RelayMQ++

A Quick Introduction

Christian Scheideler, Alexander Setzer, Thorsten Götte

SS 17

Project Group: Trusted Communication Module
Paderborn University

You can send questions to

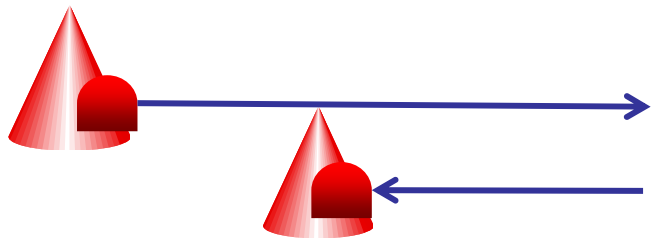
thgoette@mail.upb.de

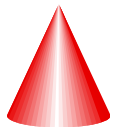
Contents

Basic Concepts

TCM Model

Abstraction:



- Processes  (at AL and TCL)
- Relays  (managed by the TCL)
- Processes have references (→) to local relays

Main Classes and Methods

RelayRef

```
1    typedef RelayRef
```

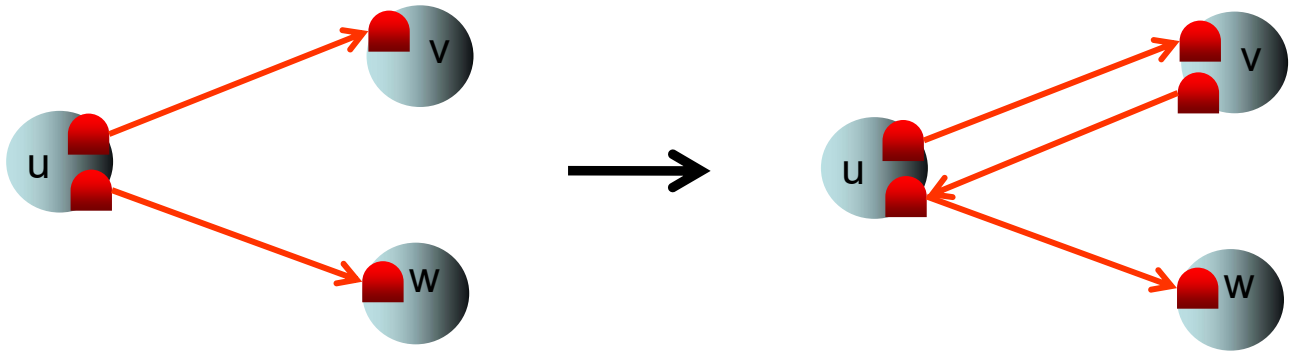
- Can send messages to a Subject
- Has no direct c'tor
- Can be shared via sending a message
- Not intended to be subclassed

RelayRef->Send

```
    /**
    * Sends a string and references
    *
    * Returns true on success
    */
    bool Send(string msg, vector<RelayRef> refs);
```

TCM Model

Recall the safe introduction rule:



Instead of introducing w to v, u can only introduce its relay to w to v.

RelayRef->Send

```
RelayRef to_v = ...;
RelayRef to_w = ...;

to_v->Send("INTRODUCE", {to_w});
```

RelayRef->CompareTo

```
/* Checks if Relays have the same target */
bool Compare(RelayRef otherRef);
```

RelayRef->Close

```
/*Closes Relay, s.t. no more messages are handled*/
void Close();
```

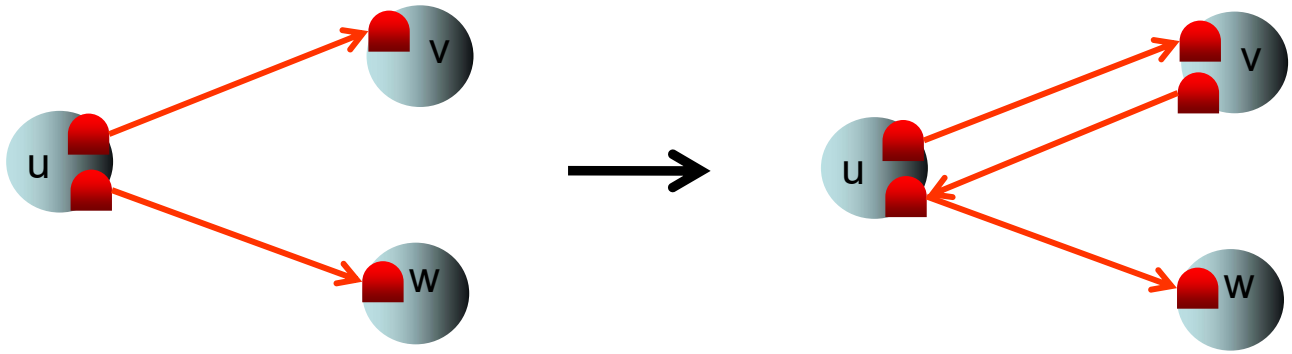
Subject

```
class RelaySubject
```

- Models a process
- Can receive messages
- Can create additional incoming relays
- Monitors its Relays
- Intended to be subclassed for custom behavior

TCM Model

Recall the safe introduction rule:



Instead of introducing w to v, u can only introduce its relay to w to v.

Subject->onMessage

```
/**
 * Executed whenever there is a message
 * for the Subject
 */
void
onMessage(RelayRef receiver,
          string msg,
          vector<RelayRef> refs);
```

Subject->onMessage

- **`receiver`** is an incoming relay that received the message
- **`msg`** is the message
- **`refs`** is a list of outgoing relays

RelayRef->Send

```
RelayRef to_v = ...;
RelayRef to_w = ...;

to_v->Send("INTRODUCE", {to_w});
```

Subject->onMessage

```
void
onMessage(RelayRef receiver,
          string msg,
          vector<RelayRef> refs){

  assert(receiver == this->Incoming());

  assert(msg == "INTRODUCE");

  assert(refs.size() == 1);
  RelayRef to_w = ref[0];

}
```

Subject->onTimeout

```
    /**
    * Executed if
    * a) there is no I/O
    * b) at least 1s has passed
    */
    bool onTimeout();
```

Subject->onClose

```
    /**
2   * Called if the Relay is closed remotely
    */
4   void onClose(RelayRef);
```

Subject->Stop

```
    /**
2   * Stops the Subject.
    *
4   * Note: Make sure that
    * all Relays are closed !!!
6   */
    void Stop();
```

Subject->CreateNewIncoming

```
/**
 * Create new IncomingRelay for this Subject
 */
RelayRef CreateNewIncoming();
```

Note: This c'tor makes sure that messages are received

Subject->Add

```
1 /* Adds a socket to the subject */
  void add(SocketRef s, function<void(SocketRef)> f);
```

- The socket **s** can be an an arbitrary ZeroMQ socket
- The function **f** is called whenever there is an incoming message

Subject->Add

This code lets the subject listen to a local address:

```
    auto s= make_shared<ZmqSocket>(ZMQ_PULL);
2   s->Bind("tcp://localhost:9001");
    this->add(s, [this](){
4       string msg = s->RecvString();
        ...
6   });
```

Subject->Add

Any program can send messages to this subject:

```
zsock_t* sock = zsock_new(ZMQ_PUSH);
zsock_connect(sock,"tcp://localhost:9001");
zstr_send(sock, "EVENT");
```

Subject->ExecuteLater

```
/**
* Executes the given function later
*/
TicketHandle ExecuteLater(function<void()> fun);
```

Via the **`TicketHandle`** the execution can be stopped

Subject->ExecuteLater

- Use this to simulate time
- Do NOT use `sleep(long millis)`

Subject->ExecuteLater

```
1 r->Send("Start", {this->Incoming()});
  sleep(1000)
3 r->Send("Finish", {});
```

- The simulation would freeze for 1s
- The **`Start`**-Message would be delivered after **`sleep`** returns

Subject->ExecuteLater

```
r->Send("Start", {this->Incoming()});
this->ExecuteLater([this](){
  r->Send("Finish", {});
});
```

- The simulation never freezes
- The **Finish**-Message will be sent 1s later

Subject->AddEdge

```
/* Displays edge in the Visualisation */
void addEdge(RelayRef to);
```

- The function this idempotent
- Blocks if no Visualistation running

ApplicationContext

```
class ApplicationContext;
```

- Handles I/O in the background
- Calls onMessage and onTimeout
- Manages your Subjects

ApplicationContext::Create

```
/**
* Creates a new Subject of class T
* and passes Args to its c'tor
*/
template<class T, typename... Args>
RelayRef Create<T>(Args... args);
```

ApplicationContext::Create

- Use this to create your Subjects
- Do NOT use the normal c'tor
- Returns a RelayRef and NOT the subject instance

ApplicationContext

```
1 int main(void){
2
3     // Set the TIMEOUT to 1s
      ApplicationContext::Init(1000);
5
      auto one = Create<MySubject>(a,b,c);
7     auto two = Create<MySubject>(x,y,z);
8
9     ApplicationContext::Start();
  }
```

Example Program

Example Subject

```
class PingPongSubject: public RelaySubject{

  // Another Subject where we send msgs
  RelayRef theOtherGuy;

}
```

Example Subject

```
void
PingSubject::onMessage(RelayRef r,
                       string msg,
                       vector<RelayRef> refs)
  {

    if(msg == "INIT")
      theOtherGuy = ref[0];
    else
    if(msg == "PING")
      theOtherGuy->Send("PONG", {});
    else
    if(msg == "PONG")
      theOtherGuy->Send("PING", {});
  }
```

Example Main

```
int main(void){
    ApplicationContext::Init();

    auto ping= Create<PingSubject>();
    auto pong= Create<PingSubject>();

    ping->Send("INIT", {pong});
    pong->Send("INIT", {ping});

    ping->Send("PONG", {})

    ApplicationContext::Start();
}
```

Setup

Setup

1. Install Ubuntu, e.g., from
    - **https://ubuntu.org/**
    - **https://lubuntu.org/**
    - **https://xubuntu.org/**
2. Download and register for Clion
    - **www.jetbrains.com/clion/**
    - **www.jetbrains.com/student/**

Install Dependencies

1. Install cmake and git via:

   `sudo apt-get install git cmake build-essential`
2. Clone Repository `git clone`

   `https://git.cs.upb.de/crc-901/RelayMQ-Example`

   `-b development`
3. Download dependencies directly via

   `cmake ./`

Starting the Simulation

1. Write your main program
2. Execute
   `python sim.py <Name> [args]`

Starting the Examples

1. Custom Programm
   `python sim.py Homework`
2. Example 1
   `python sim.py BuildClique`
3. Example 2
   `python sim.py PointerDoubling`

Useful Links

1. Cmake: **www.cmake.org/**
2. ZeroMQ: **www.zeromq.org/**
3. C++: **www.stroustrup.com/Tour.html**
4. Lambda Expressions: **en.cppreference.com/w/cpp/language/lambda**
5. JSON for C++: **www.github.com/open-source-parsers/jsoncpp**

Thanks for your attention!

Questions?