Recent Progress Toward Intelligent Robot Assistants for Non-Destructive Disassembly

Jan Jungbluth, Wolfgang Gerke and Peter Plapper

Abstract— **One key to successful and fluent human-robot collaboration in disassembly processes is equipping the robot systems with greater autonomy and intelligence. In this paper, we present our progress in developing such an intelligent robot assistant system. We present the multi-agent control architecture we developed and describe its technical implementation. Our control approach relies on two types of knowledge models: product and process models. The product model describes the structure of the product to be dismantled through a description of the parts involved and the connections between them. A connection type-related process model describes which agents and objects participate in the disassembly process and which methods of human-robot cooperation or collaboration are most useful. When supplied with a product model and a goal definition, our robotic assistant can automatically generate a partially ordered disassembly task sequence. For each disassembly task, the user can choose one of the divisions of labor defined in the process model. Using the process model, through perception and deliberation, the control system can both execute actions and coordinate and synchronize the actions of human and machine. This approach allows the system to be more autonomous when providing assistance to human coworkers in complex and one-piece disassembly processes.**

I. Introduction

The advent of safe, small and affordable robots in the industrial domain has led to a shift away from enclosed robot cells toward mobile robot assistants in recent years. Boosted by the vision of Industry 4.0, which considers mobile platforms and safe robot arms as the key end-of-line components within smart factories, most attention has been focused on the safety of such robotic systems. Therefore, the published standards [1] and technical specifications [2] now provide guidance. However, as increasing numbers of mobile robotic systems implemented in smart factories, they will bring new challenges. We will need to control these systems on multiple levels, including logistics, processing and assisting in production tasks. In the last case, a system will be interacting with a human coworker, which is a drastically different scenario than that found in classical automation. In classical automation, machines followed clear instructions and could be synchronized by means of efficient communication. This kind of communication is not possible between humans and machines. We are likely to encounter application areas in which we cannot provide an exact sequence of instructions to the machines due to economic factors (for example, when manufacturing small lot sizes or individual products). Other application areas, such as corrective maintenance, recycling or the remanufacturing domain, present even greater challenges for automation because the product condition can vary drastically, and fully automated disassembly processes will fail.

In these not yet automated domains, we can expect the greatest gain in synergy through human and robot cooperation or collaboration [3]. However, today's industrial robots do not have any domain knowledge of these application areas, nor do they know how to work with humans; thus, they cannot serve as valuable assistants.

In this work, we explain how we addressed these challenges. First, we introduce the reader to our developed agent-based control system architecture in Chapter II. Thereafter, we explain different topics in more detail, with the product model being discussed in Chapter III and the process model in Chapter V. Chapter IV explains how the product model is used for disassembly (task) planning and explains the algorithms that were employed. We examine task initialization in Chapter VI and describe the process state monitoring module in Chapter VII. In Chapter VIII, we address the topic of coordination through action planning, in which we determine workflow at the level of single human and machine actions. Thereafter, Chapter IX explores how we execute and synchronize the actions of human and machine. Before we conclude this work, we provide insights into our demonstration application and its technical implementation in Chapter X.

II. The Overall Architecture of the Intelligent Assistant System

In robotics research, three robot control paradigms for achieving intelligent behavior exist: reactive, deliberative and hybrid architectures [4]. Alternative approaches stem from the field of artificial intelligence, in which two well-known designs for intelligent behavior include agents [5] and cognitive systems [6]. Our control system architecture was inspired by a goal-oriented agent and was gradually extended to meet our challenges. We ended up with a multi-agent system containing two types of agents: device-controlling agents and an intelligent disassembly assistant agent (Figure 1

A. *Device-controlling Agents*

As the name suggests, a device-controlling agent (DCA) controls a device. Therefore, a DCA provides device-specific communication on one side and a unified communication interface on the other, which in our case is the Internet of Things (IoT) communication protocol Message Queuing Telemetry Transport (MQTT). Depending on the controlled device and its purpose, we perform further computation within the DCA.

Peter Plapper is with the University of Luxembourg, Luxembourg, Luxembourg (e-mail: peter.plapper@uni.lu).

Wolfgang Gerke is with the Trier University of Applied Sciences, Birkenfeld, Germany (e-mail: w.gerke@umwelt-campus.de).

Jan Jungbluth is with the company SEW EURODRIVE, Bruchsal, Germany (e-mail: jan.jungbluth.w@sew-eurodrive.de).

For example, a DCA in our application controls an electrically actuated two-finger gripper. Incoming commands from the unified communication interface are translated into device commands and executed through device-specific communication. Simultaneously, the DCA monitors the gripper status (for example, if the gripper is open, closed or in an in-between state) and publishes status changes on the unified communication interface.

In our approach, each device is integrated into our architecture through such a DCA. Within our architecture, we employ a centralized control approach to coordinate the actions of man and machine in a goal-oriented manner, but we also allow decentralized control within the agents. For example, a DCA that serves as a voice interface can send commands to both the robot and the gripper.

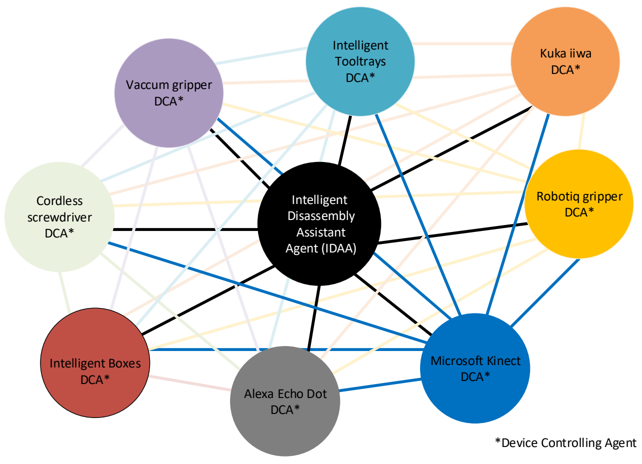

Intelligent Tooltrays DCA*
Kuka iiwa DCA*
Vaccum gripper DCA*
Cordless screwdriver DCA*
Intelligent Disassembly Assistant Agent (IDAA)
Robotiq gripper DCA*
Intelligent Boxes DCA*
Alexa Echo Dot DCA*
Microsoft Kinect DCA*
*Device Controlling Agent

Figure 1. The multi-agent system.

B. *Intelligent Disassembly Assistant Agent (IDAA)*

An IDAA serves as the centralized control in our architecture; its current shape is depicted in Figure 2. This IDAA is directly connected to the manufacturing execution system. The blue rectangle on the left side represents the developed tool chain, which supports the creation and extension of the product and process models. On the right side of the IDAA, we illustrate its interaction with its environment. In the environment, we unite the human coworker, physical objects (such as tools, boxes or parts and assemblies), and the DCAs. The individual modules that comprise the IDAA are briefly explained in the following section.

The **knowledge base** consists of two types of models: the **product model** and the **process model**. We represent the product structure by describing the kinds of parts in the assembly and the types of connections between them. The product model is created using a Siemens NX CAD (computer-aided design) plugin and exported as an Extensible Markup Language (XML) file. A **process model** contains a model of the disassembly process related to a specific connection type, such as the loosening of a screw joint or the removal of a circlip. In the process model, we declare the agents

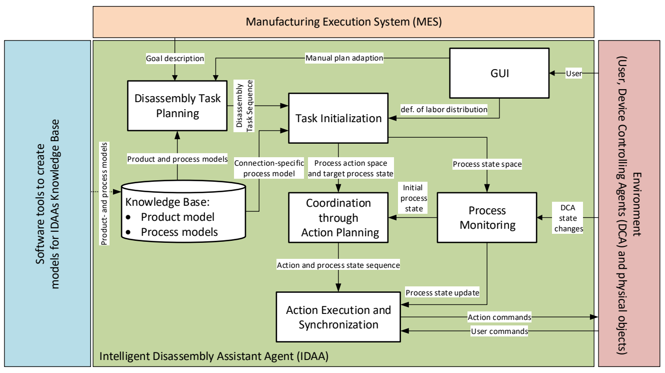

Manufacturing Execution System (MES)
Software tools to create models for IDAAs Knowledge Base
Product- and process models
Goal description
Manual plan adaption
GUI
User
Disassembly Task Planning
Disassembly Task Sequence
Task Initialization
def. of labor distribution
Product and process models
Connection-specific process model
Process action space and target process state
Process state space
Knowledge Base:
Product model
Process models
Coordination through Action Planning
Initial process state
Process Monitoring
DCA state changes
Action and process state sequence
Process state update
Action Execution and Synchronization
Action commands
User commands
Intelligent Disassembly Assistant Agent (IDAA)
Environment (User, Device Controlling Agents (DCA) and physical objects)

Figure 2. IDAA's embedding and internal architecture.

that participate in this activity and the objects involved. Furthermore, various useful distributions of labor are defined in the process model. Through the process model editor, we can also create new process models or extend existing ones. We also use XML format for importing and exporting the process model. Both models are discussed in detail in Chapters III and IV.

Depending on the loaded product model and goal definition, the **disassembly task-planning** module determines the necessary sequence of disassembly tasks. The result is a disassembly plan that consists of a sequence of steps, each of which features at least one disassembly task. In each task, we store the type of connection and the related part that needs to be removed. The steps and tasks can be manually rearranged by the user, and a suitable labor distribution for each task can be selected via the graphical user interface (GUI) to meet the user's workflow expectations. In Chapter V, we explain the module in more depth.

In the **task initialization** module, the agents and objects defined in the process model are linked with the physically occurring objects and agents associated with the task. The (dynamic) parameters defined in the process model are initialized with the values from the corresponding physical objects or agents. Therefore, the goods of production, such as manual tools, non-controllable robot effectors and transport boxes, are pre-determined. The initial and target process states are created by merging together the objects and agents' initial or target states as defined in the process model. The initial process state is transferred to the process monitoring module, and the target process state is transferred to the action planning module along with the parametrized actions from the agents.

The initial process state is updated by the **process monitoring** module to set the current process state. The current process state is then transmitted to the action planning module. The process monitoring module also updates the process state of the action execution module.

The **action planning** module examines the current process state and selects the sequence of actions needed to reach the target process state from the received action set. To determine this action sequence, we employ a search strategy that we explain in Chapter VII. The output of the action planning module is a sequence of actions that must be executed and a sequence of process states that must be met to fulfill the disassembly activity.

The **action execution and synchronization** module executes and synchronizes the determined actions by sending commands to the DCAs. After sending a command, the module waits until the predicted process state matches the current perceived process state delivered from the process monitoring module. While the sequence of predicted process states matches the states delivered by the process monitoring module, the action execution module continues action execution. If deviations are detected between predicted and perceived process states, the action planning module can be restarted to resolve the issue.

The above section provided brief descriptions of each of the IDAA's modules. Chapters III through IX describe the modules in further detail, beginning with the product model in the knowledge base.

III. The Product Model

The aim of the product model is to represent a product structure in a machine-readable way in order to allow for automated (dis)assembly planning. For this reason, a number of different representations exist in academic research. In the reviewed literature, most approaches represent product structure using undirected and directed (liaison) graphs, as well as hypergraphs [7]. Other approaches use petri nets [8], Planning Domain Definition Language (PDDL) models and object-oriented models [9]. More recent approaches are based on ontologies [10]. It is important to note at this point that knowledge representation is closely connected to knowledge processing. The reviewed literature focused on representing the product structure for automated (dis)assembly planning. In this work, we extend this approach to encompass disassembly task planning in highly flexible and collaborative human-robot workstations. We also use the product model to store necessary information concerning the disassembly process. In this way, we build a custom, object-oriented product model that can also be used as a knowledge base for the disassembly process. As noted previously, we describe different classes of parts and connections in the product model. We then create instances from both classes and define relations between the part and connection instances to represent the product structure.

A. The Part Classes

We group parts into different classes depending on the information they are required to provide in the disassembly process. For a simple part such as a cover plate, some common information about the object suffices. Examples of the type of information required include the object's grasping position and orientation with reference to the main assembly coordinate system, a unique identifier (ID) or the robot tool used to grasp the part. Other parts, which might be closely associated with a connection type (for instance, a screw with a screw connection), need to store more information. For example, to loosen a screw connection, we need information about the driver style. We therefore create a new class called "screw," derived from the part class, to hold this information. The driver style may vary for different screws in the assembly, so we store this information in each instance of a screw class. Using this approach, we can provide the information required for the disassembly process in the part instances. In some cases, we cannot strip down an assembly piece by piece because a subassembly must be removed as one. Therefore, we also need to represent subassemblies. A subassembly is initially treated like any other part in the assembly, but, once it is removed from the main assembly, it is itself treated as an assembly to be taken apart.

B. Connection Classes

As with part classes, we define connection classes for different connection types. The representation of the connections in the semantic model is a key element in process planning. A connection is defined such that loosening or establishing a connection describes a formal, representable process—that is, a process that consists of a sequence of actions that are performed by one or more agents on objects within the task. If we can determine the connections that must be removed to disassemble a product, we can coarsely determine the related process. We provide a step-by-step explanation of the disassembly process in the following chapters, but, for now, we continue with the explanation of the product model, using an example.

The object-oriented product model can be transformed into a directed graph, which is useful for further planning. In such a graph, each node represents a part instance, and all edges in the graph belong to a connection instance. An edge direction is defined from the "connection-establishing part" to the "connection-constrained part". If we consider the bolted joint of the four hex-head screws (Numbers 1.1–1.4 in Figure 3) that link the cover plate (2), the sealing (3) and the drive body (15), the four screws are all instances of the "screw" part class, while the sealing, the cover plate and the drive body are instances of the "component" part class. The bolted joint is symbolically described by only one instance from the "screw joint" connection class because the screws are identical, and they link the same parts. In the screw connection instance, we define the screws as the "connection-establishing part" and the other components as constrained parts. This screw connection is represented in Figure 3 with sixteen edges, each screw has an edge to any of the other three connected parts. The next connection we need to specify is the cover plate, which lies on the sealing. We represent it using an instance of the "lyingOn" connection class. This connection is represented by one edge from the cover plate to the sealing. An instance of the "covered" connection class is used to describe a situation in which one part prevents access to another part without any physical interaction between them. This case applies to the cover plate and the cylinder head screw (5). The "covered" instance is also illustrated on the graph in Figure 4 by an edge pointing from the cover plate to the screw.

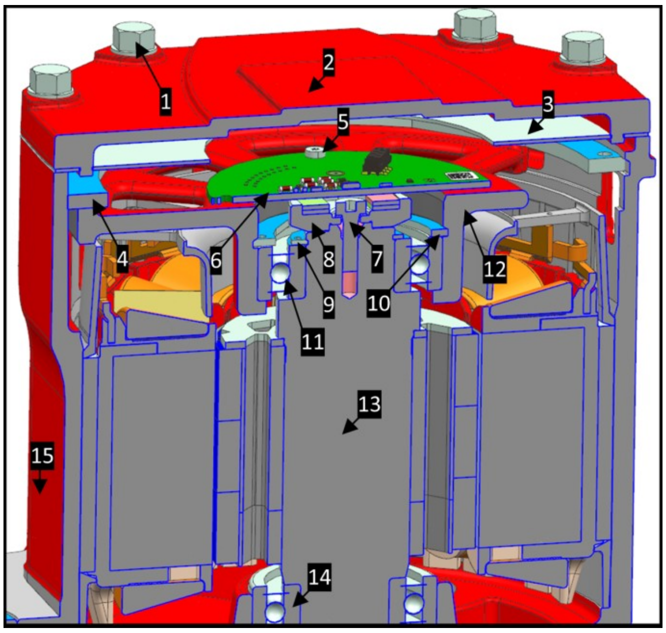


Figure 3. Cross-sectional view of an mechatronic drive with numbered parts.

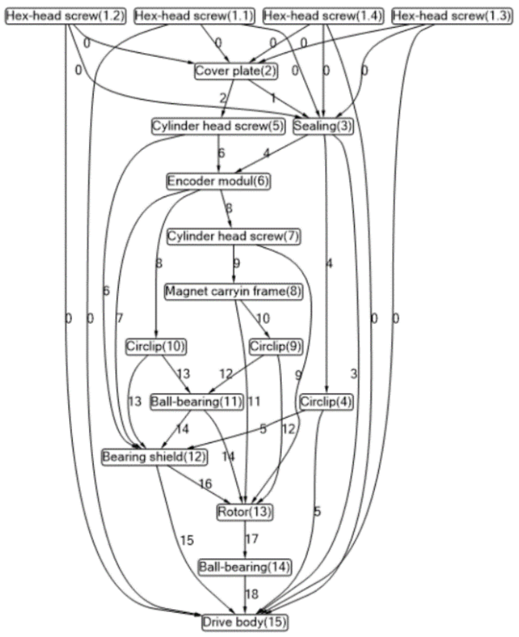


Figure 4. Product graph of the mechatronic drive (see Figure 3).

To create these product models, we developed and integrated a CAD plugin within the Siemens PLM NX software. With the help of this CAD plugin, we can create product models with a few clicks. We can also gather some part information directly from the CAD environment, such as the position of the part with reference to the main assembly coordinate system, the material that the part is made from and its weight and weight distribution. The CAD plugin exports the product model as an XML file that can be loaded into the IDAA.

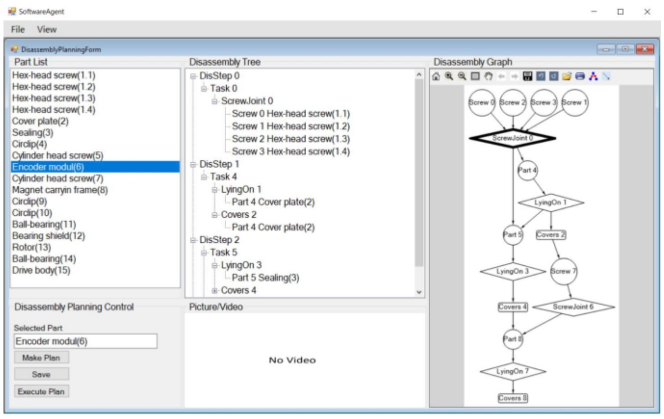
IV. Disassembly Planning

To create a valuable assistant, we must drastically reduce the programming effort involved, with the goal that no manual programming should be necessary. Therefore, the disassembly planning module automatically determines the sequence of disassembly tasks based on the product model and a given goal definition. The goal definition could be the removal of a certain part or parts or the complete stripdown of the assembly. To remove a single part from the assembly, the user selects the corresponding part. The algorithm searches for the node representing this part in the product graph, and it then produces a subgraph by following the ingoing edges of the node and adding their connected nodes to the subgraph in a breadth-first manner until no nodes or edges are available. The subgraph describes which parts and connections must be removed to reach the desired part. Next, we use topological sorting to determine the disassembly task sequence. In this way, we iteratively create new disassembly steps containing disassembly tasks. In each step, we select the nodes from the subgraph without ingoing edges. We define a task for each determined node/part with the connection type defined by the edge and the related part. After a task is created, we delete the node and its outgoing edges from the subgraph.

If all nodes without ingoing edges are threaten in a step, we create a new step. This procedure is repeated until no nodes or arcs are available in the subgraph. The result is a partially ordered list of disassembly steps with at least one disassembly task per step. Each task stands for the loosening of a connection type and the removal of a part from the assembly. Because the disassembly sequence is only partially defined, the sequence may not match the user's intentions and the user may rearrange it as he or she wishes via the GUI. Furthermore, the user can select the best suitable labor distribution from the corresponding process model. The human coworker is thus able to adapt the process according to the current situation or condition of the product. The disassembly sequence (see Figure 5) is then transmitted to the next (task initialization) module. Before discussing the task initialization module, however, we must first introduce the process model.

V. The Process Model

The purpose of the process model is to describe the process of loosening a specific connection type in a machine-understandable way. The intention is to use the model to coordinate the actions of man and machine. To formally describe a disassembly process, we must define the actions that take place in a disassembly task. One method of describing human actions that is commonly used to analyze manual assembly processes is methods-time measurement (MTM). In MTM, human activities are modularized (for example, "reach," "grasp" and "move"), and a task is described in detail through the sequence of these activities. A similar method for describing robot actions is robot time and motion (RTM). The adapted MTM model mentioned in [11] was also proposed as process logic for cognitive automated assembly.



Part List

- Hex-head screw(1.1)
- Hex-head screw(1.2)
- Hex-head screw(1.3)
- Hex-head screw(1.4)
- Cover plate(2)
- Sealing(3)
- Circlip(4)
- Cylinder head screw(5)
- Encoder modul(6)
- Cylinder head screw(7)
- Magnet carryin frame(8)
- Circlip(9)
- Circlip(10)
- Ball-bearing(11)
- Bearing shield(12)
- Rotor(13)
- Ball-bearing(14)
- Drive body(15)

Disassembly Tree

```
DisStep 0
  Task 0
    ScrewJoint 0
      Screw 0 Hex-head screw(1.1)
      Screw 1 Hex-head screw(1.2)
      Screw 2 Hex-head screw(1.3)
      Screw 3 Hex-head screw(1.4)
DisStep 1
  Task 4
    LyingOn 1
      Part 4 Cover plate(2)
    Covers 2
      Part 4 Cover plate(2)
DisStep 2
  Task 5
    LyingOn 3
      Part 5 Sealing(3)
    Covers 4
```

Disassembly Planning Control

Selected Part: Encoder modul(6)

Picture/Video

No Video

Figure 5. The IDAA's disassembly planning view. On the left sidem a part list from the product model is displayed, and, on the lower left, the encoder module has been selected. In the middle, the generated disassembly sequence can be seen. On the right, there is a graph representing the disassembly sequence.

Furthermore, an extended MTM version that combines MTM with RTM was introduced for designing work systems featuring human-robot interaction [12]. These methods are based on manually created—and thus fixed—action sequences, which are further evaluated to optimize particular (primarily ergonomic) metrics. However, relying on a fixed sequence of action is not a recommended approach because process deviation may occur, which would require adaptions to be made. To be adaptable, the system must choose its actions based on accurate perceptions of the current situation. Therefore, we need to represent the current situation in some fashion and describe what actions are applicable, when they are applicable and how they will change the situation. With a defined goal situation, the system would then be able to determine the necessary actions automatically.

One approach involves the use of description languages such as STRIPS, ADL, PDDL or derivatives of PDDL [13]. In one part of the PDDL model, called the domain, a hierarchy of object types, predicates and actions are defined. In another part, the object instances and constraints and the initial and goal states of the world are defined. The latter part is called the problem. Both models are fed into a solver to determine a feasible sequence of actions to reach the target state. These languages are powerful but complex, and it thus is difficult to create accurate representations thereof or to extend them. Furthermore, we had six specific goals that we wished to achieve, which were best accomplished by designing our own process model.

The first goal of our model was to reduce complexity by splitting the overall domain into simpler connection type-related, task-level process models. Each process model is fully independent from the other process models within the domain. We define a set of process models $\mathbf{T}$, in which each process model $\mathbf{t}$ is related to one connection type.

A. *The Process Model*

Within a **process model**, we describe all involved DCAs in a set $\mathbf{D_t}$ (e.g. robot, gripper, user, etc.) and objects in a set $\mathbf{O_t}$ (e.g. tool, part, box, etc.). We distinguish between agents and objects in that agents can execute actions. The second goal of our model was to provide the user with different options for selectable labor distributions, or workflows, to adapt to different situations (e.g., product conditions). Therefore, we describe different forms of workflows within the process model in a set $\mathbf{B_t}$. When the user selects a workflow, he or she can expect a certain behavior, and this produces greater trust in the system. In conclusion, a process model is defined as $\mathbf{t} = \{\mathbf{O_t}, \mathbf{D_t}, \mathbf{B_t}\}$.

B. *Defining a Workflow*

A **workflow**, which is a certain kind of labor distribution between man and machine, can take the form of coexistence, cooperation or collaboration. A workflow $\mathbf{b_t}$ is related to a process model and contains a number of actions $\mathbf{A_b}$. The actions within a workflow are the minimum needed to perform the task under perfect circumstances. The actions in the workflow are not necessarily ordered; we only mention that these actions are part of the workflow. Furthermore, we can define a specific goal process state $\mathbf{g_b}$ for each workflow. This is necessary because the goal process state can change with different workflows. A workflow is therefore defined as $\mathbf{b_t} = \{\mathbf{A_b}, \mathbf{g_b}\}$.

C. *Describing Objects within the Process Model*

The third goal of our model was to represent each physical object in the task as an abstract object $\mathbf{o_t}$ within the process model $\mathbf{t}$. An abstract object $\mathbf{o_t}$ can represent physical objects, such as the parts mentioned in the product model, as well as other objects, such as transport boxes, manual tools or non-controllable robot effectors. An abstract object in our process model consists of three parts—a unique name $\mathbf{n}$, a parameter set $\mathbf{P}$ and a set of states $\mathbf{S}$—and can be represented as $\mathbf{o_t} = \{\mathbf{n}, \mathbf{P}, \mathbf{S}\}$.

D. *Describing Agents Within the Process Model*

The fourth goal was to represent the agents that are involved. An agent $\mathbf{d_t}$ owns a name $\mathbf{n}$, a parameter set $\mathbf{P}$ and a set of states $\mathbf{S}$. Furthermore, we need to represent the actions that an agent provides (our fifth goal) through an action set $\mathbf{A}$. An agent is thus defined as $\mathbf{d_t} = \{\mathbf{n}, \mathbf{P}, \mathbf{S}, \mathbf{A}\}$. In the next section, we explain the meaning of a parameter $\mathbf{p}$ within the set $\mathbf{P}$.

E. *Parameters*

Most actions defined within an agent require parameters for execution. For example, if a robot needs to move to an object's location, the object location is the parameter of a move action. A parameter stores the unique name $\mathbf{n}$ of the object or agent to which it's belongs. Furthermore, we distinguish between static and dynamic parameters. A static parameter has a parameter name $\mathbf{pn}$ and a constant parameter value $\mathbf{pv}$. A dynamic parameter additionally defines a reference $\mathbf{prc}$ to a known physical object class, such as a screw class or transport box class, and a reference $\mathbf{prp}$ to a property of that class. In the task initialization module, the value of a dynamic parameter is set to the property value of the current instance. Implementing these dynamic parameters was our sixth goal. They allow us to link information via an abstract object or agent defined in the process model to real physical objects or agents as these change with each task. A dynamic parameter is thus defined as p = {**n**, **pn**, **pv**, **prc**, **prp**}.

Next, we explain the meaning of a state within the set **S**.

F. States

We had two options for representing the process state within the process model: directly defining the overall process state or creating the process state by merging the objects and agents' states. We chose the latter option because it is better suited to changing conditions.

To characterize each condition of an object or agent that is important for process control, we use a state **s**. A state holds the name **n** of the object or agent to which it belongs. Next, a state **s** defines a state name **sn** and a state value **sv,** which represents the current condition. A state can also define an initial state value **svi**, a target state value **svt,** or both. All values for **sv**, **svi** and **svt** are elements of the set of possible state values **Svp**. These values are unique strings, each of which represents a specific condition of the object or agent. In addition, to define a "connected" state (our sixth goal), we can assign an MQTT topic **st** to the state. Through the MQTT topic, a state can be automatically coupled with a DCA that perceives this condition and updates its state value. A state is thus defined as **s** = {**n**, **sn**, **sv**, **svi**, **svt**, **Svp**, **st**}. For example, to represent the conditions of a gripper, we can define two states named "DeviceStatus" and "FingerStatus." For the "DeviceStatus" state, we define the possible state values as {"Unknown", "Off", "On", "Initializing", "Ready"} and set the initial state value to "Unknown." For the "FingerStatus" state, we define the possible values as {"Unknown", "Open", "Closed", "ObjectGripped"} and set the initial state value to "Unknown." To link these two states directly to the gripper DCA, we assign them the MQTT topic to which the gripper DCA publishes its status. The possible object or agent state space can then be computed by the Cartesian product of each state's set of possible state values **Svp**.

G. Actions

An action also stores the name **n** of the agent to which it belongs and has an action name **an**, as well as three sets of state spaces. The precondition state set **SP** describes the condition of the process state required to execute an action, the transition state set **ST** describes the intermediate condition of the activity and the effect state set **SE** describes the final condition after successful execution of the action. With the help of the process model editor, these action state sets are configured using drag-and-drop states from the state sets defined in the process model for objects or agents. For example, the grasp action of the gripper agent would define the **SP**, **ST** and **SE** state sets listed in Table 1. Furthermore, an action defines a parameter set **P**, which contains the necessary action parameters for its execution. With the process model editor, the parameter set of an action can be configured using drag-and-drop parameters from other objects or agents in the model. To execute an action, we must send a command and its parameters to the DCA to which the action belongs. Therefore, we define in the action a command word **cm**, indicating the action, and an MQTT topic **cto**, from which the DCA receives its commands. Furthermore, we assign cost to each action **ac**. An action is completely defined as **a** = {**n**, **an**, **SP**, **ST**, **SE**, **P**, **cm**, **cto**, **ac**}.

Table 1. The state sets **SP**, **ST**, **SE** of the grasp action

Precondition	Transition	Postcondition
Robot.DeviceStatus.isIdle	Gripper.Finger.Moving	Part.Position.atGripper
Robot.Effector.Gripper		Gripper.Finger.ObjectGripped
Robot.Position.atPartLocation		
Part.Position.atAssemblyLocation		
Gripper.DeviceStatus.Ready		
Gripper.Finger.Open		

VI. Task Initialization

In this module, we work through the disassembly sequence step by step and task by task. Only one task at a time is handled by the task initialization, interaction planning and action execution modules. Depending on the current connection type, as defined in the task from the disassembly sequence, we load the related process model from the knowledge base. Next, we determine the goods of production based on the part-type instance defined in the task. In this process, we use the information stored in the part instance to determine a suitable transport box and the appropriate robot effector and manual tool. We then initialize all of the dynamic parameter values defined in the objects, agents and actions. Thereafter, the process state is generated by merging the objects and agents' states with the initial state values. This process state is then transmitted to the process monitoring module. Since we defined more than one workflow in a process model, the user must select the desired one at this point. This selection causes the actions defined in the workflow to be assigned with zero costs. The selection of the workflow will also affect the target process state. If no target process state is defined in the workflow, it is generated by combining the target states from the objects or agents. Finally, the target process state and all actions of the agents are transmitted to the interaction planning module.

VII. Task Monitoring

Before we can use the interaction planning module to determine the necessary sequence of actions, we need to perceive the real process state through the process monitoring module. Upon the initial process state, we can find all states that define MQTT topics that update their state values. When we subscribe to these MQTT topics, we receive the last valid state value because all status messages are published with the "retained message" feature. In other words, the MQTT broker will store the last status message for each topic and automatically send this status message to an MQTT client when it subscribes to the topic. To obtain a complete new process state, we wait until all states that are connected to MQTT topics have been updated once. Using the MQTT "last will and testament" feature, we can define a state message for a topic that will be published if communication with the agent is lost. We also use the "quality of service level 2" feature to ensure that messages are received exactly once. The perceived process state is then transmitted to the interaction planning module.

VIII. Action Planning

In this module, we attempt to find a sequence of actions that produces the goal process state and matches the workflow desired by the user. A state transfer system using an uninformed branch-and-bound search algorithm is employed for this purpose. The first and most important advantage of this approach is that no programming is necessary. The second advantage is that we do not have to rely on fixed action sequences and can produce adaptive action sequences based on the current situation. Thus, necessary additional actions that were not defined in the workflow can be included, or actions that are defined in the workflow but are not required can be removed. This approach can also adapt if the team composition changes (e.g., if an agent joins or leaves the process) or if the selected workflow is not feasible. The search process is straightforward: First, we create the first search node, which contains both the perceived process and a unique id. This search node is added to a queue. Thereafter, the following steps will continue until the target process state is reached.

The first search node is popped from the queue and added to a list of extended nodes. We then compute the actions that are applicable to this search node. This is done by comparing the states values from the precondition states of an action with the state values defined in the process state of the search node. If the state values match, the action is added to the applicable action set. We then apply all determined actions to the search node. When an action is applied to a search node, it creates a new search node with its own unique ID, as well as the search node ID of the parent. Furthermore, the state values in the stored process state change as described in the effect state set of the applied action. The action also adds its costs to the new search node. As the creator of the search node, the action will also be stored in the new search node. All search nodes thus created are added to the queue, except if a new search node contains a process state that is equal to a process state of a search node that exists in the extended node list. The elements in the queue are sorted such that the node with the lowest cost is placed on the top. After checking whether one of the search nodes contains the target process state, we continue with the next search step.

By repeating this process until a target process state is found, we create a tree-like structure of possible action sequences, as illustrated in Figure 6. The branch and bound algorithm thereby finds either an optimal solution or no solution for the given target process state. If a goal search node is reached, we can determine the sequence of parent search nodes, as well as the sequence of actions. Because of the lower cost assigned to the actions in the specified workflow, these actions are preferred in the search process. The solution is a sequence of actions that need to be executed and a sequence of process states that must be met to loosen the connection and separate the part from the assembly. Both sequences are transmitted to the action execution module.

IX. Action Execution and Synchronization

This module works from the determined action sequence. The module executes an action by instructing the corresponding DCA (which is the provider of the action) to execute it. This is done by sending the command word via the assigned command MQTT topic, followed by the comma-separated parameters from the action parameter set. This is done for all agents, with the exception of the human user. To advise/inform the user, we use a GUI displayed on a touch screen monitor. While different agents are working together, we need to synchronize their actions. This is done using the sequence of predicted process states delivered by the action planning module. The controller can observe the action execution by comparing the predicted, transition and effect process states with the perceived process state that is permanently updated by the process monitoring module. If an action is successfully executed, the controller moves on to the next action in the sequence. If the action sequence is finished, we select the next task in the disassembly sequence and jump back to the task initialization module. This procedure continues until all tasks are done. In the event that the predicted process state does not match the current process state, we can force a re-planning in the action planning module.

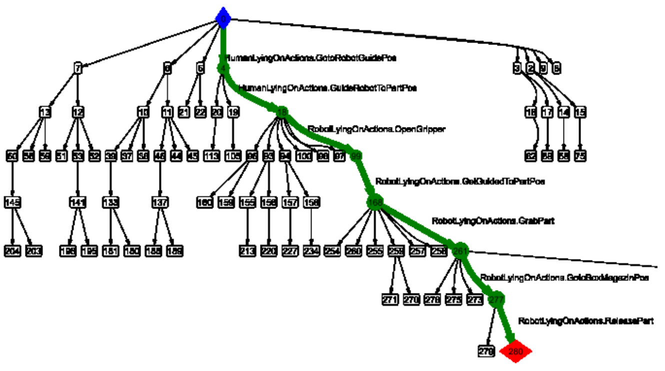

HumanLyingOnActions.GotoRobotGuidePos
HumanLyingOnActions.GuideRobotToPartPos
RobotLyingOnActions.OpenGripper
RobotLyingOnActions.GetGuidedToPartPos
RobotLyingOnActions.GrabPart
RobotLyingOnActions.GotoBoxMagazinPos
RobotLyingOnActions.ReleasePart

Figure 6. The figure portrays a tree structure generated by the action planning module. The blue diamond is the initial and the red diamond the target process state. The action sequence is illustrated by the arrows on the green path and the predicted process states by the green dots.

Because some objects (e.g., screws) do not have microprocessors and sensors attached that can determine their states and send state descriptions to the process monitoring module, we need to perceive those objects states using other DCAs. Another approach is to consider actions as deterministic in the following sense: If an agent whose state we can perceive successfully executes an action on an object that we cannot perceive, then we believe that the object state has changed as described in the action effect state set. In our application, we also rely on the human participant and his or her ability to intervene in the process. If an action does not have the intended effect, the user must engage and resolve the issue. With experience, the user will then be able to anticipate the success of actions depending on the overall situation and choose the workflow accordingly. Furthermore, the user's feedback concerning unsuccessful actions could provide valuable data for the implementation of more advanced planning technics. For example, machine learning could be used to predict the desired workflow based on the user's preferences. Learning is a key element in assistance systems, but, before learning can occur, we need to provide basic functionality that enables the system to work with humans. Learning will then assure the improvement of the system's performance over time.

X. The Demonstration Application

In this chapter, we illustrate our demonstration application, the hardware used and how we interconnect all of the devices involved.

A. *Introduction*

In our demonstration application, we explore the robot-assisted non-destructive disassembly of mechatronic drives. The broader goal is to boost the economic fitness of remanufacturing processes by reducing or supporting the heavy and costly manual labor of disassembling. The drives we focus on come in three performance/size classes, each of which could be equipped with mechatronic elements such as brakes and electronics such as sensors, communication modules or built-in frequency converters. When variations in size, gear ratio and attachments are taken into consideration, there are many distinct parts in the drive assembly. We also need to consider product design changes caused by product optimization. All in all, the number of variants exceed one million, hence the need for product model-based disassembly planning. The dominant connection technics are screw joints, press fittings, inserted parts and subassemblies, circlips and electrical connectors. To loosen these connection technics, we developed tools, fixtures and processes to determine the required disassembly knowledge that is represented through the product and process models.

B. *Hardware*

Our demonstrator consists of two robots, a Kuka KR 125 and a Kuka iiwa 14 R820. The Kuka KR 125 is equipped with a manual tool change system and a fixture to hold the 25-kg electrical drive in an appropriate work position. With the use of the measurement plate on the drive fixture, we can determine the transformation from the Kuka iiwa to the main drive assembly coordinate system. To teach the reference frame, the user hand-guides the robot's calibration tool into the three measurement holes with a ball tip. The Kuka iiwa is mounted on a mobile platform. Inside the platform, the robot controller, additional electronics (e.g., bus coupler, input/output terminals, etc.), pneumatic valves, a network router and a control PC are integrated. We installed a pneumatic-actuated tool change system with electrical and pneumatic feed-through to automatically mount different tools on the Kuka iiwa's flange. We designed and manufactured three robot effectors: the calibration ball tip tool, a ratchet with a universal wrench socket to loosen screw joints and a gear set holding plate to remove the complete gear assembly at once. A Robotiq two-finger gripper was also made mountable on the tool change system.

C. *Software*

We decided to use a distributed software approach that couples the IDAA with the DCAs over the MQTT protocol in a loose manner. Figure 7 provides an overview of the implemented DCAs and their interconnectedness. We integrated a DCA as an interface for the Kuka iiwa, which receives commands via the MQTT protocol and routes them through a TCP/IP connection to a robot program running on the robot controller. A thread running parallel to the robot program listens for those command messages on the TCP/IP socket. Receipt of a command message triggers the execution of the corresponding action or skill in the main robot program.

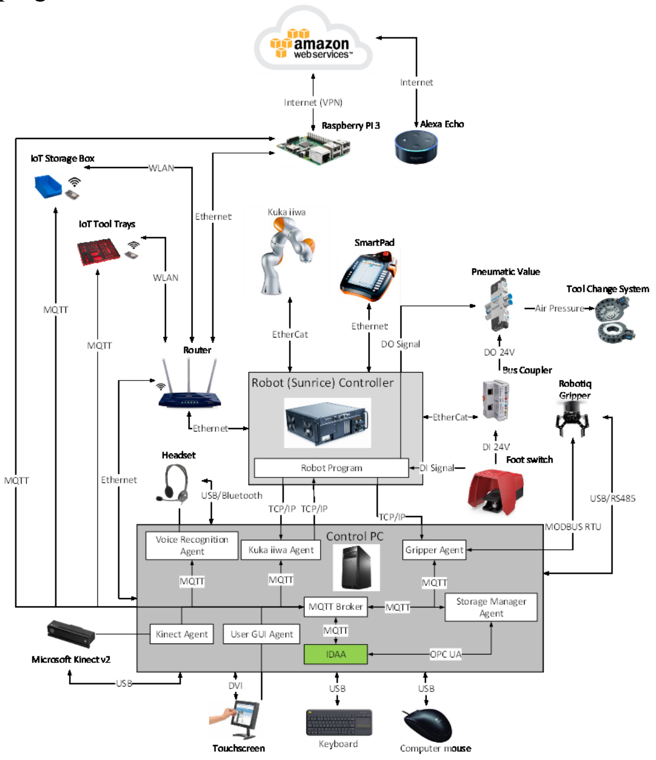


Figure 7. This figure illustrates the communication between the different devices and software in our demonstrator application.

While executing an action or skill, we change the robot status accordingly. If the robot status changes, we transmit the new status through the second TCP/IP connection back to the Kuka iiwa's DCA. The Kuka iiwa's DCA then publishes the received message on the MQTT status topic. In our application, we also use the Kuka iiwa as a human-robot communication interface for action synchronization. The user has two options to confirm his or her action execution to the system. The first is a haptic interaction in which the user pushes or pulls on the robot structure, and the second is pushing a button located at the robot flange to confirm action execution.

Because this form of human-robot communication is very limited and the robot might not within reach of the user arm to confirm action execution, we investigated further communication modalities. The first approach was a voice recognition agent which uses the speech recognition libraries available in .NET Framework. To improve speech recognition, we also connected an Amazon Alexa to our application through an Amazon Echo Dot. Our Alexa skill uses Amazon's Alexa Voice Service, Lambda and IoT Web Service, as well as a Raspberry Pi 3 functioning as an IoT device, to connect our robot assistant with Amazon's Alexa digital assistant. The improved voice recognition and additional assistive functions contributed by the Alexa device provide an example of a promising application of cloud computing in robotics. When the disassembly process continuously varies, natural language human-robot communication is essential for the specification, coordination and synchronization of tasks, workflows or actions.

We also designed a DCA to control the Robotiq gripper. The DCA establishes device-specific communication using Modbus RTU to communicate with the gripper hardware. While Modbus RTU communication is based on reading and writing bits, the gripper DCA must map commands to its bits representation and bits representation to a gripper status.

The openness of the MQTT protocol also allows for the integration of small IoT devices such as "intelligent" storage boxes or tool trays (e.g., equipped with an ESP8266 Wi-Fi module). In our application, these boxes and tool trays register with the storage manager agent upon activation. The storage manager agent therefore knows which boxes and box sizes are available, as well as which tools are in the tool tray. When, for example, a tool tray is registered, the storage manager agent prompts the robot and the user to teach its tray coordinate system so that the stock manager agent can directly determine the grasping position of a tool.

The Kinect DCA is an ongoing project in which we are working on user tracking via the Kinect v2 sensor. Our goal is to track the user and his or her hands in order to automatically monitor and update the user's status description. With the use of virtual areas around meaningful locations, such as the box or part position, we plan to recognize the moment when the user's hand enters or leaves the area of interest. This should also contribute to the improvement of action synchronization.

D. *Task and Workflows*

We address four different tasks: the removal of simply stacked parts, loosening of screw connections, the removal of the gear set and the assisted dismantling of press-fitted gear wheels and bearings. In the following section, we describe the possible removal procedures (defined through the workflows in the process model) for simply stacked parts as an example.

- In the first specified workflow, the user solves the task manually, without support from the system. The assistant therefore waits until a push, a button press or a signal from the voice recognition agent indicates the completion of the task. The waiting condition of the robot is visualized by a strip of green lights at the robot flange.
- In the second workflow, the robot grasps a matching transport box and holds it near the working area. The user places the part in the box and confirms the action, and the robot returns the box to its dock. Before moving, the robot sets its light strip to red, indicating that it will start moving in one second.
- The third workflow describes a collaborative approach to carrying out the task. The robot is set to a hand-guided mode and signals its condition with a blue light strip. The user then leads the robot to a grasping position, and, after confirmation, the robot grasps the part and places it in the corresponding box. A benefit of this workflow is that the robot remembers the grasping position for the next time (i.e., it learns). This learning is accomplished through changing the part grasp position in the product model.
- The last workflow describes non-human actions only: the robot approaches the grasp position, grasps the part, moves it to the drop position over the box and releases the part.

Please note that the grasp action is carried out by the gripper agent in the above description.

XI. Conclusion

Through the multi-agent system architecture and the implementation of the device-controlling software agents, we gained the ability to integrate a wide variety of device. Through the unified, lose coupled communication between the agents we can use both centralized and decentral control approaches. Furthermore, we built the base level of an intelligent assistant (IDAA) that is capable, upon explanation of the product model, of automatically determining the sequence of disassembly tasks required to reach a specified goal. Furthermore, we introduced the process model, which stores object and agent models that describe a connection-related disassembly process with different forms of workflows. Through action planning, the system can determine the necessary actions of man and machine upon the current process state. The system is also able to synchronize the actions of man and machine during execution using process monitoring and human-robot communication. Derivation between the process state and the foreseen process state after action execution are detected and used to force an action re-planning. We developed a demonstrator application and used it to verify the information flow and algorithms used. We successfully performed tests with simple stacked objects to validate the human-robot interaction. However, some functionalities are still missing, such as the collision-free robot path planning, grasp planning, and the integration of vision systems. Moreover, learning is an important aspect of such assistance systems. One application of machine learning could be to learn the user's preferred workflows and thus automatically choose the appropriate labor distribution to create a more fluent human-robot interaction.

REFERENCES

[1] ISO 190218-1/2:2011, Robots and robotics devices- Safety requirements for industrial robots, Part 1an 2.

[2] ISO/TS 15066 Robots and robotic devices - Collaborative robots

[3] W. Gerke, Technische Assistenzsysteme: Vom Industrieroboter zum Roboterassistenten. Berlin: De Gruyter Oldenbourg, 2015.

[4] R. C. Arkin, Behavior-based robotics, 3rd ed. Cambridge, Mass.: MIT Press, 2000.

[5] S. J. Russell and P. Norvig, Artificial intelligence: A modern approach. Upper Saddle River: Prentice Hall, 1995.

[6] D. Vernon, Artificial cognitive systems: A primer. Cambridge, MA: The MIT Press, 2014.

[7] J. M. Henrioud and A. Bourjault, "Computer aided assembly process planning," Laboratoire d'Automatique de Besancon, Besancon, France, 1992.

[8] G. D. GRADI, "Petri-Net Modelling of an Assembly Process System"

[9] Xu, C. Wang, Z. Bi, and J. Yu, "Object-Oriented Templates for Automated Assembly Planning of Complex Products," (en), IEEE Trans. Automat. Sci. Eng., vol. 11, no. 2, pp. 492–503, 2014.

[10] M. Merdan, W. Lepuschitz, T. Meurer, and M. Vincze, "Towards ontology-based automated disassembly systems," in IECON 2010 - 36th Annual Conference of IEEE Industrial Electronics, Glendale, AZ, USA, pp. 1392–1397.

[11] B. Britzke, MTM in einer globalisierten Wirtschaft: Arbeitsprozesse systematisch gestalten und optimieren. s.l.: mi-Wirtschaftsbuch, 2013.

[12] D. Schröter, P. Kuhlang, T. Finsterbusch, B. Kuhrke, and A. Verl, "Introducing Process Building Blocks for Designing Human Robot Interaction Work Systems and Calculating Accurate Cycle Times," (en), Procedia CIRP, vol. 44, pp. 216–221, 2016.

[13] M. Ghallab et al., PDDL: The Planning Domain Definition Language.