Artificial Intelligence supported Tool Chain in Manufacturing Engineering

ITEA 3 – 19027

Work package 6

Integration of plug & play modules and tools

Deliverable 1

Requirements for integration functions and architecture

Document type	: Deliverable
Document version	: 1
Document Preparation Date	: 2022-11-16
Classification	: public
Contract Start Date	: 2021-01-01
Contract End Date	: 2024-02-29

Final approval	Name	Partner
Review Task Level	Sabino Roselli	Chalmers
Review WP Level	Gamze Kecibas	Ford Otosan
Review Board Level	Klaus Fischer	DFKI

Executive Summary

The aim of the AIToC platform is to create a modular architecture with the implementation of the basic modules demonstrating its fitness in the use cases provided by the industrial partners. To maintain interoperability between systems, standard data exchange formats must be supported. In addition, internal data exchange within the framework should at least follow the framework wide internal standard, that needs to be clearly described to allow extensibility of the framework with any number of modules. A distributed architecture is preferred to allow leveling of the computational and storage load between multiple systems, and the deployment in cloud environments, which provides scalability. This document brings in requirements from the project partners towards the AIToC framework, that should be satisfied in order to create a comprehensive tool, tackling key manufacturing industrial problems. Deliverable 6.4b will at the end of the project summarize the use case realization utilizing the framework and its modules, demonstrating final implementation results of the herein stated requirements.

Content

Executive Summary 3
1 Introduction 5
2 Requirements for integration 5
2.1 Interoperability with other systems 5
2.2 Simple and compact data exchange within framework components 5
2.3 Integration to existing Product Data Management (PDM) systems 6
2.4 Modularity 6
3 Framework components and architecture 6
3.1 Data marketplace 6
3.2 PAIP platform 7
3.3 Task and operation planning and reasoning 7
3.4 Layout planning 8
3.5 Overall architecture of the AIToC platform 9
3.6 Framework to world communication 10
3.7 User interfaces 11
4 Summary and conclusions 11

1 Introduction

This document summarizes technical requirements from the use case providers towards the AIToC framework. Requirements are divided into integration related and architecture ones. The Core focus on WP6 is the integration and homogenization of architectures defined in other use cases to form one common platform. Requirements are the baseline for those integration actions and for key achievements definition for project progress reporting.

2 Requirements for integration

2.1 Interoperability with other systems

The AIToC platform is designed to provide connectivity internally between its modules. Interoperability with other systems should be provided through adaptation of open data formats that are standardized. The car industry requires interoperability with AutomationML, software developers appreciate JSON format for message data exchange as well as transactional information exchange. We have selected glTF as main solution for CAD data exchange. In addition to single file exchange operations, network connectivity is required for online data exchange between software modules and the AIToC platform. REST interface will be adopted to accommodate need for short request-response transactions, and MQTT and web sockets will be utilized for message passing that can be bandwidth intensive. A messaging system need to support subscription to topics and publishing data within topics to alleviate many-to-many routing.

2.2 Simple and compact data exchange within framework components

Based on discussions in the project meetings we have decided to support primarily JSON format as inter framework communication data exchange format. The decision is based on the human readability of the format, extensibility that does not break backward compatibility, and availability of parsing modules for any programing language that would be practically used to create framework modules. In addition, this format allows seamless integration with the CAD data representation format that is glTF, which is also based on JSON. As described in deliverables from work package two, the glTF has been selected as standard CAD data exchange for the use within AIToC framework. The decision is dictated by the optimization of the format for the network and Internet use and wide adaptation of the format within tools for creating 3D web interfaces. In addition, the XML-based AutomationML format will be utilized for specific modules that target communication of the framework with the outside components and systems. This is dictated by the need to adhere to the automotive industry requirements. Task and operation reasoning which is the scope of WP3 will utilize common data format based on JSON to describe:

- Parts annotations – that brings semantic meaning to the geometrical features of individual parts, assemblies, and components.
- Assembly definitions.
- Tasks and operations.

- Instructions based on tasks and operations and including links to interactive content and supportive content like sound, video, images, or documents.

2.3 Integration to existing Product Data Management (PDM) systems

The integration layer of the AIToC framework should be based on a single master module that will facilitate communication within the framework using internal standard and communication with the external systems through one end point. This is required to limit integration effort of the framework and internal PDM and PLM systems in factories to build a single data exchange module between framework data layer and framework components, without need to any modification of the later ones. It is required that at least one common data platform is defined within the project demonstrating use of the AIToC registry as the top framework management layer so that the framework would be deployment ready with fully functional backend for data management and storage. Documentation for extension or replacement of the open-source data marketplace with industrial PLM/PDM systems must be provided as one of the key results of this project.

2.4 Modularity

AIToC framework will be designed in a modular fashion with extendibility in mind. To achieve that goal clear data exchange between modules must be defined with preferably uniform API allowing service discovery, subscription, dependency, maintenance, and resolution. Modules of the framework should be isolated to the extent that replacement of a single module should be a simple operation. If certain module uses dependent modules, mechanism for resolving those dependencies need to be implemented in the framework to automatically install required submodules. Connectivity between modules should preferably be based on a network protocol to allow distributing modules between separate machines without requiring all the modules to be working on a single system. Data exchange should be possible between all the modules. User access with a single sign-in mechanism should be implemented to control access to resources and data, based on user and group policies. Modules that require user interface should all be based on the same platform to allow seamless integration of user interfaces to a common system. It would be desired to create a single web-based user interface platform for all the modules, where each user interface would be a plugin for the main user interface system. This would simplify deployment of the AIToC platform. Modules should be packaged so that it is easy to move them from one system to another and install in the framework. Preferably new modules should integrate into all tools that can utilize them using service discovery system. User interface available outside a single computer must provide secure connection to maintain passwords and data secrecy.

3 Framework components and architecture

3.1 Data marketplace

To facilitate user management and access rights management, a single sign-in system needs to be part of the data marketplace integration layer. Keycloak – a free and open-source module has been selected as a major candidate for this job. Integration of the single sign-in

mechanism for all tools in the web-based platform utilizing OpenID and OAuth2 standards will enable easy integration of existing tools for data management, database access, storage management without need for developing our own user management system.

Data marketplace will be directly connected to the Active registry data access layer and the frontend for the framework to enable seamless transitions between direct access to platform using web-based user interface and transactional access channels based on the same API for the component communication with the framework.

3.2 PAIP platform

We have identified the need for data processing as part of data collection and usage scenarios. The sources for digital information are very broad and use of information often requires data preconditioning, filtering, transformation, or aggregation before useful output is obtained. As those operations are repetitive and can be templated for individual use cases, the idea of the PAIP platform came to life. A graphical user interface allowing to map data inputs, processing blocks, and data outputs, is required to enable usage of this pipeline by engineers and not only data specialists. In addition, such GUI provides far more control over visual data flow correctness evaluation. The processing, data providing, and outputting modules share the same GRPC communication architecture. This enables modules to aggregate more than one function. For example, the data loading module could integrate also common filtering or data mapping functions that are required. As data flows through the pipeline, it can be shared by multiple modules, enabling for instance implementation of online data monitoring and processing that provides output to common message broker in a form of events, which in turn can be used by other system modules to take specific actions and therefore synchronize or adapt process control to changing conditions.

3.3 Task and operation planning and reasoning

To derive the tasks and operations required to assemble a product, different aspects of the product and the production site have to be considered during the reasoning process. The most obvious are the parts assembled in the final product. According to the single elements and their special features, the required tasks and operations for the assembly can be derived using corresponding rules, for example, a screw will lead to a screwing task. However, during the reasoning it also must be considered that not all required tasks are always directly derivable from the product and its components itself. For example, temporary parts are applied and later removed to protect design parts, or some parts require some processing such as cleaning which cannot be retraced from the product geometry. Further, it is also necessary to include knowledge about the actual production site and the surrounding. According to the availability of tools or parts – parts and tools might be available on the worktable so it is sufficient to pick them up, but they might be located in toolboxes or on racks what involves first walking toward them before they can be picked up. In some cases it is also required to prepare the tools prior to use. Such tasks are not directly related to the product and will require the input of further information.

To perform this task and operation derivation and scheduling, the so-called Operation Reasoner, an ASP based reasoning is integrated into the agent system AJAN. To serve different

data sources, multiple REST endpoints are realized in AJAN to, inter alia, receive data from the database where the rules are stored and from the annotation editor, which provides product and assembly related information. Further APIs could be used to add more information for example about the assembly site or tool availability to derive non-product related operations. The final command to start the actual reasoning process using the so far received information is also triggered through the REST API.

The result of the reasoning process is a JSON file with parts, operations, tools, and operation parameters. To visualize the result the so-called Task and Operation Editor is offered as a web service where the result is received. The user can then, if necessary, fine tune the results of the Operation Reasoner using visual user interface and clean presentation of the tasks and operations.

From the generated and finetuned tasks and operations, the human-readable assembly instructions must be prepared. This is done by the Instruction Editor in a semi-automatic way. It uses the JSON file as well as the 3D models of the assembly as an input and pre-processes instructions for each task and operation. The result is a definition of visible parts and highlights for the current operation, viewpoints, and basic animations. The work planner can manually optimize the instructions and add animations, further annotations and descriptions. The resulting instructions are added to the input JSON file.

Finally, an Instruction Viewer can read this file and use it to automatically show the prepared work instructions. Different kinds of viewers will be able to show instructions as text, videos or in 3D. The Instruction Viewer can be controlled by the Worker Assistance System to always show the correct instructions.

3.4 Layout planning

Layout planning and optimization is one of the main outcomes of the project. Factory layout has large influence on the task and operation generation, as tasks at some point need to be attributed to individual workstations, and tools and parts need to be assigned to the workstations. The part and tools assignments to workstations on the other hand influences layout optimization problem, as internal workstation layout should be adapted to specific tasks and operations performed there and facilitate parts transfer from and to workstation.

Layout optimization can be multilevel and references to higher layers should be always maintained. This means that factory layout (the highest level) can optimize placement of workstations and common material zones based on process flow diagram and warehouse constraints. Then detailed workstation layout optimization can happen where actual parts, material storage places, worktables, etc. are arranged within single work cell. The second level optimization can influence for example task assignment to individual stations or can influence tool and component assignment to stations. And that should be possible in the framework. Also, notification to modules that depend on the specific layout should be provided.

To tackle that problem common data format describing layout is developed. This format can be then used in WP3 for task and operation planning as well as in WP5 for the layout optimization. In addition, process workflow format needs to be defined that can be used as

input to factory layout optimization (to serve as a constrain between individual workstations) and for task and operation planning to designate tasks to individual workstations and workers.

3.5 Overall architecture of the AIToC platform

AIToC data related framework architecture has been planned realized in work package 2. In addition, data processing pipeline architecture has been developed in work package 4. Work package 3 provided additional architecture requirements towards task and operation editing with special focus on data flow between process editors, layout planning personnel, factory workers being end users for instruction viewers, and production engineers supervising automatic task and operation derivation. In WP6, those intermediate layouts are combined into single AIToC framework and data communication, messaging system, and data handling and storage system.

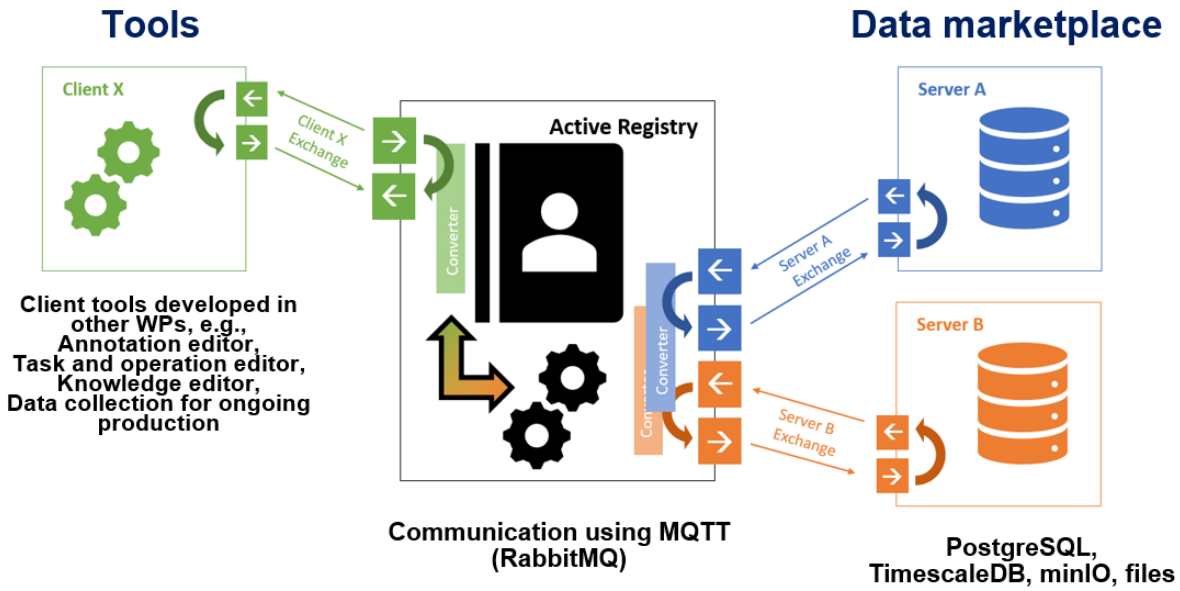

Figure 1 Active registry architecture

Data management is performed by a centralized platform that offers common API for wide range of data storage mechanisms. The idea behind active registry is to enable separation of data storage mechanisms, that should be flexible and adaptable to specific use case needs, from key AIToC framework components that interact with the data. This way every client – being part of AIToC framework modules – needs to only implement one data input and output method to communicate with the active registry. For the modules the way that data is stored and organized is irrelevant, what is important, is the unified data access layer and user rights control implemented outside the module to remove the burden of data management from each module and locate it in the central place where all modules can utilize it as a service. Active registry provides communication mechanisms: the first one is based on HTTP requests and provides convenient access layer for clients that require data in portions or provide data in portions. For example, editors, data collection from sensors, or consuming data by pipelines. This is a stateless interface which means simplicity in the implementation and usage – single request is followed by a single response. Using polling mechanism, clients can update their status based on data related events that can happen in other modules. For use cases like

online collaboration on the data, or data processing that need to reflect changes in real time and not after undetermined delay, MQTT protocol is provided. This protocol is based on the idea of subscriber/publisher and allows clients to subscribe to events related to data, and then receive information immediately when such events happen – the core difference here is that active registry can push notification to clients at any time and even buffer messages for temporarily unavailable clients if such buffering is required by the client. Behind active registry service there is data marketplace storage pool, which comprises of relational databases, NoSQL databases, object storage compatible with Amazon's S3 architecture, to allow unlimited scaling of the data backend, and file storage for simple use cases. The API for the clients provides interfaces that allow to rely on the active registry to select the right storage method and provide data to the client without client implementing direct access to the storage backend, and for those clients that have specific requirements the storage method can also be selected by the client and proxied to the client by the active registry service. This allows migration of the storage backend without affecting the clients, what in turn allows simple adjustment on the server side to cope with the changing data types, access frequency and size.

By utilizing plugin models for data access blocks, active registry offers opportunity for one place integration to internal company data systems. This means that adopting AIToC framework to individual data storage mechanism needs only be done in one place for the whole framework to function with the new data provider.

The active registry is designed with the data workflows identified in the use cases of the AIToC project. Allowing for flexibility in data movement through the framework and providing intermediate results storage and management, including knowledge storage mechanisms for reasoning engines.

Modules in the AIToC framework communicate through MQTT protocol provided by the data marketplace service. The access to the resources and constrains of the communication channels is provided by the active registry identity management system, which in this moment is the open-source Keycloak project. This facilitates centralized user and user privileges management and implements single sign in mechanism.

3.6 Framework to world communication

Active registry service will be used as the gateway between AIToC framework components and inter framework communication, and the external services that will utilize AIToC framework or provide services, data storage, etc. to the AIToC framework. Common internal API will allow to simplify data mangling to the gateway and utilize common data and messaging platform for internal components to simplify module development and reduce need for reimplementation of format exchange in individual modules.

Active registry will provide plug and play modules that tackle individual data exchange needs to facilitate communication between AIToC framework and the external tools/services/end users. For manual data management, module development, user management, verification purposes and audits, user interface will be provided to give overview as well as low level data access methods. For regular or large-scale integrations with external systems, API will be provided that will facilitate connecting external clients/servers to the AIToC framework. In

addition to data exchange and mangling, also a messaging service will be provided that will enable sending notifications to subscribers when selected events occur – for example modification of data, incoming data, or adding/removing data source/sink. This will enable creating data pipelines that are event driven.

3.7 User interfaces

Web interface is preferred for all the modules. Common interface should be developed that allows to extend its functionality with modules. It is expected that many modules will require only basic configuration by specifying numerical and textual values. For such use cases a template-based user interface should be offered to minimize need for such simple interfaces to be developed separately for each module. Template-based user interfaces will be configurable by adding a module manifest file in JSON format with specification of parameter types and allowable ranges. A module will be then provided with the parameter values obtainable through request to the Active registry API. In the same way, such module will be allowed to register its need for user interface and parametrization options.

For use cases where a more advanced user interface is required, or a web-based user interface is already part of the module, the active registry will provide proxy functionality with single sign on functionality. This way, complete user interfaces will be integrated into common web interface.

For standalone tools, the AIToC Active Registry service will provide an installation package location, that will be conveniently available to end users, so that they could download and install the tool on local machine.

4 Summary and conclusions

The technical requirements and status of the current integration is provided in this document. Ideas for the implementation will be put into production during the last year of the project and documented accordingly. The core components of the framework are planned as open-source tools with permissive licenses allowing to royalty free reuse of the framework in new applications and in industrial and commercial products and services. Data storage and management is taken care of centrally to offload module developers from the burden of data access control and data sharing implementation. A common data access API provides integration opportunities with minimum dependencies and allows for integration of modules written in different programming languages. The network communication protocols chosen, adhere to industrial and open commercial standards, providing wide compatibility of the framework out of the box. Internal data formats have been defined around mostly JSON messages to limit burden required to support multiple formats and thus data checking libraries.