Technische Hochschule Augsburg

soar_ros: A ROS 2 Interface for the Cognitive Architecture Soar

Moritz Schmidt

Agenda

- What is Soar?
- Soar & ROS 2: Challenges and Requirements
- Mapping Soar to ROS 2 communication
- Preliminary Performance Evaluation
- What’s next?

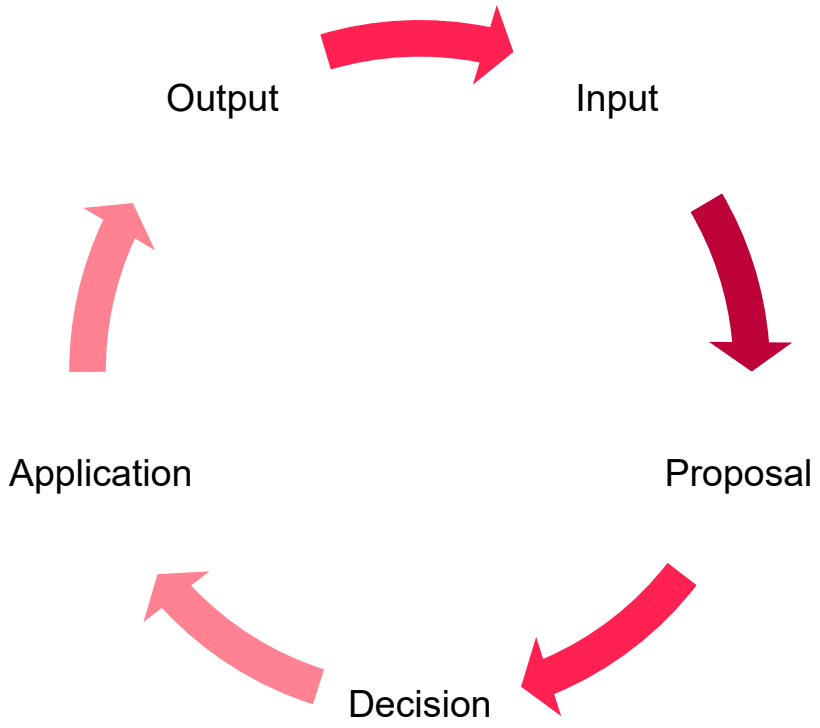
THA

What is Soar?

Definition of Soar

Soar is a general cognitive architecture for developing systems that exhibit intelligent behavior.

- Rule-based system (white box)
- Chunking (New rule generation based on impasse)
- RL capabilities
- Open Source (UoM, CIC)



Soar & ROS 2: Challenges and Requirements

Challenges

- Soar synchronous architecture blocks ROS 2
- Run Soar agents during time consuming tasks in ROS
- Time of ROS2 message != Soar input phase
- Fixed callback interfaces for ROS 2 and Soar

Requirements

- Soar Kernel runs continuously
- Add ROS 2 interfaces via builder pattern
- Only parsing of Soar WMEs and ROS 2 messages by dev
- Debug
  - Support Soar Java Debugger
  - Hook into VS Code ROS 2 debug tooling
  - Stop kernel via ROS 2 messages
  - Full ROS Tooling integration (logging, debug)
- No maintenance of additional Soar fork

THA

How to use?

Example Code

```
class TestClient : public SoaROS::Client<example_interfaces::srv::AddTwoInts>
{
public:
  TestClient(sml::Agent * agent, rclcpp::Node::SharedPtr node, const std::string & topic)
  : Client<example_interfaces::srv::AddTwoInts>(agent, node, topic) {}
  ~TestClient() {}

  example_interfaces::srv::AddTwoInts::Request::SharedPtr parse(sml::Identifier * id) override
  {
    example_interfaces::srv::AddTwoInts::Request::SharedPtr request =
      std::make_shared<example_interfaces::srv::AddTwoInts::Request>();
    auto a = std::stoi(id->GetParameterValue("a"));
    auto b = std::stoi(id->GetParameterValue("b"));
    request.get()->a = a;
    request.get()->b = b;
    RCLCPP_INFO(m_node->get_logger(), "Request computation: %d + %d", a, b);
    return request;
  }

  void parse(example_interfaces::srv::AddTwoInts::Response::SharedPtr msg) override
  {
    sml::Identifier * il = getAgent()->GetInputLink();
    sml::Identifier * pId = il->CreateIdWME("AddTwoIntsClient");
    pId->CreateIntWME("sum", msg.get()->sum);
    RCLCPP_INFO(m_node->get_logger(), "Result: %ld", msg.get()->sum);
  }
};
```

How to use?

Example Code

```
class TestClient : public SoaROS::Client<example_interfaces::srv::AddTwoInts>
{
public:
  TestClient(sml::Agent * agent, rclcpp::Node::SharedPtr node, const std::string & topic)
  : Client<example_interfaces::srv::AddTwoInts>(agent, node, topic) {}
  ~TestClient() {}

  example_interfaces::srv::AddTwoInts::Request::SharedPtr parse(sml::Identifier * id) override
  {
    example_interfaces::srv::AddTwoInts::Request::SharedPtr request =
      std::make_shared<example_interfaces::srv::AddTwoInts::Request>();
    auto a = std::stoi(id->GetParameterValue("a"));
    auto b = std::stoi(id->GetParameterValue("b"));
    request.get()->a = a;
    request.get()->b = b;
    RCLCPP_INFO(m_node->get_logger(), "Request computation: %d + %d", a, b);
    return request;
  }

  void parse(example_interfaces::srv::AddTwoInts::Response::SharedPtr msg) override
  {
    sml::Identifier * il = getAgent()->GetInputLink();
    sml::Identifier * pId = il->CreateIdWME("AddTwoIntsClient");
    pId->CreateIntWME("sum", msg.get()->sum);
    RCLCPP_INFO(m_node->get_logger(), "Result: %ld", msg.get()->sum);
  }
};
```

How to use?

Example Code

```
class TestClient : public SoaROS::Client<example_interfaces::srv::AddTwoInts>
{
public:
  TestClient(sml::Agent * agent, rclcpp::Node::SharedPtr node, const std::string & topic)
  : Client<example_interfaces::srv::AddTwoInts>(agent, node, topic) {}
  ~TestClient() {}

  example_interfaces::srv::AddTwoInts::Request::SharedPtr parse(sml::Identifier * id) override
  {
    example_interfaces::srv::AddTwoInts::Request::SharedPtr request =
      std::make_shared<example_interfaces::srv::AddTwoInts::Request>();
    auto a = std::stoi(id->GetParameterValue("a"));
    auto b = std::stoi(id->GetParameterValue("b"));
    request.get()->a = a;
    request.get()->b = b;
    RCLCPP_INFO(m_node->get_logger(), "Request computation: %d + %d", a, b);
    return request;
  }

  void parse(example_interfaces::srv::AddTwoInts::Response::SharedPtr msg) override
  {
    sml::Identifier * il = getAgent()->GetInputLink();
    sml::Identifier * pId = il->CreateIdWME("AddTwoIntsClient");
    pId->CreateIntWME("sum", msg.get()->sum);
    RCLCPP_INFO(m_node->get_logger(), "Result: %ld", msg.get()->sum);
  }
};
```

```
int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);


  const std::string package_name = "soaros";
  const std::string share_directory =
    ament_index_cpp::get_package_share_directory(package_name);

  std::string soar_path = share_directory + "/Soar/main.soar";
  auto node = std::make_shared<SoaROS::SoarRunner>("Test Agent", soar_path);

  std::shared_ptr<SoaROS::Client<example_interfaces::srv::AddTwoInts>> client =
    std::make_shared<TestClient>(node.get()->getAgent(), node, "AddTwoIntsClient");
  node->addClient(client, "AddTwoIntsClient");

  node->startThread();

  rclcpp::executors::MultiThreadedExecutor executor;
  executor.add_node(node);
  executor.spin();
  rclcpp::shutdown();

  return 0;
}
```

Application

Preliminary Performance Evaluation

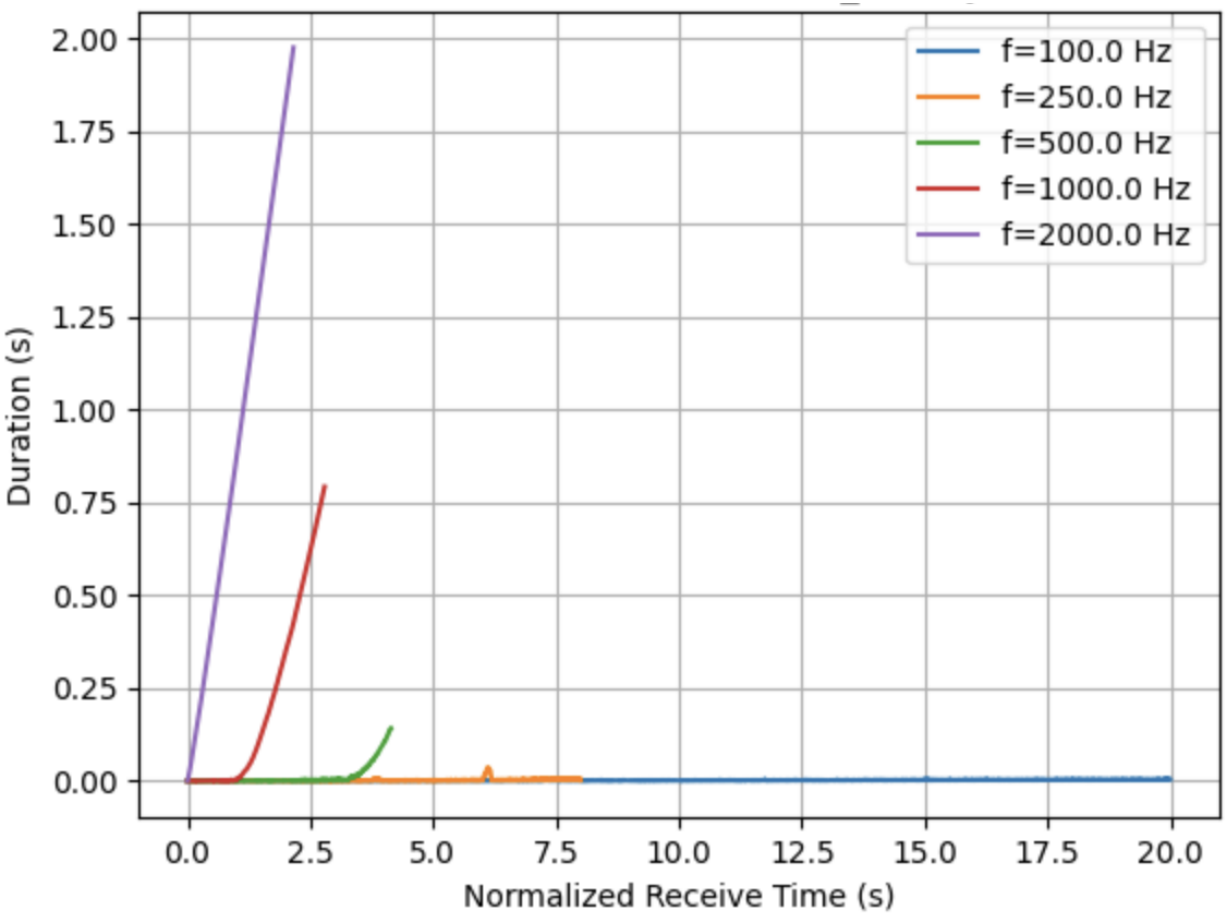
SISO

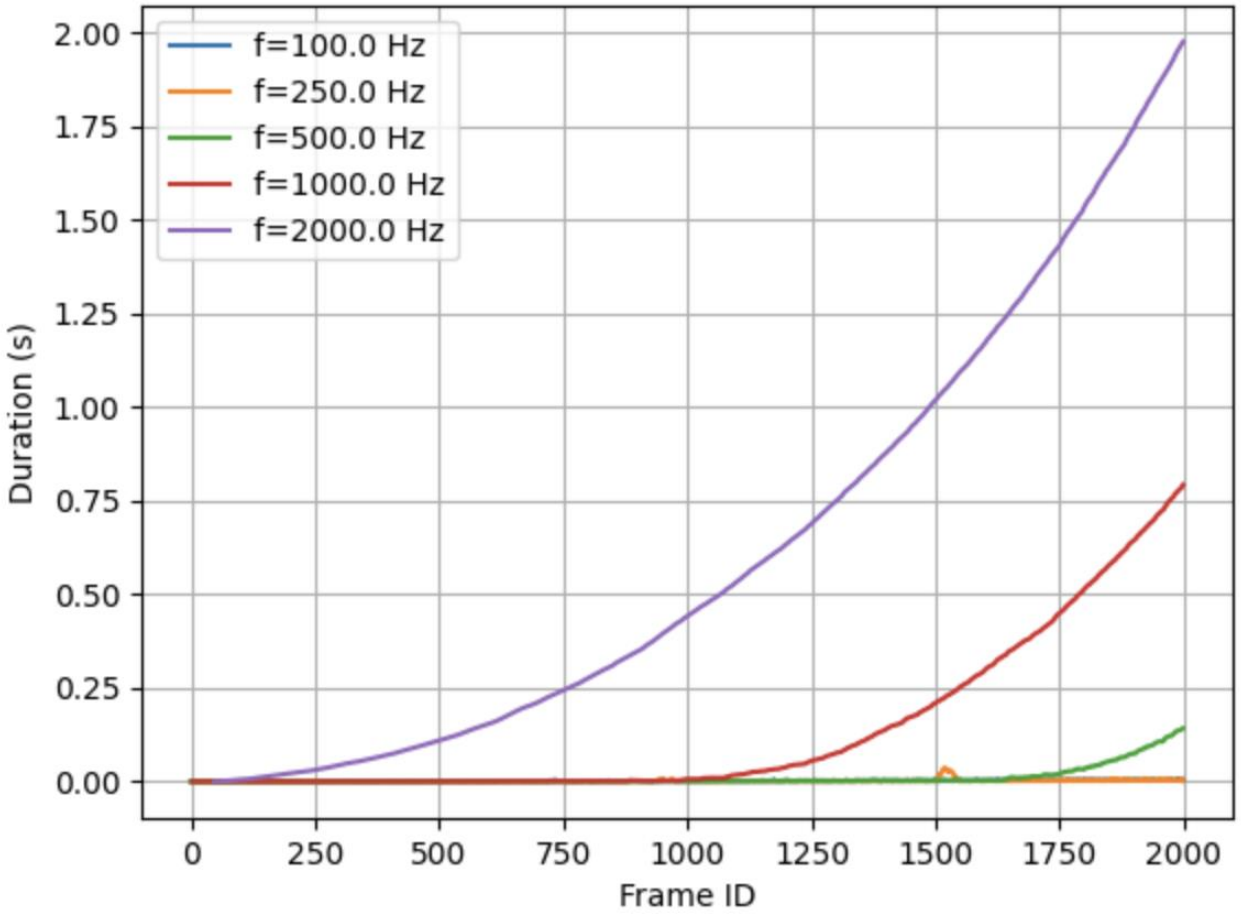
- Sender publishes `std_msgs::msg::Header` messages to the `input` topic at configurable frequencies. Use `frame_id` as counter
- System node copies input messages to output
- Receiver subscribes to the `output` topic and logs received messages with timestamps.

THA

Preliminary Performance Evaluation

SISO





Conclusion

- Threading managed in the background. Not exposed to API/ interface.
- ROS 2 ←→ Soar parsing is the only required implementation
- High Code reusability due to templates and generics

THA

What’s next?

- Launch tests fail in CI due to bad shutdown (zombie process); but only in combination with launch testing
- Multi-input/ multi-output performance tests
- Extension for common ROS 2 message definition parsing between ROS 2 and Soar
- Packaging & addition to ROS apt repository via build farm
  - Probably requires packaging of Soar base library (currently via Cmake *FetchContent_Declare*)

THA

Links

Soar Documentation

soar_ros GitHub

Project @ THA (update will follow 12/2025)

Technische Hochschule Augsburg

Moritz Schmidt

Faculty of Electrical Engineering
Research Associate

Technical University of Applied Sciences Augsburg
An der Hochschule 1
D-86161 Augsburg
T +49 821 5586 1010
moritz.schmidt@tha.de
www.tha.de

ROS 2 Communication applied to Soar

Defined via **message type** (pub/sub, service, action)**, name** (topic) and quality of service (**QoS**)

Pub/Sub

Publisher: Soar output[*topic*]

Subscriber: Soar input[*topic*]

Services

Service: Soar input[*topic*] → process → Soar output[*topic*]

Client: Soar output → wait for answer → Soar input[*topic*]

Actions

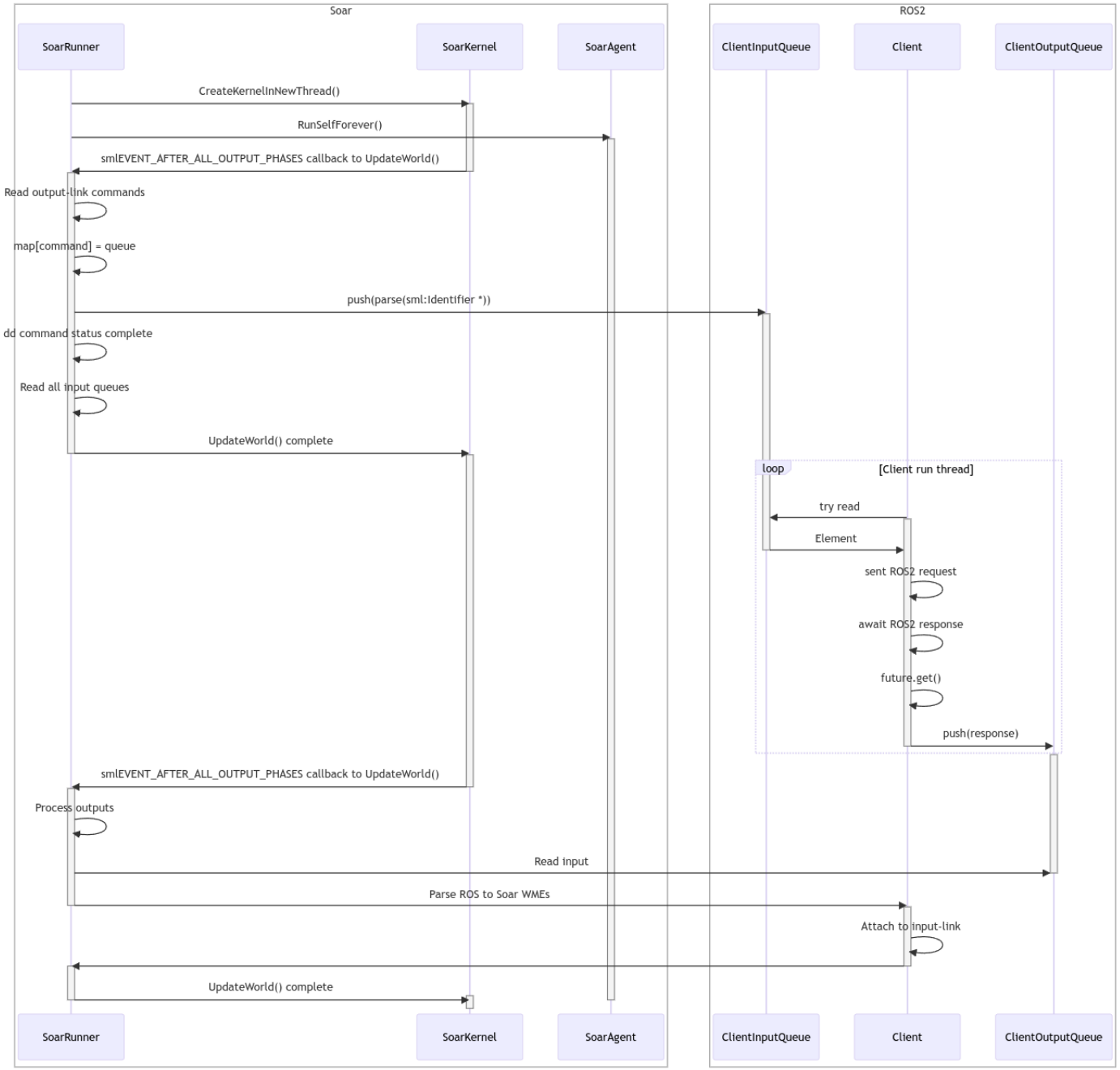
Server: Not implemented

Client: Implemented[*topic*]

THA

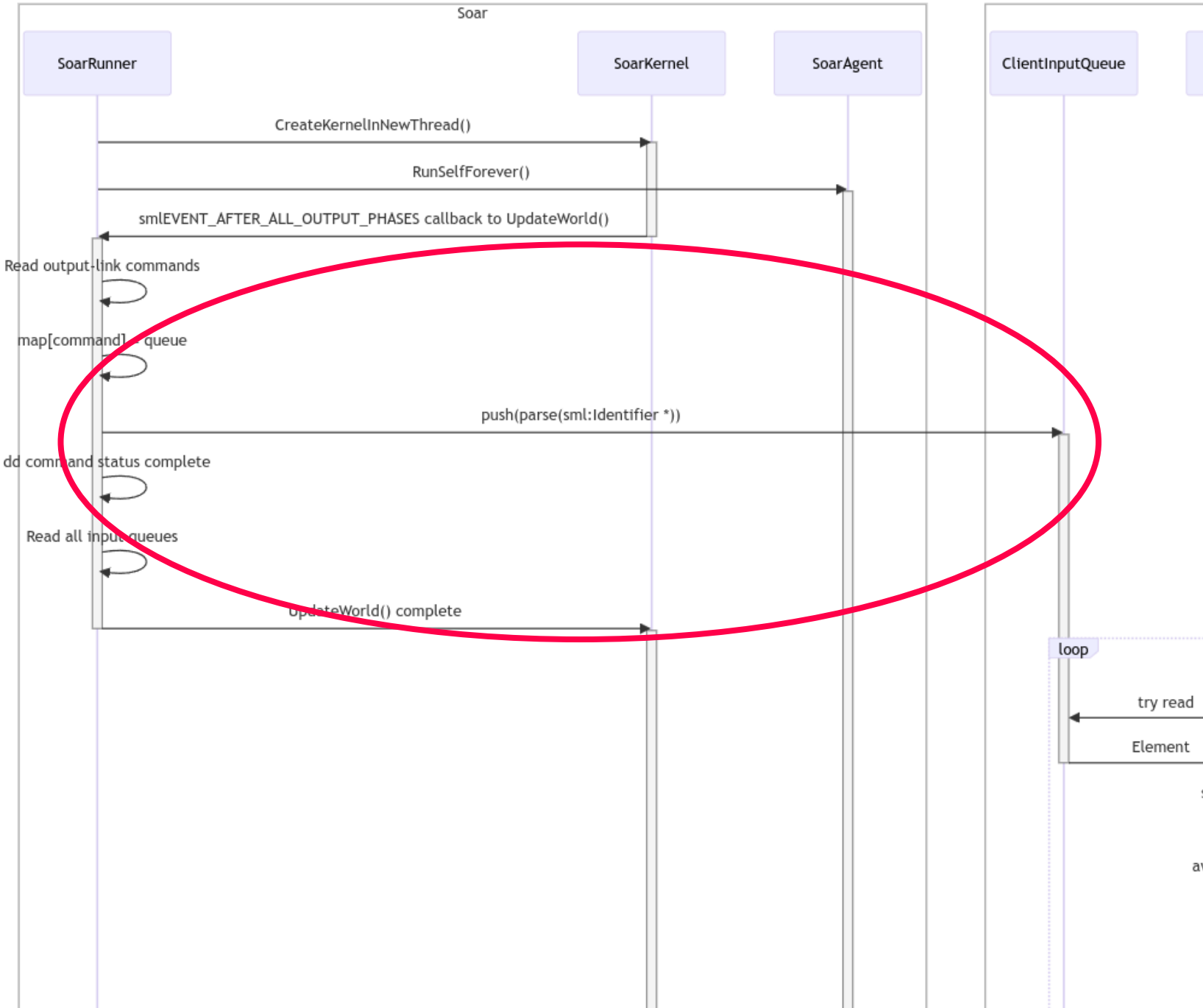
Theoretical Example

Client



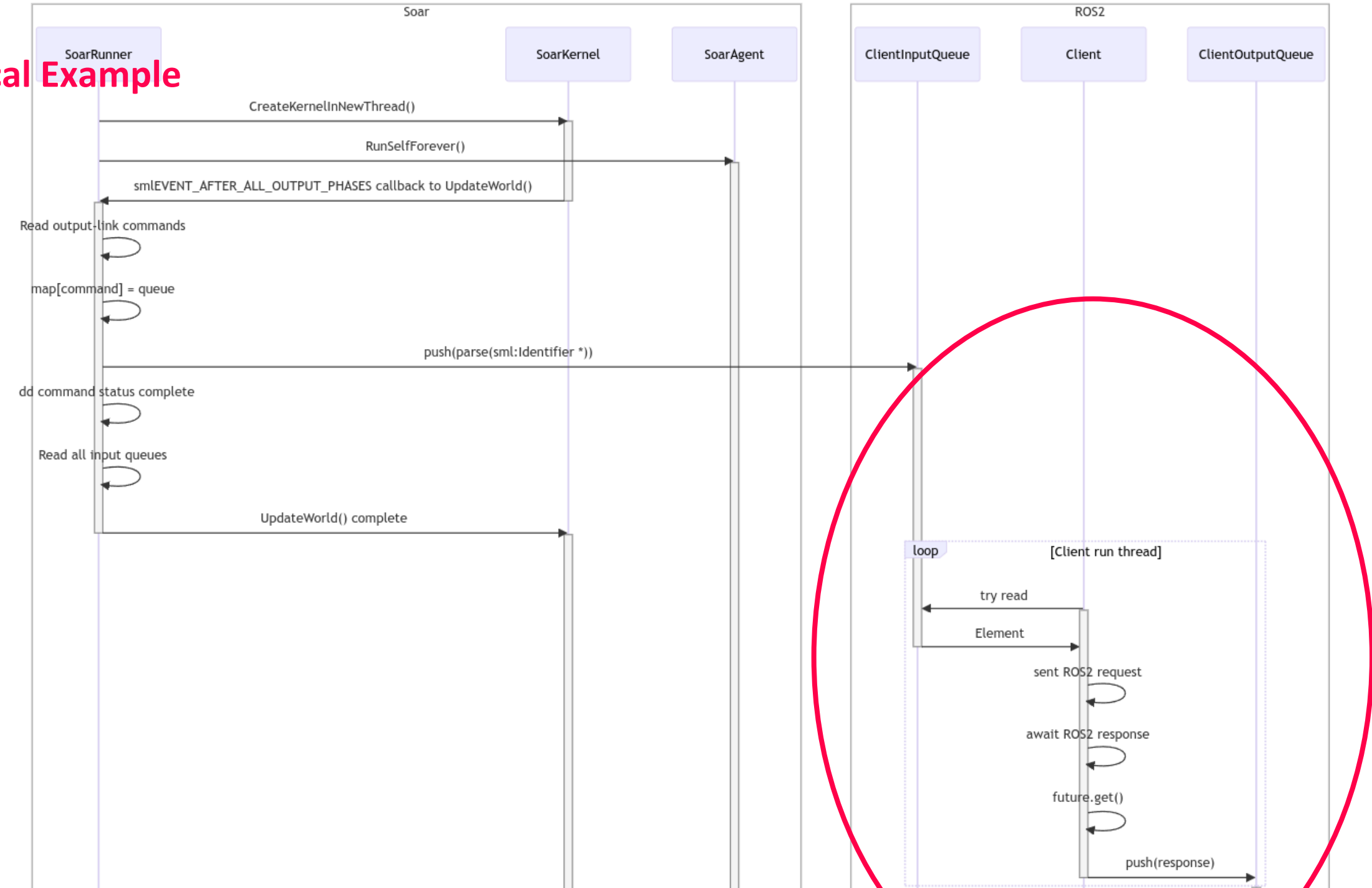
Theoretical Example

Client



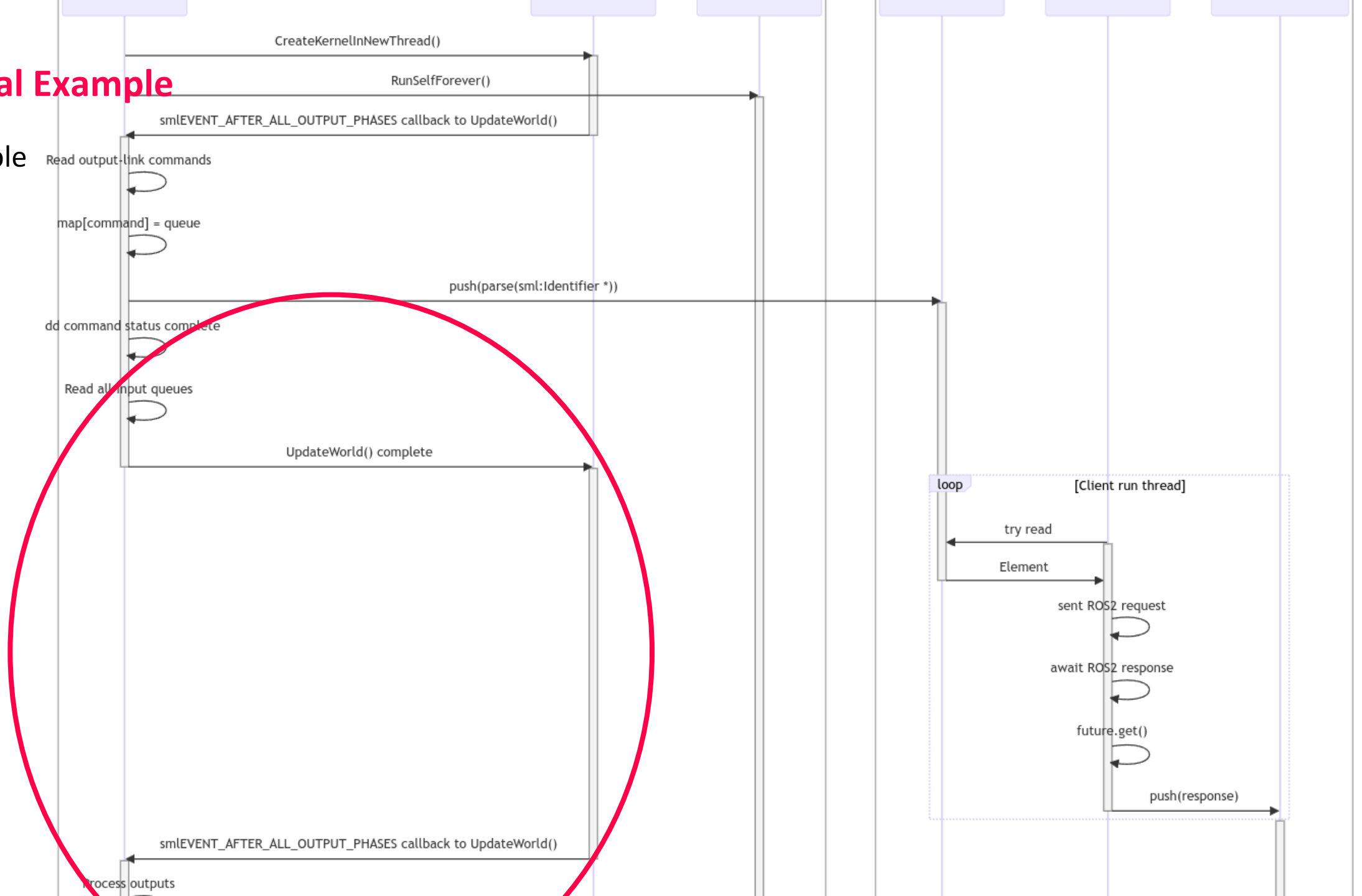
Theoretical Example

Client



Theoretical Example

Client Example



Theoretical Example

Client