Artificial Intelligence supported Tool Chain in Manufacturing Engineering

ITEA 3 – 19027

Work package 3
Assisted generation of process instruction

Deliverable 3.4
Standardized context description format

Document type	: Deliverable
Document version	: 1.0
Document Preparation Date	: 21.12.2023
Classification	: private
Contract Start Date	: 2021-01-01
Contract End Date	: 2024-02-29

Final approval	Name	Partner
Review Task Level	Mario Thron	IFAK
Review WP Level	Adam Kłodowski	LUT
Review Board Level	Thomas Bär	DAIMLER

Executive Summary

The general goal of the context description is to depict the environment where a process is taking place and to characterize its state. Therefore, identifying the context is a supporting method for other work-packages within AITOC project and standardizing the way it is described will allow the integration of the various developed solutions.

Within deliverable 3.1 “Context Definition with industrial Relevance” four categories of context have been defined and were confirmed during all further considerations within AIToC:

- Human-centered Context,
- Task-centered Context,
- Environmental-centered Context,
- Interaction-centered Context.

Derived from these categories and based on several prototypical implementations this document proposes a context description format which is harmonized among the project consortium members. It is based on JSON due to its various advantages, including human-readable format and wide language support. The document suggests utilizing the generic 100% set of context parameters outlined in D3.1 as a comprehensive solution. For each specific use-case, relevant context parameters should be implemented, enabling a tailored approach to capture the required context information.

Content

1 Introduction

To determine a context description format with respect to industrial relevance in the field of manufacturing, a definition of context with its underlying parameters is required. Within deliverable 3.1, such a context definition and context parameters with industrial relevance was defined. A generic context definition according to Abowd et al. [1] forms the basis of the underlying activities within work package 3 of the research project AIToC. Within the definition of [1, p. 304], context is presented as "any information that can be used to characterize the situation of an entity, where an entity can be a person, place, or physical or computational object". Moreover, this definition is enhanced by [[2], p. 5], whereby an entity is further described as "relevant to the interaction between a user and an application." Consequently, there is no restriction on context in terms of underlying parameters like the location; rather, it encompasses any knowledge that defines the present status and everything that can be beneficial and utilized by a specific context-aware application. Hence, it can be adopted to multifaceted use-cases like the manufacturing industry. Furthermore, considering the literature, a categorization of industrial relevant context parameters into four main categories emerges. Regarding the underlying term to define generic context, the authors [1] emphasized the general importance of the user/human and the task to be performed. Besides that, within the literature, the dimensions of significant context categories for manual assembly assistances are expanded to parameters that can be classified in an environmental [3] – [5] and interaction model [5], [6].

A variety of context-aware systems have been developed, explored, and presented to the scientific community, due to the wide range of crucial context inside the multifaceted use-case of manufacturing. Because augmented reality (AR) applications are context-aware, having regard to the spatial registration of AR content in the external world [7], a wide range of assistances is developed in AR and characterized thus, as context-aware. To enable the mentioned spatial correct registration of digital content, information about the layout and object data is necessary [8], i.e., environmental-centered context data. Furthermore, approaches that are more complex exist, including additional algorithms and sensors to ensure an adaptive information presentation. In this context, the retrieving of static layout data or static task sequences from databases is combined with more sophisticated data such as continuous/dynamic data, to ensure a real-time adaptation. From this point of view, a static data acquisition is defined by preliminary collected data, that are pre-loaded and made available for the assistance system. These data are indeed updateable, but not during the specific information provision i.e., task execution. On the contrary, dynamic data are not pre-loaded, but constantly collected during the task execution. As an example, for dynamic context acquisition, besides the target assembly sequence, Mura et al. [9] utilized a force and a camera sensor to reason task progress, task type as well as assembly errors. Rodriguez et al. [10] gathered motion capture data by a Microsoft Kinect camera to determine equal context parameters, as well as Funk et al. [11]and thus predict the cognitive state. Task complexity is utilized and therefore predicted by Zaeh et al. [12] by a multi-dimensional measure across the four context categories with consideration of predetermined task time, general qualification as well as the cognitive state. Apart from task-centered or human-centered context parameters, continuous motion data retrieved from cameras [13], electromyography sensors [14] or inertial measurement units [15], can also be used to identify interaction-centered

context. Whilst implicit interaction needs no additional movement besides the ones that must be conducted to execute the specific tasks, explicit interaction is realized by performing extra tasks by the user (e.g., hand or eye gestures, user input by physical buttons or graphical user interfaces [16]). Furthermore, Binder et al. [16] utilized the explicit user input to determine human-centered context parameters such as the user's preference and personal constraints like the existence of visual impairment or color blindness. Implicit interaction data are used by Bannat et al. [17]to gather the cognitive state. Westerfield et al. [18] used an ontology-based approach to reason the cognitive state by implicit camera data and knowledge about task constraints.

The presented context-aware systems are intended to serve as a basis for creating a list of possible context parameters. Thus, in the following the literature-based context parameters are assigned to one of the four main categories: human-centered, task-centered, environmental-centered, and interaction-centered context. However, the interaction context in particular shows that no one-dimensional context view can be used. More precisely, the collected interaction data can be used to determine human-centered context (e.g., search movements to determine the cognitive state) or task-centered context (e.g., task progress information from motion data). Therefore, no exact separation of the four context categories can be made, but rather must be seen in a multi-dimensional approach with dependencies between the individual parameters and models. Thus, the following allocation is a dynamic propose.

1.1 Human-centered Context

Regarding the literature, the human-centered context comprises parameters that can be directly linked to a specific human or a group of individuals (e.g., working group at a specific workplace).

Table 1: Literature-based human-centered context.

Generic Context Parameter	Description	Example: Specific Parameters	Context Acquisition	Example: Data Input
Anthropometry	The user's human body during the use-case-specific task execution as well as in general with information about individual constraints.	Restrictions, Bones, Posture	Static or Dynamic	Static: User-ID Dynamic: Camera-based Posture Tracking
Cognitive State	Information about the individual user's cognitive processes, with the possibility to reason cognitive overload or underload.	Stress, Cognitive Load	Dynamic	Smartwatch
Preference	The user's individual opinion on different alternatives (e.g., visualization devices,	Learning Type/Styles	Static	User-ID

	information provision methods…).			
Qualification	Assessment of general knowledge as well as specific task-related skills that gives the user the ability to perform the task.	General Qualification, Task-specific Qualification	Static	User-ID: Company-Specific Qualification Matrix
Demographic Data	Specific information about the user and his/her affiliation to certain groups of people.	Age, Gender	Static	User-ID

1.2 Task-centered Context

The task-centered context category encompasses data that defines or specifies the task that should be executed by the user. Accordingly, more general parameters such as the task type as well as detailed calculations like the underlying complexity can be considered as relevant to adapt the assistance.

Table 2: Literature-based task-centered context.

Generic Context Parameter	Description	Example: Specific Parameters	Context Acquisition	Example: Data Input
Task Complexity	Assessment of the task characteristics that describes the need to integrate complicated mental and physical aspects independent of the individual user's skills.	Complexity Metrics (e.g. [39,40]), Error Rate	Static	Annotations, Historical Data
Task Type	Categorization of the specific tasks to higher-level task groups.	Higher-level Task Categories (e.g., MOSIM Task List)	Static	Annotations, Company-Specific Work Plan
Task Specifications	Meta-Data, detailed information, or constraints about the task with high importance to consider during performance.	Safety Information, Predetermined Task Time	Static	Annotations, Company-Specific Work-Plan or EDM-System
Task Progress	The active task, according to granularity and possibility, tracking in terms of progress and correct execution.	Task Sequence, Performed Tasks, Correctness of Performed Tasks (i.e., Errors)	Static or Dynamic	Static: Company-Specific Work-Plan

				Dynamic: Hand Tracking (Data Gloves)

1.3 Environmental-centered Context

Parameters that are related to the specific environment in which the tasks that should be performed by the user take place, are linked to the environmental-centered context model. Thus, layout as well as object data and specific requirements are comprised by this category.

Table 3: Literature-based environemntal -centered context.

Generic Context Parameter	Description	Example: Specific Parameters	Context Acquisition	Example: Data Input
Layout Data	Meta-data as well as transforms and geometry of layout-related data.	Data of Stations, Workplaces, Resources (e.g., Racks, Trolleys), QR-Markers	Static or Dynamic	Static: Annotations Dynamic: RFID-Tags/Reader
Object Data	Meta-data (e.g., Grasping Points, Bill of Material, Assembly Path) as well as transforms and geometry of object-related data.	Data of Products, Tools, Sensors, Visualization Devices	Static or Dynamic	Static: Annotations Dynamic: RFID-Tags/Reader

1.4 Interaction-centered Context

The interaction-centered context can be seen as a context model category with a strong interdependency to the other three context models. This is, due to the fact, that the interaction-centered context comprises parameters and data which are used to derive different parameters of the other context categories (e.g., task progress derived by the implicit interaction of the user by motion data).

Table 4: Literature-based interaction -centered context.

Generic Context Parameter	Description	Example: Specific Parameters	Context Acquisition	Example: Data Input
Implicit Interaction	Interaction of the user is derived from measurements during the use-case-specific task execution.	Behavioural/ Physicological/ Motion Measurements	Dynamic	Camera-based Posture Tracking
Explicit Interaction	Interaction of the user is done by additional tasks,	Commands (Voice, Eyemovement,	Dynamic	Voice Commands

	besides the use-case-specific task execution.	Gesture, Taktile Information)		

2 Implementation of context

2.1 Annotations of parts and assemblies

Identifying part features from geometry is a complex task that have been separated from the task reasoning routine to simplify the problem. Currently, the annotations of parts, geometrical features within parts, and assemblies is a manual effort. However, it is a good example of work that, with enough manually annotated examples, could be automated and used as training data for the AI. There are three levels of annotations: assembly level – describing relations between part-level markers to form semantic assembly representation; part level – identifying part types, material, categories, weight, overall dimensions, and function – this level is used in reasoning operation to determine which type of operation is applicable for specific part or which handling method to use for moving part around; finally feature-level annotations describe position, orientation and geometric parameters of a feature, a good example is threaded hole – to mate with proper screw, thread parameters like pitch, length, direction need to be known, in addition, position of the threaded hole and its orientation is required to position mating part in the assembly properly.

Annotations of parts is done in annotation editor, that is a standalone tool based on Unity. This tool allows to annotate geometrical features, assign part level information, as well as define assemblies. Data is saved in the JSON format for readability and portability between systems, and task and operation editor can transform this JSON format to ASP definitions directly usable by the operation reasoner.

For the annotations of features and functions a common dictionary of allowed and well-defined properties need to exist at least on a company level, but preferably on the world-level to enable data and model exchange between part vendors and system integrators. Annotations are supplementary information for the CAD data and are closely related to the CAD data – for instance sharing common coordinate system. Therefore, we have adopted glTF CAD data exchange format, which is natively based on JSON and allows introducing extensions into the data format. This way annotations and CAD data can be merged into one open and widely supported data file. If alternative CAD format is required, annotations can accommodate that file in a separate JSON file.

To maintain feature types and allowed values as a dictionary commonly available, the annotation editor synchronizes annotation definitions with a selected server, be it internal company or global standard providing server. The ultimate goal is to provide open living standard for part, feature and assembly-level annotations that will be updated as new features, part types are emerging.

Assembly annotation in the JSON representation is currently under development, as parallel format ASP assembly annotation is already implemented and used directly. However, the JSON exchange format will ease editing of the annotations and cross-validation with the

template. The idea of assembly-level annotation requires simple array structure, where each array entry represents individual connection between two parts. The single array record contains reference always to two features at two different parts.

Assembly annotation structure contains two parts: assembly-level key-value pair annotations that are used for assembly component characterization and categorization, and geometrical connection section that defines part relations. Currently there are no assembly level coordinate system definitions included in the assembly annotations, as connections between assemblies can be described based on the assembly components feature markers. An example is presented in the listing below.

```
{
 "Properties": [
        "Name": "Front brake",
        "Subsystem": "Braking system",
        "Location": "Front left",
        "Vendor": "Brembo"
 ],
 "Assemblies": [
        {
        "ID": "A20",
        "Name": "Brake calliper",
        "Connections": [{
                "BaseAssembly": "",
                "BasePart": "Caliper",
                "BasePartFeature":"1",
                "PartAssembly": "",
                "Part": "Piston",
                "PartFeature":"3",
                },
                {
                "BaseAssembly": "",
                "BasePart": "Caliper",
                "BasePartFeature":"2",
                "PartAssembly": "",
                "Part": "Piston",
                "PartFeature":"3",
                }]
        }]
}
```

The interpretation of the annotation is that geometrical feature marker of part A should be aligned with the geometrical feature marker of part B to form a connection. There is no constrain of which features can be mated together, so for example a surface marker can be aligned with a thread marker if so desired. Restrictions on which geometrical feature marker on one part can be aligned with which geometrical feature marker on the other part are only

part of editor implementation, and should not be hard-coded, instead should be based on easily editable external definition. Currently, the Annotation Editor does not impose any restrictions on feature mating, but it will be introduced in future versions as aid for the user to limit the number of possible choices to sane selections.

Part-level annotations comprise of key-value pairs describing parameter name and value. Both parameter keys as well as allowed values can be defined in the annotation template standard. Four different value types are supported: integer and floating-point numbers; string values (free input text), and single choice list items. This allows covering geometrical parameters using floating-part numbers to describe sizes, dimensions, weights, etc.; normalized parameters that are of integer type, descriptive fields, that might be more useful for human than automated systems string inputs – for example part names, comment fields, or features not yet supported by standard in transition phase; and lists that are great to store descriptive options like types, part group names, or subsystems names. Lists have the benefit over string fields in sense they do not allow any mistype in the feature value, so ensuring each definition is consistent with the standard.

Feature level annotations comprise of the key-value pairs to define descriptive feature parameters and geometric definition of position within part (3D vector), orientation (as quaternion), translational and rotational ranges – for example to describe tolerances or permissible rotation angles or plane element extents. The definition for JSON representation of annotations follows *MConstraint* definition from the MOSIM project to maintain full compatibility with the MOSIM+ simulation tools.

2.2 How tasks and operations are described

In the AIToC project, common operation definitions are used. Each operation has a unique ID that allows translation of the operations to any language without affecting the context. The semantic meaning of each operation available currently is provided below:

- Position (ID=9) – moving object to a desired position from current location, move can be made by hands or using a tool. This operation does not attach rigidly moved part to the base where it was moved, additional operations are required to secure part in place.
- Stick on (ID=11) – similar action to positioning, except it implies the part will stay in place when force is removed as it contains glue. Please note that stick on assumes the glue is already on the part or base so glue application is not needed.
- Tighten loose (ID=12) – tightening screw/nut operation without torque sensing tool, mostly performed using hand, but can also be performed using tool. After this operation part is still to move within limited clearance between screw hole and screw.
- Tighten fully (ID=13) – tightening screw/nut operation that causes rigid fixing of the part to the base part. After this operation it is impossible to move the fastened part by applying force.
- Tighten with torque (ID=14) – action wise it is equivalent to tighten fully, it additionally implies using torque wrench, so torque should be defined in operation parameters.

- Untighten (ID=15) – loosening screw or nut to enable its motion, for example for adjustment. The fasteners are not removed completely but only loosened.
- Visual check (ID=16) – visual inspection of a part or connection, in simulation showed by gazing at the point of interest. It is a non-value adding operation.
- Manual check (ID=17) – manual inspection, implies testing with a hand if part is in place, can move, have proper surface etc. Implies using hand for inspection.
- Insert part (ID=18) – part is insert into another part hole or cavity. Inserting always means operation that does not require overcoming friction, or the friction is negligible. Inserted part is secured with maximally one translation and one rotation left free degree of freedom.
- Press in part (ID=19) – similar to insert with the exception, that in this operation use of force is expected, and part is assumed to be held in place by friction.
- Glue (ID=20) – similar to stick on, the difference is that glue operation requires first glue application and only then sticking the part on the base.
- Snap clip (ID=21) – similar to position, except the snap on action fixes the part in place. Specific part motion close to the final position can be expected in this operation.
- Unsnap clip (ID=22) – removing snappable clip from assembly, part becomes detached and free to move in all directions.
- Clean (ID=23) – hand motion or tool motion along cleaning path defined within the target part.
- Cut to length (ID=24) – cutting stock material to length, the cut-out element becomes active element for the next action.
- Make a cut (ID=25) – cutting operation where material can be cut along straight line, the cut element does not become active part for the next operation.
- Cut to shape (ID=26) – similar to make a cut operation with the exception that the cut shape can be any profile.
- Subassembly (ID=27) – performing subassembly operation on a group of parts.
- Assemble (ID=28) – positioning of a subassembly.
- Remove (ID=29) – opposite to position, the part is taken from its final location and held in hand or put in material zone.
- Check and adapt (ID=30) – manual check combined with final positioning.
- Insert electrical connector (ID=31) – similar to inserting part, the additional context is operation on electrical system, and in future it might mean simulating cable motions.
- Grease (ID=32) – applying grease on a surface, greasing path can be defined as annotation on the part being greased.
- Route cable (ID=33) – threading cable through hole.
- Open/close (ID=34) – Opening/closing door, parameters describe whether opening or closing motion is intended.
- Control (ID=35) – abstract action of controlling a device, it is simulated as taking a pose and persisting in it, while other mechanism is moving according to a prescribed program. Useful for modeling general interaction between user and systems. For example, pressing button to wait for an elevator, where the elevator motion constraints waiting time before next action can be performed, or controlling a crane to move object from one place to another.

- Move (ID=36) – move object from one place to another. Object is not fixed at the final position; object must be in hand already in order to be moved. It is a lower-level positioning action, where reasoning does not consider grasping and reaching for an object. It is mostly useful if object at hand should be manipulated.
- Document/read (ID=37) – reading text or preparing documentation, it is an abstract action and in simulation is just indicated symbolically.
- Fix (ID=38) – attaching part to the base part rigidly. It assumes no motion, but simple application of a geometrical constraint between part and base in the position and orientation that is available on the initial phase of the fix.
- Lay cable (ID=39) – placing cable on support, does not require threading through holes, but just laying out cables.
- Thread cable (ID=40) – exactly same as route cable.
- Open (ID=41) – opening a hatch, or hinged door panel.
- Close (ID=42) – closing a hatch or a hinged door panel.
- Press momentarily (ID=43) – pressing a button and releasing it immediately
- Press persistently (ID=44) – pressing a button and holding it until another instruction is issued.
- Tilt forward (ID=45) – tilting joystick forward; joystick needs to be already grasped.
- Tilt backward (ID=46) – tilting joystick backward; joystick needs to be already grasped.
- Tilt left (ID=47) – tilting joystick left; joystick needs to be already grasped.
- Tilt right (ID=48) – tilting joystick right; joystick needs to be already grasped.
- Tilt random (ID=49) – tilting joystick in a random direction, mostly for used for simulating joystick operations, joystick needs to be already grasped.
- Walk (ID=50) – walking from current point to target point, this action includes path planning.
- Jog (ID=51) – jogging from current point to target point, this action includes path planning.
- Run (ID=52) – running from current point to target point, this action includes path planning.
- Jump (ID=53) – making single jump in place.
- Sit down (ID=54) – sitting down from standing position, action can include turning around to position avatar with respect to the chair.
- Stand up (ID=55) – standing up from a sitting position.
- Squat (ID=56) – performing a squat in place, provided there is sufficient space.
- Enter (ID=57) – entering a vehicle – this action incorporates posture transitions required to take place inside the vehicle.
- Exit (ID=58) – exiting a vehicle – this action incorporates posture transitions required to leave the vehicle.
- Wave hand (ID=59) – waving a hand while looking at specific target.
- Grab handrail (ID=60) – grabbing a handrail defined as a constraint. If walking is performed in the same time it will keep the hand following the handrail.
- Release handrail (ID=61) – releasing a handrail and removing the walk path constraint imposed by the arm length and handrail shape.
- Write (ID=62) – writing on paper wit h a pen. Action can be simulated using actual objects for writing or can be just described using simple visualization.

- Read (ID=63) – abstract reading action, that means gazing at specific object for fixed duration of time.
- Stamp (ID=64) – placing a stamp on paper.
- Scan (ID=65) – scanning barcode using handheld scanner.
- Tie (ID=66) – attaching tie wrap around certain object, for example fixing cable in place.
- Clamp (ID=67) - Clamping an object to a fixture. This action locks all degrees of freedom from the object.
- Group tasks (ID=68) – reference to a task performed by a leader in which worker should take part or support it. Single user performs any tasks, and the other workers utilize group task to join to this task. This allows changing task leader task without need to update followers' tasks.
- Synchronize (ID=69) – waiting for another worker to finish certain task. This is implicitly used in group tasks< but can also be used explicitly to synchronize operations performed by independent workers.
- Meet (ID=70) – going to a meeting point or waiting in the meeting point for the others. The location of the avatar defines whether the person stays in place (if he is already at the meeting place) or walks to the meeting place. Until all participants are not at the meeting place all are avatars cannot perform additional actions. When last avatar in this action arrives at the meeting place all avatars continue executing their own actions that follow the meet operation.
- Sidestep (ID=71) – explicit request to make sidestep locomotion to a target point. Target point needs to be reachable by sidestepping. Currently walking does not use sidestepping even if it would be optimal motion to reach the goal, therefore if sidestep is required, it should be defined explicitly.
- Adjust (ID=72) – adjusting position of a part. No motion is performed just abstract notion of adjusting real part position for a fixed amount of time.

All those operations are defined framework wide. The semantic meaning of each operation is well defined and used in the same way across tools. Operations can be divided into three groups – assembly related value adding, assembly related non-value adding, and direct operations for human control. The first group focuses on building up an assembly, the second one describes additional operations that are required but do not change the product structure (like cleaning). The final group is used mostly for simulation control to provide ways of creating motion scenarios for the virtual workers. A good example is to send a worker to a specific place to the patrol neighborhood, or toggling control buttons to visualize actions required to control equipment to make a training video. For every operation there is associated set of rules that govern which operations are related to each other or must be executed in a certain order or are prerequisites for the selected operation. Tasks are logically grouping elements for combining several operations together to from one larger action. They do not need standardization and can be defined as the user wants. On company level some constraints about tasks definitions can be introduced, like using dictionaries of tasks, to ease their reading and understanding by workers or support additional information for the simulation reasoner.

2.3 Active registry – central place for context definitions

Active registry is the REST API and web-based front end for the data marketplace. It provides a central place for the AIToC platform user management, data management and storage space organization. It provides interfaces for definition of annotation templates as a central source of common knowledge that need to be synced between multiple applications belonging to the AIToC framework. It holds part types definitions, as well as serves for the backend of the knowledge organization and storage mechanism. As a central data management place, the data generated by each software tool can be attributed to projects within active registry.

2.4 Context annotation with different data formats and models

2.4.1 Data Formats and Models and their use in AIToC: An Overview

In the AIToC project, a wide variety of data serialization formats and data models are used to store, use, and exchange context information. The respective types are selected for the respective application area and the models are optimized for the information to be held.

Data Formats: For example, in REST, i.e., mostly web-based environments, the JSON data interchange format has been able to establish itself as a quasi-standard even outside the AIToC project. Another widely used data format and markup language, which is used mainly in the manufacturing sector, but is increasingly being replaced by JSON, is XML. Besides the biggest difference that JSON uses key-value pairs and XML is tree-based, both data formats have other differences. XML, for one, is older than JSON, which is why the former has more mature tools available for processing and is more commonly used in long-established software environments. XML is generally more complex and more difficult to write and read, which is because it is optimized for machine readability. JSON, on the other hand, has been optimized primarily for data exchange, which favors the integration of such data into one's own software environment. JSON and XML are text-based, which makes them rather unsuitable for high performance communication in e.g., RPC (Remote Procedure Call) environments. For this reason, more compact binary data is used in such environments.

Data Models: Data models are serialized into data formats. The choice of which format to use depends among other things on its expressiveness, needed performance and if required, ease of use. If, for example, XML or JSON, are to be compared with this in mind, there are a few differences. Apache Thrift, on the other hand, is not suitable for representing complex models but it is very compact. In case of ***geometric context information***, the data model gLTF is used in AIToC, which can be serialized to JSON or in binary. To define additional information such as constraints and semantic annotations (see Section 2.1) for use in MOSIM+ and other AIToC components, the gLTF standard has been extended to include additional JSON-based language constructs. For the definition of ***engineering related context information*** such as products, resources, processes, their internal dependencies as well as for the integration of external data such as geometry data, the standard AutomationML is used in AIToC, which is discussed in detail in Deliverable 3.1. and is widely used in the automation sector in general. AutomationML-based data is not only used as a basis for the manual and planning of processes at the respective industrial project partners, but it also serves as a basis for the automated process planning implemented in the project. For the representation of **detailed task and**

operation context information, which can be easily adapted by an end-user, a model developed in the MOSIM project and extended in AIToC is used, which was introduced in Section 2.2. As already mentioned, this model is primarily used for interaction with a user and simple exchange within the software architecture, which is why it is serialized in JSON. This data model is used for visualization and 3D simulation of worker instructions. The declarative logical programming language ASP (Answer Set Programming) is used to **define context and planning rules** or expert knowledge to generate assembly plans automatically. ASP in general and how it is used in AIToC was discussed in D2.2. RDF/RDFS and OWL are used in AIToC for the description of Natural Language information, knowledge graphs, agent models and agent behavior as well as for the integration of various context information.

Combined Use of Context Information: For example, the ***Knowledge Editor***, which can be used to define ASP-based expert knowledge, uses AutomationML and annotation information as the modeling basis for this knowledge to obtain contextual information about resources, tools, and processes. In addition, further meta information for the Natural Language-based interaction with the user is defined in OWL. The ***Instruction Viewer*** in turn uses geometry and annotation information stored in gLTF files as well as task and operations information to visualize and communicate worker instructions step by step. Another component used in AIToC to control simulated workers in 3D simulations of worker instructions and to automatically generate process plans is the agent engineering tool AJAN. It is used to implement the ***Operation Reasoner***, which needs to collect a wide variety of contextual information such as resource information, annotations, and expert knowledge to infer process plans automatically. AJAN is also used to implement agents that control individual workers in a ***MOSIM+*** simulation, which requires contextual information about the simulation environment, object annotations, and task and operation sequences.

As illustrated, the AIToC project uses a variety of software components and data that are either newly developed (e.g., *Annotation Editor*), standard in the respective domains (e.g., *AutomationML*), used for high-performance real-time processing (e.g., *MOSIM+ with Thrift*), or used for the integration and processing of heterogeneous data (e.g., *Operation Reasoner*). As an overview of data used in WP3 with their field of application, Table 1 is presented.

Through the data management tool of context information, called ***Active Registry*** (see Section 2.3), developed in WP2, all information to be distributed in the AIToC project (as shown in Table 1) is available to the individual software components. The way chosen in AIToC to provide context information to the individual software components in the most optimal representation for the respective application area, and thus using different formats and models, has several reasons. On the one hand, the task of creating a general data format and model that meets the individual requirements of all application areas is a very complex and time-consuming procedure. To realize this undertaking, a single project with exactly this task is required. Furthermore, a central data format and model would require an additional, not to be underestimated, integration effort for the respective new and already established software components and for the project partners and their optimized internal processes. In addition, the components to be used only require a specific section of context information, which is why it is also impractical to develop a general data format and model. Furthermore, it is

uncertain to what extent such a common standard will be established outside AIToC, as other domains have different requirements.

Table 1: Data Formats and Models and their use in AIToC

Application	Partner	Data Model	Data Format	Transport Layer
Model generation from real data	EKS	FMU models	Binary	AI-M Platform
	TWT			
	isb			
		AI models		
Requirements Engineering	TWT	Required model, Boilerplate	JSON	REST
			XML	ReqIF
	ifak	AutomationML model	AutomationML (XML)	File Exchange
	Eryaz			
		Modelica model	Modelica Language	
Geometry exchange	EvoBus	gLTF	JT	File Exchange
	LUT		JSON	REST
Time Series	ifak	gRPC	JSON	MQTT
	EvoBus			
	TWT			
	eks			
Task List	LUT	MOSIM-Task	JSON	REST
	EvoBus			
	DFKI			
Rule exchange	EvoBus	ASP	JSON	REST
	TWT			
	LUT			
	DFKI			
			RDF/RDFS	
MOSIM+	DFKI	MOSIM+	Thrift	Thrift
	EvoBus			
	LUT			
Data collection & (pre-) processing	TWT	raw data, processed data, models	CSV	Kafka
			JSON	
Exchange of Meta Information & models	TWT	Ontology	OWL	REST
	DFKI			
		Vocabulary	RDF/RDFS	
Reasoning	DFKI	SPARQL-BT	RDF/RDFS/OWL, JSON	REST, MQTT, THRIFT, File Exchange
		ASP		
		PDDL		
		Ontology (gLTF, MOSIM+, MOSIM-Task, AutomationML)		

Nevertheless, in certain cases, context information must be homogenized in AIToC to be able to reason about it to automatically derive plans and actions, as it happens in the ***Operation Reasoner*** and the ***MOSIM+ Agents***. For this reason, we will follow up on how the presented contextual information is homogenized and used in AJAN using Semantic Web technologies.

2.4.2 Homogenizing Context Information

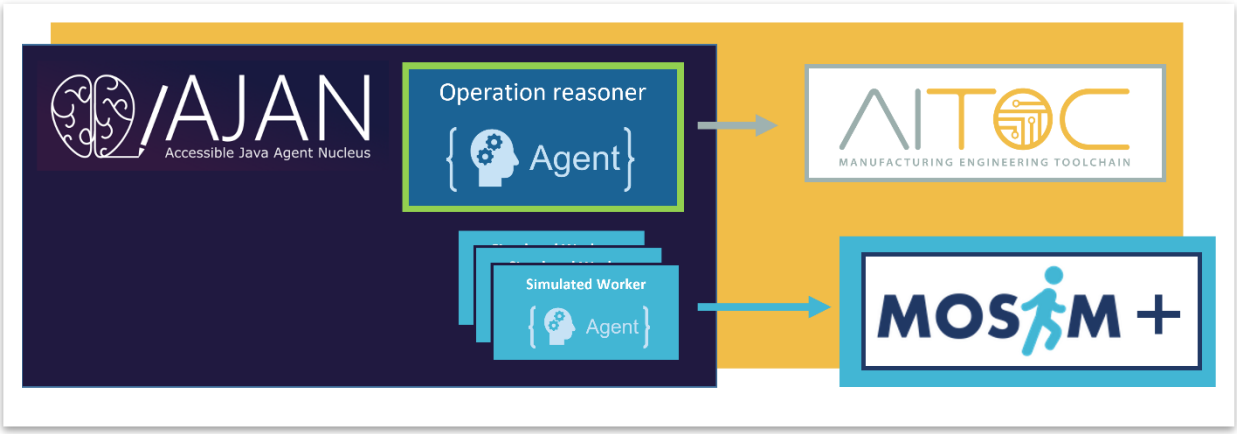


Figure 1: AJAN embedded in the AIToC Framework

As can be seen in Table 1, a wide variety of models such as ***MOSIM+***, ***AutomationML*** or ***gLTF*** annotations are used, particularly in the area of reasoning. These models must be combined or homogenized so that conclusions can be drawn about the combined model. A central reasoning component in AIToC is AJAN (see Figure 1). The internal knowledge model or knowledge graph of an AJAN agent, such as the ***Operation Reasoner*** or a **MOSIM+ agent**, is available as RDF/RDFS or OWL. Therefore, for the internal reasoning of an AJAN agent, incoming data must first be translated into RDF. As can be seen in Figure 2, e.g., the Operation

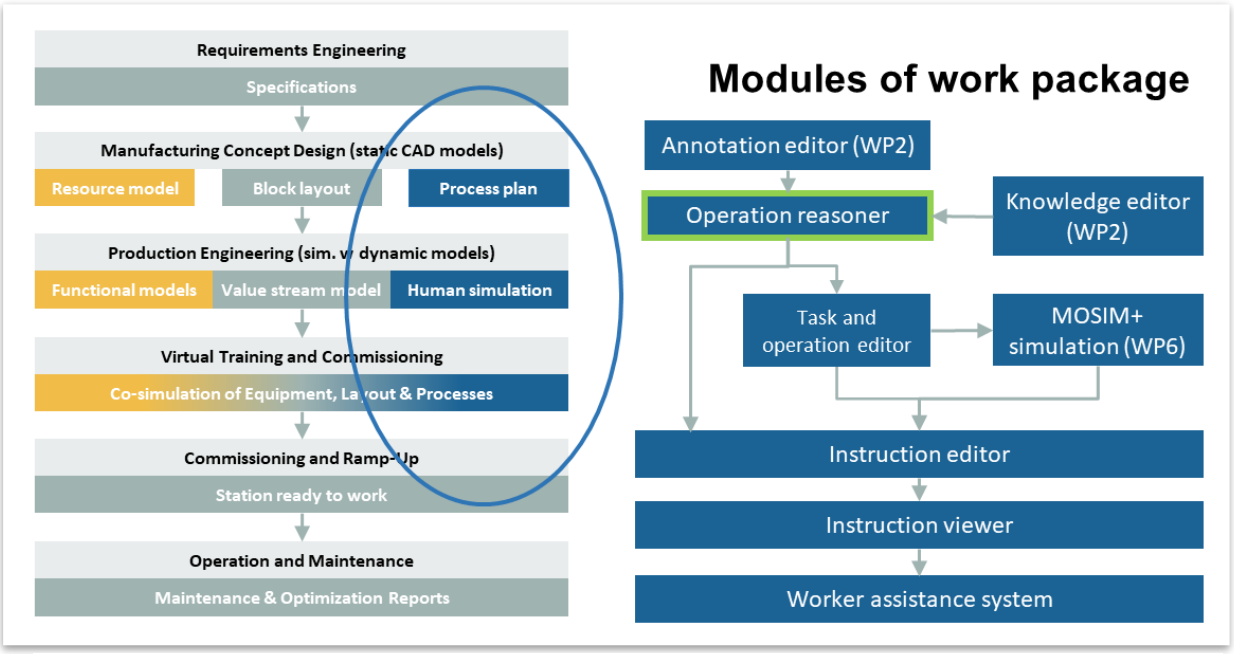


Figure 2: Operation Reasoner within AIToC WP3

Reasoner receives from different sources, different models for processing. For example, the agent receives from the Annotation Editor and assembly data in the form of ASP and gLTF data in JSON format. From the Knowledge Editor, context information is received in the form of expert knowledge in ASP. Additional context information from the requirements engineering domain is available as AutomationML data (which is XML serialized), for reasoning purposes.

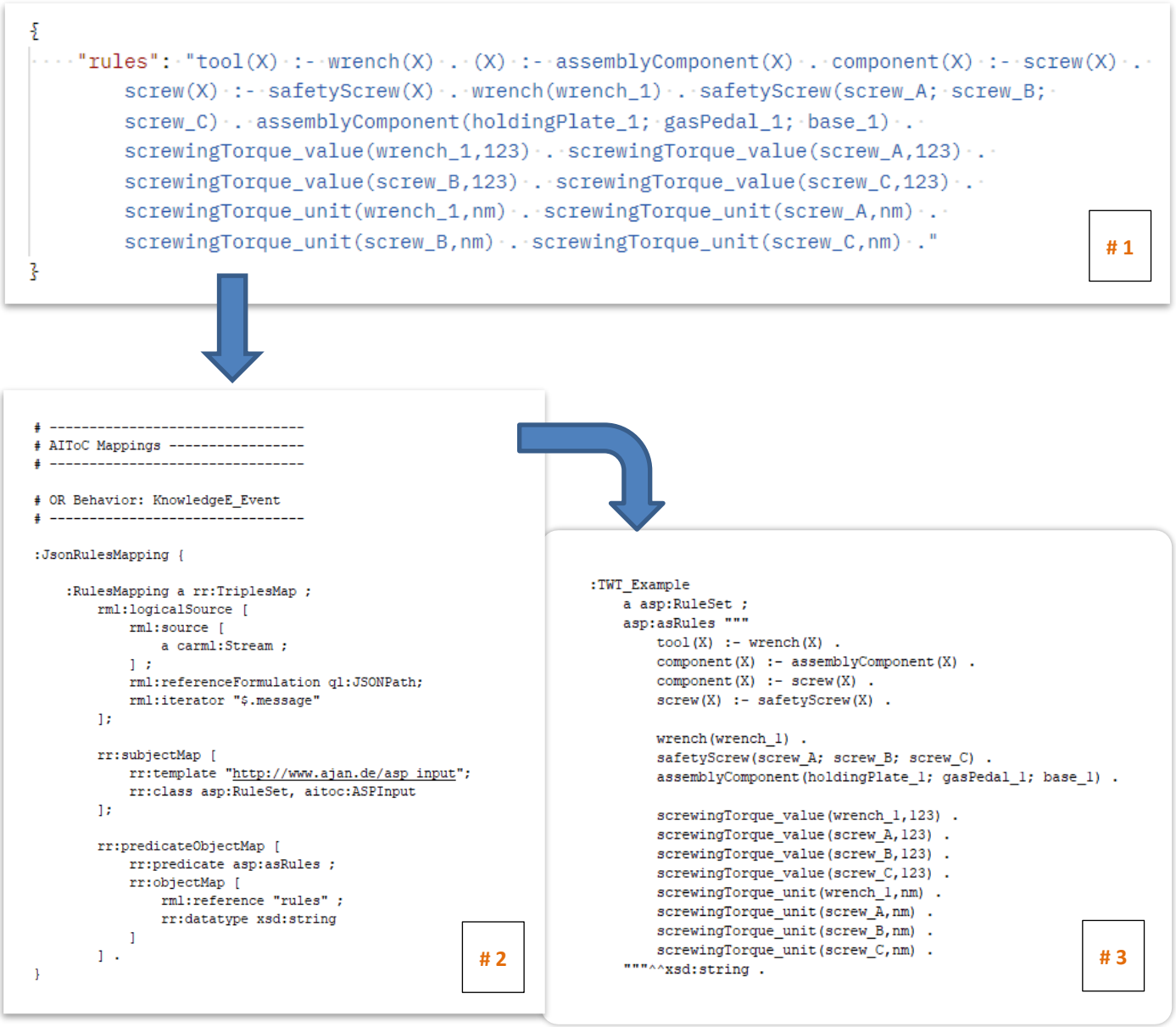
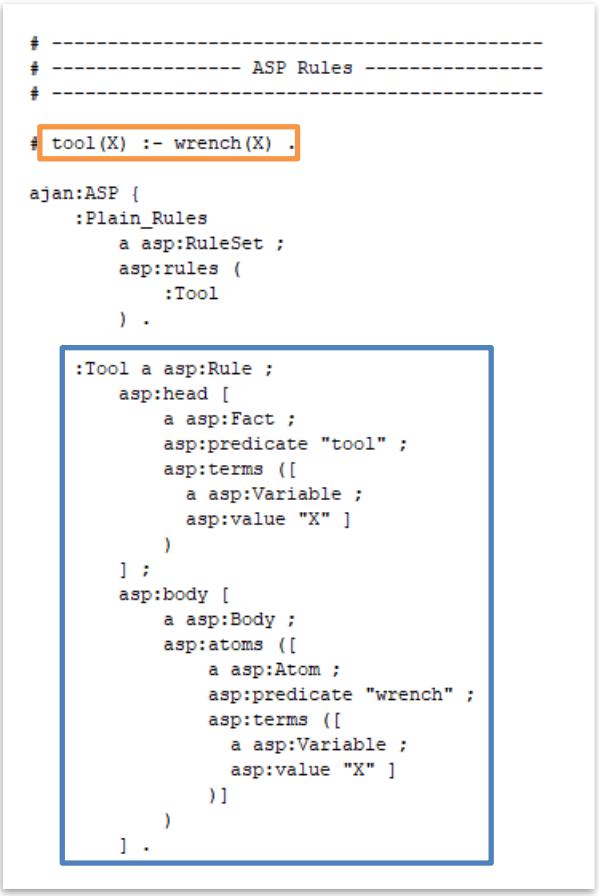


Figure 3: Mapping from incoming Expert Knowledge (from Knowledge Editor) into RDF/RDFS data

Mapping From <X> to RDF: Figure 3 shows the mapping of incoming ASP/JSON information from the Knowledge Editor into RDF/RDFS data. To translate data serialized in JSON, XML or CSV into RDF, RML mapping [21] is used. This requires an RDF-based mapping file (Figure 3, #2) that translates a JSON-based model (Figure 3, #1) into an RDF-based one (Figure 3, #3). In the mapping process, a JSONPath is mapped to an RDF element. For example, *$.message.rules* (JSONPath) becomes --> asp:asRules (RDF predicate). For translating a specific model, an



```
# ------------------------------------------
# ---------------- ASP Rules ---------------
# ------------------------------------------

# tool(X) :- wrench(X) .

ajan:ASP {
    :Plain_Rules
        a asp:RuleSet ;
        asp:rules (
            :Tool
        ) .

    :Tool a asp:Rule ;
        asp:head [
            a asp:Fact ;
            asp:predicate "tool" ;
            asp:terms ([
              a asp:Variable ;
              asp:value "X" ]
            )
        ] ;
        asp:body [
            a asp:Body ;
            asp:atoms ([
                a asp:Atom ;
                asp:predicate "wrench" ;
                asp:terms ([
                  a asp:Variable ;
                  asp:value "X" ]
                )]
            )
        ] .
```

Figure 4: ASP rules defined in RDF/RDFS using ASP vocabulary

RDF/RDFS vocabulary or an OWL ontology must be available into which it is translated. In the case of ASP data, a vocabulary developed by DFKI is used. With this vocabulary, the incoming ASP data can be stored as a string as shown in Figure 3 or as RDF-based ASP rules in a triplestore, as shown in Figure 4, where the ASP rule *tool(X) :- wrench(X)* is presented in **RDF** using said vocabulary. Even annotation information that is available in gLTF/JSON is translated into RDF via RML mapping and, depending on the reasoning used, left in RDF (in the case of SPARQL-BT-based reasoning) or translated into ASP rules (in the case of ASP-based reasoning. AutomationML data is also translated into OWL via RML mapping. In contrast to ASP, no new vocabulary or ontology had to be developed for this model. Instead, in AJAN the XML based data is translated into the official OWL-based AutomationMLOntology model. As soon as all context information required for a specific task is available for reasoning, the respective reasoning method can be executed with this homogenized data. In the case of the Operation Reasoner, ASP reasoning is primarily used, whereas SPARQL-BT and PDDL-based reasoning is used for MOSIM+ agents. For the reasoning in MOSIM+ agents, context information includes not only MOSIM-task sequences, gLTF-based annotation information, and information of the simulated MOSIM+ environment, but also the MMUs to be used. All mapping files used for the Operation Reasoner and MOSIM+ agents can be found under https://github.com/aantakli/AJAN-service/tree/master/executionservice/use-case/domains.

Mapping From RDF to <X>: After the reasoning of an AJAN agent, the result is available in RDF and needs to be translated into the data format and model required for the particular domain in order to distribute it within it. In the case of the Operation Reasoner, the ASP-reasoning result is back-translated into plain ASP rules and sent via JSON to the Task and Operation Editor as well as the Active Registry. POSER is used for the mapping from RDF to JSON. Like RML, RDF-based mapping files are also required for the mapping. The individual RDF statements are translated into JSON key-value pairs. Figure 5 shows an example of such a mapping file that translates ASP reasoning results (Figure 5, **#1**) into JSON (Figure 5, **#2**). In the case of a MOSIM+ agent, where communication takes place over Thrift, a programmatic mapping takes place that translates MMU instructions from an RDF graph and communicates with the simulation environment over Thrift. A detailed explanation is available at: https://github.com/aantakli/AJAN-service/wiki/MOSIM-Plugin

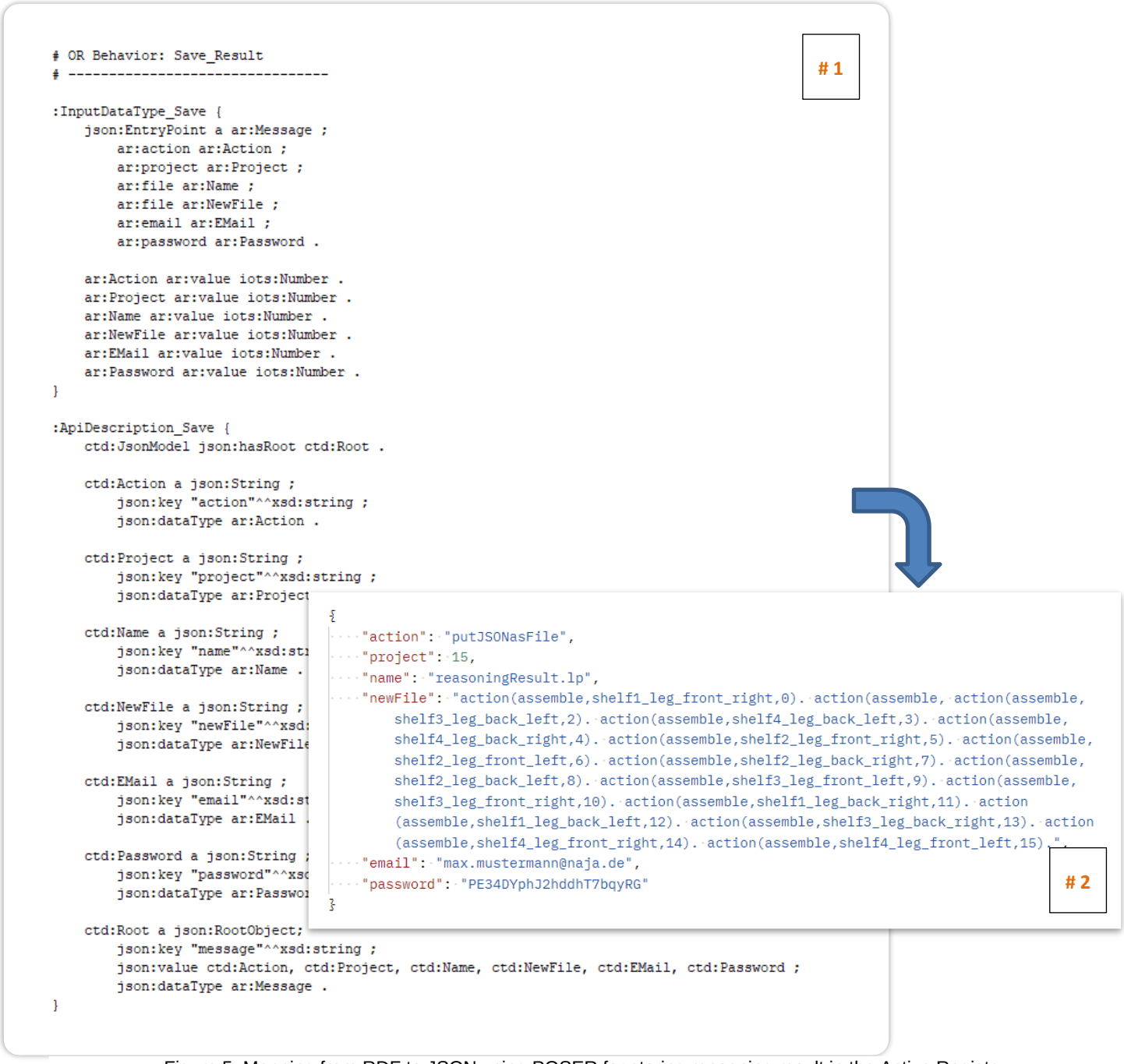


Figure 5: Mapping from RDF to JSON using POSER for storing reasoning result in the Active Registry

2.5 Instruction description

Instructions within the AIToC project can be seen as an extension of the description of tasks and operations. The latter define the correct order and steps of an assembly which can be used for simulating the assembly process as well as for instructing workers which step to execute at the assembly line. However, the task and operation description initially holds only

the minimum information necessary for executing and simulating an assembly process, such as the part, the place to be assembled and required tools. While this is sufficient for simulation purposes, for adequately instructing human workers additional information may be needed: Highlights, animations, additional documents, images or videos may hold the relevant information for a worker to fulfill his task correctly and timely.

Another aspect of worker-centered instructions is the correct granularity. Operations for simulations must include every single (small) step. For human workers, e.g., screwing operations can easily be grouped to a “tighten all screws” instruction, making sure only relevant information is conveyed.

Due to the very close interplay between tasks, operations and instructions, the standardized instruction description is integrated with the task and operation description in one common file format. This format can be used in an instruction viewer only visualizing the tasks and operations as well as visualizing previously added instructions. This way, a manual authoring step is not mandatory. In the current state of the project, this is realized using a JSON-based file structure facilitating the integration into the AIToC web-based tools. During the last project period, it is planned to moreover realize a redundant format using Automation ML.

Building instructions on tasks and operations, the first part of the instruction description holds this information:


```
{
    "Header": {
        "BaseURL": "https:\/\/active-registry.aitoc.eu\/api.php?action=getFile&project=13",
        "CadExtension": ".gltf",
        "AnnotationExtension": ".json",
        "ProjectId": 27,
        "Name": "Ikea hyllis",
        "Description": "Use case",
        "Station": "Full example"
    },
    "Instances": [{
            "Id": "490",
            "Name": "Shelf 1",
            "PartClass": "Shelf"
        }
        ...
    ],
    "Workplaces": {
        "Id": "125",
        "Name": "Full example",
        "BasePart": "494",
        "Tasks": [{
                "Id": "263",
                "Name": "legl_494_shelfl_screw_connections",
                "Description": "",
                "Operations": [{
                        "Type": "Position",
                        "Tool": "hand",
                        "Part": "490",
                        "Subassembly": "",
                        "Location": {
                            "BasePart": "494",
                            "BaseMarker": "3",
                            "Marker": "2"
                        }
                    }
                    ...
                ]
            }
        ]
    },
```


Figure 6: Task and operation part of the instruction description

The main sections contain basic information (“Header”), the underlying CAD data references (“Instances”) and the tasks and operations regarding a specific workplace (“Workplaces”). The operations moreover refer to annotations from the annotation editor, making sure the

instruction editor as well as the viewer are able to position the subassemblies correctly in 3D space.

Following this kind of information, in the second part of the actual instructions are described:


```
  "Cameras": {
    "05e93a47-ec25-41ad-aldf-16f279c32246": {
      "tags": "modified",
      "position": [
        -0.41,
        0.053,
        0.611
      ]
    }
  },
  "Instructions": [
    {
      "Name": "Position Base plate [1] with Hand",
      "Description": "",
      "Id": "084a5a45-203d-44af-be84-1e2b46b1b310",
      "Operations": [
        "d88b44a3-274b-47cb-b8dd-7b4dd38fce20"
      ],
      "Views": [
        {
          "Id": "5f32c79b-483a-4b58-b2b3-754c22acf950",
          "Name": "Default View for Position Base plate [1] with Hand",
          "Operations": [
            "d88b44a3-274b-47cb-b8dd-7b4dd38fce20"
          ],
          "Cameras": [
            "05e93a47-ec25-41ad-aldf-16f279c32246"
          ],
          "Visuals": [
            "31e1a747-d98e-41d4-92f9-ae7228ab7e53"
          ],
          "PartVisibilityOverrides": {}
        },
        ...
      ]
    }
  ],
  "Visuals": [
    {
      "Name": "New Highlight Visual",
      "Id": "a6289f0b-901c-439d-824a-08efb57706c3",
      "LocalPartId": "Screw4.A6290000530_Gas_Pedal.N910105008016, 0004.001, SEC..."
      "Type": "Highlight",
      "Position": {
        "x": 0.050192008328629856,
        "y": 0.006151425694486845,
        "z": 0.006823663421304804
      },
      "ColorA": 1,
      "ColorR": 0.9647058823529412,
      "ColorG": 0.10196078431372549,
      "ColorB": 0.10196078431372549
    }
  ]
}
```


Figure 7: Instruction part of the description

Firstly, camera positions defined in the instruction editor are specified (“Cameras”). The “Instructions” set contains a number of instructions. Each instruction references 1-n operations and 1-n views, each containing 1-n cameras, a subset of the operations and 0-n visuals. The views are the central element of an instruction: They are meant as one view for the human worker containing a view position (the camera), the set of operations to describe and a number of visuals to clarify the task. Visuals can be seen as annotations that are located in the 3D space at the current state of the assembly. The type of a visual can be a text, a URL, an image, a video, a 3D model or an animation. The following overview helps to understand the instruction data hierarchy:

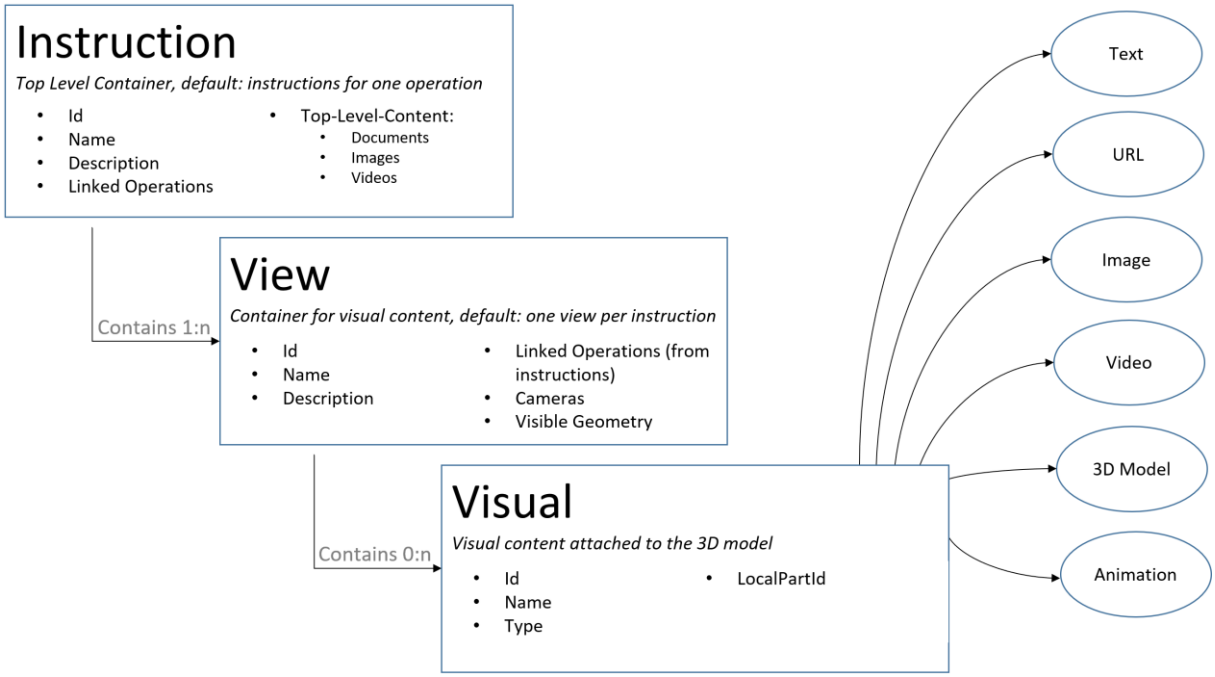


Figure 8: Hierarchy of the instruction description

2.6 Communication between viewer and assistance system for viewer control

2.6.1 Phase I: Initialization of Contextual Data

In the initialization phase, three categories of contextual data are integrated: Human-Centered Context (HCC), Task-Centered Context (TCC), and Environmental-Centered Context (ECC). In this phase, the viewer communicates with the Worker Support System to identify the user's specific work station. This exchange of information enables the viewer to retrieve task-specific data. Within the Task-Centered Context, variables such as task type, task specification, and task progress are considered. The Environmental-Centered Context incorporates variables related to the layout and object data.

Following the data initialization, the user profile selection process is initiated to optimize the user experience. Upon login, the data platform customizes the interface by presenting templates that are specific to the user's profile. In the Human-Centered Context, preferences related to the color scheme are included. Additionally, qualifications are considered to define the level of support needed and to provide customized instructions.

2.6.2 Phase II: Real-time Adaptation of Visualization Data

After completing the initialization, the system transitions to a state of real-time adaptation. During this phase, the Worker Support System continuously monitors activities and updates the viewer with notifications upon the completion of actions or when deviations from the plan are detected. Specifically, the Task-Centered Context is updated to reflect the current task progress.

Alongside these automated system updates, different options for user interaction are available. Users have the option to engage with the system through explicit and implicit means. In terms of explicit interactions, direct voice and touch commands are supported, known as ICC Explicit Interaction. On the other hand, implicit interactions utilize data gathered from Mimetik data gloves, categorized as ICC Implicit Interaction.

Explicit Interaction via Voice Commands

Explicit interactions employ voice commands as an option of user input. Voice commands are systematically translated into standard messages by the Worker Support System. Once translated, the subsequently publishes those commands as messages to viewers. Feedback mechanisms are also in place; the viewer issues a confirmation message upon the successful execution of a command. Additionally, the viewer uses a descriptor file, typically in YAML format, to communicate its range of supported commands and capabilities. This descriptor file is initially transmitted upon the viewer's first registration with the WSS.

Implicit Interaction via Mimetik Data Gloves

In addition to explicit interactions via voice commands, Mimetik Data Gloves facilitate implicit interactions by processing and converting raw motion data into standard messages. These messages are interpreted as specific types of operations by the Mimetik MiTracker software. Future iterations of the system are anticipated to incorporate gesture recognition capabilities as part of ongoing research.

Feedback mechanisms are similar to those for explicit interactions. The viewer issues a confirmation message upon successful command execution and employs a descriptor file, typically in YAML format, to communicate its range of supported commands. This descriptor file is initially sent upon the viewer's first registration with the WSS.

Architectural notes:

The figure below illustrates the communication infrastructure based on the MQTT protocol. An MQTT broker acts as the central hub for message exchange. Context providers act as MQTT publishers and are responsible for providing context information. On the other hand, MQTT subscribers, including the 3D viewer and video player, act as components that evaluate the received context and perform appropriate actions. This architecture allows the components to communicate seamlessly and exchange information in real time, resulting in efficient and responsive interaction.

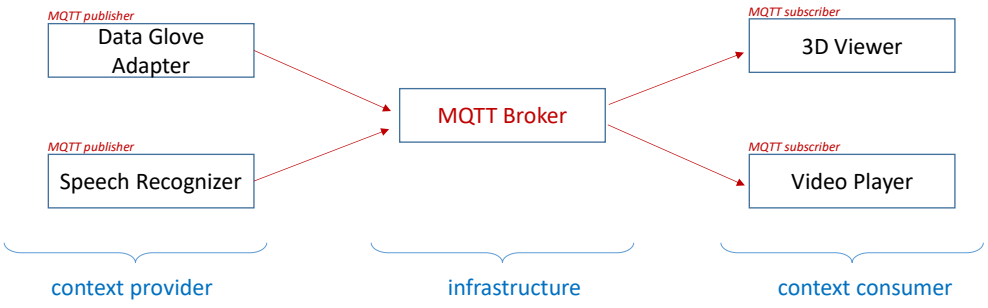


Figure 2: Architecture of the communication infrastructure for context providers and consumers

In the following sections we will look at two important aspects of the communication infrastructure. First, we will look at the configuration of the context providers, which is essential to ensure their functionality. Then we will explain the structure and coding of the messages to be transmitted in more detail in order to gain an understanding of the data exchange within the system.

The **data glove adapter** serves as a bridge between a data glove and an MQTT broker. It utilizes gRPC to communicate with the data glove on one side, facilitating data exchange. The adapter's **configuration** is divided into three sections and each section has parameters:

- **header**: This section contains meta-information used to enhance log messages for better readability.
  - **title**: This is the component name, usually “MPU Adapter”, where MPU stands for Mimetik Processing Unit and Mimetik is the manufacturer of the data gloves.
  - **description**: a short description of the adapter.
- **client**: Here, the gRPC client's configuration details are specified, enabling seamless communication with the data glove.
  - **address**: The "address" parameter in the configuration specifies the location or network address of the data glove, indicating where it can be reached.
  - **port**: The "port" parameter defines the specific port number through which the data glove communicates using the gRPC protocol.
- **broker**: This section defines the configuration settings for the MQTT broker, allowing the adapter to publish messages to the broker.
  - **address**: MQTT broker host address.
  - **port**: MQTT broker port number.
  - **client_id**: Identification of the adapter at the broker.
  - **username**: Part of the credentials used to access the broker.
  - **password**: The other part of the credentials required for broker access.
  - **topic**: The MQTT topic which the data glove adapter uses to publish information about current actions of the worker.

An example of the data glove adapter configuration in YAML format is as follows:

```
header:
  title: MPU Adapter
  description: This is a gRPC client providing MPU actions to an MQTT broker.
client:
  address: "169.254.30.47"
  port: "50051"
broker:
  address: "127.0.0.1"
  port: "1883"
  client_id: mpu_adapter_001
  username: guest
  password: guest
  topic: "evobus/pilotstation/dataglove"
```

YAML (YAML Ain't Markup Language) is a human-readable data serialization format commonly used for configuration files. It uses indentation and colons to represent data structures and

key-value pairs, making it easy to read and write. YAML is often used as an alternative to JSON (JavaScript Object Notation) and shares a similar relationship with JSON, as they both serve as data interchange formats. However, YAML provides a more concise and human-friendly syntax compared to the more verbose and machine-oriented nature of JSON.

The adapter serves as the intermediary, supplying messages to the MQTT broker, which, in turn, disseminates them to all its subscribers. The payload of each message follows a specific structure, ensuring uniformity and ease of data interpretation. This standardized structure allows efficient communication and seamless integration between the adapter and the broker, enabling smooth data transmission to all interested parties.

Here's a description of all **payload** parameters:

- **hdr**: An object containing header information for the message.
    - **sessionId**: A unique identifier representing the session when the message was generated (in our implementation we use a string providing time of opening the session in ISO 8601 date-time format).
    - **reqId**: A numerical value indicating the request ID associated with the message. A single call to the gRPC server is answered with multiple return values sent at different times (a similar approach to WebSockets). We use the time of initiating the call. The encoding is UNIX epoch, the number of seconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC) on January 1, 1970.
- **processId**: A numerical value specifying the process ID.
- **timeStamp**: A numerical value representing the timestamp when the message was generated (UNIX epoch, but in milli-seconds).
- **actionId**: A numerical value representing the ID of the action performed. This is the original code sent by the Mimetik data glove. The value range is described below.
- **actionName**: A string indicating the name or type of the action performed, as defined by the manufacturer of the data glove (Mimetik).
- **err**: A numerical value indicating the error code associated with the message. A value of 0 typically means no error occurred. The error code is defined by the manufacturer of the data glove (Mimetik).
- **aitocActionId**: A numerical value representing the ID of the action performed. This is the AIToC specific action code to be understood by other AIToC applications. The mapping is given by a value table below.
- **aitocActionDescription**: A string indicating the description of the action performed, as defined by the AIToC consortium.

“err” and “actionID” require more explanation. Only a gRPC interface was available describing codes and numeric values. In the following descriptions of the value ranges for those two parameters, the codes and values are valid but the descriptions reflect our assumptions of the meanings of the codes without knowing the details of the data gloves:

- err
    - eSUCCESS (0): The operation was successful.
    - eNO_DEVICE_FOUND (1): No data glove device was found.

  - eNO_MODEL_FOUND (2): The required artificial neural network model was not found.
  - eNO_DATA_IN_DB (3): There is no data available in the database for evaluation.
  - eCANNOT_EXECUTE_FEATURE (4): The data evaluation feature cannot be executed.
  - eDATA_SIZE_MISMATCH (5): The size of the data does not match the expected format.
  - eNO_PERSON_FOUND (6): No person was found or recognized.
  - eCANNOT_ADD_SESSION (7): It's not possible to add a new session for data evaluation.
  - eCANNOT_DELETE_SESSION (8): The session for data evaluation cannot be deleted.
  - eNO_SESSION_FOUND (9): The specified session for data evaluation was not found.
  - eNO_SKILL_NOT_MATCH_WITH_DEVICE (10): The user's skill level does not match the required skill level for the data glove device.
  - eCANNOT_ADD_PROCESS (11): A new data evaluation process cannot be added.
  - eCANNOT_REMOVE_PROCESS (12): An existing data evaluation process cannot be removed.
- actionId:
  - eRandom (0): Represents non-predetermined movements.
  - eToGet (1): "to fetch or get something".
  - eToPosition (2): "to position or place something".
  - eToTightenLoose (3): "to tighten loosely".
  - eToTighten (4): "to tighten firmly".
  - eToCutToLength (5): "to cut something to length".
  - eToPutAway (6): "to put something away".
  - eToPlug (7): "to plug or insert something".
  - eToEquip (8): "to equip or prepare something".
  - eToCarryOut (9): "to carry out or perform something".
  - eToInstall (10): "to install or lay something, e.g., a cable harness".
  - eToFix (11): "fix or secure something".
  - eToGlue (12): "to glue or adhere something".
  - eToTightenWithTorque (13): "to tighten with torque".
  - eToGrease (14): "to grease or lubricate something".
  - eToApplyAdhesive (15): "to apply adhesive or glue".

This JSON payload provides a structured and comprehensive representation of the message, with header information, action details, and error status, facilitating clear and consistent communication between the adapter and the MQTT broker, ensuring seamless distribution to all subscribers.

To enable other AIToC applications to utilize data glove actions, a mapping between the manufacturer's "actionId" and the project's "aitocActionId" as introduced in section 2.2 is

established by AIToC project members. This allows for seamless integration and harmonized communication between the data glove and AIToC applications:

actionId	aitocActionId	aitocActionDescription
0	0	Unknown action
1	29	Remove
2	9	Position
3	12	Tighten loose
4	13	Tighten fully
5	24	Cut to length
6	29	Remove
7	18	Insert part
8	28	Assemble
9	28	Assemble
10	39	Lay cable
11	67	Clamp
12	20	Glue
13	14	Tighten with torque
14	32	Grease
15	20	Glue

An example of a payload is given by the following JSON code:


```
{
 "hdr": {
  "sessionId": "2023-07-25T17:14:42.375423",
  "reqId": 1690298082
 },
 "processId": 1,
 "timeStamp": 1690298085426,
 "actionId": 2,
 "actionName": "position",
 "err": 0,
 "aitocActionId": 9,
 "aitocActionDescription": "Position"
}
```


The payload structure is not optimized for speed and data efficiency but for human readability because it was designed for research purposes and debugging of the applications. Of-course, the payload could be reduced to a structure like {timeStamp, actionId, err}.

The following sections are related to a configuration file, which is utilized by a speech recognizer. It enables configuration of actions and their corresponding IDs. This file plays a pivotal role in tailoring the **speech recognizer** to the specific needs of the user or application (e.g. 3D viewers or video players). By employing a YAML format, the configuration file offers

a human-readable and easily editable structure. Through this file, users can define custom actions and assign unique IDs, allowing the speech recognizer to accurately identify and execute the desired actions based on recognized voice commands. The precise layout and syntax of the YAML configuration is described in the following:

- **header**: Contains meta-information about the speech recognizer.
  - **title**: The title of the application.
  - **description**: A brief description of the application, highlighting its purpose as an MQTT client for processing command IDs.
- **broker**: Defines the MQTT broker configuration details.
  - **address**: The IP address of the MQTT broker.
  - **port**: The port number through which the MQTT broker is accessed.
  - **client_id**: The unique client ID for the MQTT client.
  - **username**: The MQTT broker username.
  - **password**: The MQTT broker password.
  - **topic**: This attribute is of no relevance and should have the value "none". It stems from a generic application structure and should be removed in production systems.
- **application**: Specifies application-specific details.
  - **topic**: The MQTT topic to which the speech recognizer subscribes for receiving commands.
  - **commands**: A list of different commands that the speech recognizer can recognize and execute.

Each command in the commands list consists of:

- **id**: A numerical value representing the unique ID of the command.
- **title**: A descriptive title for the command, providing context for what the command does.
- **voice_commands**: A mapping of voice commands for different languages (e.g., "de" for German and "en" for English). Each language has a list of voice command variations that users can say to trigger the associated action.

Additionally, here is a complete example of the configuration file structure:

```
header:
  title: vplay - A video player
  description: This is an MQTT client, which receives and processes command IDs.
broker:
  address: "127.0.0.1"
  port: "1883"
  client_id: vplay_001
  username: guest
  password: guest
  topic: "none"
application:
  topic: "evobus/pilotstation/vplay"
  commands:
    - id: 1
      title: play the video from current position
```

```
  voice_commands:
    de:
      - start
      - anfangen
      - fang an
      - beginnen
      - beginne
      - weiter
      - fortsetzen
      - setze fort
    en:
      - start
      - start video
      - continue
      - continue video
      - resume
      - resume video
- id: 2
  title: stop the video play back
  voice_commands:
    de:
      - halt
      - stopp
      - stoppen
      - anhalten
    en:
      - stop
      - stop video
      - hold on
- id: 4
  title: jump in the video stream <number> seconds forward
  voice_commands:
    de:
      - <number> sekunden vorwärts
      - <number> sekunden vor
    en:
      - <number> seconds forward
      - <number> seconds ahead
```

This is the configuration file for the speech recognizer that processes voice commands and publishes messages to an MQTT broker. The MQTT broker is located at IP address "127.0.0.1" and port "1883". The client ID for the MQTT client is "vplay_001", and it uses the username "guest" with the password "guest" for authentication.

According to this example configuration, the application listens to the topic "evobus/pilotstation/vplay" and recognizes specific voice commands for two commands:

- Command ID: 1 - "Play the video from the current position"
    - Supported voice commands in German: "start", "anfangen", "fang an", "beginnen", "beginne", "weiter", "fortsetzen", "setze fort"

  - Supported voice commands in English: "start", "start video", "continue", "continue video", "resume", "resume video"
- Command ID: 2 - "Stop the video playback"
  - Supported voice commands in German: "halt", "stopp", "stoppen", "anhalten"
  - Supported voice commands in English: "stop", "stop video", "hold on"
- Command ID: 4 - "Jump in the video stream <number> seconds forward "
  - Supported voice commands in German: "<number> sekunden vorwärts", "<number> sekunden vor"
  - Supported voice commands in English: "<number> seconds forward", "<number> seconds ahead"

The configuration file structure enables easy customization of the speech recognizer, allowing users to define their own set of new commands and their respective voice variations, tailoring the system to recognize and execute desired actions based on recognized voice inputs.

A first example of the payload of the speech recognizer is as follows:

```
{
 "id": 1,
 "title": "play the video from current position",
 "text": "start video",
 "number": null
}
```

We can describe this payload as follows:

- **id**: An integer field that represents the command ID. In this example, the command ID is 1, which corresponds to the "play the video from the current position" command.
- **title**: A string field that provides a descriptive title for the command. In this case, it describes the action of playing the video from the current position.
- **text**: A string field that contains the voice command text. In this example, the voice command is "start video," which is one of the recognized voice commands for the "play" action.
- **number**: Since this command (ID 1) doesn't require a numeric value, the number field is set to null.

A second example demonstrates on how to get back spoken numbers:

```
{
 "id": 4,
 "title": "jump in the video stream <number> seconds forward",
 "text": "10 seconds forward",
 "number": 10
}
```

Since this command (ID 4) requires a numeric value (the number of seconds to jump forward), the number field is set to 10, representing the specified number of seconds.

The payload structure is not optimized for speed and data efficiency but for human readability because it was for research purposes and debugging of the applications. Of-course, the payload could be reduced to a structure like {id, number}.

3 Summary and next steps

The research document proposes a context description format harmonized in the project consortium. It uses JSON due to its various advantages, including human-readable format and wide language support. The document suggests utilizing the generic 100% set of context parameters outlined in D 3.1 as a comprehensive solution. For each specific use-case, relevant context parameters should be implemented, enabling a tailored approach to capture the required context information.

It emphasizes that the choice and individual implementation of context parameters, such as the selection of sensors and data sources, is company-specific, domain-specific, workplace-specific, and use-case specific. This approach allows for flexibility and adaptability based on unique requirements.

The document highlights the significance of process complexity in ensuring reliable context-capturing. Higher complexity is associated with a medium context capturing rate, while lower complexity results in a higher context capturing rate. For instance, in a task tracking scenario with data gloves, manual effort increases with higher complexity (e.g. assembly of individual products), but with a mass production with recurring activities, the automatic context capturing rate is high.

The future work within the AIToC Project involves creating a generic example of context-aware worker assistance, specifically focusing on the assembly of an Ikea Hyllis Shelf. The objective is to implement and demonstrate the entire toolchain/workflow in action, showcasing how context information can be effectively captured and utilized in a real-world scenario.

4 References

[1] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, und P. Steggles, „Towards a Better Understanding of Context and Context-Awareness“, in *Handheld and Ubiquitous Computing*, Springer, Berlin, Heidelberg, 1999, S. 304–307. doi: 10.1007/3-540-48157-5_29.

[2] A. K. Dey, „Understanding and Using Context“, *Pers. Ubiquitous Comput.*, Bd. 5, Nr. 1, S. 4–7, 2001.

[3] H. Bubb, H. Müller, A. Schubö, G. Rigoll, F. Wallhoff, und M. F. Zäh, *CoTeSys Progress Report 2008: ACIPE - Adaptive Cognitive Interaction in Production Environments*. 2008.

[4] K. Doerr und A. Arreola-Risa, „A worker-based approach for modeling variability in task completion times“, *IIE Trans.*, Bd. 32, S. 625–636, 2000.

[5] S. Hinrichsen, D. Riediger, und A. Unrau, „Assistance Systems in Manual Assembly“, in *Production Engineering and Management*, Lemgo, 2016.

[6] M. Aehnelt und S. Bader, „From Information Assistance to Cognitive Automation: A Smart Assembly Use Case“, in *Agents and Artificial Intelligence*, B. Duval, J. van den Herik, S. Loiseau, und J. Filipe, Hrsg., in Lecture Notes in Computer

Science. Cham: Springer International Publishing, 2015, S. 207–222. doi: 10.1007/978-3-319-27947-3_11.

[7] J. Grubert, T. Langlotz, S. Zollmann, und H. Regenbrecht, „Towards Pervasive Augmented Reality: Context-Awareness in Augmented Reality“, *IEEE Trans. Vis. Comput. Graph.*, Bd. 23, Nr. 6, S. 1706–1724, 2017, doi: 10.1109/TVCG.2016.2543720.

[8] E. Lampen, J. Lehwald, und T. Pfeiffer, „A Context-Aware Assistance Framework for Implicit Interaction with an Augmented Human“, in *Virtual, Augmented and Mixed Reality. Industrial and Everyday Life Applications*, J. Y. C. Chen und G. Fragomeni, Hrsg., in Lecture Notes in Computer Science, vol. 12191. Cham: Springer International Publishing, 2020, S. 91–110. doi: 10.1007/978-3-030-49698-2_7.

[9] M. D. Mura, G. Dini, und F. Failli, „An Integrated Environment Based on Augmented Reality and Sensing Device for Manual Assembly Workstations“, *Procedia CIRP*, Bd. 41, S. 340–345, 2016, doi: 10.1016/j.procir.2015.12.128.

[10] L. Rodriguez, F. Quint, D. Gorecky, D. Romero, und H. R. Siller, „Developing a Mixed Reality Assistance System Based on Projection Mapping Technology for Manual Operations at Assembly Workstations“, *Procedia Comput. Sci.*, Bd. 75, S. 327–333, Jan. 2015, doi: 10.1016/j.procs.2015.12.254.

[11] M. Funk, T. Dingler, J. Cooper, und A. Schmidt, „Stop helping me - I’m bored!: why assembly assistance needs to be adaptive“, in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers - UbiComp ’15*, Osaka, Japan: ACM Press, 2015, S. 1269–1273. doi: 10.1145/2800835.2807942.

[12] M. F. Zaeh, M. Wiesbeck, S. Stork, und A. Schubö, „A multi-dimensional measure for determining the complexity of manual assembly operations“, *Prod. Eng.*, Bd. 3, Nr. 4–5, S. 489–496, 2009, doi: 10.1007/s11740-009-0171-3.

[13] J. A. Diego-Mas und J. Alcaide-Marzal, „Using Kinect™ sensor in observational methods for assessing postures at work“, *Appl. Ergon.*, Bd. 45, Nr. 4, S. 976–985, 2014, doi: 10.1016/j.apergo.2013.12.001.

[14] U. Côté-Allard *u. a.*, „Deep Learning for Electromyographic Hand Gesture Signal Classification Using Transfer Learning“, *ArXiv180107756 Cs Stat*, 2018, Zugegriffen: 10. August 2018. [Online]. Verfügbar unter: http://arxiv.org/abs/1801.07756

[15] H. Koskimaki, V. Huikari, P. Siirtola, P. Laurinen, und J. Roning, „Activity recognition using a wrist-worn inertial measurement unit: A case study for industrial assembly lines“, in *2009 17th Mediterranean Conference on Control and Automation*, Thessaloniki, Greece: IEEE, 2009, S. 401–405. doi: 10.1109/MED.2009.5164574.

[16] E. Binder, M. Romer, P. Engesser, und J. Lehwald, „Extensible Worker Assistance (EWA): Presenting a Comprehensive Framework for Context-Aware Assistance

in Manual Assembly“, *Procedia CIRP*, Bd. 112, S. 501–506, 2022, doi: 10.1016/j.procir.2022.09.055.

[17] A. Bannat *u. a.*, „Towards Optimal Worker Assistance: A Framework for Adaptive Selection and Presentation of Assembly Instructions“, *1st Intern Cotesys Workshop 2008*, S. 7, 2008.

[18] G. Westerfield, A. Mitrovic, und M. Billinghurst, „Intelligent Augmented Reality Training for Motherboard Assembly“, *Int. J. Artif. Intell. Educ.*, Bd. 25, Nr. 1, S. 157–172, 2015, doi: 10.1007/s40593-014-0032-x.

[19] B. Alkan, D. Vera, M. Ahmad, B. Ahmad, und R. Harrison, „A Model for Complexity Assessment in Manual Assembly Operations Through Predetermined Motion Time Systems“, *Procedia CIRP*, Bd. 44, Nr. Supplement C, S. 429–434, 2016, doi: https://doi.org/10.1016/j.procir.2016.02.111.

[20] T. Faessberg, A. Fasth, F. Hellman, A. Davidsson, und J. Stahre, „Interaction between Complexity, Quality and Cognitive Automation“, in *Proc. 4th CIRP Conference on Assembly Technologies and Systems (CATS 2012), Ann Arbor, 21-22 May 2012*, Amsterdam: Elsevier Procedia, 2012.

[21] RDF Mapping Language (RML), Unofficial Draft 06 October 2020. https://rml.io/specs/rml/