Technische Hochschule Augsburg

Soar as Cognitive Control Unit for a Robot Task Delegation Interface based on ROS2

Moritz Schmidt

Agenda

- Use Case: Soar-based Sander Agent
- SoaROS: ROS2 Interface for Soar

Use Case

Context

- Dull and dirty sanding tasks are not automated in SMEs
- Skilled labor shortage in Germany
- Strong SME sector in Germany w.r.t. GDP
- Industrial grade products for cobots and end-effectors available
- Variable lot sizes or single-unit production → Reprogramming cobot not feasible
- Most operators have no desire to learn programming

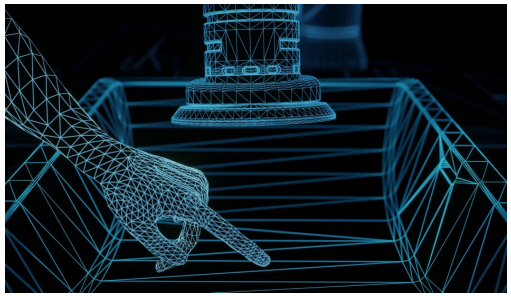
Use Case

Schematic User Interaction with Sander Agent

Website animatic

1. Approach cobot

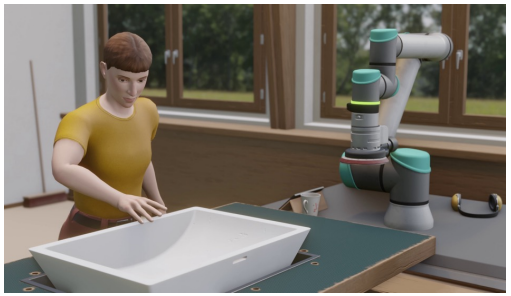
2. Delegate task



3. Scan object

4. Start process

5. Cobot is working

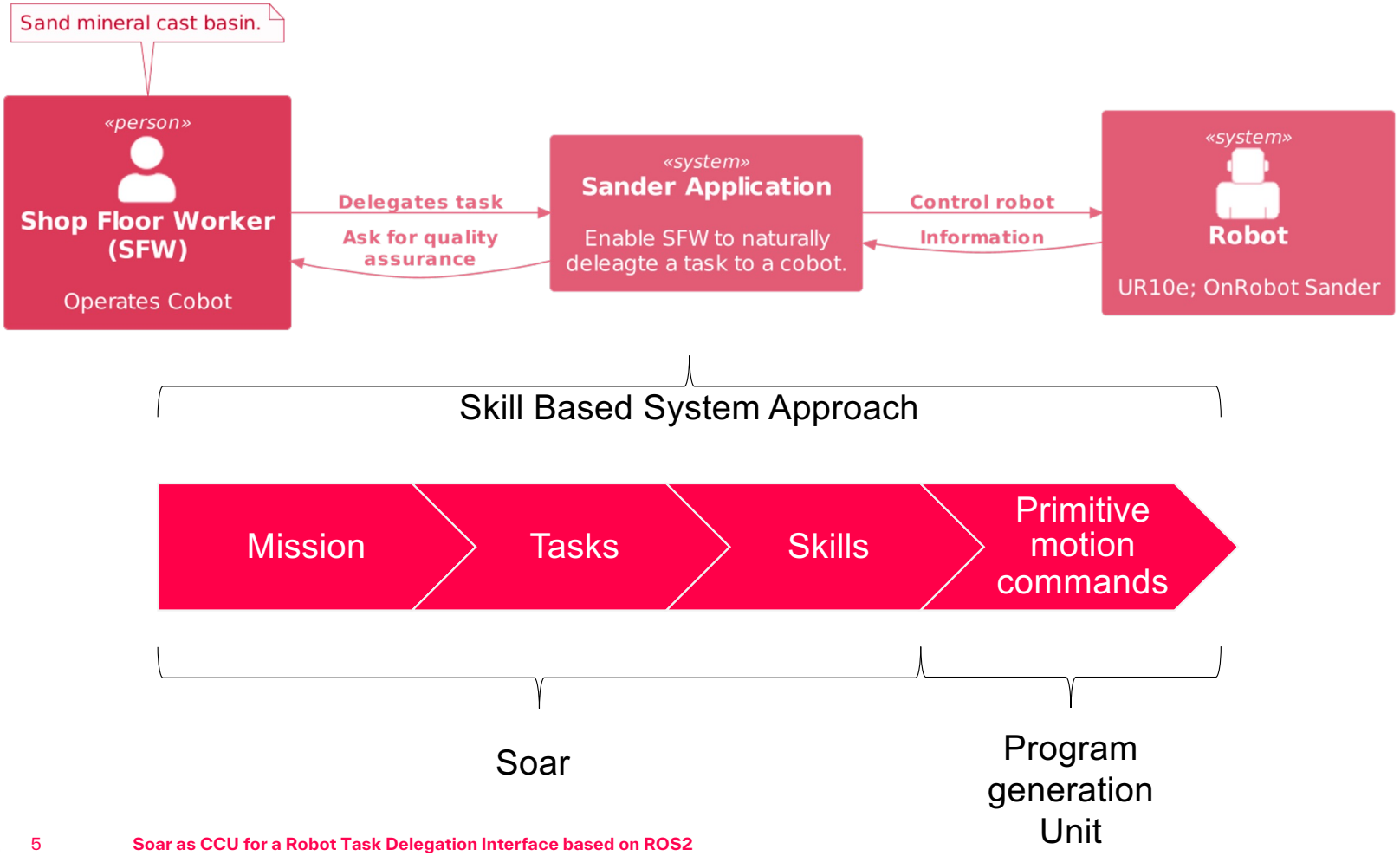


6. Quality assurance

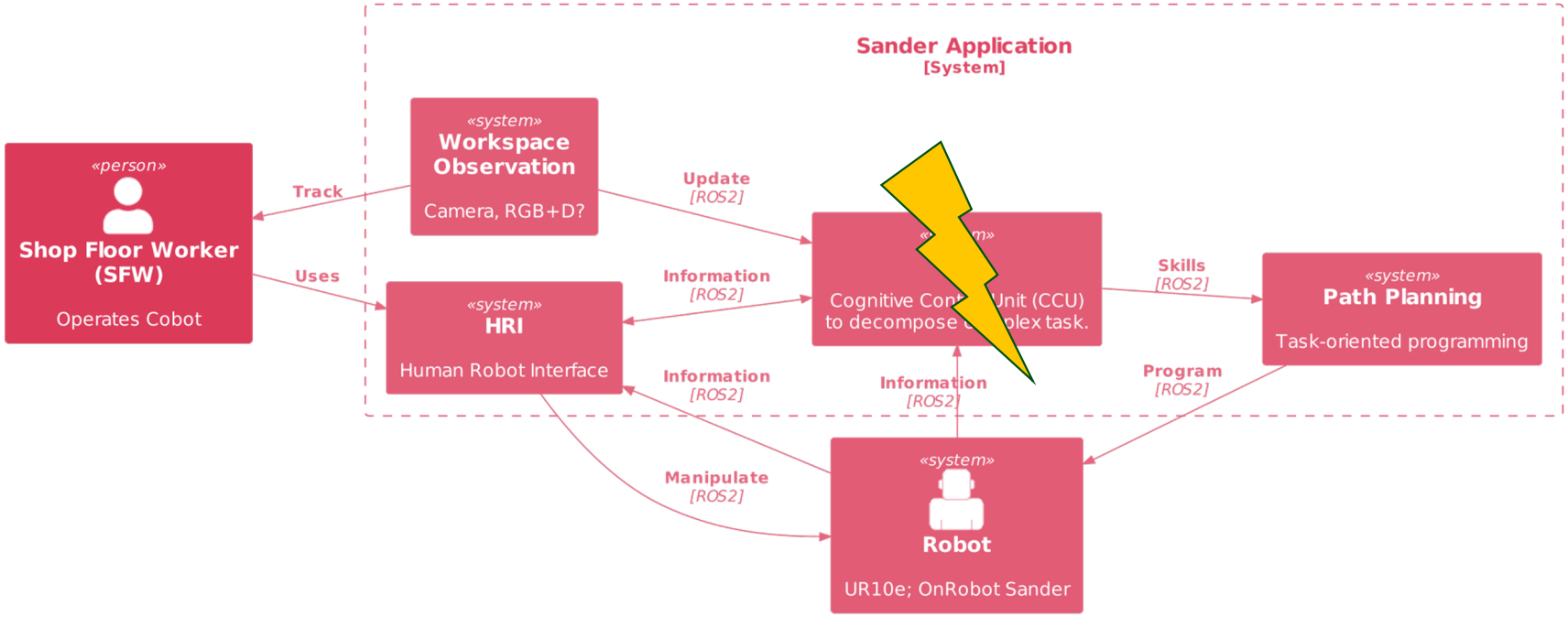
7. Specify refinishing location

8. Final quality assurance

Soar: Sander Agent



Soar: Sander Agent



SoaROS: ROS2 Interface for Soar

ROS2 Communication applied to Soar

- Defined via **message type** (topic, service, action)**, name** (topic) and quality of service (**QoS**)
- Topics:
  - Publisher: Soar output
  - Subscriber: Soar input
- Services
  - Service: Soar input → process → Soar output
  - Client: Soar output → wait for answer → Soar input
- Actions:
  - Action Server: Not implemented
  - Action Client: Not implemented

SoaROS: ROS2 Interface for Soar

Challenges

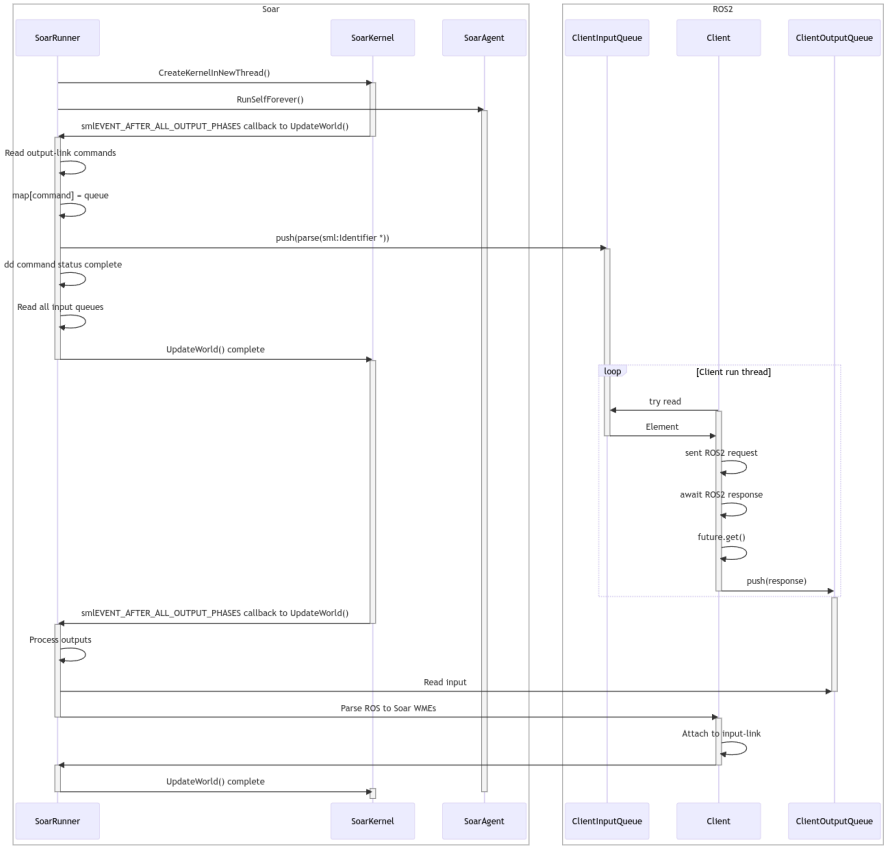
- Time of ROS2 message != Soar input phase
- Soar synchronous architecture blocks ROS
- Fixed callback interfaces for ROS and Soar
- Soar and ROS tooling incompatibilities (wait for service)
- No fork of Soar
- Run Soar Agents during ROS2 time consuming tasks

Requirements

- Multithreading, Thread safe queues
- Soar Kernel runs continuously
- Add ROS2 Interfaces via builder pattern
- Parsing of Soar WMEs and ROS2 messages by Developer
- Debug
  - Support Soar Java Debugger
  - Hook into VS Code ROS debug tooling
  - Stop kernel via ROS messages
- Full ROS Tooling integration (logging, debug)

SoaROS: ROS2 Interface for Soar

Client Example



SoaROS: ROS2 Interface for Soar

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::Publisher<std_msgs::msg::String>> p = std::make_shared<TestOutput>(
    node.get()->getAgent(), node, "test");
  node->addPublisher(p);

  std::shared_ptr<SoaROS::Subscriber<std_msgs::msg::String>> s = std::make_shared<TestInput>(
    node.get()->getAgent(), node, "testInput");
  node->addSubscriber(s);

  std::shared_ptr<SoaROS::Subscriber<std_msgs::msg::String>> sattic = std::make_shared<Trigger>(
    node.get()->getAgent(), node, "Trigger");
  node->addSubscriber(sattic);

  std::shared_ptr<SoaROS::Service<example_interfaces::srv::AddTwoInts>> service =
    std::make_shared<TestService>(node.get()->getAgent(), node, "AddTwoInts");
  node->addService(service);

  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;
}
```

SoaROS: ROS2 Interface for Soar

Conclusion

- Threading managed in the background. Not exposed to API.
- ROS ←→ Soar parsing is the only required implementation
- High Code reusability due to templates and generics
- Looking for code review from Soar Group
- Pain points/ Blockers
  - Installation of Soar manually required
- Release scheduled for Q3/2024

Technische Hochschule Augsburg

Moritz Schmidt

Faculty of Electrical Engineering
Research Assistant

Augsburg Technical University of Applied Sciences
An der Hochschule 1
D-86161 Augsburg

moritz.schmidt@tha.de
www.tha.de

THA Research Group

LinkedIn