1. ScalABLE4.0 Project Overview;
   1.1 Project Objective;
   1.2 Partners and Use-cases;
   1.3 Project Work Packages;
   1.4 Use-case Videos;

2. Open Scalable Production System (OSPS);
   2.1 Advanced Plant Model (APM);
   2.2 Production Manager (PM);
   2.3 Task Manager (TM);
   2.4 A Skill-Based Programming;
   2.5 Horizontal Integration: ROS-CODESYS Bridge;

3. Application Examples;

4. APM Overview;

5. TM Overview;

6. OSPS Hands-On;

7. Skills;

8. ROS-CODESYS Bridge;

Appendix I. Task Manager Stack Installation
1.1 ScalABLE4.0 Project Overview: Project Objective

- Development and demonstration of an Open Scalable Production System (OSPS) framework enabling optimization and maintenance of production lines.

- Using:
  - Digital Representation of the industrial shop floor;
  - Cyber-Physical Systems;
  - Plug’n’Produce;
  - Simulation;
1.1 ScalABLE4.0 Project Overview: Project Objective
1.2 ScalABLE4.0 Project Overview: Partners & Use-cases

**PSA (Autonomous Mobile Platform with 2 Arms)**
- Engine Assembly Production Line:
  - Piston Insertion
  - Screwing

**Simoldes Plásticos (Mobile Platform with 1 Arm)**
- Multi-Product Assembly Line:
  - Legacy Machine / Robot interoperability
  - Packaging of Plastic Part
# 1.3 ScalABLE4.0 Project Overview: Project Work Package Structure

<table>
<thead>
<tr>
<th>WP8 Management [INSC]</th>
</tr>
</thead>
<tbody>
<tr>
<td>WP6 Simulation and Decision Support Systems [INSC]</td>
</tr>
<tr>
<td>WP5 Vertical enterprise integration [CMF]</td>
</tr>
<tr>
<td>WP4 Flexible programming for skills based robotics [AAU]</td>
</tr>
<tr>
<td>WP3 Plug-n-Produce for flexible production systems [FHG]</td>
</tr>
<tr>
<td>WP7 Knowledge transfer, dissemination and exploitation [AAU]</td>
</tr>
<tr>
<td>WP1 Ethics [INSC]</td>
</tr>
<tr>
<td>WP2 Use case definition, validation and evaluation [PSA]</td>
</tr>
</tbody>
</table>
1.4 ScalABLE4.0 Project Overview: Automated pick and pack of objects on a conveyor
1.4 ScalABLE4.0 Project Overview: Augmented reality HMI cooperative assembly operations
1.4 ScalABLE4.0 Project Overview: Dual-arm Assembly of a Piston on an Engine Block
1.4 ScalABLE4.0 Project Overview: Vertical Integration
1.4 ScalABLE4.0 Project Overview: Simoldes Test Sprints
2. Open Scalable Production System

Addressed Problem

The 4th Industrial Revolution

1st
Mechanisation, Steam and Water Power

2nd
Mass production, Assembly lines, electricity

3rd
Computer & Automation

4th
Cyber Physical Systems, networks, AI

2. Open Scalable Production System

Addressed Problem

Open Challenges and Opportunities of CPS

- Interoperability
- Flexibility
- Usability
- Reliability
- Safety
2. Open Scalable Production System

Motivation
2. Open Scalable Production System

Motivation

Complex Robotic Applications often require the **INTEGRATION** of several software modules.

The **ORCHESTRATION** of Robotic Applications is not a trivial problem, even if interfaces are well defined.

Usually, strategies rely on **PROBLEM SPECIFIC** orchestrators.

( monolithic conditional cascading structures, nested switch statements, or ad hoc task planning )

https://en.wikipedia.org/wiki/Spaghetti_code
2. Open Scalable Production System

Background
2. Open Scalable Production System

Robotics Objectives

**Skill-Based Robot Programming**
Reducing Costs of Adapting Robot Applications by promoting Reusability

**Task-Level Orchestration**
Promoting Intuitive and Flexible Robot Programming

**Vertical & Horizontal Integration**
Establishing Generic Interoperability with Manufacturing Management Systems and Industrial Equipment

COMMUNITY-SUPPORTED TOOLS & STANDARDS
2. Open Scalable Production System

Proposed Architecture

Production Manager

- Kitting, Machine Tending, ...

Manages the execution of a Production Tasks:
- Industrial Robots, AGV, Mobile Manipulators, ...

Objectives

- Task Assigner
- Task Tracker
- Navigation Manager

Production Resource #1 (ROS)
- Industrial Robot
- Task Manager → Skills → HAL

Production Resource #2 (ROS)
- AGV
- Task Manager → Skills → HAL

Production Resource #3 (ROS)
- Mobile Manipulator
- Task Manager → Skills → HAL

Logistic Equipment
- Automated Warehouse
- Automated Warehouse System
- Connectors

MES / ERP / ...
- Manufacturing Execution System
- Enterprise Resource Planning

SIM
- Simulation & Decision Support
- Connectors

Production Equipment
- 3D Printer
- 3D Printer System
- Connectors

APM
- Advanced Plant Model
- Connectors

IoT Platform
- Connectors

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Grant Agreement No 723658

ROS Industrial EU Fall’19 Workshop
2. Open Scalable Production System

Proposed Architecture

- **MES (Manufacturing Execution System)**: Defines operations & routing, task assignments to production resources.
- **APM (Advanced Plant Model)**: Defines 3D model of assembly line or logistics warehouse, 3D model of parts and finished products, robot task state machines and goals for task execution.
- **PM (Production Manager)**: Task context model, task status.
- **Task Manager**: Production schedule, task state machine goals.
- **Robot Profile**
- **Robot Heart Beep**
2.1 Open Scalable Production System

Advanced Plant Model (APM)

- Production Manager
  - Manages the execution of a Production Tasks: Industrial Robots, AGV, Mobile Manipulators, ...

- Task Assigner
- Task Tracker
- Navigation Manager

- Logistic Equipment: Automated Warehouse
  - Connectors

- Production Equipment: 3D Printer
  - Connectors

- MES / ERP / ...
  - Connectors

- SIM
  - Simulation & Decision Support
  - Connectors

- Production Equipment: Automated Warehouse System
  - Connectors

- Production Equipment: 3D Printer System
  - Connectors

- APM
  - Advanced Plant Model
  - Connectors

- Production Resource #1
  - (ROS) Industrial Robot
  - Task Manager ↔ Skills ↔ HAL

- Production Resource #2
  - (ROS) AGV
  - Task Manager ↔ Skills ↔ HAL

- Production Resource #3
  - (ROS) Mobile Manipulator
  - Task Manager ↔ Skills ↔ HAL

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Grant Agreement Nº 723658
2.1 Open Scalable Production System
Advanced Plant Model (APM)

- **Central Entity** that stores a near real-time **Digital Representation** of the ongoing state of the shop floor, in the form of **Semantic** and **Geometrical** information:
  - **Logistic Warehouse**: Racks, Boxes, Palettes, Kits, Parts
  - **Assembly Line**: WorkStation, Manufacturing Line.
  - Fixed + mobile robotic manipulators
  - Production Schedule

- **Synchronizes** a **Digital Twin** representation between multiple software modules in the system
2.1 Open Scalable Production System
Advanced Plant Model (APM)
2.1 Open Scalable Production System

Advanced Plant Model (APM)
2.1 Open Scalable Production System
Advanced Plant Model (APM)
2.1 Open Scalable Production System

Advanced Plant Model (APM)

IoT with Grafana
2.1 Open Scalable Production System
Advanced Plant Model (APM)

- Manufacturing Area Model (navigations tasks)
- Task Context Model (manufacturing tasks)
2.1 Open Scalable Production System
Advanced Plant Model (APM)

- 2D editor within the APM allows to specify which physical objects are implanted thus building the World Model of the physical area

- Simoldes use case

- PSA use case
2.2 Open Scalable Production System

Production Manager (PM)

- Task Assigner
- Task Tracker
- Navigation Manager

Production Manager

- Kitting, Machine Tending, ...
- Manages the execution of a Production Tasks: Industrial Robots, AGV, Mobile Manipulators, ...

Production Resource #1 (ROS)
- Industrial Robot
- Task Manager ↔ Skills ↔ HAL

Production Resource #2 (ROS)
- AGV
- Task Manager ↔ Skills ↔ HAL

Production Resource #3 (ROS)
- Mobile Manipulator
- Task Manager ↔ Skills ↔ HAL

MES / ERP / ...
- Manufacturing Execution System
- Enterprise Resource Planning

SIM
- Simulation & Decision Support

IoT Platform

Logistic Equipment
- Automated Warehouse
- Automated Warehouse System

APM
- Advanced Plant Model

Production Equipment
- 3D Printer
- 3D Printer System

MES / ERP / ...
- Manufacturing Execution System
- Enterprise Resource Planning

SIM
- Simulation & Decision Support

IoT Platform
2.2 Open Scalable Production System

Production Manager (PM)

- Responsible for managing a set of Production Resources in a Production Environment.

- Issues & Controls the execution of production schedules defined by MES. *(Task Assigner)*

- Monitors the ongoing performance of previously issued production tasks. *(Task Tracker)*

- Can provide a set of services for aiding the execution of the issued tasks, that require a centralized approach. *(Ex.: Navigation Manager - TEA)*
## 2.2 Open Scalable Production System

**Production Manager (PM)**

![Production Manager Interface](image.png)

### Physical Area

<table>
<thead>
<tr>
<th>Physical Area</th>
<th>Start Date</th>
<th>End Date</th>
<th>Released on</th>
</tr>
</thead>
<tbody>
<tr>
<td>IILAB-fasten</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Production Order

<table>
<thead>
<tr>
<th>Production Order</th>
<th>#</th>
<th>Start Date</th>
<th>End Date</th>
<th>Due Date</th>
<th>Final Product</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMB_KITTING_0001</td>
<td>1</td>
<td>11:00:00</td>
<td>11:00:00</td>
<td>11:00:00</td>
<td>EMB TargElKit 3 cells</td>
<td></td>
</tr>
<tr>
<td>single_side Bracket</td>
<td>Execution</td>
<td>11:00:00</td>
<td>11:00:00</td>
<td>14:27:00</td>
<td>14:29:14</td>
<td></td>
</tr>
<tr>
<td>Drive_Warehouse</td>
<td>Finished</td>
<td>11:00:00</td>
<td>11:00:00</td>
<td>14:27:00</td>
<td>14:28:25</td>
<td></td>
</tr>
<tr>
<td>Drive_Rack1</td>
<td>Finished</td>
<td>11:00:00</td>
<td>11:00:00</td>
<td>14:28:25</td>
<td>14:28:27</td>
<td></td>
</tr>
<tr>
<td>PickAndPlace_single_side Bracket</td>
<td>Execution</td>
<td>11:00:00</td>
<td>11:00:00</td>
<td>14:28:39</td>
<td>14:29:14</td>
<td></td>
</tr>
<tr>
<td>reinforced Bracket</td>
<td>Execution</td>
<td>11:00:00</td>
<td>11:00:00</td>
<td>14:29:27</td>
<td>14:29:59</td>
<td></td>
</tr>
<tr>
<td>PickAndPlace_reinforced Bracket</td>
<td>Execution</td>
<td>11:00:00</td>
<td>11:00:00</td>
<td>14:29:27</td>
<td>14:29:59</td>
<td></td>
</tr>
<tr>
<td>double_side Bracket</td>
<td>Execution</td>
<td>11:00:00</td>
<td>11:00:00</td>
<td>14:30:00</td>
<td>14:30:54</td>
<td></td>
</tr>
<tr>
<td>Drive_Rack2</td>
<td>Execution</td>
<td>11:00:00</td>
<td>11:00:00</td>
<td>14:30:00</td>
<td>14:30:09</td>
<td></td>
</tr>
</tbody>
</table>

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Grant Agreement No. 723658.
2.3 Open Scalable Production System

Task Manager (TM)

Production Manager

Kitting, Machine Tending, ...

Manages the execution of a Production Tasks:
Industrial Robots, AGV, Mobile Manipulators, ...

Task Assigner
Task Tracker
Navigation Manager

Production Resource #1 (ROS)
Industrial Robot

Production Resource #2 (ROS)
AGV

Production Resource #3 (ROS)
Mobile Manipulator

MES / ERP / ...
Manufacturing Execution System
Enterprise Resource Planning

SIM
Simulation & Decision Support

Logistic Equipment
Automated Warehouse

Automated Warehouse System

APM
Advanced Plant Model

Production Equipment
3D Printer

3D Printer System

Connectors

IoT Platform

ROS Industrial EU Fall’19 Workshop
2.3 Open Scalable Production System
Task Manager (TM)

- Central Module running onboard of the Robot.
- Provides Integration Mechanisms between the Robot, APM & PM.
- Orchestrates production tasks in the form of sets of robotic Skills.
- Task Scripting approach based on Hierarchical & Concurrent State Machines. (ROS SMACH)
- Supports Task Scripting based on SCXML files.
2.3 Open Scalable Production System
Task Manager (TM)

Nominal (iterative) Behaviour

Cycle Control Mechanism

Fault Tolerance Mechanism
SkiROS

Software platform for the coordination industrial robots

Main features

- Skill-based robot control architecture
- Behavior trees execution system, for reactive behavior in dynamic environments
- Hardware-abstracted task description
- Semantic database server
- Integrated with PDDL task planner
SkiROS in Scalable

High-level
Integration and synchronization with PM/APM

SkiROS
Task design and execution on robot

Low-level
Integration of hardware through HAL
2.4 Open Scalable Production System

Skill-Based Programming

- **MES / ERP / ...**
  Manufacturing Execution System
  Enterprise Resource Planning

- **SIM**
  Simulation & Decision Support

- **APM**
  Advanced Plant Model

- **Production Manager**
  Kitting, Machine Tending, ...
  Manages the execution of a Production Tasks:
  Industrial Robots, AGV, Mobile Manipulators, ...

- **Task Assigner**
- **Task Tracker**
- **Navigation Manager**

- **Production Resource #1**
  Industrial Robot (ROS)
  Task Manager
  Skills
  HAL

- **Production Resource #2**
  AGV (ROS)
  Task Manager
  Skills
  HAL

- **Production Resource #3**
  Mobile Manipulator (ROS)
  Task Manager
  Skills
  HAL
• Meant to be **Hardware Agnostic**.

• Should be **Reusable** by different platforms, for **different tasks**, and in **different environments**.

• Built on top of **ROS Actions**.

• Each **Skill** is constructed as a **ROS Action Server**.

• **TM** implements the **ROS Action Client**.
2.5 Open Scalable Production System
Horizontal Integration ROS-CODESYS Bridge - Concept

Production Manager
Kitting, Machine Tending, ...
Manages the execution of a Production Tasks:
Industrial Robots, AGV, Mobile Manipulators, ...

MES / ERP / ... Manufacturing Execution System Enterprise Resource Planning
SIM Simulation & Decision Support
APM Advanced Plant Model
Production Equipment 3D Printer
Logistic Equipment Automated Warehouse

Production Resource #1 (ROS)
Industrial Robot
Task ManagerSkillsHAL

Production Resource #2 (ROS)
AGV
Task ManagerSkillsHAL

Production Resource #3 (ROS)
Mobile Manipulator
Task ManagerSkillsHAL

MES / ERP / ... Manufacturing Execution System Enterprise Resource Planning
SIM Simulation & Decision Support
APM Advanced Plant Model
Production Equipment 3D Printer
Logistic Equipment Automated Warehouse

Production Manager
Kitting, Machine Tending, ...
Manages the execution of a Production Tasks:
Industrial Robots, AGV, Mobile Manipulators, ...

MES / ERP / ... Manufacturing Execution System Enterprise Resource Planning
SIM Simulation & Decision Support
APM Advanced Plant Model
Production Equipment 3D Printer
Logistic Equipment Automated Warehouse

Production Resource #1 (ROS)
Industrial Robot
Task ManagerSkillsHAL

Production Resource #2 (ROS)
AGV
Task ManagerSkillsHAL

Production Resource #3 (ROS)
Mobile Manipulator
Task ManagerSkillsHAL

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Grant Agreement No 723658
2.5 Open Scalable Production System
Horizontal Integration: ROS-CODESYS Bridge - Concept

• **Problems:**

  • Time consumed developing and maintaining drivers for industrial communication protocols and actuators;
  
  • Inability for automation technicians to program complex robotic systems.

• **Solution:**

  • Shared memory interface between ROS and CODESYS softPLCs.
• Shared memory is the fastest way to pass data between two processes;
• Semaphores can be used as synchronization mechanism;
• CODESYS provides libraries to handle shared memory and semaphores.
2.5 Open Scalable Production System
Horizontal Integration: ROS-CODESYS Bridge - System Architecture

ROS Node
- ROS Messages
- CODESYS Compatible Structures

ROS Nodes
- ROS topics

Semaphores
- write()
- read()

Shared Memory

CODESYS Application
- write()
- read()

Industrial Equipment
- industrial protocols
2.5 Open Scalable Production System
Horizontal Integration: ROS-CODESYS Bridge - ROS Messages and IEC 61131-3 Data Types

<table>
<thead>
<tr>
<th>Description</th>
<th>ROS Messages Primitive Type</th>
<th>C++</th>
<th>IEC 61131-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsigned 8-bit Integer</td>
<td><code>bool</code></td>
<td><code>uint8_t</code></td>
<td><code>USINT</code></td>
</tr>
<tr>
<td>Signed 8-bit Integer</td>
<td><code>int8</code></td>
<td><code>int8_t</code></td>
<td><code>SINT</code></td>
</tr>
<tr>
<td>Unsigned 8-bit Integer</td>
<td><code>uint8</code></td>
<td><code>uint8_t</code></td>
<td><code>USINT</code></td>
</tr>
<tr>
<td>Signed 16-bit Integer</td>
<td><code>int16</code></td>
<td><code>int16_t</code></td>
<td><code>INT</code></td>
</tr>
<tr>
<td>Unsigned 16-bit Integer</td>
<td><code>uint16</code></td>
<td><code>uint16_t</code></td>
<td><code>UDINT</code></td>
</tr>
<tr>
<td>Signed 32-bit Integer</td>
<td><code>int32</code></td>
<td><code>int32_t</code></td>
<td><code>DINT</code></td>
</tr>
<tr>
<td>Unsigned 32-bit Integer</td>
<td><code>uint32</code></td>
<td><code>uint32_t</code></td>
<td><code>UDINT</code></td>
</tr>
<tr>
<td>Signed 64-bit Integer</td>
<td><code>int64</code></td>
<td><code>int64_t</code></td>
<td><code>LINT</code></td>
</tr>
<tr>
<td>Unsigned 64-bit Integer</td>
<td><code>uint64</code></td>
<td><code>uint64_t</code></td>
<td><code>ULINT</code></td>
</tr>
<tr>
<td>32-bit IEEE Float</td>
<td><code>float32</code></td>
<td><code>float</code></td>
<td><code>REAL</code></td>
</tr>
<tr>
<td>64-bit IEEE Float</td>
<td><code>float64</code></td>
<td><code>double</code></td>
<td><code>LREAL</code></td>
</tr>
<tr>
<td>ASCII String</td>
<td><code>string</code></td>
<td><code>std::string</code></td>
<td><code>STRING</code></td>
</tr>
<tr>
<td>Time (secs/nsecs)</td>
<td><code>time</code></td>
<td><code>ros::Time</code></td>
<td><code>TIME</code></td>
</tr>
<tr>
<td>Time (secs/nsecs)</td>
<td><code>duration</code></td>
<td><code>ros::Duration</code></td>
<td><code>TIME</code></td>
</tr>
</tbody>
</table>

Supported by current implementation

Converted to a fixed length array of char on ROS implementation

ROS Industrial EU Fall’19 Workshop
2.5 Open Scalable Production System
Horizontal Integration: ROS-CODESYS Bridge - Real World Applications
2.5 Open Scalable Production System
Horizontal Integration: ROS-CODESYS Bridge - Ongoing and future developments

• Ongoing:
  • Public release in the scope of the ROSIN project;
  • Support for more data types and custom data structures.

• Future:
  • Standard interfaces for commonly used components;
  • Easier reconfiguration of mapped variables;
  • Support for ROS services and actions.
3. Application Examples
Demonstration with a mobile robot

Drive Skill → Dock Skill → Actuate Hook Skill → Drive Skill → Drive Skill → Undock Skill → Actuate Hook Skill → Drive Skill

- **Move Robot closer to Object**
- **Find Object and Approximate**
- **Move Hook Down**
- **Move Robot to Undock Position**
- **Find Undock Zone and Approximate**
- **Move Hook Up**
- **Move Robot to New Task**
3. Application Examples
Demonstration with a collaborative robot

**Pick Skill**
- Move Arm Skill
  - Approximate Gripper to Object
- Actuate Gripper Skill
  - Close Gripper
- Move Arm Skill
  - Move Manipulator to rest position

**Place Skill**
- Move Arm Skill
  - Approximate Object to the Table
- Actuate Gripper Skill
  - Open Gripper
- Move Arm Skill
  - Move Manipulator to rest position
3. Application Examples
Demonstration with a mobile manipulator

- **Drive Skill**
  - Move Mobile Manipulator to Workstation 1

- **Move Arm Skill**
  - Approximate Gripper to Object

- **Actuate Gripper Skill**
  - Close Gripper

- **Move Arm Skill**
  - Move Manipulator to rest position

- **Drive Skill**
  - Move Mobile Manipulator to Workstation 2

- **Move Arm Skill**
  - Approximate Object to the Table

- **Actuate Gripper Skill**
  - Open Gripper

- **Move Arm Skill**
  - Move Manipulator to rest position
3. Application Examples

H2020 ColRobot Demonstration
3. Application Examples
H2020 ScalABLE4.0 Preliminary Demonstration
3. Application Examples
H2020 FASTEN Preliminary Demonstration
4. 
OSPS - Advanced Plant Model & Production Manager
Advanced Plant Model (APM) & Production Manager (PM)

Advanced Plant Model (APM)

Configuration of an Assembly Line
5. OSPS - Task Manager
5.1. Features Overview and Compatibility;

5.2. APM Interface;

5.3. Task Manager:
   5.3.1 System Architecture;
   5.3.2 Sequence Diagram;
   5.3.3 Component Diagram;
   5.3.4 Skill-Based Programming;
5.1. Features Overview and Compatibility

- Python based;
- Fully compatible with OSPS APM & OSPS PM;
- Communication API for ROS, Web Services, FIWARE (NGSI), APACHE (MQTT) and Manufacturing Service Bus (MSB);
- Support for ROS Action Protocol:
  - Supports Custom Actions (skills) (drive, pick, place, dock, move hook, ...);
  - Interfaces for easy configuration of new Actions (skills);
- Hierarchical State Machines powered by SMACH.
- Can be modified to work as a standalone Python library, i.e. being ROS Agnostic.
- Test Driven Development:
  - Unit, Integration, and System Tests
- Continuous Integration
5.1. Features Overview and Compatibility

- Fully compatible with APM and World Model through APM Interface.
5.2. APM Interface

Node Tree

- The APM Interface allows for the update and retrieval of data concerning the environment in which a task is executed;

- The context model for the task being executed is represented as a tree of nodes

<table>
<thead>
<tr>
<th>Node Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Node's unique identifier</td>
</tr>
<tr>
<td>parent_id</td>
<td>Parent node’s unique identifier</td>
</tr>
<tr>
<td>type</td>
<td>&quot;What does the node represent?&quot;</td>
</tr>
<tr>
<td>friendly_name</td>
<td>A simpler and generic identifier</td>
</tr>
<tr>
<td>bounding_volume</td>
<td>The object dimensions</td>
</tr>
<tr>
<td>properties</td>
<td>Metadata</td>
</tr>
<tr>
<td>transform</td>
<td>TF between node and its father</td>
</tr>
<tr>
<td>children</td>
<td>Identifiers of children nodes</td>
</tr>
</tbody>
</table>
Inside a node's 'properties' field one can store additional information regarding the node:

- A physical object might have associated grasping poses;
- A cell in a rack might require a status occupied/unoccupied flag;

Each property must have the following structure:

- **Key**: an identifier for the property
- **Data Type**: the type of data being stored
- **Data Value**: the property value

For convenience, some API calls allow for the direct retrieval of certain data typed properties.
5.2. APM Interface

API Calls

• Under the hood API calls are translated into a ROS service calls. The tree of nodes is consulted at which call in order to retrieve the desired data.

• The APM Interface provides the following API calls:

<table>
<thead>
<tr>
<th>Services provided by the APM Interface API</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_model(url, path)</code></td>
</tr>
<tr>
<td><code>get_nodes_with_friendly_name(friendly_name)</code></td>
</tr>
<tr>
<td><code>get_node_bounding_volume(node_id)</code></td>
</tr>
<tr>
<td><code>get_nodes()</code></td>
</tr>
<tr>
<td><code>get_node_grasping_poses(node_id)</code></td>
</tr>
<tr>
<td><code>sync_context_model()</code></td>
</tr>
<tr>
<td><code>get_node_information(node_id)</code></td>
</tr>
<tr>
<td><code>update_node_bounding_volume(node_id, volume)</code></td>
</tr>
<tr>
<td><code>get_node_models(node_id)</code></td>
</tr>
<tr>
<td><code>update_node_father(node_id, father_id)</code></td>
</tr>
<tr>
<td><code>get_node_properties(node_id)</code></td>
</tr>
<tr>
<td><code>update_node_properties(node_id, properties)</code></td>
</tr>
<tr>
<td><code>get_node_types()</code></td>
</tr>
<tr>
<td><code>get_node_vertex(node_id)</code></td>
</tr>
</tbody>
</table>
The APM Interface node also provides a mechanism to inform the APM of changes in the task context model.

- **update_node_bounding_volume**: to use whenever a pose of a physical object changes;
- **update_node_father**: to use whenever an object is inserted or removed from within another object;
- **update_node_properties**: to set or update node's metadata.

To send the updated task context model to the APM is required to call **sync_context_model**. All changes are sent at once.
5.3.1 Task Manager
System Architecture

- **Task Manager Server**
  - Receives/processes requests from PM to assign/execute Tasks;
  - SCXML preprocessing, SCXML parsing and SMACH conversion;
  - Orchestrates task execution.
5.3.1 Task Manager

System Architecture

- **Task Manager APM Interface**
  - Allows the query of data regarding the task context model of the task being executed
5.3.1 Task Manager

System Architecture

- Task Manager Heart Beep
  - Periodically provides information about the robot.

- Robot Id
- Robot Type
- Status
- Manufacturing Area
- Bounding volume
- Properties
• Task Manager Robot Map
  • Gets information about the robot map.
5.3.1 Task Manager

System Architecture

- Task Manager Robot Profile
  - Periodically provides information about the robot
    - Robot Id
    - Robot Type
    - Manufacturing Area
    - Skills
    - Properties
ASSIGNING A TASK:

• PM sends request for assigning Task (PMAssignTaskListReq);

• TM adds processed task to internal DB if valid;

• TM responds to PM with the assigned Tasks (TMAssignTaskListResp).
EXECUTING A TASK:

• PM sends request for executing Task (PMExecuteTaskReq);

• If Task still not in DB, repeats assigning process;

• TM reallocates Task to TaskQueue and starts execution;

• TM responds to PM with the Task that will be executed (TMExecuteTaskResp).
5.3.3 Task Manager

- Task Manager Server launch file
  - Receives Robot and TM configuration

- Task Manager Server Executable

- Task Manager Class
  - Waiting a request from PM
5.3.3 Task Manager
Component Diagram

PM Assigns/Executed Task
5.3.3 Task Manager
Component Diagram

- Task Manager receives request
- Asks APM for the Task Context Model
- SCXML preprocessing, SCXML parsing and SMACH conversion;
- Skills (SMACH) sent to execution module
Configuration of the Skill

When Skill's action client is invoked:

- Action Goal is constructed
- Action Name is constructed
- Action Type is constructed
5.3.3 Task Manager
Component Diagram

- Execution of the Skill
5.3.4 Task Manager
Skill-Based Programming

- The TM acts as the **Action Client** for the existing skills, sending the *goal* and receiving feedback from each specific **Action Server** (Skill);

- Each Skill has its own Action Client that inherits from TM, allowing method overloading.

*Parallelism between ROS Actions and Skills: .action file as the definition of the Skill*
• The addition of new Skills to the system is easy;
• Create them with the help of the Skill Generator Tool;
• Using a yaml configuration file this tool will create the desired Skill.
6. OSPS - Hands on

6.1. Task Manager
6.2. Skill Generator
6.3. Advanced Plant Model
6.4. Bridge ROS-CODESYS
7.
OSPS - Skills
7. OSPS - Skills: Summary

7.1. Skill Generation;

7.2. Skill Implementation;
   7.2.1. Server
   7.2.2. Client

7.3. Task Creation;

7.4. Task Execution;

7.5. Task Examples.
7.1. Skill Generation

- To create a skill execute the following command:
- The input of this script is a yaml configuration file;

Yaml file for wait skill

Yaml file for random_outcome skill

Yaml file for test skill (generic skill)

Skill with outcomes besides succeeded, preempted and aborted

Mandatory Fields

Optional Fields

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Grant Agreement Nº 723658
7.1. Skill Generation

- To create a skill, simply execute the following command:

Example: Creating a wait skill with the skill generator.
7.1. Skill Generation

- The script generates required folders and files; relocates the `yaml` file to the skill folder.

Wait skill `scxml` visualized on Qt Creator (SCXML editor)
7.2.1 Skill Implementation

Server

```python
import rospy
import actionlib
from wait_skill_msgs.msg import WaitSkillAction, WaitSkillResult, WaitSkillFeedback

class WaitSkill(object):
    def __init__(self, action_name='WaitSkill'):
        ...

    def execute_skill(self, goal):
        ...
        The execution of the skill should be coded here. In order to save you time,
        the methods check_preemption(), feedback(), success() and aborted() should be used.
        The check_preemption() method should be called periodically.
        The variable self.percentage should be updated when there is an evolution
        in the execution of the skill.
        The feedback() method should be called when there is an evolution in
        the execution of the skill.
        ...

    def feedback(self, status=None):
        ...

    def success(self, status=None, outcome='succeeded'):
        ...

    def aborted(self, status=None, outcome='aborted'):
        ...

    def check_preemption(self):
        ...

    def result_constructor(self, status, percentage=None, outcome=None):
        ...

@staticmethod
def log_info(status):
```

**Wait skill server generated by the Skill Generator (wait_skill_class.py)**

```python
import rospy
import actionlib
from wait_skill_msgs.msg import WaitSkillAction, WaitSkillResult, WaitSkillFeedback

class WaitSkill(object):
    def __init__(self, action_name='WaitSkill'):
        ...

    def elapsed_time(self):
        ...

    def execute_skill(self, goal):
        ...
        # Sets starting time as current time
        self.start_time = time.time()  # While not preempted
        # If elapsed_time() < goal.waitTime:
        if self.elapsed_time() < goal.waitTime:
            # Waits until the time in goal passes
            skill_status = 'Elapsed Time: ' + str(round(self.elapsed_time())) + 's. Remaining Time: ' + str(round(goal.waitTime - self.elapsed_time())) + 's'  # Skill Status
            self.feedback(skillStatus)  # Skill feedback
            # Defining percentage of the skill done
            self.percentage = int(round(self.elapsed_time() / goal.waitTime * 100))
            time.sleep(1.0)
        else:
            # If skill terminates normally sets success and breaks loop.
            self.success('Waited Successfully ' + str(goal.waitTime) + 's')
            break

    # Continues with same methods as the generated server. Methods feedback(),
    # success(), aborted(), check_preemption(), result_constructor() and log_info()
```

**Implementation of the Wait skill server (wait_skill_class.py)**
7.2.2 Skill Implementation

Client

```
import sys
import rospkg

from task_manager_common.skill_class import SkillSetup, SkillExecution, SkillAnalysis

class WaitSkillSetup(SkillSetup):
    pass

class WaitSkillExecution(SkillExecution):
    pass

class WaitSkillAnalysis(SkillAnalysis):
    pass
```

*Wait skill client generated by the Skill Generator (wait_skill_class.py)*
• Implementing changes example in the Client:

```python
class UsescoreSkillSetup(SkillSetup):
    # Overloads Default Client Method present in TM
    def action_goal_constructor(self, goal, ud):
        """""" Gets result from previously executed skill
        """

        # Gets previous result for TestscoreSkill from userdata
        result_test_score = ud.previousSkillsResults['TestscoreSkill']

        # Sets score goal getting the score attribute from previous result
        ud.actionGoal['score'] = getattr(result_test_score, 'score')

        return SkillSetup.action_goal_constructor(self, goal, ud)
```

Overloading Skill Client – action_goal_constructor()

In this example we can retrieve a result from a Previously executed skill.
7.2.2 Skill Implementation
Client

• Implementing changes example in the Client:

```python
class MoveArmSkillAnalysis(SkillAnalysis):
    # ... #

def update_apm(self, userdata):
    if 'objectId' in userdata.actionGoalDict:
        objectId = str(userdata.actionGoalDict['objectId'])
        if objectId and 'partId' in userdata.actionGoalDict:
            nodeId = str(userdata.actionGoalDict['partId'])
            apm.update_node_father(nodeId, objectId)
            node_bw = apm.get_node_bounding_volume(nodeId)  # obtains bounding volume
            target_bw = apm.get_node_bounding_volume(objectId)  # obtains target bounding volume
            apm.update_node_bounding_volume(nodeId, node_bw)
            apm.update_node_bounding_volume(objectId, target_bw)
            apm.sync_context_mode('UPDATE')  # APM Updated
    # ... #
```

Overloading Skill Client – update_apm()
In this example we can update a node and its bounding volume.
7.3. Task Creation

- A **task** is a **scxml** file and can be **created on Qt Creator** ([https://www.qt.io/](https://www.qt.io/)), following the next steps:

  1. Open *Qt Creator* and **create a new scxml file** under the `task_manager_pseudo_pm/resources`;
  2. Add an initial state;
  3. Import the generated scxml of the desired skill state;
  4. Copy the desired instances of the skill state to the **task scxml**;
  5. Connect the states using the appropriate transitions;
  6. If necessary change the default values to the goals of the skill;
  7. Save the task **scxml** file.
7.3. Task Creation

- A task is a *scxml* file and can be created on Qt Creator ([https://www.qt.io/](https://www.qt.io/)), following the next steps:

*Example: Creating a task with two wait skills.*
7.3. Task Creation

Task with wait skills developed in Qt Creator

Definition of the WaitSkill1 goal
7.4. Task Execution

- **Edit** the `task_manager_pseudo_pm` launch file:
  - Link to the desired task (`scxml` file);
  - You may change: `recipient_id` (robot_id), `task_id`, `priority`;
  - You may choose to **assign a task** or to **execute** it:
    - Assign task: sends SCXML but doesn’t execute;
    - Execute: requests execution (sending or not the SCXML file).

- **Run** the servers of the skills contained in the task:
  - Run `wait_skill_server` launch file.

- **Run** `task_manager` launch file;

- **Run** `task_manager_pseudo_pm` launch file.

- Optional: Run `qt_smach_viewer` for visualization
7.4. Task Execution

Example: Running the task and visualizing with qt_smach_viewer
With `qt_smach_viewer` it is possible to see the task's evolution with multiple depth levels;

The **executing skill** is marked as **green** and the **final states** as **red**.
7.5. Task Examples
Task containing skills with non-default outcome

• This task uses the **Wait Skill** and the **Random Outcome Skill**;

• The **Random Outcome Skill** chooses a random outcome and delays the success for a given delayTime.

• This example explores the use of outcomes besides **succeeded**, **aborted** and **preempted**.

---

**Task in Qt Creator** (available in the resources folder of the pseudo_pm package)
## 7.5. Task Examples

Task containing skills with non-default outcome

---

**Example:** Running the task and visualizing with `qt_smach_viewer`
7.5. Task Examples
Task containing macro skill (set of skills) and parallel states

- This task uses the **Wait Skill**, and **Outcome Skill**;

- The **Outcome Skill** accordingly to the outcome set on `outcomeType` ('A','B' or 'C'), **delays the success for that outcome for a given delayTime**.

- Explores the **use of macro skills** (skill containing more than one skill) and **parallel states** (several skills running simultaneously).
7.5. Task Examples
Task containing macro skill (set of skills) and parallel states

Example: Running the task and visualizing with qt_smach_viewer
8.

OSPS - ROS-CODESYS Bridge
8.1. ROS-CODESYS Bridge Usage
8.1. ROS-CODESYS Bridge Usage
Main ROS class

- Topic-based implementation;
- Shared memory written automatically on subscriber callback;
- Shared memory read periodically and published to topic.

```cpp
class Robin {
  std::string name_;  
  Semaphore semaphore_;  
  SharedMemory shared_memory_;  
  ros::NodeHandle nh_;  
  ros::Publisher pub_;  
  ros::Subscriber sub_;  
  std_msgs::Bool msg_;  
  const uint32_t queue_size_ = 100;  
  const bool latch_ = true;
  void write(const std_msgs::Bool::ConstPtr & msg);
public:
  Robin(std::string name, bool mode=READ, bool open=true);
  bool isOpen();
  bool isClosed();
  void read();
  void open(bool mode=READ);
  void close();
  ~Robin();
};
```

Public interface

- Executed internally
- Executed externally
8.1. ROS-CODESYS Bridge Usage

Example ROS node

```c
#include "robin/robin.h"
#include <ros/ros.h>
int main(int argc, char **argv)
{
    ros::init(argc, argv, "robin");  // Object creation
    Robin move_conveyor("move_conveyor", WRITE);
    Robin wait_conveyor("conveyor_finished", READ);
    ros::Rate read_rate(10);
    while (ros::ok())
    {
        wait_conveyor.read();
        ros::spinOnce();
        read_rate.sleep();
    }
    return 0;
}
```

Must be executed manually when reading

Periodic shared memory reading
8.1. ROS-CODESYS Bridge Usage

Example CODESYS project

**Project structure**

**Main program**

```
MAIN
1 PROGRAM MAIN
2 VAR_INPUT
3 msgFromRos : BOOL;
4 END_VAR
5 VAR_OUTPUT
6 msgToRos : BOOL;
7 END_VAR
8
9 CONVEYOR_PRG.on := msgFromRos;
10 msgToRos := CONVEYOR_PRG.finished;
```

**Variable mapping program**

```
ROBIN_PRG
1 PROGRAM ROBIN_PRG
2 VAR
3 msgFromRos : Robin('move_conveyor', RobinConstants.READ);
4 msgToRos : Robin('conveyor_finished', RobinConstants.WRITE);
5 END_VAR
6
7 // read
8 MAIN.msgFromRos := msgFromRos.read();
9 // write
10 msgToRos.write(MAIN.msgToRos);
```
8.1. ROS-CODESYS Bridge Usage

Demo

![Diagram showing ROS to CODESYS interaction](image-url)

- move_conveyor
- conveyor_finished

CODESYS Web Visualization
Appendix I.

OSPS - Task Manager Stack Installation
Summary

- Repository;
- Installation;
- Usage;
This guide is meant to teach how to install the Task Manager software stack and everything necessary for the Workshop.

This guide assumes that you already installed ROS Kinetic and that you have a catkin workspace prepared.

Please install Qt Creator 4.10.0 for Linux 64-bit (https://www.qt.io/offline-installers). Make sure you install the minimum required.

CLONE THE FOLLOWING REPOSITORIES:

• Task_Manager_Stack:
• Wait_Skill:
• Outcome_Skill:
• Random_Outcome_Skill:
• Conveyor_Skill:
• Update_APM_Skill:
• Skill_Generator:
Installation

1. Please clone the repositories in the previous slide to your src folder;
2. Please read all the README files carefully and make sure you install all the required dependencies;
3. Build everything
Usage

1. Please launch the Task Manager node by executing the command `roslaunch task_manager run.launch`;

2. Please launch the Wait Skill with `roslaunch wait_skill_server run.launch`;

3. Please launch the Pseudo Production Manager (simulates the task execution request) by executing the command `roslaunch task_manager_pseudo_pm run.launch scxml_file:wait.scxml`;

4. The task has only one skill that will perform a wait time of 10s and then succeed;

5. You should be able to see the feedback in the Task Manager node;

6. Your setup for the Workshop is now completed.