct info {ssi_time_t time; ssi_time_t dur; STATUS status;};

  void consume_enter (ssi_size_t stream_in_num,
    ssi_stream_t stream_in[])

  void consume (IConsumer::info consume_info,
    ssi_size_t stream_in_num,
    ssi_stream stream_in[]);

  void consume_flush (ssi_size_t stream_in_num,
    ssi_stream_t stream_in[]);

  ssi_object_t getType () { return SSI_CONSUMER; };
}
```

Consumer Example

```
class MyConsumer : public IConsumer {
public:
  static const ssi_char_t *GetCreateName () { return "myconsumer"; };
  static IObject *Create (const ssi_char_t *file) {
    return new MyConsumer (file);
  };
  ~MyConsumer ();

  IOptions *getOptions () { return 0; };
  const ssi_char_t *getName () { return "myconsumer"; };
  const ssi_char_t *getInfo () { return "outputs stream on console"; };
...
protected:

  MyConsumer (const ssi_char_t *file = 0);
  File *_file;
};
...
```

Consumer Example

```
...
void MyConsumer:: consume_enter (ssi_size_t stream_in_num,
  ssi_stream_t stream_in[]) {
  _file = File::Create (File::ASCII, File::WRITE, 0);
  _file->setType (stream_in[0].type);
}

void MyConsumer::consume (IConsumer::info consume_info,
  ssi_size_t stream_in_num,
  ssi_stream_t stream_in[]) {
  for (ssi_size_t i = 0; i < stream_in_num; i++) {
    _file->write (stream_in[i].ptr,
      stream_in[i].dim,
      stream_in[i].dim * stream_in[i].num);
  }
}

void MyConsumer::consume_flush (ssi_size_t stream_in_num,
  ssi_stream_t stream_in[]) {
  delete _ file; _ file = 0;
}
```

Pipeline Example

```
void ex_pipeline () {

  ITheFramework *frame = Factory::GetFramwork ();

  MySensor *sensor = ssi_factory_create (MySensor, 0, true);
  sensor->getOptions ().sr = 5.0;
  ITransformable *sensor_p =
              frame->AddProvider (sensor, MYSENSOR_PROVIDER_NAME);
  frame->AddSensor (*sensor);

  MyConsumer *writer =
  frame->AddConsumer (sensor_p, writer, 0.5s);

  frame->Start ();
  getchar ();
  frame->Stop ();
  frame->Clear ();

}
```

```
stream#0
  0.29   0.13
  0.30   0.07
  0.32   0.04
stream#0
  0.33   0.03
  0.33   0.03
  0.31   0.09
stream#0
  0.27   0.27
  0.27   0.28
  0.27   0.34
stream#0
  0.26   0.38
  0.21   0.43
  0.20   0.51
stream#0
  0.16   0.51
  0.17   0.48
  0.17   0.52
```

Pipeline Example

```
...

FileWriter *filewrite = ...
filewrite->getOptions ()->type = File::ASCII;
filewrite->getOptions ()->setPath ("cursor.txt");
frame->AddConsumer (sensor_p, filewrite, 0.5s);

SocketWriter *sockwrite = ...
sockwrite->getOptions ()->port = 1111;
sockwrite->getOptions ()->setHost ("localhost");
sockwrite->getOptions ()->type = Socket::UDP;
frame->AddConsumer (sensor_p, sockwrite, "0.5s");

SignalPainter *sigpaint = ...
sigpaint->getOptions ()->setName ("cursor");
sigpaint->getOptions ()->size = 10.0;
sigpaint->getOptions ()->setMove (0, 0, 300, 300);
frame->AddConsumer (sensor_p, sigpaint, "0.5s");

...
```

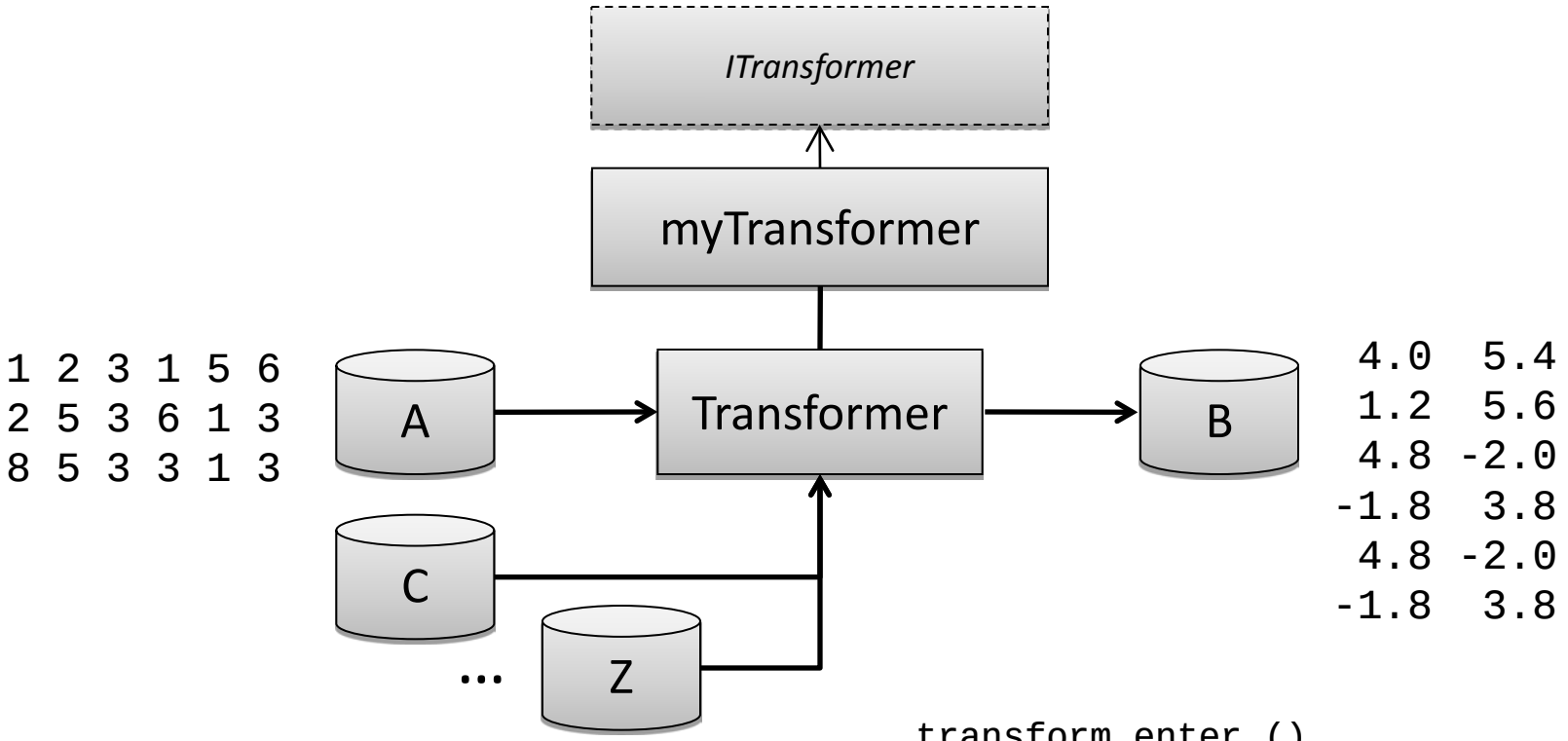
cursor.txt:

```
SSI@15.000000 2 4 9
0.0  225
0.244792  0.423333
0.236979  0.440000
0.238542  0.354167
0.234375  0.190000
0.147917  0.150000
```

Social Signal Interpretation

TRANSFORMER

Transformer




```
transform_enter ()
Loop:
    transform ()
transform_flush ()
```

ITransformer

```
class ITransformer : public IObject {
  struct info {
    ssi_time_t time;
    ssi_size_t frame_num;
    ssi_size_t delta_num;
  };
  ssi_size_t getSampleDimensionOut (ssi_size_t sample_dimension_in);
  ssi_size_t getSampleBytesOut (ssi_size_t sample_bytes_in);
  ssi_size_t getSampleNumberOut (ssi_size_t sample_number_in);
  ssi_type_t getSampleTypeOut (ssi_type_t sample_type_in);
  void transform_enter (ssi_stream_t &stream_in,
    ssi_stream_t &stream_out,
    ssi_size_t xtra_stream_in_num = 0,
    ssi_stream_t xtra_stream_in[] = 0);
  void transform (ITransformer::info info,
    ssi_stream_t &stream_in,
    ssi_stream_t &stream_out,
    ssi_size_t xtra_stream_in_num = 0,
    ssi_stream_t xtra_stream_in[] = 0);
  void transform_flush (...);
  ssi_object_t getType () { return SSI_TRANSFORMER; };
}
```

Example: Transformer

- Removes every second sample from the input stream

```
class MyTransformer : public ITransformer {
public:
...
  ssi_size_t getSampleDimensionOut (ssi_size_t sample_dimension_in) {
    return sample_dimension_in;
  }
  ssi_size_t getSampleBytesOut (ssi_size_t sample_bytes_in) {
    return sample_bytes_in;
  }
  ssi_size_t getSampleNumberOut (ssi_size_t sample_number_in) {
    return (sample_number_in + 1) / 2;
  }
  ssi_type_t getSampleTypeOut (ssi_type_t sample_type_in) {
    return sample_type_in;
  }
...
```

Example: Transformer

```
...
void MyTransformer::transform (ITransformer::info info,
  ssi_stream_t &stream_in,
  ssi_stream_t &stream_out,
  ssi_size_t xtra_stream_in_num,
  ssi_stream_t xtra_stream_in[]) {

  ssi_byte_t *ptr_in = stream_in.ptr;
  ssi_byte_t *ptr_out = stream_out.ptr;
  ssi_size_t n_bytes = stream_in.byte * stream_in.dim;

  for (ssi_size_t i = 0; i < (stream_in.num + 1) / 2; i++) {
    memcpy (ptr_out, ptr_in, n_bytes);
    ptr_in += 2 * n_bytes;
    ptr_out += n_bytes;
  }
}
```

Example: Transformer

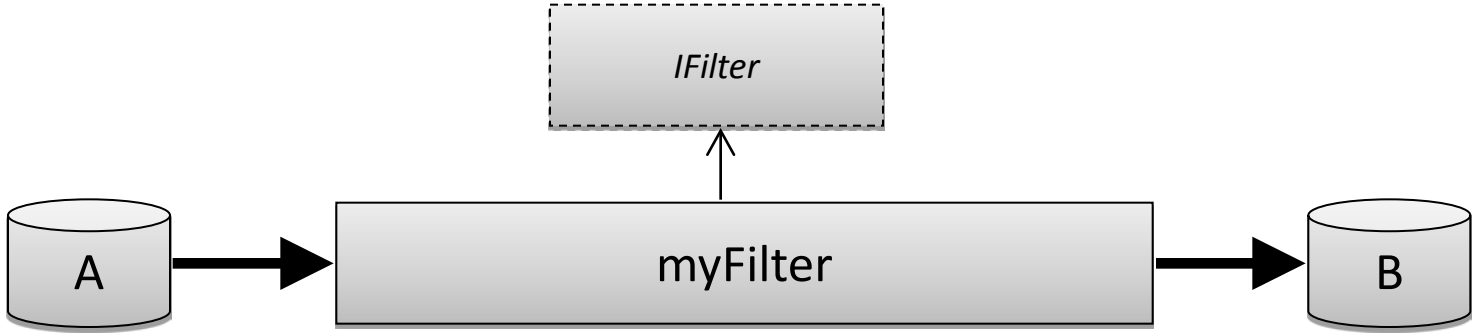
```
void ex_transf () {

  ...

  MyTransformer *transf = ...
  Transformer *transf_t = frame->AddTransformer (
                          sensor_p, transf, "0.5s");

  Itransformable *source[] = { sensor_p, transf_t };
  MyConsumer *consumer = ...
  frame->AddConsumer (2, source, consumer, "0.5s");

  ...

}
```

```
stream#0
  0.22   0.45
  0.21   0.44
  0.21   0.43
  0.21   0.43
  0.21   0.43
  0.21   0.43
  0.21   0.44
  0.21   0.44
  0.21   0.44
  0.20   0.44
stream#1
  0.22   0.45
  0.21   0.43
  0.21   0.43
  0.21   0.44
  0.20   0.44
```

Social Signal Interpretation

FILTER

Filter

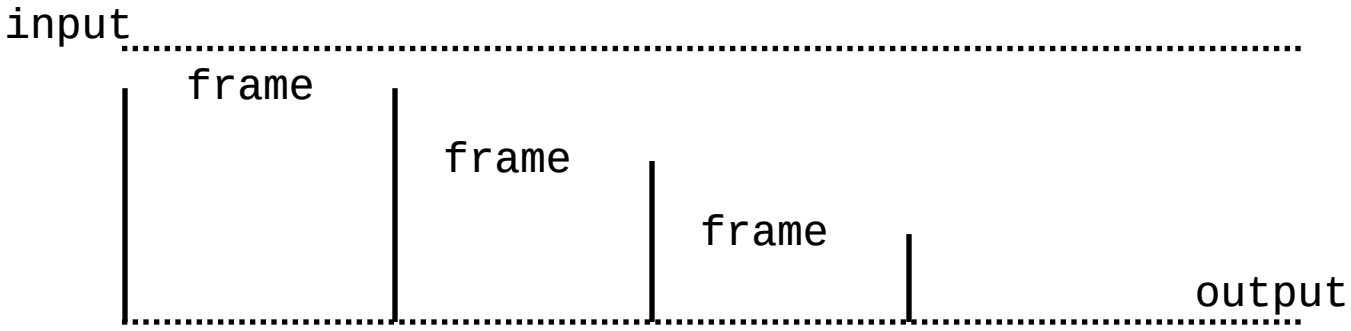



```
1 2 3 1
2 5 3 6
8 5 3 3
8 5 3 3
3 4 2 8
3 4 2 5
```

```
ssi_size_t getSampleNumberOut (...) {
      return sample_number_in;
}
```

#samples remains unchanged

```
 4.0  5.4
 1.2  5.6
 4.8 -2.0
-1.8  3.8
 4.8 -2.0
-1.8  3.8
```



Filter Example

- Swaps dimensions of input stream

```
class MyFilter : public IFilter {
public:
...
  ssi_size_t getSampleDimensionOut (ssi_size_t sample_dimension_in) {
    return sample_dimension_in;
  }
  ssi_size_t getSampleBytesOut (ssi_size_t sample_bytes_in) {
    return sample_bytes_in;
  }
  ssi_type_t getSampleTypeOut (ssi_type_t sample_type_in) {
    return sample_type_in;
  }
...
```

Filter Example

```
void MyFilter::transform (ITransformer::info info,
  ssi_stream_t &stream_in,
  ssi_stream_t &stream_out,
  ssi_size_t xtra_stream_in_num,
  ssi_stream_t xtra_stream_in[]) {

  ssi_byte_t *ptr_in = stream_in.ptr;
  ssi_byte_t *ptr_out = stream_out.ptr;
  ssi_size_t n_bytes = stream_in.byte * stream_in.dim;

  for (ssi_size_t i = 0; i < stream_in.num; i++) {
    for (ssi_size_t j = 0; j < stream_in.dim; j++) {
       memcpy (ptr_out + (stream_in.dim - j - 1) * stream_in.byte,
       ptr_in + j * stream_in.byte,
       stream_in.byte);
    }
    ptr_in += n_bytes;
    ptr_out += n_bytes;
  }
}
```

Filter Example

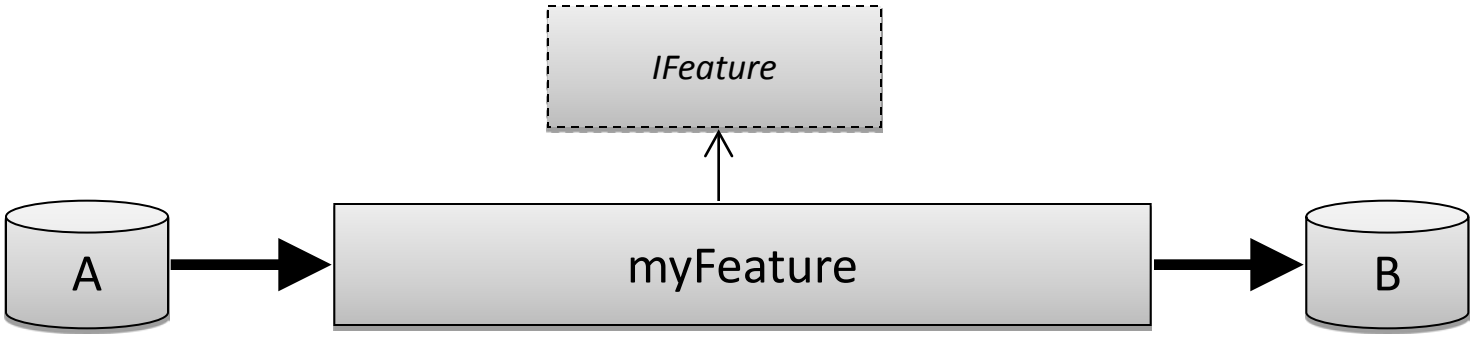
```
void ex_filter () {

  ...

  MyFilter *filter = ...
  ITransformable *filter_t = frame->AddTransformer (
                                    sensor_p, filter, "0.5s");
  ITransformable *source[] = { sensor_p, filter_t };
  MyConsumer *consumer = ...
  frame->AddConsumer (2, source, consumer, "0.5s");

  ...
}
```

```
stream#0
  0.35   0.59
  0.36   0.55
  0.36   0.52
  0.34   0.49
  0.32   0.46
stream#1
  0.59   0.35
  0.55   0.36
  0.52   0.36
  0.49   0.34
  0.46   0.32
```

Social Signal Interpretation

FEATURE

Feature

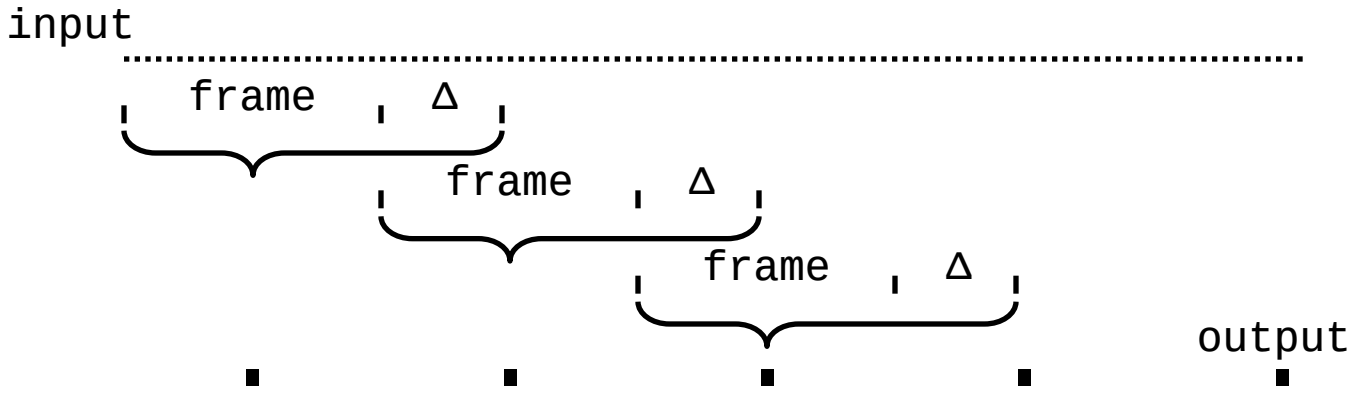



```
1 2 3 1 5 6
2 5 3 6 1 3
8 5 3 3 1 3
```

```
ssi_size_t getSampleNumberOut (...) {
        return 1;
}
```

```
1.0 2.4 -3.1
```

#samples are reduced to 1



Feature Example

- Calculates for each dimension mean value of input stream

```
class MyFeature : public IFeature {
public:
...
  ssi_size_t getSampleDimensionOut (ssi_size_t sample_dimension_in) {
    return sample_dimension_in;
  }
  ssi_size_t getSampleBytesOut (ssi_size_t sample_bytes_in) {
    return sample_bytes_in;
  }
  ssi_type_t getSampleTypeOut (ssi_type_t sample_type_in) {
    if (sample_type_in != SSI_REAL) {
       ssi_err ("type '%s", SSI_TYPE_NAMES[sample_type_in]);
    }
    return SSI_REAL;
  }
...
```

Feature Example

```
void MyFilter::transform (ITransformer::info info,
  ssi_stream_t &stream_in,
  ssi_stream_t &stream_out,
  ssi_size_t xtra_stream_in_num,
  ssi_stream_t xtra_stream_in[]) {

  ssi_real_t *ptr_in = ssi_pcast (ssi_real_t, stream_in.ptr);
  ssi_real_t *ptr_out = ssi_pcast (ssi_real_t, stream_out.ptr);

  for (ssi_size_t i = 0; i < stream_in.dim; i++) {
    ptr_out[i] = 0;
  }
  for (ssi_size_t i = 0; i < stream_in.num; i++) {
    for (ssi_size_t j = 0; j < stream_in.dim; j++) {
      ptr_out[j] += *ptr_in++;
    }
  }
  for (ssi_size_t i = 0; i < stream_in.dim; i++) {
    ptr_out[i] /= stream_in.num;
  }
}
```

Feature Example

- Calculates for each dimension min/max value of input stream

```
class MyFeature2 : public IFeature {
public:
...
  ssi_size_t getSampleDimensionOut (ssi_size_t sample_dimension_in) {
    return sample_dimension_in * 2;
  }
  ssi_size_t getSampleBytesOut (ssi_size_t sample_bytes_in) {
    return sample_bytes_in;
  }
  ssi_type_t getSampleTypeOut (ssi_type_t sample_type_in) {
    if (sample_type_in != SSI_REAL) {
       ssi_err ("type '%s", SSI_TYPE_NAMES[sample_type_in]);
    }
    return SSI_REAL;
  }
...
```

Feature Example

```
void MyFilter::transform (...) {

  ssi_real_t *ptr_in = ssi_pcast (ssi_real_t, stream_in.ptr);
  ssi_real_t *ptr_out = ssi_pcast (ssi_real_t, stream_out.ptr);
  ssi_real_t value = 0;

  for (ssi_size_t i = 0; i < stream_in.dim; i++) {
    value = *ptr_in++;
    ptr_out[i*2] = value;
    ptr_out[i*2+1] = value;
  }
  for (ssi_size_t i = 1; i < stream_in.num; i++) {
    for (ssi_size_t j = 0; j < stream_in.dim; j++) {
      value = *ptr_in++;
      if (value < ptr_out[j*2]) {
        ptr_out[j*2] = value;
      } else if (value > ptr_out[j*2+1]) {
        ptr_out[j*2+1] = value;
      }
    }
  }
}
```

Filter Example

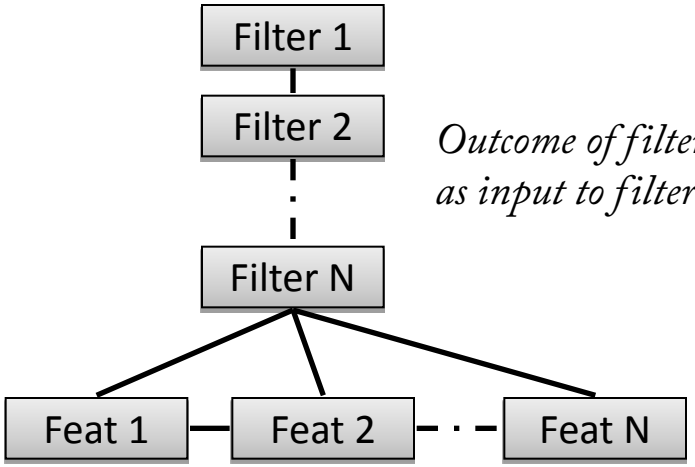
```
void ex_feature () {

  ...

  MyFeature *feature = ...
  ITransformable *feature_t = frame->AddTransformer (
                              sensor_p, feature, "0.5s");

  MyFeature2 *feature2 = ...
  ITransformable *feature2_t = frame->AddTransformer (
                              sensor_p, feature2, "0.5s");

  ITransformable *source[] = { sensor_p, feature_t, feature2_t };
  MyConsumer *consumer = ...
  frame->AddConsumer (3, ids, consumer, "0.5s");

  ...

}
```

```
stream#0
  0.10   0.27
  0.08   0.28
  0.07   0.28
  0.07   0.29
  0.07   0.32
stream#1
  0.07   0.28
stream#2
  0.07   0.10   0.27   0.32
```

Social Signal Interpretation

CHAIN

Feature



Outcome of filter n is used as input to filter n+1

output of feature n+1 is concatenated with output of feature n

```
class Chain : public ITransformer {

  void chain->set (ssi_size_t filter_number,
     SSI_Filter **filter,
     ssi_size_t feature_number,
     SSI_Feature **feature);
};
```

Feature Example

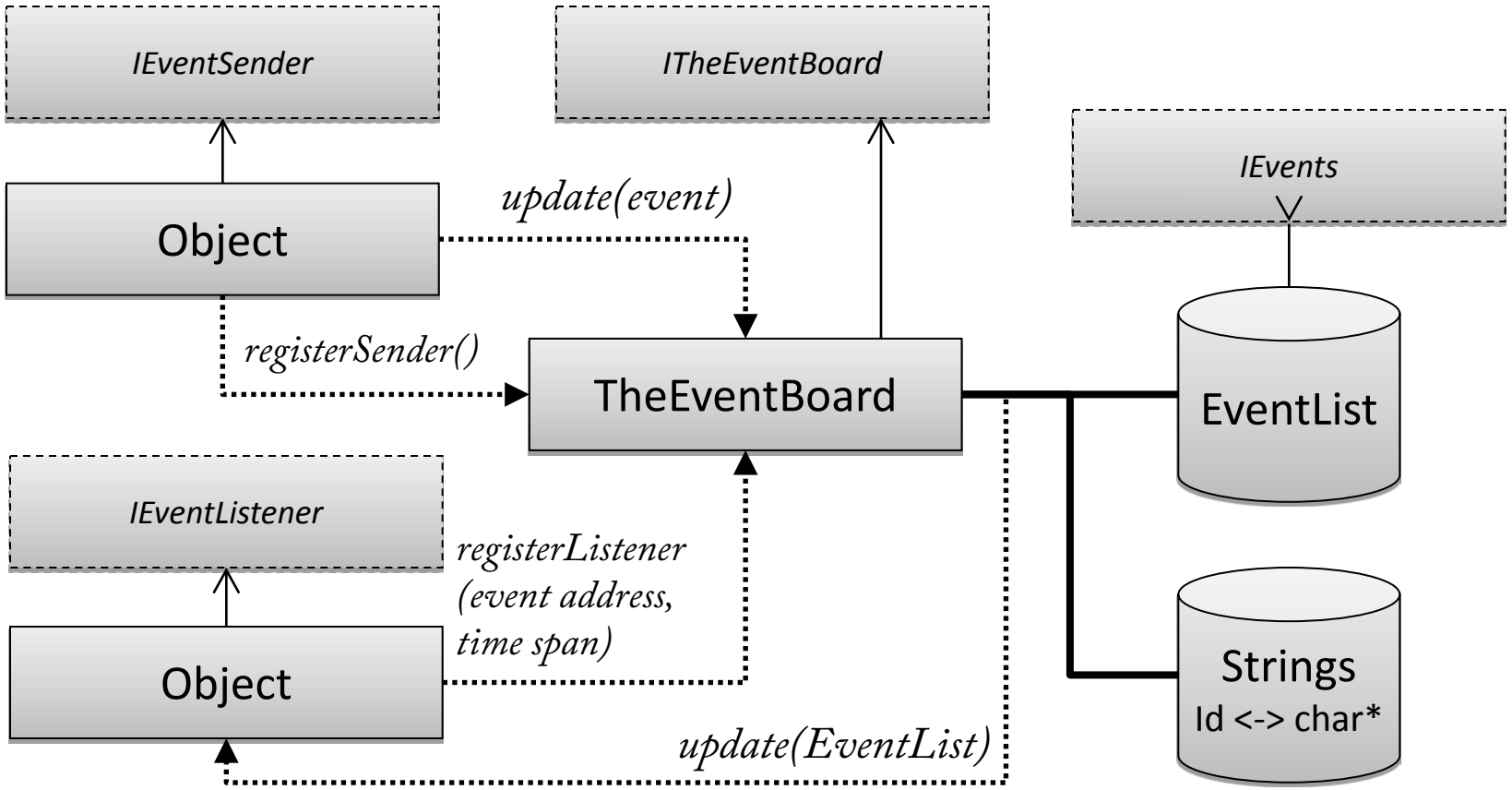
```
stream#0
  0.10   0.27
  0.08   0.28
  0.07   0.28
  0.07   0.29
  0.07   0.32
stream#0
  0.07  0.07  0.10  0.28  0.27  0.32
```

```
void ex_chain () {
...

  MyFilter *filter = ...
  MyFeature *feature = ...
  MyFeature2 *feature2 = ...

  IFilter *filters[1] = { filter };
  IFeature *features[2] = { feature, feature2 };
  Chain *chain = ...
  chain->set (1, filters, 2, features);
  ITransformable *chain_t = frame->AddTransformer (sensor_p, chain, "0.5s");

  MyConsumer *printer = ...
  frame->AddConsumer (sensor_p, printer, "0.5s");

  MyConsumer *printer_t = ...
  frame->AddConsumer (chain_t, printer_t, "0.5s");

...

}
```

Social Signal Interpretation

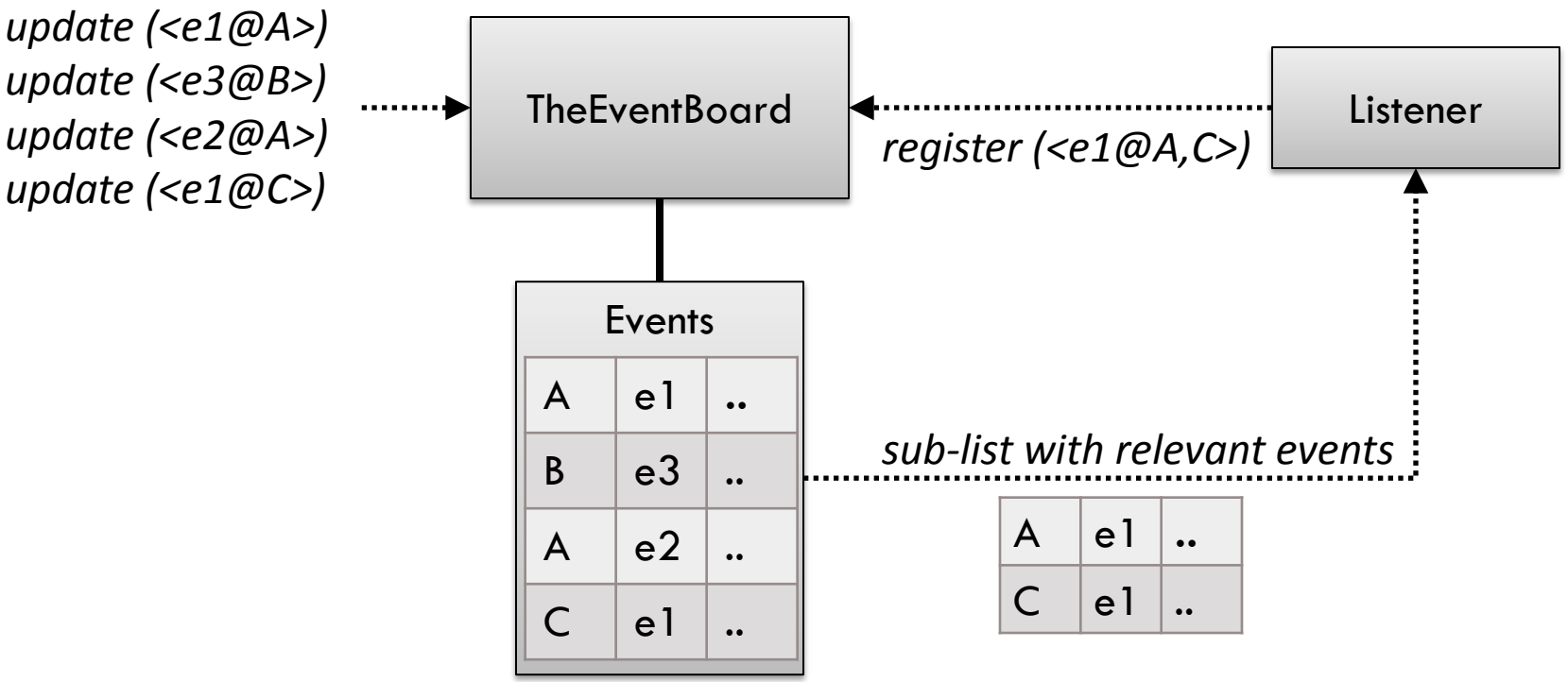
EVENTS

Events


IEventSender
IEventListener
Object
Object
update(event)
registerSender()
registerListener
(event address,
time span)
update(EventList)
ITheEventBoard
TheEventBoard
IEvents
EventList
Strings
Id <-> char*

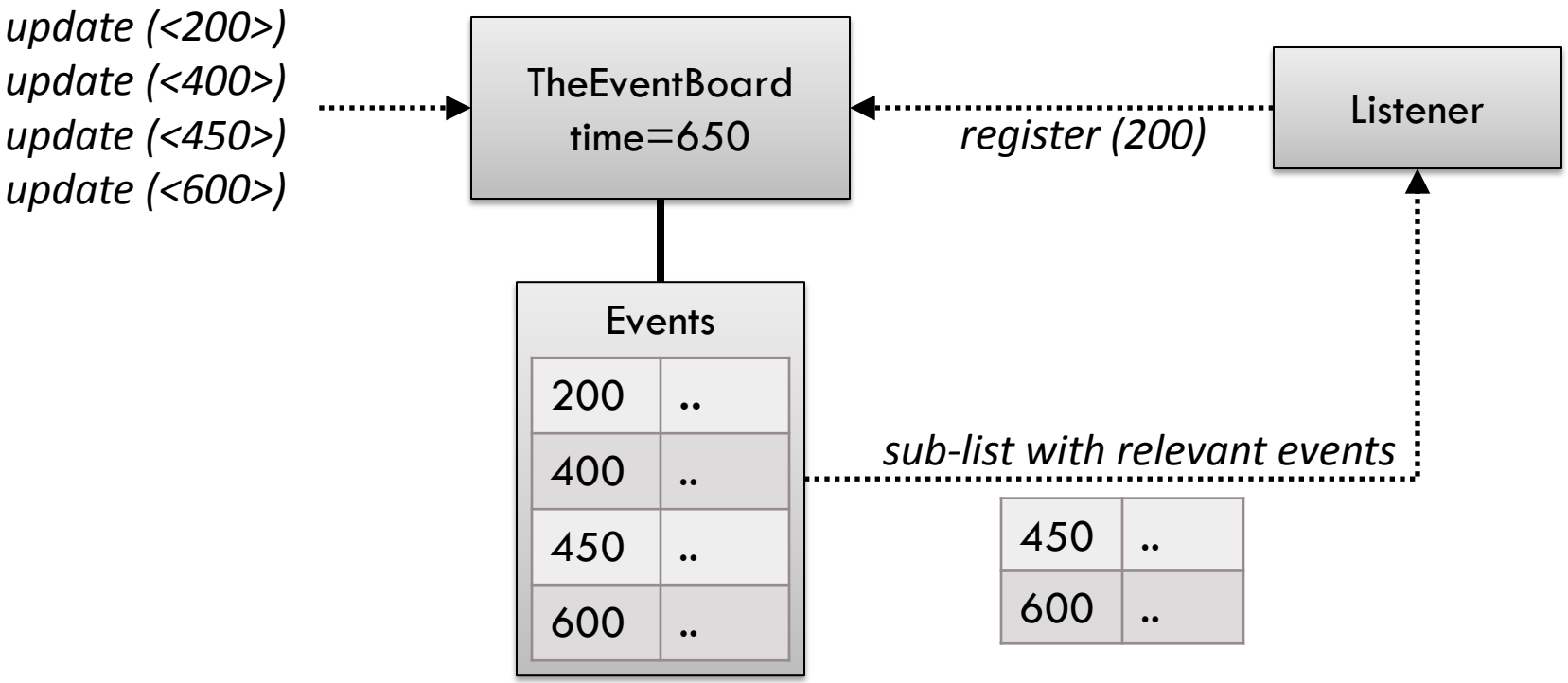
Event Address

- Listener receive events matching their event address mask
- An event address mask is made of N event and M sender names <e1,…,eN@s1,…sM> (<@> receives all!)



Time Span

- Listener receives relevant events in the last N milliseconds (0 to receive all)



Interfaces

```
class IEventListener {
    virtual void listen_enter () {};
    virtual bool update (ssi_event_t &e) { return false; };
    virtual bool update (IEvents &events, ssi_size_t n_new_events,
                         ssi_size_t time_ms) { return false; };
    virtual void listen_flush () {};
};

class IEventSender {
   virtual void send_enter () {};
   virtual bool setEventListener (IEventListener *listener) { return false; };
   virtual void sender_flush () {};
   virtual const ssi_char_t *getEventAddress () { return 0; };
};
```

Interfaces

```
class IEvents {
  virtual void reset () = 0; // moves pointer to latest event
  virtual ssi_event_t *get (ssi_size_t index) = 0;
  virtual ssi_event_t *next () = 0; // returns latest event and moves pointer
  virtual ssi_size_t getSize () = 0;
};

class ITheEventBoard : public IObject, public IEventListener {
  virtual void Start () = 0;
  virtual void Stop () = 0;
  bool RegisterSender (IEventSender &sender) = 0;
  bool RegisterListener (IEventBoardListener &listener,
    const ssi_char_t *address = 0,
    ssi_size_t time_span_ms = 0) = 0;
}
```

Events

- An event…
  - represents a discrete period of time
  - has a name and a sender name
  - may carry meta data
- Data structure:

```
struct ssi_stream_t {
    ssi_size_t sender_id; // unique sender id
    ssi_size_t event_id; // unique event id
    ssi_size_t time; // start time in ms
    ssi_size_t dur; // duration in ms
    ssi_real_t prob; // probability [0..1] to express confidence
    ssi_etype_t type; // event data type
    ssi_size_t tot; // size in bytes
    ssi_size_t tot_real; // total available size in bytes
    ssi_byte_t *ptr; // pointer to event data
    ssi_estate_t state; // events status
};
```

Events

- Types:
  - SSI_ETYPE_EMPTY: empty meta data
  - SSI_ETYPE_STRING: a string value of variable length
  - SSI_ETYPE_FLOATS: a series of float values
  - SSI_ETYPE_NTUPLE: a series of string/value tuples

```
struct ssi_event_tuple_t {
  ssi_size_t id; // string id
  ssi_real_t value; // value
};
```

- States:

```
enum ssi_estate_t {
    SSI_ESTATE_COMPLETED, // event is complete
    SSI_ESTATE_CONTINUED  // incomplete, another event will follow
};
```

Sender Example

```
class MyEventSender : public IConsumer {
public:
...
  void consume_enter (ssi_size_t stream_in_num, ssi_stream_t stream_in[]);
  void consume (IConsumer::info consume_info, ssi_size_t stream_in_num,
           ssi_stream_t stream_in[]);
  void consume_flush (ssi_size_t stream_in_num, ssi_stream_t stream_in[]);

  bool setEventListener (IEventListener *listener);
  const ssi_char_t *getEventAddress () {
    return _event_address.getAddress ();
  }
protected:
...
  IEventListener *_elistener;
  ssi_event_t _event;
  EventAddress _event_address;
};
```

Sender Example

```
MyEventSender::MyEventSender (const ssi_char_t *file) {
  ssi_event_init (_event, SSI_ETYPE_FLOATS);
}

bool MyEventSender::setEventListener (IEventListener *listener) {

  _elistener = listener;
  _event.sender_id = Factory::AddString ("myevent");
  _event.event_id = Factory::AddString ("mysender");

  _event_address.setSender ("myevent");
  _event_address.setEvents ("mysender");

  return true;
}

void MyEventSender::consume_enter (ssi_size_t stream_in_num,
  ssi_stream_t stream_in[]) {
  ssi_event_adjust (_event, stream_in[0].dim * sizeof (ssi_real_t));
}
```

Sender Example

```
void MyEventSender::consume_enter (ssi_size_t stream_in_num,
  ssi_stream_t stream_in[]) {
  ssi_event_init (_event, SSI_ETYPE_FLOATS);
  ssi_event_adjust (_event, stream_in[0].dim * sizeof (ssi_real_t));
}
void MyEventSender::consume (IConsumer::info consume_info,
  ssi_size_t stream_in_num, ssi_stream_t stream_in[]) {
  ssi_real_t *in = ssi_pcast (ssi_real_t, stream_in[0].ptr);
  ssi_real_t *out = ssi_pcast (ssi_real_t, _event.ptr);
  ssi_mean (stream_in[0].num, stream_in[0].dim, in, out);
  _event.time = ssi_cast (ssi_size_t, consume_info.time * 1000);
  _event.dur = ssi_cast (ssi_size_t, consume_info.dur * 1000);
  _elistener->update (_event);
}
void MyEventSender::consume_flush (ssi_size_t stream_in_num,
  ssi_stream_t stream_in[]) {
  ssi_event_destroy (_event);
}
```

Listener Example

```
class MyEventListener : public IEventBoardListener {
public:
...
  void listen_enter (ITheEventBoard &board);
  void update (IEvents &events,
               ssi_size_t n_new_events,
               ssi_size_t time_ms);
  void listen_flush (ITheEventBoard &board);
...
};
```

Listener Example

```
void MyEventListener::update (...) {

  EventAddress ea;
  ssi_event_t *e = 0;
  for (ssi_size_t i = 0; i < n_new_events; i++) {
    e = events.next ();
    ea.clear ();
    ea.setSender (Factory::GetString (e->sender_id));
    ea.setEvents (Factory::GetString (e->event_id));
    ssi_print ("received event %s of type %s at %ums for %ums\n",
                ea.getAddress (), SSI_ETYPE_NAMES[e->type], e->time, e->dur);
    if (e->type == SSI_ETYPE_FLOATS) {
      ssi_real_t *ptr = ssi_pcast (ssi_real_t, e->ptr);
      ssi_size_t n = e->tot / sizeof (ssi_real_t);
      for (ssi_size_t j = 0; j < n; j++) {
        ssi_print ("%.2f ", *ptr++);
      }
      ssi_print ("\n");
    }
  }
}
```

Events Example

```
void ex_event () {
  ...
  ITheEventBoard *board = Factory::GetEventBoard ();

  MyEventSender *sender = ...
  frame->AddConsumer (sensor_p, sender, "2.5s");
  board->RegisterSender (*sender);

  MyEventListener *listener = ...
  board->RegisterListener (*listener,
    sender->getEventAddress ());

  frame->Start ();
  board->Start ();
  getchar ();
  board->Stop ();
  frame->Stop ();
  board->Clear ();
  frame->Clear ();
};
```

```
received event mysender@myevent
  of type FLOATS at 0ms for
  2500ms
0.35 0.30
received event mysender@myevent
  of type FLOATS at 2500ms for
  2500ms
0.08 0.40
received event mysender@myevent
  of type FLOATS at 5000ms for
  2500ms
0.01 0.47
received event mysender@myevent
  of type FLOATS at 7500ms for
  2500ms
0.02 0.38
received event mysender@myevent
  of type FLOATS at 10000ms for
  2500ms
0.06 0.37
```

Social Signal Interpretation

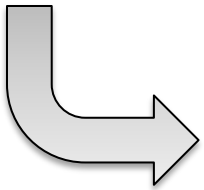
XML PIPELINES

XML Pipelines

- SSI allows the definition of pipelines in XML language instead of code
- Advantages:
  - Microsoft Visual Studio not required,
  - no C++ knowledge
  - no re-compilation of pipelines if a component changes
- Writing of XML pipelines is supported by a graphical editor (xmledit.exe) with object browser, syntax highlighting, error checking, option settings per dialogue and immediate execution of the pipeline
- The interface of the XML editor is covered in a separate tutorial (see xml.pdf)

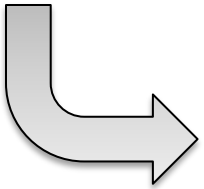
Specification

```
Factory::RegisterDLL ("ssigraphic.dll");
Factory::RegisterDLL ("ssisignal.dll");
```



```
<register>
  <load name="ssigraphic.dll"/>
  <load name="ssisignal.dll"/>
</register>
```

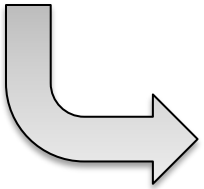
```
Mouse *mouse = ssi_create (Mouse, "mouse", true);
mouse->getOptions ()->mask = Mouse::RIGHT;
ITransformable *button = frame->AddProvider (mouse, "button");
ITransformable *cursor = mouse->AddProvider (mouse, "cursor");
frame->AddSensor (mouse);
```



```
<sensor create="ssi_sensor_Mouse" mask="2" option="mouse">
  <provider channel="button" pin="button"/>
  <provider channel="cursor" pin="cursor">
</sensor>
```

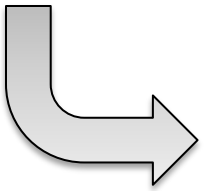
Specification

```
Derivative *derivative = ssi_create (Derivative, 0, true);
Transformer *derivative_t = frame->AddTransformer (cursor_p, derivative, "0.2s");
```



```
<transformer create="ssi_filter_Derivative">
  <input pin="cursor" frame="0.2s"/>
  <output pin="derivative"/>
</transformer>
```

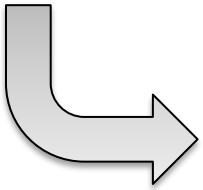
```
SignalPainter *sigpaint = ssi_create (SignalPainter, 0, true);
frame->AddConsumer (derivative_t, sigpaint, "0.2s");
```



```
<consumer create="ssi_consumer_SignalPainter">
  <input pin="derivative" frame="0.2s"/>
</consumer>
```

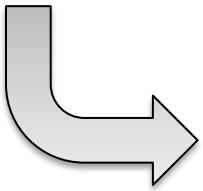
Specification

```
ZeroEventSender *ezero = ssi_create (ZeroEventSender, "ezero", true);
ezero->getOptions ()->setSender ("zsender");
ezero->getOptions ()->setEvent ("zevent");
frame->AddConsumer (button_p, zerotr, "0.2s");
board->RegisterSender (*ezero);
```



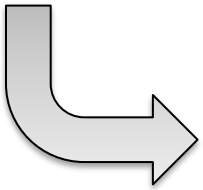
```
<consumer create="ssi_consumer_ZeroEventSender" option="ezero"
          sname="zsender" ename="zevent" >
  <input pin="button" frame="0.25s"/>
</consumer>
```

```
sigpaint = ssi_create (SignalPainter, 0, true);
frame->AddEventConsumer(cursor_p, sigpaint, board, ezero->getEventAddress(), derivative)
```



```
<consumer create="ssi_consumer_SignalPainter">
  <input pin="cursor" listen="zevent@zsender">
    <transformer create="ssi_filter_Derivative>
  </input>
</consumer>
```

```
EventMonitor *monitor = ssi_create (EventMonitor, 0, true);
board->RegisterListener (ezero->getEventAddress ());
```



```
<object create="ssi_listener_EventMonitor">
  <listen address="zevent@zsender"/>
</object>
```

Social Signal Interpretation

DLL EXPORT

DLL Export

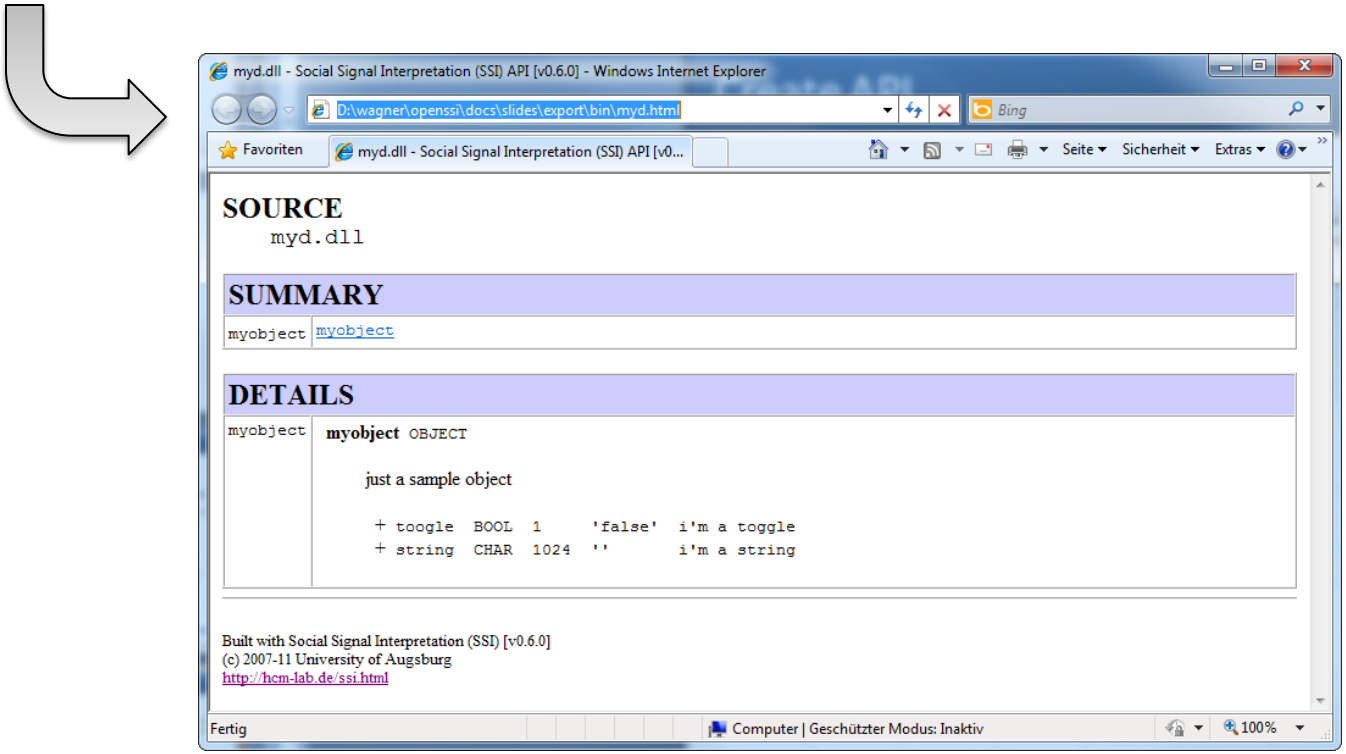
- Objects can be exported to a DLL and dynamically loaded at runtime through the Factory:

```
#include "MyObject.h"
#include "base/Factory.h"

#ifndef DLLEXP
#define DLLEXP extern "C" __declspec( dllexport )
#endif

DLLEXP bool Register (ssi::Factory *factory, FILE *logfile, ssi::IMessage *message) {

  ssi::Factory::SetFactory (factory);
  if (logfile) {
    ssiout = logfile;
  }
  if (message) {
    ssimsg = message;
  }

  return ssi::Factory::Register (MyObject::GetCreateName (), MyObject::Create);
}
```

API Generation

- API documentation is automatically extracted from a DLL using APIGenerator:

```
APIGenerator:: APIGenerator::CreateAPI ("my.dll");
```



Social Signal Interpretation

MACHINE LEARNING

Machine Learning

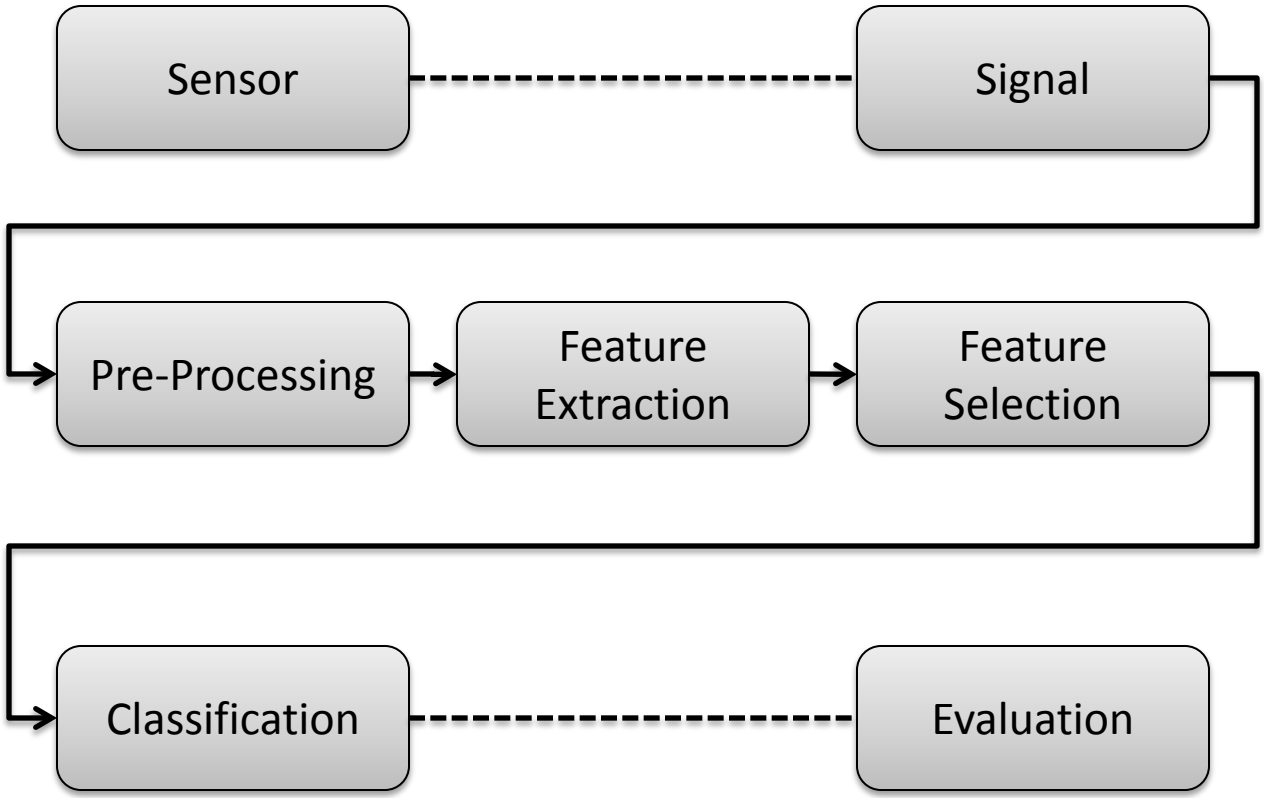
- Machine learning is concerned with the design and development of algorithms that allow computers to evolve behaviors based on empirical data, such as from sensor data or databases
- A learner can take advantage of examples (training data) to capture characteristics of interest of their unknown underlying probability distribution.
- A major focus of machine learning research is to automatically learn to recognize complex patterns and make intelligent decisions based on training data

Example

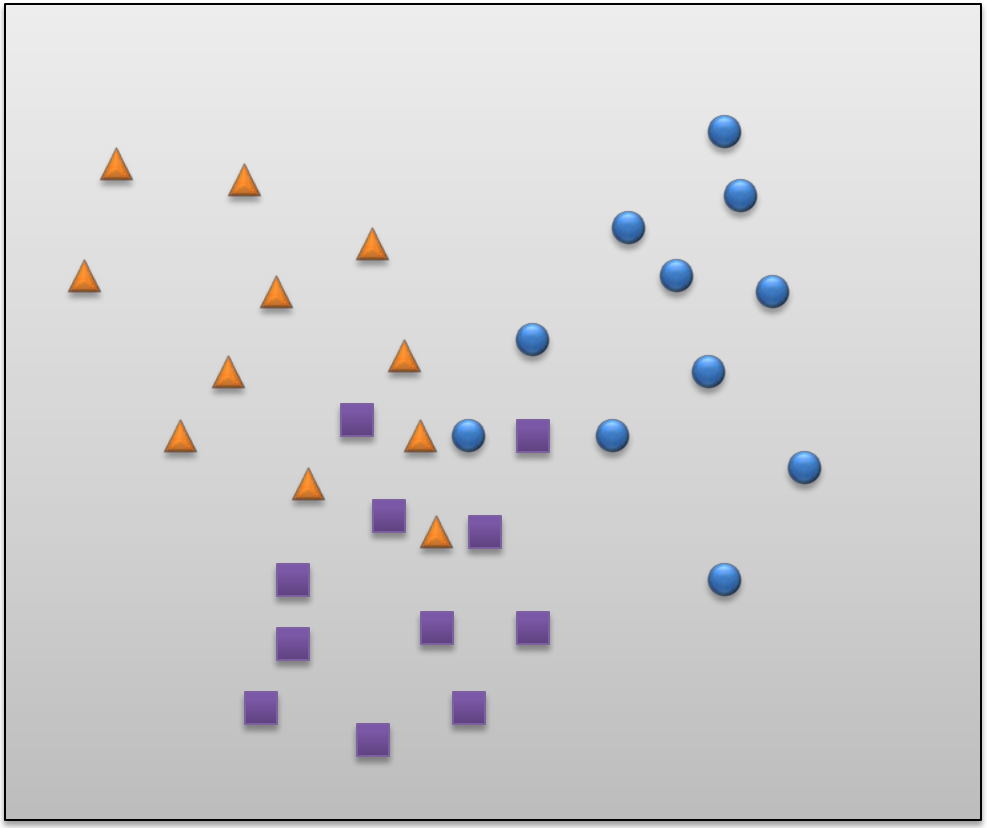
Sensor	2-D cursor stream captured from mouse sensor
Training Data	Recorded movements
Learner	Dollar$1 algorithm (finds best matching training example)


```
[trigger___] update (0.72@0.52)
[recog_c__1] recognized class
  circle
circle {0.84}
```

Classification Pipeline



Classification

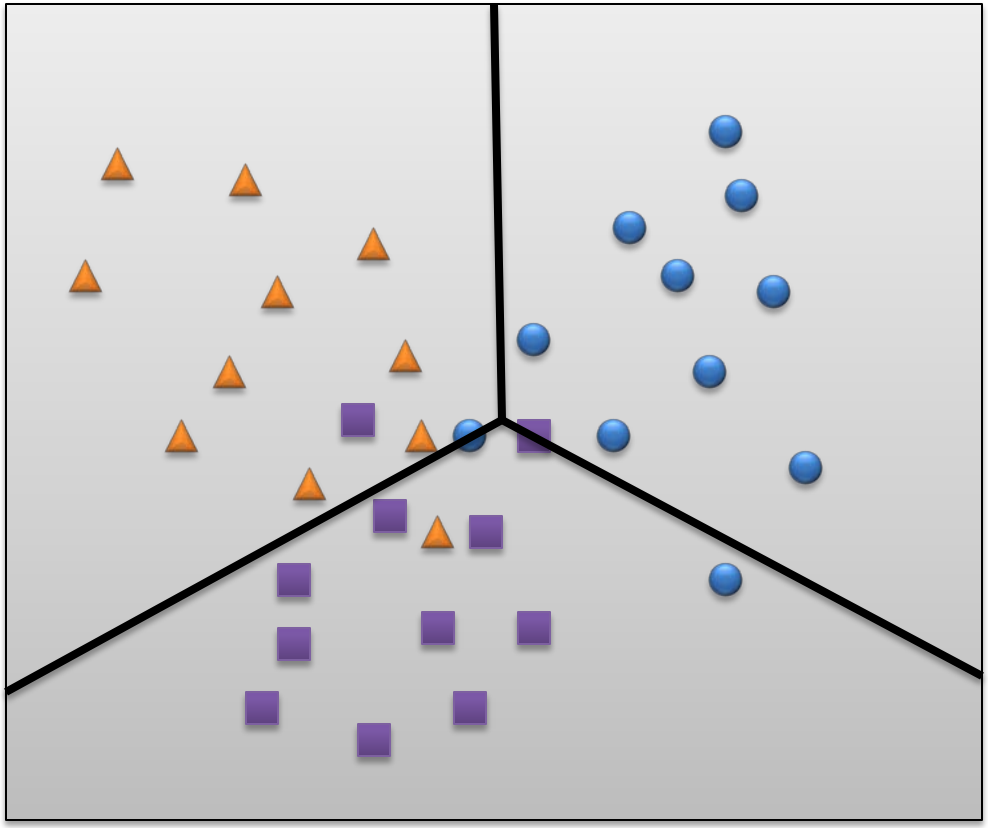


Samples Class 1

Samples Class 2

Samples Class 3

Classification



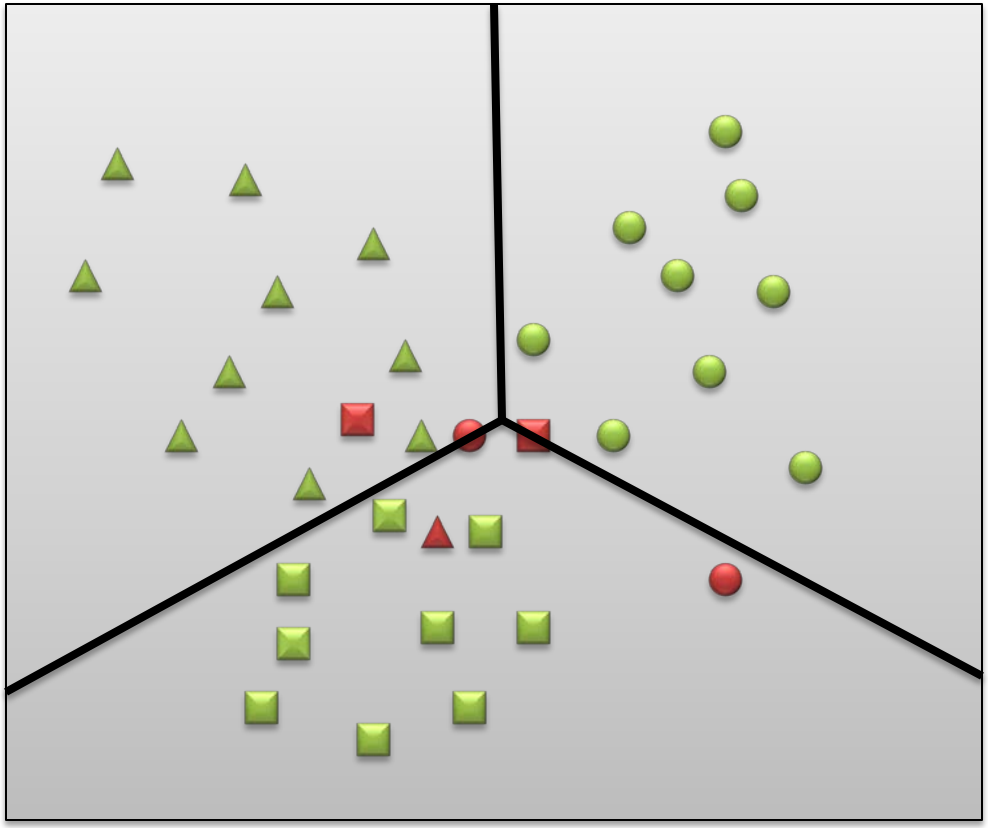
Samples Class 1

Samples Class 2

Samples Class 3

Decision Boundary of Classifier

Classification



Samples Class 1

Samples Class 2

Samples Class 3

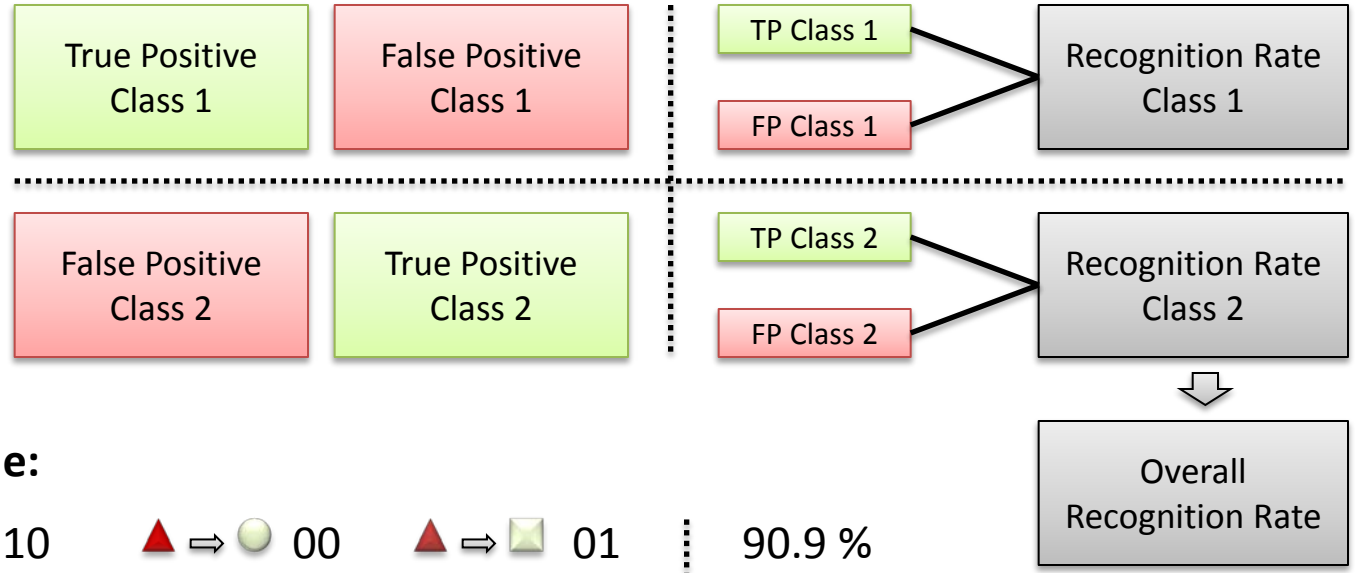
Decision Boundary of Classifier

Correctly classified

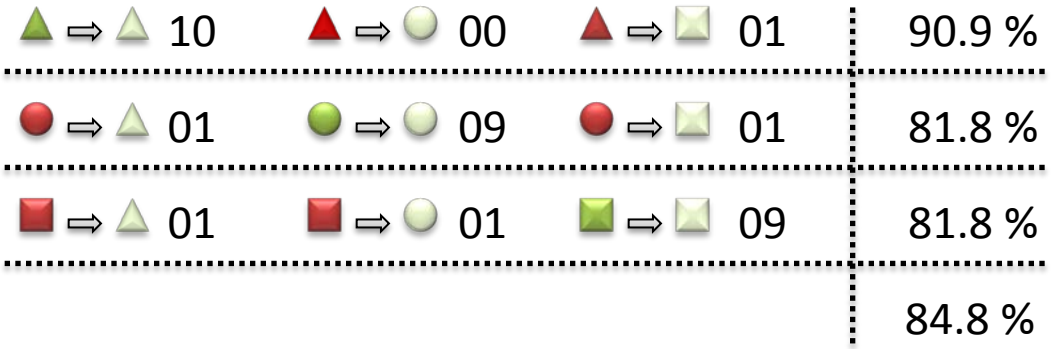
False detections

Evaluation

Confusion Matrix:

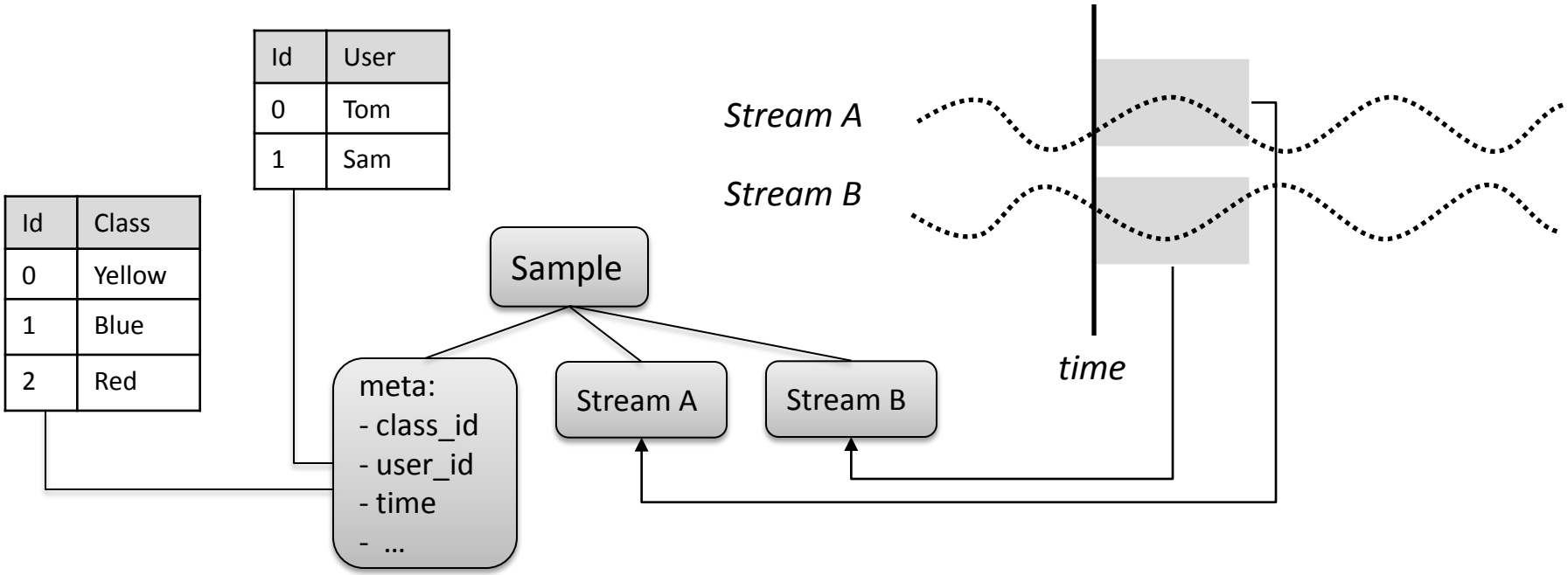


Example:



SAMPLES

Sample




```
struct ssi_sample_t {
  ssi_size_t num; // number of streams
  ssi_stream_t **streams; // streams
  ssi_size_t user_id; // id of user name
  ssi_size_t class_id; // id of label name
  ssi_time_t time; // time in seconds
  ssi_real_t prob; // probability [0..1] to express confidence
};
```

ISamples

```
class ISamples {
public:

  virtual void reset () = 0;
  virtual ssi_sample_t *get (ssi_size_t index) = 0;
  virtual ssi_sample_t *next () = 0;
  virtual ssi_sample_t *next (ssi_size_t class_index) = 0;

  virtual ssi_size_t getSize () = 0;
  virtual ssi_size_t getSize (ssi_size_t class_index) = 0;

  virtual ssi_size_t getClassSize () = 0;
  virtual const ssi_char_t *getClassName (ssi_size_t class_index) = 0;

  virtual ssi_size_t getUserSize () = 0;
  virtual const ssi_char_t *getUserName (ssi_size_t user_index) = 0;

  virtual ssi_size_t getStreamSize () = 0;
  virtual ssi_size_t getStreamDim (ssi_size_t stream_index) = 0;

  virtual bool supportsShallowCopy () = 0;
};
```

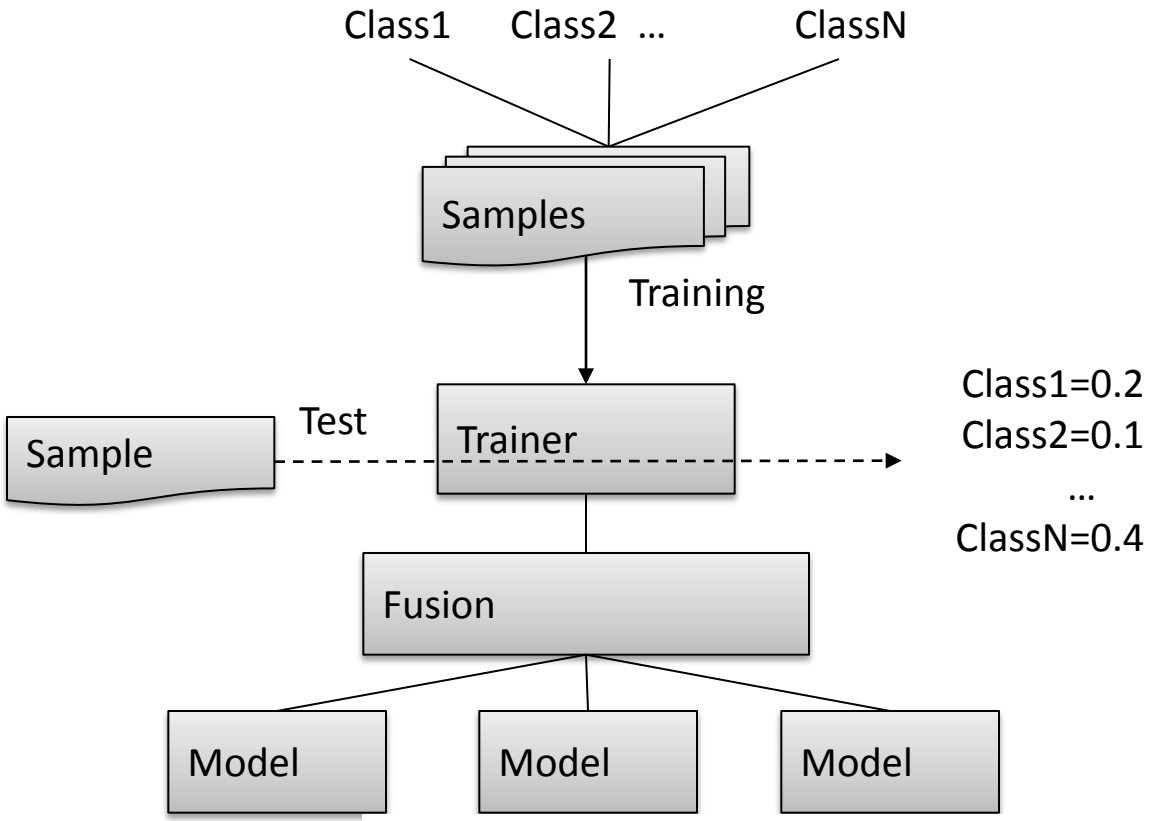
ISamples Wrapper

```
class ISHotClass : public ISamples {
public:
  ISHotClass (ISamples &samples);
  bool setHotClass (ssi_size_t class_id);
...
}

class ISSelectDim : public ISamples {
public:
  ISSelectDim (ISamples &samples);
  bool setSelection (ssi_size_t index, ssi_size_t n_dims, ssi_size_t dims[]);
...
}

class ISTransform : public ISamples {
public:
  ISTransform (ISamples &samples);
  bool setTransformer (ssi_size_t index, ITransformer &transformer);
  void callEnter ();
  void callFlush ();
...
}
```

TRAINER

Trainer

- Wrapper for model or fusion class:



Trainer

```
class Trainer {
public:

  Trainer ();
  Trainer (IModel *model, ssi_size_t stream_index = 0);
  Trainer (ssi_size_t n_models, IModel **models, IFusion *fusion);

  bool train (ISamples &samples);
  bool forward (ssi_size_t num,
    ssi_stream_t *streams[],
    ssi_size_t &class_index);
  bool forward_probs (ssi_size_t num,
    ssi_stream_t *streams[],
    ssi_size_t class_num,
    ssi_real_t *class_probs);
  bool cluster (ISamples &samples);

  void release ();
  static bool Load (Trainer &trainer, const ssi_char_t *filename);
  bool save (const ssi_char_t *filename);
...
```

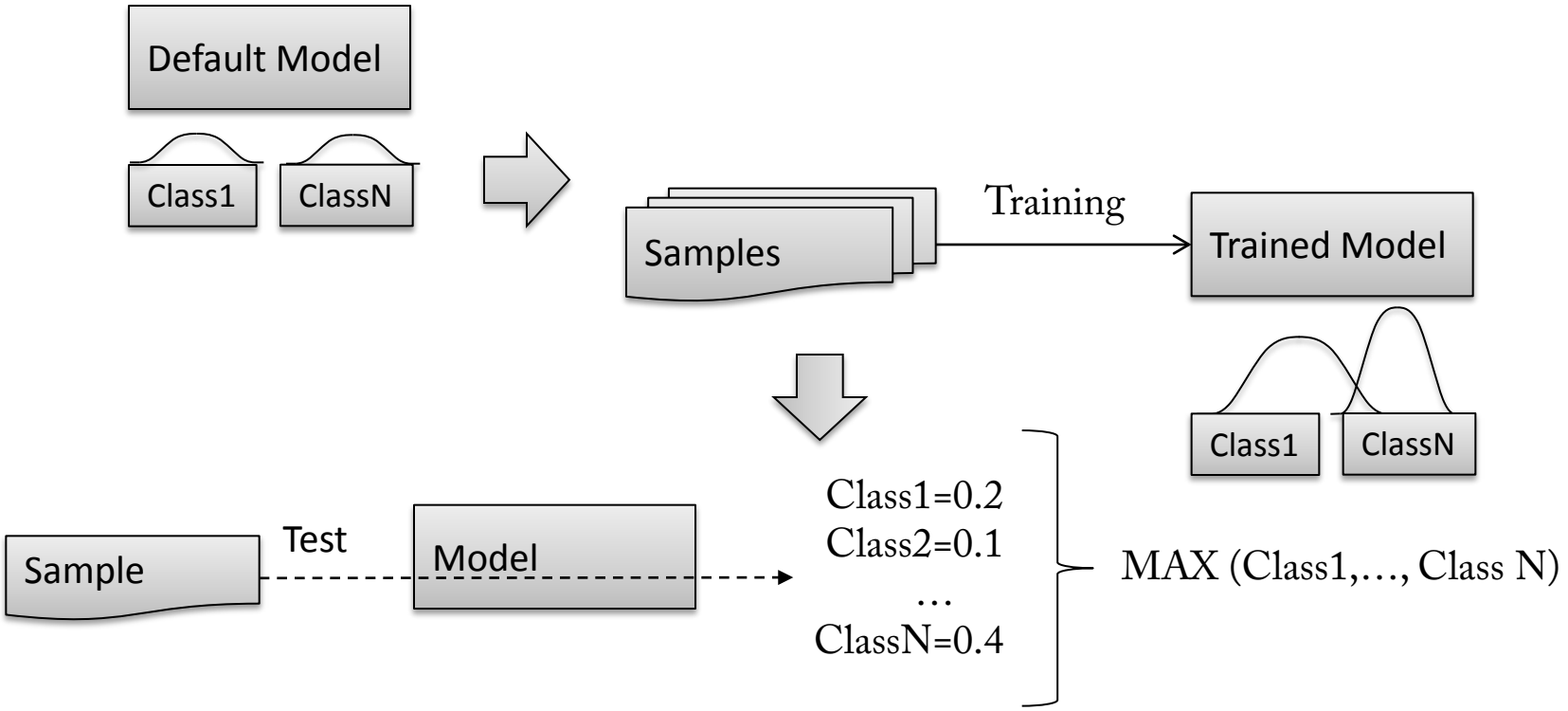
Evaluation

```
class Evaluation {
...
  // evaluiert gegen test set
void eval (Trainer &trainer, ISamples &samples);

  // trainiert mit (100*split)% und testet mit rest
void evalSplit (Trainer &trainer, ISamples &s, ssi_real_t split);

  // bildet k folds und testet jedes einmal gegen den rest
void evalKFold (Trainer &trainer, ISamples &samples, ssi_size_t k);

  // wie k folds, wobei k = #samples
void evalLOO (Trainer &trainer, ISamples &samples);

  // gibt confusion matrix aus
void print (FILE *file = stdout);

  // setzt confusion matrix zurück
void release ();
...
}
```

MODEL

Model

- Training: present samples including class labels
- Test: calculate confidence value for each class and assign sample to class with highest probabily



IModel

```
class IModel : public IObject {
public:
  virtual bool train (ISamples &samples, ssi_size_t stream_index) = 0;
  virtual bool isTrained () = 0;
  virtual bool forward (ssi_stream_t &stream,
    ssi_size_t n_probs,
    ssi_real_t *probs) = 0;
  virtual void release () = 0;

  virtual bool save (const ssi_char_t *filepath) = 0;
  virtual bool load (const ssi_char_t *filepath) = 0;

  virtual ssi_size_t getClassSize () = 0;
  virtual ssi_size_t getStreamDim () = 0;
  virtual ssi_size_t getStreamByte () = 0;
  virtual ssi_type_t getStreamType () = 0;

  virtual const ssi_char_t *getName () = 0;
  virtual const ssi_char_t *getInfo () = 0;

  ssi_object_t getType () { return SSI_MODEL; };
};
```

Model Example

```
class MyModel : public IModel {
public:
...
  bool train (ISamples &samples, ssi_size_t stream_index);
  bool isTrained () { return _centers != 0; };
  bool forward (ssi_stream_t &stream, ssi_size_t n_p, ssi_real_t *p)
  void release ();
  bool save (const ssi_char_t *filepath);
  bool load (const ssi_char_t *filepath);
  ssi_size_t getClassSize () { return _n_classes; };
  ssi_size_t getStreamDim () { return _n_features; };
  ssi_size_t getStreamByte () { return sizeof (ssi_real_t); };
  ssi_type_t getStreamType () { return SSI_REAL; };

protected:

  MyModel ();
  static ssi_real_t dist (ssi_real_t *x1,ssi_real_t *x2,ssi_size_t d);
  ssi_size_t _n_classes;
  ssi_size_t _n_samples;
  ssi_size_t _n_features;
  ssi_real_t **_centers;
};
```

Model Example

```
bool MyModel::train (ISamples &samples, ssi_size_t stream_index) {
  _n_classes = samples.getClassSize ();
  _n_features = samples[0].streams[stream_index]->dim;
  _centers = new ssi_real_t *[_n_classes];
  for (ssi_size_t i = 0; i < _n_classes; i++) {
    _centers[i] = new ssi_real_t[_n_features];
    for (ssi_size_t j = 0; j < _n_features; j++)
      _centers[i][j] = 0;
  }
  ssi_sample_t *sample;
  samples.reset ();
  ssi_real_t *ptr = 0;
  while (sample = samples.next ()) {
    ptr = ssi_pcast (ssi_real_t, sample->streams[stream_index]->ptr);
    for (ssi_size_t j = 0; j < _n_features; j++)
      _centers[sample->class_id][j] += ptr[j];
  }
  for (ssi_size_t i = 0; i < _n_classes; i++) {
    ssi_size_t num = samples.getSize (i);
    for (ssi_size_t j = 0; j < _n_features; j++)
      _centers[i][j] /= num;
  }
}
```

Model Example

```
bool MyModel::forward (ssi_stream_t &stream,
  ssi_size_t n_probs,
  ssi_real_t *probs) {

  ssi_real_t *ptr = ssi_pcast (ssi_real_t, stream.ptr);
  ssi_real_t sum = 0;
  for (ssi_size_t i = 0; i < _n_classes; i++) {
    probs[i] = 1 / dist (ptr, _centers[i], _n_features);
    sum += probs[i];
  }
  for (ssi_size_t i = 0; i < _n_classes; i++) {
    probs[i] /= sum;
  }

  return true;
}
```

Model Example

```
bool MyModel::save (const ssi_char_t *filepath) {

  if (!_centers) {
    ssi_wrn ("not trained");
    return false;
  }

  File *file = File::CreateAndOpen (File::BINARY, File::WRITE, path);

  file->write (&_n_classes, sizeof (_n_classes), 1);
  file->write (&_n_samples, sizeof (_n_samples), 1);
  file->write (&_n_features, sizeof (_n_features), 1);
  for (ssi_size_t i = 0; i < _n_classes; i++) {
    file->write (_centers[i], sizeof (ssi_real_t), _n_features);
  }

  delete file;

  return true;
}
```

Model Example

```
bool MyModel::load (const ssi_char_t *path) {

  File *file = File::CreateAndOpen (File::BINARY, File::READ, path);

  release ();

  file->read (&_n_classes, sizeof (_n_classes), 1);
  file->read (&_n_samples, sizeof (_n_samples), 1);
  file->read (&_n_features, sizeof (_n_features), 1);
  _centers = new ssi_real_t *[_n_classes];
  for (ssi_size_t i = 0; i < _n_classes; i++) {
    _centers[i] = new ssi_real_t[_n_features];
    file->read (_centers[i], sizeof (ssi_real_t), _n_features);
  }

  delete file;

  return true;
}
```

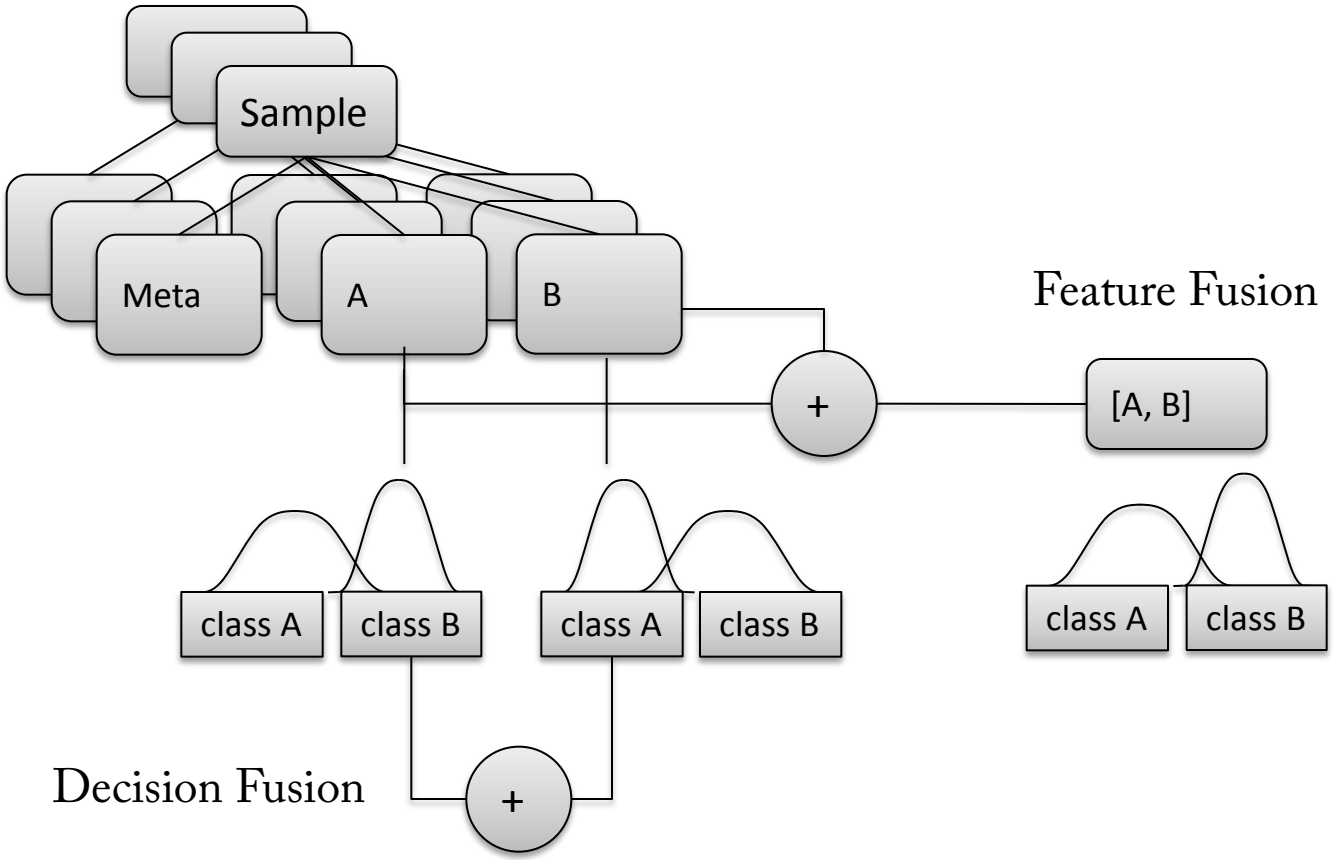
Model Example

```
void ex_model () {
...
  {
    MyModel *model = ...

    Trainer trainer (model, 0);
    trainer.train (strain);
    trainer.save ("mymodel");
  }

  {
    Trainer trainer;
    Trainer::Load (trainer, "mymodel");

    Evaluation eval;
    eval.eval (trainer, sdevel);
    eval.print ();
  }
}
```

FUSION

Fusion

- Feature Fusion: combine feature and train single model
- Decision Fusion: one model per class and combine class probabilities



IFusion

```
class IFusion : public IObject {
public:

  virtual bool train (ssi_size_t n_models,
    IModel **models,
    ISamples &samples) = 0;
  virtual bool isTrained () = 0;
  virtual bool forward (ssi_size_t n_models,
    IModel **models,
    ssi_size_t n_streams,
    ssi_stream_t *streams[],
    ssi_size_t n_probs,
    ssi_real_t *probs) = 0;
  virtual void release () = 0;
  virtual bool save (const ssi_char_t *filepath) = 0;
  virtual bool load (const ssi_char_t *filepath) = 0;

  virtual const ssi_char_t *getName () = 0;
  virtual const ssi_char_t *getInfo () = 0;

  ssi_object_t getType () { return SSI_FUSION; };
};
```

Fusion Example

```
class MyFusion : public IFusion {
public:
...
  bool train (ssi_size_t n_models,IModel **models,ISamples &samples);
  bool isTrained () { return _is_trained; };
  bool forward (ssi_size_t n_models,
    IModel **models,
    ssi_size_t n_streams,
    ssi_stream_t *streams[],
    ssi_size_t n_probs,
    ssi_real_t *probs);
  void release ();
  bool save (const ssi_char_t *filepath);
  bool load (const ssi_char_t *filepath);

protected:

  MyFusion ();
  bool _is_trained;
};
```

Fusion Example

```
bool MyFusion::train (ssi_size_t n_models,
  IModel **models,
  ISamples &samples) {

  ssi_size_t n_streams = samples.getStreamSize ();

  for (ssi_size_t n_model = 0; n_model < n_models; n_model++) {
    if (!models[n_model]->isTrained ()) {
      models[n_model]->train (samples, n_model);
    }
  }

  _is_trained = true;

  return true;
}
```

Fusion Example

```
bool MyFusion::forward (ssi_size_t n_models,
  IModel **models,
  ssi_size_t n_streams,
  ssi_stream_t *streams[],
  ssi_size_t n_probs,
  ssi_real_t *probs) {

  ssi_real_t *tmp_probs = new ssi_real_t[n_probs];
  models[0]->forward (*streams[0], n_probs, probs);

  for (ssi_size_t n_model = 1; n_model < n_models; n_model++) {
    models[n_model]->forward (*streams[n_model], n_probs, tmp_probs);
    for (ssi_size_t n_prob = 0; n_prob < n_probs; n_prob++) {
      if (probs[n_prob] < tmp_probs[n_prob]) {
        probs[n_prob] = tmp_probs[n_prob];
      }
    }
   }

  delete[] tmp_probs;
  return true;
}
```

Fusion Example

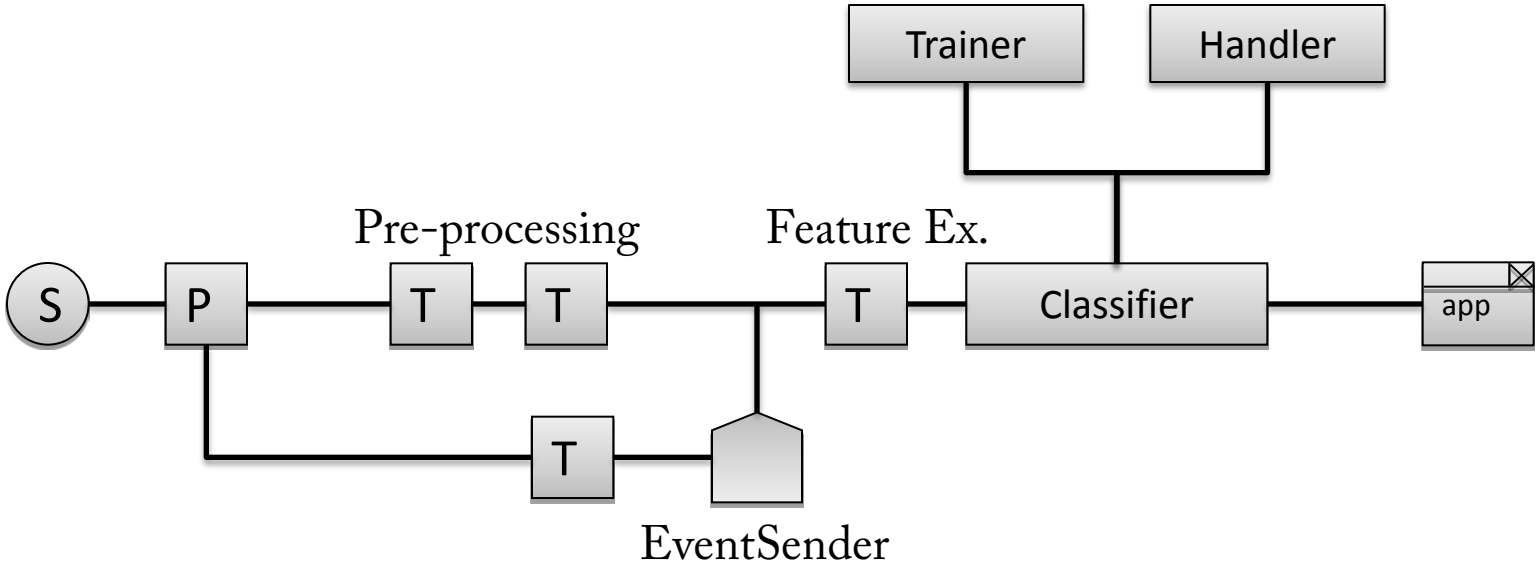
```
void ex_fusion () {
...
  {
    IModel **models = new IModel *[n_streams];
    for (ssi_size_t i = 0; i < n_streams; i++) {
      models[i] = ...
    }
    MyFusion *fusion = ...

    Trainer trainer (n_streams, models, fusion);
    trainer.train (strain);
    trainer.save ("myfusion");
  }

  {
    Trainer trainer;
    Trainer::Load (trainer, "myfusion");
    Evaluation eval;
    eval.eval (trainer, sdevel);
    eval.print ();
  }
}
```

Social Signal Interpretation

ONLINE CLASSIFICATION

Online Classification

- Trigger: decides when classifier is invoked
- Classifier: calculates feature vector passes it to trainer
- Handler: knows how to proceed with result



Example

```
void ex_online () {

...

  Trainer trainer;
  Trainer::Load (trainer, mymodel);

  Classifier *classifier = ...
  classifier->setTrainer (trainer);


  ITransformable *transformer = ...

  frame->addEventConsumer(cursor_p, classifier, board, "event@sender", transformer);

...

}
```