Information Model for Auto-generation of Assembly Work Instructions

Using information from manufacturing preparation by increasing system interoperability

Master's thesis in Production Engineering

William Bäckström
Christopher Westberg

DEPARTMENT OF INDUSTRIAL AND MATERIALS SCIENCE
DIVISION OF PRODUCTION SYSTEMS

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Information Model for Auto-generation of Assembly Work Instructions

Using information from manufacturing preparation by increasing system interoperability

WILLIAM BÄCKSTRÖM
CHRISTOPHER WESTBERG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Industrial and Materials Science
Division of Production Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021

Information Model for Auto-generation of Assembly Work Instructions:
Using information from manufacturing preparation by increasing system interoperability

WILLIAM BÄCKSTRÖM
CHRISTOPHER WESTBERG

Supervisor: Ph.D. candidate Dan Li, Department of Industrial and Materials Science, Division of Production Systems

Examiner: Professor Åsa Fasth-Berglund, Department of Industrial and Materials Science, Division of Production Systems

Industrial Supervisor: Ph.D. Pierre Johansson, R&D Engineer, Research & Technology Development, Quality & Engineering, Volvo Group Trucks Operations

Department of Industrial and Materials Science
Division of Production Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 (0)31-772 100

Gothenburg, Sweden 2021

Information Model for Auto-generation of Assembly Work Instructions:
Using information from manufacturing preparation by increasing system interoperability

WILLIAM BÄCKSTRÖM
CHRISTOPHER WESTBERG

Department of Industrial and Materials Science, Division of Production Systems
Chalmers University of Technology

Abstract

This thesis aims at finding an information model suitable to handle the data needed for auto-generation of work instructions. The purpose is to enable the generation of work instructions without requiring manual work by production planners, making it possible to better handle the increasing variation in production which the case company Volvo GTO currently faces. The effort is trying to take the digital technology into advantage not only for the sake of the operator but for the sake of the engineers performing the preparation processes.

By iterative development together with representatives from the software companies and Volvo GTO, an information model has been developed. A simple demonstrator in Java was designed to develop the model and prove its functionality. A pedal car assembly case taken from closely related research projects has been used in the development process. Finally, the model was implemented within the ThingWorx IoT platform. At this stage, solving interoperability issues between different involved systems turned out to be key to move forward in the process.

The most important characteristic of the information model presented is the possibility to break down tasks into smaller elements, representing simpler movements, and let parts and tools be related to these. Input to manikin simulation software has been proven useful for providing such context as well as data from line balancing software. The information model designed is considered relatively simple to expand, both in terms of complexity of the objects and the number of objects.

Simulation and planning software used have previously served other main purposes, thus their functionality and interfaces are designed with this in mind. Using a modular design with the IoT platform ThingWorx as centerpiece has proven to help overcome some interoperability issues as its interface is easily adapted to fit a wide range of software. Its flexible handling of data structures enables imported data to be mapped and used for other purposes, like work instructions.

The next step is to fit the data to an HMI so that the information model and its abilities can be utilized to support the operator. To increase the technology readiness level, the information model also needs to be verified and most probably extended to be able to serve more complex tasks in assembly.

Keywords: work instructions, information model, system interoperability, cognitive support, Industry 4.0, manikin simulation, manufacturing preparation

Acknowledgments

This thesis project has been carried out together with Volvo Group Trucks Operations during the autumn of 2020 with the aim to contribute to the MOSIM research project as well as the journey towards digitalization and Industry 4.0 maturity at Volvo GTO.

The project has required close cooperation with partners such as Industrial Path Solutions, Solme, PTC and Virtual Manufacturing. We want to thank all of them and their representatives for this opportunity and their involvement. Thanks to Johan, Tobias, Niclas and Peter at IPS for valuable input on the topic and for developing needed functionality. Thanks to Oskar Ljung at Solme who has provided valuable insights on AviX and how it can be part of the presented solution. Thanks to Per Lönnehed at PTC for helping us navigating the ThingWorx ecosystem. Without them, this project would not follow through.

Special thanks to Dr. Pierre Johansson, our industrial supervisor at Volvo GTO who presented this interesting project to us and has put trust and hope in us and our proceedings. His great dedication, ideas and interest in the topic have been valuable throughout this five-month journey. We also want to thank him and his colleagues within the Research & Technology Development team at Volvo GTO for their hospitality and welcoming atmosphere. Thanks for all the enjoyable moments and your confidence in having us.

Further, we would like to express our gratitude to our academic supervisor, Ph.D. candidate Dan Li & examiner, professor Åsa Fasth Berglund at the division of Production Systems. Their valuable contributions, feedback and support, especially on report writing and ensuring academic quality.

The project has been a challenging and educative experience in which we have had the opportunity to make use of previous knowledge as well as developing a better understanding of Industry 4.0 and digital tools within manufacturing. To us, this project is not just another delivery, but it also marks the end of our journey towards the master's degree.

William & Christopher

Gothenburg, January 2021

Table of Contents

List of Figures

List of Tables

Abbreviations

BoM	Bill of Material
DHM	Digital Human Modeling
DSR	Design Science Research
ERP	Enterprise Resource Planning
FCC	Fraunhofer Chalmers Centre
GAIS	Global Assembly Instruction Strategy
GTO	Group Trucks Operations
GUI	Graphical User Interface
HVLV	High Variety Low Volume
HMI	Human-Machine Interface
ICT	Information Communication Technology
IoT	Internet of Things
IMMA	Intelligently Moving MAnikin (module of IPS)
IPS	Industrial Path Solutions
MES	Manufacturing Execution System
PDM	Product Data Management
PLM	Product Lifecycle Management
PMTS	Pre-determined Motion Time System
TACO	insTruction innovAtion for Cognitive Optimisation
TRL	Technology Readiness Level
TDM	Time Data Management
UID	Unique Identity

1 Introduction

This chapter presents the project's background and context as well as its academic and organizational contributions. The aim, research questions and delimitations of the issue under investigation are presented at the end of the chapter.

1.1 Background

Research in the domain of manufacturing engineering claims that we approach or even find ourselves in the middle of the fourth industrial revolution, more commonly cited as Industry 4.0. The key concept in this revolution is digital transformation and connectivity of devices in production. Improved productivity, shorter time to market and increased competitiveness will be the result, according to Schuh et al. (2017). The digital transformation will also help organizations to better handle disturbances and swiftly adapt to changes in products and manufacturing systems.

Hu (2013) illustrates how the manufacturing industry has gone from the paradigm of mass production to mass customization and further on to personalization, see figure 1. A company that still lives by Henry Ford's (1926) saying, *"Any customer can have any color that he wants so long as it is black"* will not be able to stay competitive. Today, customers demand more complex products which are made according to individual preferences and specifications. This trend increases the complexity in manufacturing as it demands higher flexibility while maintaining levels of high quality and productivity. According to Kagermann et al. (2013), Industry 4.0 will unlock the possibility to produce unique products while still making a profit and staying competitive on the market.

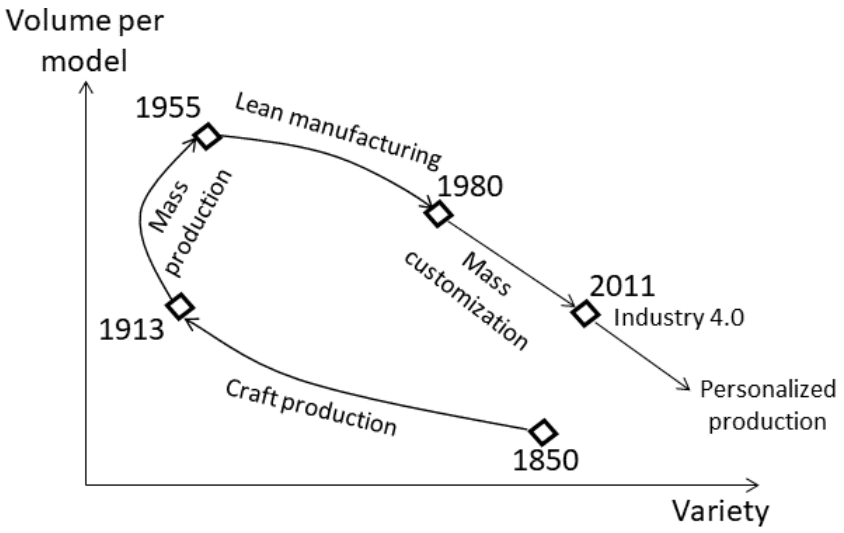


Figure 1. The change of volume and variety over the last 160 years (Hu, 2013)

According to Schuh et al. (2017), efficient communication between agents, both human and machine is a cornerstone in Industry 4.0. Berlin & Adams (2017) stress the importance of cognitive support especially as the product complexity increases to ensure a good operator work

environment, product quality and productivity. According to Fast-Berglund et al. (2018), work instructions must be relevant, accurate, accessible and on time to be useful to the operator. To be able to provide well-designed work instructions, Berlin & Adams (2017) claim that a digital information carrier is beneficial as it allows a wide range of content. It also makes it possible to move instructions closer to the operator, thus enabling more efficient information transfer (Thorvald et al., 2014). Degerman & Lindgren (2010) have identified that operators of dissimilar experience need different levels of detail in their instructions and earlier studies at Volvo Group Truck Operations, e.g., Eriksson & Johansson (2017) have come to the same conclusion. Dynamic work instructions will only be possible by digitalizing the information content and its operator interface.

Volvo Group has global market coverage and facilities on six continents manufacturing heavy commercial vehicles. Volvo Group Trucks Operations runs the manufacturing operations of trucks and engines for the Volvo Group brands (Volvo Group, 2019). In Sweden final assembly is found in Gothenburg, along with a body-in-white plant in Umeå, engine assembly in Skövde and transmission assembly in Köping. As of today, plants have different solutions for designing and presenting work instructions, some digital and some paper-based, as found by Delin & Jansson (2015) who mapped internal processes for creating assembly work instructions. Volvo GTO wants to standardize the way of working to be able to handle increased variety and to improve quality. An important part of this process is being able to provide appropriate assembly work instructions in operations since humans are and will still be an integral part of production (Sony & Naik, 2020; Schuh et al., 2017).

To be able to simultaneously face the increasing product variety while providing the right cognitive support to operators, a new way of generating work instructions is needed. Earlier research has proven digital work instructions to be advantageous from many points of view. Bringing data together from manufacturing preparation processes and systems within the organization is key to generate rich work instruction content. The amount of additional manual work to create instructions should also be kept reasonable, which is especially important as product variety increases and product life cycles are short. To be able to present the data and make it useful to the operator, it must be well structured and have the application area in mind.

1.2 Aim

The aim of the thesis is to prove the possibility to auto-generate assembly work instructions. Auto-generated means that the information is brought together on-demand by reading raw data about the assembly process. Reading from raw data means that no predefined text strings are available. Instead, the instructions can be generated dynamically, adapted to the individual needs and wants of the operator. As the amount of data may be great, both in terms of amount and diversity, its structure is important. Structured data helps to build context and informative content therefore, designing an information model for this purpose is within the aim of the thesis.

The information model needs to be populated with data to be able to serve its main purpose. To make it possible to make work instructions with little manual work, data from manikin simulations,

balancing systems, CAD and other manufacturing preparation software can be used. Software and data sources will have to be brought together which have been designed to serve other needs. This requires a better understanding on how to integrate different systems and have them interoperate. The information model designed and its functionality will be implemented in a proof-of-concept which will be populated with data from manikin simulation and line balancing software.

The information model and system interface designed must be scalable to meet future needs. Enabling the ability to expand the information model and the system in which it resides is necessary to be able to reach higher technology maturity. Once mature, it can serve a real case purpose in manufacturing and then also be considered valuable.

1.3 Specification of the issue under investigation

RQ1: What are the essential characteristics of an information model designed to orchestrate auto-generation of work instructions in manual assembly?

RQ2: How can data from manufacturing preparation systems be mapped to the information model presented in RQ1?

1.4 Research projects at Volvo GTO

Volvo Group has been involved in multiple projects that strive to increase digitization and digitalization within manufacturing. Two projects are presented below related to the topic of assembly work instructions.

MOSIM is a project which focuses on improving both productivity and ergonomics in manufacturing using virtual tools and methods (Vinnova, 2018). The project aims to enable more precise body motion simulation of operators having different physical prerequisites by utilizing machine learning and AI. It will hopefully improve the possibility for effective manufacturing preparation processes by enabling easier handling of more complex situations. As the simulation makes it possible to focus on both productivity and ergonomics in an earlier stage of development, better design for assembly and a better work environment for the operators can be ensured.

insTruction innovAtion for Cognitive Optimisation (TACO) strives to improve the cognitive support in manual assembly by making it more adapted to the prerequisites of the individual (Vinnova, 2019). This will be achieved by adapting the instructions to new available technology and forming strategies for future digitalization. The project is expected to lead to a lower cognitive workload and increasing the digitalization and Industry 4.0 maturity of the partner companies.

1.5 Delimitations

The information model will be designed and developed using a case encompassing the assembly of a pedal car. The case is used in related research projects like MOSIM and TACO. That means understanding and knowledge of the case and related data such as CAD files is extensive among

other research project participants. Also, the pedal car assembly case has been designed to include a diverse set of assembly tasks found in any production site. The thesis aims to design an information model that can handle assembly work instruction data for the pedal car case while being scalable and flexible to fit more complex needs in the future.

The information content to be stored in the information model will focus on how the assembly tasks are performed from an objective point of view. That includes sequence and position information, combined with data on parts and tools which got the characteristics of explicit knowledge. For an operator to get a full view on how to perform their work, additional information may be needed such as change notices, alerts or other notes, some of which got characteristics similar to tacit knowledge. It is not within the aim of the thesis to handle such information.

The instructions generated will be presented in a simpler format, mainly using text strings and tables on a computer screen. It belongs to the future to use the data and present it within an optimized Human-Machine Interface (HMI) to provide better operator support. This project will be followed by a thesis aiming to design appropriate HMI, and therefore the focus is on information modeling and interoperability as it needs to be in place before taking the next step.

1.6 Report disposition

The report is divided into seven chapters. The following section presents the methodology used to generate the information model and its proof-of-concept. The frame of reference for relevant topics can be found in chapter 3. As the computer programs used are important for the development phase, they are mapped and presented separately in chapter 4. The information model design as well as the architecture of the demonstrators, acting as proofs-of-concept are presented in chapter 5. The result section is followed by a discussion in chapter 6 where the information model and its implementation are discussed in relation to current research. In chapter 7 the conclusions are to be found.

2 Methodology

In this chapter, the methodology used is presented. As this project focuses on the development and design of an information model and proofs-of-concept, the design science research methodology is thoroughly described. Efforts to ensure research quality and generalizability are presented at the end of this chapter. The workflow is visualized in figure 2.

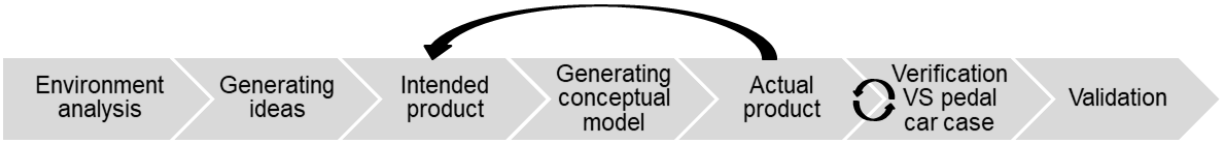


Figure 2. Workflow of the master thesis

2.1 Information system development

Before starting any development activities, the environment needs to be mapped. At the project initiation phase, it was decided that ThingWorx, IPS IMMA and AviX should be the software to include as those are also used for other purposes at Volvo GTO. The software was mapped in terms of functionality, information models and data exports at an early stage to be able to start the iterative development phase as presented later. Continuous refinement of the software mapping has taken place throughout the project.

The development of software is complex and to make it easier to understand Kruchten (2012) suggests a conceptual model as an effective tool to communicate architecture and functionality. Kruchten (2012) emphasizes not to make the model too complex as it is counterproductive and eliminates the benefits of making a model. The approach of the Design Science Research Methodology has been used to guide the information system development that has taken place. Deng & Ji (2018) has made a literature study on academic Design Science Research (DSR) contributions and claims that there is no status quo in its definition. Hevner & Chatterjee (2010) define DSR as:

> "Design science research is a research paradigm in which a designer answers questions relevant to human problems via the creation of innovative artifacts, thereby contributing new knowledge to the body of scientific evidence. The designed artifacts are both useful and fundamental in understanding that problem."

According to Deng & Ji (2018), information system research is often different from traditional research as solutions and systems are built to simplify and solve problems among humans. Traditional research on the other hand has an approach trying to better understand, explain and explore a certain phenomenon. Deng & Ji (2018) claim that DSR serves the purpose of bridging the gap between research in the traditional domain and solving problems in a business context. The aim is to solve a problem and gain scientific knowledge at the same time. In this thesis, a problem is defined, and its solution is designed in a business context. The solution is structured and designed in such a way that its generalizability is higher so that it can be applied outside the borders of the case company. The pedal car case from the MOSIM project has been used in the design phase of the information model and proofs-of-concept to ensure generalizability. Its

assembly includes the common and basic operations in manual assembly across a wide range of manufacturing companies.

Peffers et al. (2007) are one of the most cited papers on DSR within information system development and therefore their six-step framework has formed the basis of the approach taken in this project.

1. Identify the problem and the value of a solution. Knowledge of the problem area is necessary to successfully perform this mapping.
2. Define objectives, which can be quantitative or qualitative. Comparison can be made to the existing state, but the system designed can also be something completely new.
3. Design and development of the artifact, including both architecture and functionality. Hevner & Chatterjee (2010) define the term *artifact* as something artificial, created by humans in comparison to things that exist by nature.
4. Demonstration by including the artifact in a proof-of-concept, case study or simulation.
5. Evaluation of how well the solution solves the problem and reaches the objective. Perform a comparison, user testing or simulation. The feedback is used for future research, or one can return to step 3, design and development.
6. Communication of knowledge gained by submitting a research paper or similar.

The value of the solution is presented and explained in the introduction chapter. As this is a greenfield development project, objectives are defined according to what work instruction content is identified as valuable. The instruction content will be evaluated from a qualitative point of view as that is considered most appropriate at an early technology maturity stage. The design and development phase is itself an iterative process. Peffers et al. (2007) claim that iteration takes place after demonstration and evaluation. Since this is a greenfield project, continuous input, discussions and reflections from involved stakeholders have been needed and used within the design and development phase itself. Kruchten (2012) supports the iterative approach in information system development.

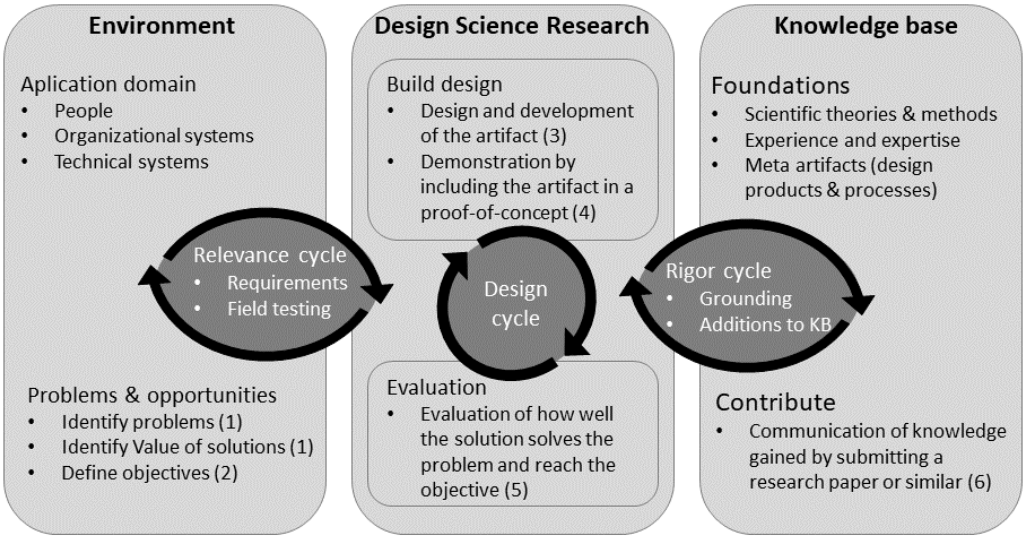


Figure 3. Six-step framework (Peffers et al., 2007) mapped to three cycles of DSR (Hevner & Chatterjee, 2010)

Figure 3 above shows the three iteration cycles included in DSR, according to Hevner & Chatterjee (2010) and the six-step framework of Peffers et al. (2007) mapped to it. The three cycles themselves and their interconnectivity shows and confirms the need for an iterative approach. The relevance cycle shows that the problem must be relevant and addressed by the artifact designed, which is why continuous communication with involved stakeholders has taken place. Discussions have taken place mainly with the industrial and academic supervisor, but also with company representatives from the three software companies involved. The rigor cycle depicts that current knowledge and expertise must be included within the development process, which is presented in the frame of reference.

To ensure validity and applicability of the information model design, a proof-of-concept is used, in accordance with the six-step framework suggested. The proof-of-concept has been designed using Java programming language and at a later stage implemented in ThingWorx IoT platform. As Peffers et al. (2007) claim, sufficient knowledge is required in the development stage. The authors of this thesis have basic programming knowledge sufficient for building light applications. The applications will serve the purpose as proofs-of-concept, however not as robust and working applications. As this project is an initial effort to show the ability to auto-generate work instructions, this way of demonstrating and evaluating the information model has been considered appropriate.

Interviews and discussions within the project have mainly focused on what additional information and value the manufacturing preparation systems can bring, besides their core functionality. The meetings can be seen as unstructured interviews, according to the definition of Bryman & Bell (2015). A topic has been agreed upon when summoning the meeting and the meeting itself has often started with a short status update of the project. Since figuring out the next step requires an open mindset and discussion, the unstructured interview setup has been considered useful. These meetings have contributed to valuable insight on what functionality is to find in the software, which is not listed in product sheets or manuals. Involved software has complex functionality, so some communication focused on helping the project participants to increase their system knowledge.

2.2 Research quality

To evaluate the research project, the quality is important, according to Bryman & Bell (2015). The three main criteria for a study of high quality are reliability, replication, and validity.

Deng & Ji (2018) have studied the literature on DSR and found that the six-step suggested by Peffers et al. (2007) is among the most used methodologies in this area and should therefore be considered a reliable method for this purpose. Combined with the iterative approach recommended by Kruchten (2012) and Hevner & Chatterjee (2010), reliability is strengthened as input from stakeholders has continuously been taken into consideration.

Replication defines if the study can be replicated by someone else to compare results. As this study will focus on IT systems, which might change in terms of functionality over time, the repeatability of the study might be limited. Also, as the development is an iterative process with lots of different and dynamic input sources, another project with the same objective might come to another design

solution. However, by thoroughly reporting the method and results of this project, the replicability can be kept at an appropriate level.

The level of validity measures how well showed concept correlates with the actual concept. In this case, the concept will be evaluated using the pedal-car case. Follow-up studies on other cases from manufacturing will be needed to further strengthen the validity and increase the technology maturity of the proposed design.

3 Frame of reference

The frame of reference starts with an introduction of Industry 4.0, which gives the project a context. It then dives into the subject of work instructions and appropriate information content. The later part of this chapter presents research and literature on interoperability within manufacturing. Motion and time data management systems are described further as they are relevant for this research context.

3.1 Industry 4.0

The fourth industrial revolution, Industry 4.0 focuses on the digital transformation and connectivity in production. The idea is to take advantage of digital aids to make the organization more competitive by decreasing time to market, increasing productivity while handling more complexity (Schuh et al., 2017).

3.1.1 Maturity framework

To measure and better understand the maturity in this transformation, Schuh et al. (2017) have designed a six-step framework. The first two steps, computerization and connectivity are considered parts of the digitalization and thus not a part of the Industry 4.0 transformation. Computerization and connectivity act as enablers for efficient information flow between different departments to support better decision making, according to Schuh et al. (2017). When digitalization has taken place, the Industry 4.0 maturity journey can be initiated, illustrated in figure 4.

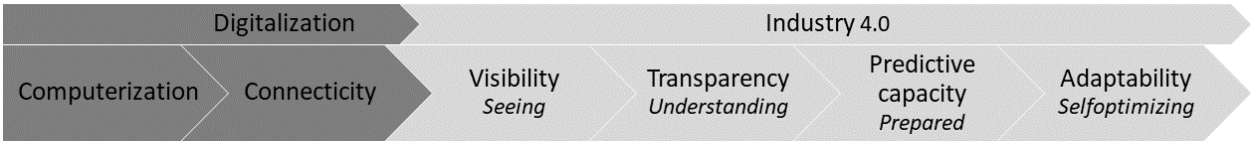


Figure 4. Maturity index for Industry 4.0 (Schuh et al., 2017)

Visibility is the first step in Industry 4.0 and has great importance to be able to reach higher maturity levels. The aim is to have all information in one place instead of separate silos. When implemented fully, an up-to-date version of the factory can exist which often is recalled as a digital twin. This makes it possible for the management to make decisions on real data. Hehenberger & Bradley (2016) describe the digital twin as a tool that uses the relevant data in IT systems to make simulations and predictive analysis.

In the transparency step, an understanding why things are happening is achieved by performing root cause analysis using big data, enabling complex and rapid decision making (Schuh et al., 2017). Predictive ability builds further on the transparency step as it makes it possible to simulate future scenarios and thus prevent failures and maximize efficiency. The final maturity level is adaptability, where the decision-making is automated to a great extent, allowing quick decisions without human interaction (Schuh et al., 2017). To reach higher maturity levels, integration of systems is needed through standard interfaces to enhance quick data exchange.

Sony & Naik (2020) state that there are two kinds of system integrations important in Industry 4.0, horizontal and vertical. The horizontal integration encompasses cooperation and collaboration between different organizations in the supply chain. The vertical integration focuses on the sub-systems and their interconnection within the organization. To succeed, all parts of the system must be considered to avoid sub-optimization. Kagermann et al. (2013) also describe end-to-end integration as a third integration mechanism. End-to-end integration spans the entire engineering process from the stage of product design and development to manufacturing, including production planning and optimization. End-to-end integration covers all aspects from customer requirements to the finished product. Kagermann et al. (2013) claim that end-to-end integration is necessary to successfully achieve vertical integration.

3.1.2 Technology Readiness Level

To assess the maturity level of a certain technology, NASA (2012) has created the nine-step Technology Readiness Level (TRL) framework seen in table 1. When in TRL 1 initial research has been conducted and the results are to be used as a base for future research and development. In TRL 2 the results from the work within TRL 1 are used to evaluate and speculate what the potential of the technology might be. In TRL 3 the technology has generally been studied both analytical and in a laboratory context, often a proof-of-concept is created. TRL 4 is achieved when the proof-of-concept has been tested successfully in a laboratory environment. TRL 5 is a continuation of TRL 4 where the technology has completed tests in a relevant environment. In TRL 6 the prototype has been tested and is fully functional in an operational environment with partial integration to existing systems. The technology is in TRL 7 when it is fully integrated with existing systems. In TRL 8 the technology has been completely developed. As it is fully implemented and proven successful, the final stage, TRL 9 is reached.

Table 1. Technology Readiness Level framework (NASA, 2012)

TRL	Description
9	Actual system "flight-proven" through successful mission operations
8	Actual system completed and "flight qualified" through test and demonstration
7	System prototype demonstration in a space environment
6	System/subsystem model or prototype demonstration in a relevant environment
5	Component and/or breadboard validation in a relevant environment
4	Component and/or breadboard validation in a laboratory environment
3	Analytical and experimental critical function and/or characteristic proof-of-concept
2	Technology concept and/or application formulated
1	Basic principles observed and reported

3.1.3 Scalable and modular infrastructure

Åkerman et al. (2013) emphasize the importance of horizontal integration and decentralization when designing new systems, meaning the hierarchies are flat. Åkerman et al. (2013) describe IoT platforms as a good base for building system-of-systems as they by design make it possible to connect a diversity of systems as illustrated in figure 5. A modular system can connect with

autonomous subsystems and exchange data. It also allows simpler upscaling and thereby the possibility to introduce continuous improvements to the system. Subsystems can be both local and cloud-based.

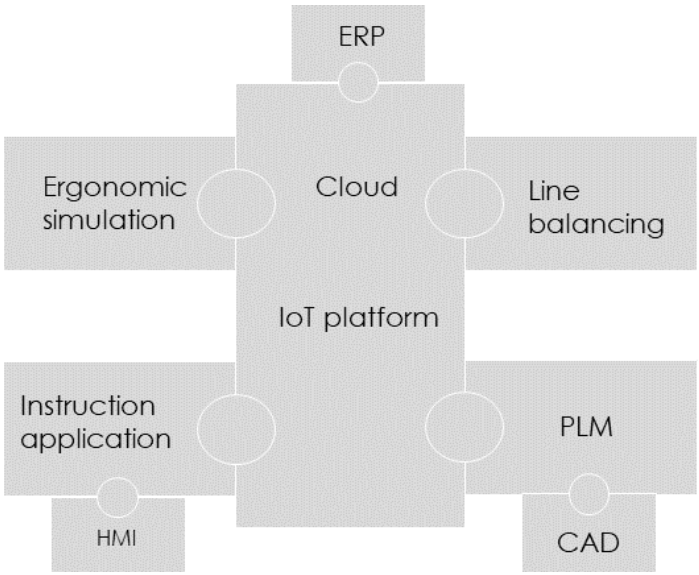


Figure 5. Modular infrastructure example (Åkerman et al. 2018)

Cloud computing could be defined as leasing computer resources such as CPU and storage located in datacenters through the internet (Zhang, Cheng & Boutaba, 2010). Cloud solutions are compelling as they require relatively small investments within the organization, lowering operating costs, business risks and maintenance costs while still being scalable. The services are also often accessible via a browser interface, meaning that they can be reached from a wide range of devices.

3.2 Work instruction content and quality

To understand which output will be needed from the subsystems, a proper understanding of work instruction content is needed. Firstly, cognitive support in operations will be presented. Secondly, work instruction quality framework and recent research in work instruction content. Thirdly, identified information content in work instructions at Volvo GTO is brought forward.

3.2.1 Operator 4.0. Cognitive automation strategy

According to Sony & Naik (2020), the aim of Industry 4.0 is not to replace humans with automation but to assist them in becoming more productive. Mattsson et al. (2020) have defined a strategy for Operator 4.0 cognitive automation, see figure 6. The strategy builds on the operator's assembly mode which can be learning, operational or disruptive. The goal is to be in the operational mode as much as possible as it results in high efficiency.

An operator in learning mode preferably has a knowledge-based behavior, requiring the work instructions to be clear and detailed to give the operator knowledge and understanding of how tasks are performed. The instructions should also support reasoning as the operator is in a state where constant attention is needed. The degree of cognitive automation should be limited to supervision that notifies the operator but does not correct any mistakes. More help tends to interfere with the operator and its learning process.

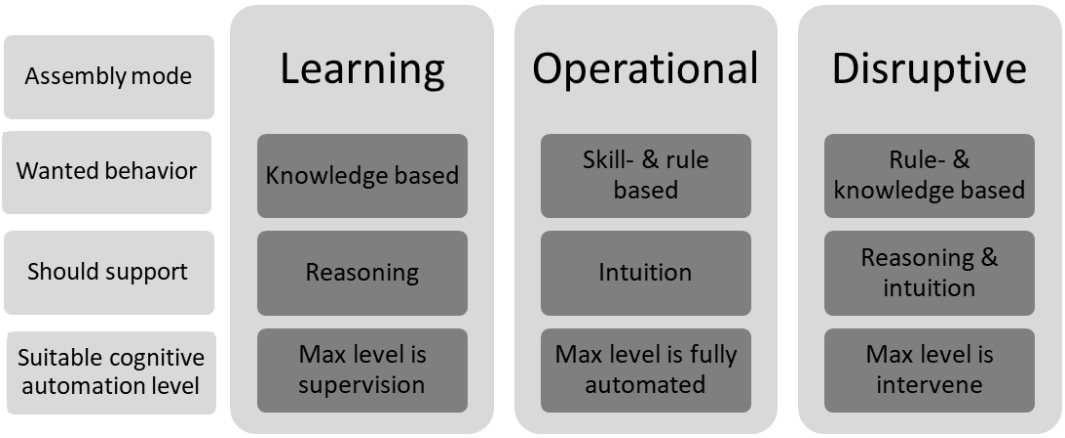


Figure 6. Strategy for cognitive automation depending on the assembly mode (Mattsson et al. 2020)

The knowledge gained in learning mode eventually takes the operator to operational mode. Here, the behavior is skill- and rule-based, supported by intuition which allows the operator to work fast and efficiently. The aim is to have the operator stay in this mode as tasks can be completely automated with the operator acting as a supervisor. The work instructions should be short and concise.

When the operator is facing a problem, like when a new variant is introduced, it can push the operator to the disruptive mode. When in this mode the operator needs more information than usual and rule and knowledge-based behavior is preferred. Such behavior is supported both by reasoning and intuition, the latter one being a valuable tool of a skilled worker when facing disruptions. In this scenario, the cognitive support can be higher than for an operator in learning mode as the operator already has a good understanding of the operation. In this case, the automation can intervene and take over or even correct the task, allowing the operator to learn from the situation and get back to operational mode.

3.2.2 Cognitive support in operations

Berlin & Adams (2017) claim that cognitive support is important for operators, especially as product variety increases and products become more complex. By being able to design work instructions that contain the right information and presented the right way, the right cognitive support can be provided to the operators. It offers a better work environment and also higher productivity and quality of the product, by minimizing the risk of assembly errors. Fast-Berglund et al. (2013) identified that increasing cognitive support can help increase assembly quality by decreasing the number of assembly errors when the number of variants is high. According to Bäckstrand (2009), human factors stand for 93% of the internal rejects due to missing parts or

wrong assemblies. Falck & Rosenqvist (2014) found that errors identified late in the value chain cost 9.2 times more than an error that was found early, and an error discovered on the market were 12.2 times more costly.

Today most work instructions are paper-based, the trend in the industry turns towards digital work instructions provided on screens (Berlin & Adams, 2017). Digital instructions can look in many ways, including text, pictures and animations, often in combination. Li et al. (2018) compared various kinds of instructions and found that a screen with picture and text-based instructions decreased the cognitive workload at the workstations compared with just showing text or having paper instructions. This effect became more evident as the product variants increased. Brolin et al. (2017) performed a full factorial design testing how kitting and picture-based work instructions affected productivity and came to a similar conclusion.

Proper work instructions will not alone provide good cognitive support to the operator but act as an important aid in manufacturing. According to Fasth & Stahre (2013), cognitive support is considered one dimension of automation, where physical automation constitutes the other. The level of cognitive automation depends on if the technology provides instruction or also provides decision support and fault management. Higher product diversity, meaning more variants require that the operator gets more cognitive support. Fässberg et al. (2011) recognize the lack of focus on cognitive automation compared to mechanical automation, which needs to change to enable the operator to effectively handle the increased information flow. The case study conducted by Fasth & Stahre (2013) identified that different operators with different skillsets and experience need a different amount of cognitive support and automation. Degerman & Lindgren (2010), who made a study in a High Variety Low Volume (HVLV) environment, also found that information need depends on operator experience.

According to Fässberg et al. (2011), operators must be taken into consideration when designing the cognitive automation. If the operators do not see obvious benefits, the aids will most probably not be used. Fast-Berglund et al. (2018) highlight the importance of only delivering relevant and accurate instructions, on time. If these criteria are not fulfilled, they risk not being used. Accessibility is key as only the readily available information will be used. Fässberg et al. (2011) also emphasize the importance of letting the operators control the amount of information they receive to fit personal preferences. The operators most often only need the product-specific information but sometimes it can be important to get more information, so additional information needs to be easily accessible.

Haug (2015) has defined a framework for work instruction quality problems, presented in table 2. In total, the framework holds 5 intrinsic factors, and 10 extrinsic factors, extrinsic ones being more dependent on context and environment. Haug (2015) emphasizes that the perspective of part/product designers and operators is not the same on work instructions as operators need information on *what* and *how* something should be assembled, rather than *why*.

Table 2. Work instructions quality problem framework (Haug, 2015)

Intrinsic
Deficient (incomplete) work instructions
The instruction being too ambiguous, so multiple needed instructions come from the same received instruction.
Unneeded instructions. Not necessary or already known by the operator.
Incorrect instructions, giving wrong information.
Repetitive instructions, same information being delivered multiple times.
Extrinsic
Representational
Inconsistent, different symbols and figures are used to express the same thing.
Verbose, using words that do not add any information.
Difficult to understand. If instructions are written in a language the receiver cannot fully interpret.
Unmatched
Too complex content. The receiver needs more knowledge to understand.
Too large amount. It should instead match the cognitive capabilities of the receiver
Untimely. That they are outdated and describe an old situation. Information should be delivered when needed.
Questionable
Poor believability. If instructions are prepared in a way that is not trustworthy.
Poor reputation. When the receivers speak negatively about the instructions.
Inaccessible
Security barriers. Not having access rights to the instructions needed
Other accessibility barriers. Having problems identifying the correct instructions.

3.2.3 Work instructions at Volvo GTO

Sprint is one of the manufacturing preparation systems used at Volvo GTO, supporting material and task planning while also having the ability to generate work instructions. All information is fed manually into the system and kept updated during the entire product lifecycle. The design of Sprint sets limitations on its functionality. For example, it is not possible to add any pictures to the work instructions, nor is it possible to have longer part names as the number of characters per row is limited. Sprint is used in Volvo GTO Tuve however, as identified by Delin & Jansson (2015) different plants within Volvo GTO use different systems for production planning and work instruction generation.

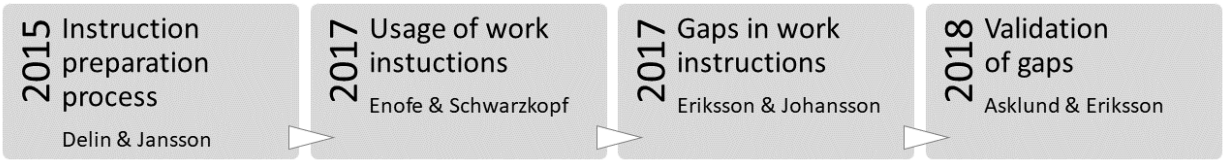


Figure 7. Timeline over earlier master theses on Volvo about work instructions

The thesis of Delin & Jansson (2015) was followed by three other theses on the topic of work instructions at Volvo GTO, see the timeline in figure 7. Eriksson & Johansson (2017) have focused on identifying the information gaps in current work instructions of three plants and an ideal state,

presented in table 3. An overall finding is that instructions need to be individualized and dynamic as senior operators need less information compared to junior. New operators prefer step by step instructions while experienced ones want information about changes, rare products and notice about common quality problems. Eriksson & Johansson (2017) also found that the plants which used digital instructions, where the information content was filtered used instructions to a greater extent partly because they were easier to understand. When too much information is available as in the paper instructions, operators tend not to use them at all, thus causing quality problems. Instead, operators trust their tacit knowledge which is transferred between operators upon the learning phase. Another finding was that there is a long lead time when making changes to the instruction in the current systems, meaning that it is common that instructions are not up to date.

Table 3. The information gaps in work instructions at Volvo GTO (Eriksson & Johansson, 2017)

Work instruction gaps	
Used today	**Additional needs**
Chassis #	Pictures
Truck product #	Feedback of work
Engine product#	Real-time information
Transmission product #	Sequence
Part #	Where to assemble (position)
Part name	How to perform assembly
Comments (special considerations)	Change notes (product/balance/material)
Quantity	Common problems and how to solve them
Screw length	General knowledge about the tasks to gain understanding
Where to assemble (position) – if provided	Mobile information
Fundamental need: The right amount of information, when needed	

Eriksson & Johansson (2017) emphasize operators' call for pictures and other graphical representation to aid assembly. However, from interviews in the factories using pictures, it was found important that pictures of the right quality and detail must be used. The number of pictures and figures must also be controlled to avoid information overflow. Asklund & Eriksson (2018) designed a prototype of digital work instructions and performed interviews in manufacturing to validate the results of the study performed by Eriksson and Johansson (2017).

Enofe (2017) and Schwarzkopf (2017) completed a study at three plants of Volvo GTO by conducting direct observations, semi-structured interviews and surveys. At the Tuve plant, almost half of the operators did not use the work instructions provided, instead relying on their knowledge and experience. They concluded that it was caused by an information overload from the instructions generated in Sprint. Such overload was not discovered in the other two plants which utilized digital work instructions. Enofe (2017) and Schwarzkopf (2017) also mapped that there is a gap in what preparation process engineers and operators consider essential information content in assembly work instructions.

3.2.4 Information need in work instructions

According to Hart (1996), most of the information need in journalism can be met by answering the questions *what, who, where, when* and *why,* also known as the *5W's.* Instructions often focus on *what* but often forget answering the other four *W*'s fully. Hart (1996) argues that the 5W's can be used when designing technical information as it puts the need of the user in focus. His paper is based on manuals for computer programs however, he claims that the same methodology can be used for other technical communication purposes as well. The 5W's are the following:

1. What are the actions to perform to succeed and what is the result of them?
2. Who performs and are responsible for the actions?
3. Where do actions take place? Where are the tools located?
4. When should the actions occur? Suitable sequence of tasks?
5. Why did the actions occur? Why did they happen in this particular sequence?

In some cases, the question *how* is also be added, but Hart (1996) claims that this question can be answered by any of the 5*W*'s instead. To answer the *where* question, Hart (1996) claims that pictures and graphic aids can be useful for the operator as they are easier to remember as they can be put in relation to the physical environment. To answer the questions *when*, a note should also be made if the sequence is unimportant, e.g., in which order to fill in rows in a form. It is also good to answer the question *why* at the same time as it helps the operator to understand why a certain order is important. The question *what* answers operations to proceed, while *why* answers the reason this is preferable and thus increases understanding and knowledge.

The information required from the work instructions ranges from the chassis numbers to the lengths of screws to be used, see table 4. Most of the information content needed is identified by Eriksson & Johansson (2017) but additional information is added by answering the *5W's & 1H* of Hart (1996). Enofe (2017) also identified part number, name, quantity and product ID as valuable content in assembly work instructions.

Table 4. Information needs in work instructions combining Hart (1996) and Eriksson & Johansson (2017)

Work Instruction Information		
Tool & parts (what)	Chassis #	Sequence
Operators (who)	Truck product #	Real-time information
Initial & end position (where)	Engine product#	Comments (special considerations)
Order of operations (when)	Transmission product #	Quality reminders
Process hierarchy (why)	Part #	Screw length
How (e.g., screw, twist, etc.)	Part name	General knowledge about a task
	Quantity	Mobile information
	Where to assemble	Pictures
	How to assemble	Feedback of work
	Change notes (product/balance/material)	Common problems and how to solve them

3.2.5 Work instruction infrastructure

According to Fasth & Stahre (2013), digital work instructions enable different information carriers such as TV-screens, tablets, and smartwatches to be used. Thorvald et al. (2014) claim that in automotive and truck manufacturing, the operators often must choose between moving larger distances to find the information source or rely on their memory. They claim that the information range can be increased by using larger screens, using audio aids or handheld computer devices. By performing a case study, Thorvald et al. (2014) could confirm that the decreased physical effort and time needed to access the information increased quality levels, however not productivity. Being able to provide information closer to the operator, using mobile devices makes it easier to use and increases the value of work instructions (Fasth & Stahre, 2013). Fässberg et al. (2011) claim that information in the format of pictures, animations and videos are easier to keep up to date as compared to detailed instructions in paper binders that easily become obsolete.

Fasth-Berglund et al. (2014) argue that the information content should be separated from the information carrier when designing and analyzing cognitive aids and information systems in manufacturing, see figure 8. The information content could be of many kinds such as text, pictures and videos. The information carrier is the tool used to present the information, such as paper, tv-screen, tablet or phone. The information systems involved affect how the information is structured in subsystems such as Manufacturing Execution System (MES) and Enterprise Resource Planning (ERP) systems. Fasth-Berglund et al. (2014) finally claim that companies can become more competitive in cases where front and backend systems can be integrated without building new separate systems.

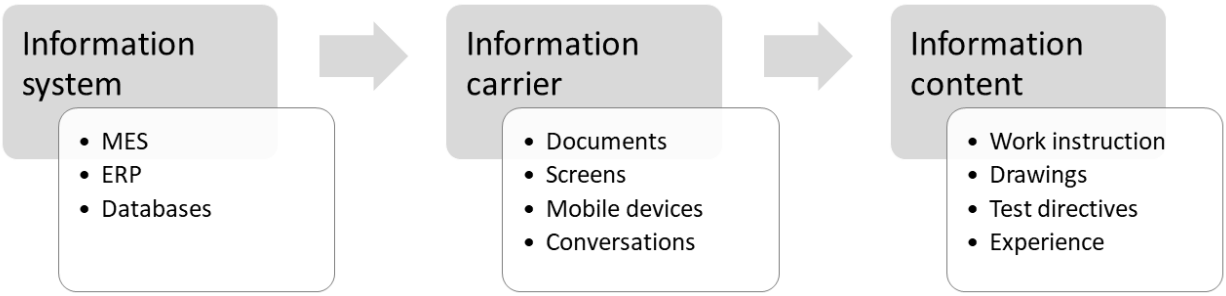


Figure 8. Information structure in overview (Fasth-Berglund et al. 2014)

3.3 System interoperability

Semantic interoperability is important as it defines how humans and other systems should act on the information that is received (Rezaei et al. 2014). It is therefore important that the instructions are understandable and contain the right information so that actions can be performed as intended. Furthermore, the technical interoperability needs to be sufficient so that the data has a format compatible with any involved software (Rezaei et al. 2014), which will be presented in this section.

3.3.1 The four layers of interoperability

Panetto et al. (2019) describe four levels of interoperability: legal, organizational, semantic and technical, the last two of special interest in this project. The framework consists of two different systems; cyber-world and physical-world systems as depicted in figure 9. Cyber-world systems are technical systems such as software while physical-world systems are physical agents such as machines or humans. Technical interoperability is the possibility for cyber-world systems to communicate with each other, which is dependent on syntax, interactions and interfaces of the involved systems. Semantic interoperability involves the connection between a cyber-world system and a physical-world system. The simplest example is an information system which delivers information to an operator. The receiving system will interpret the information, thus possibly changing the meaning from the sending system, add meaning to it and act accordingly.

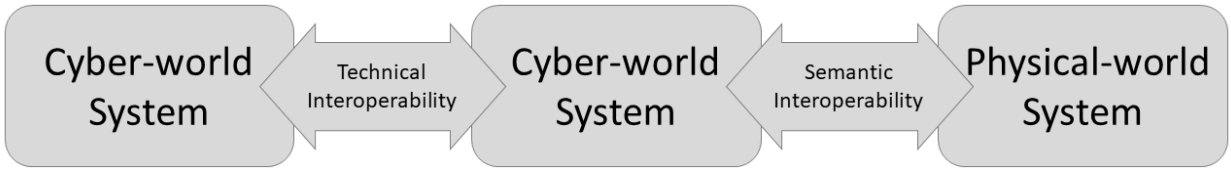


Figure 9. The four levels of interoperability according to Panetto et al. (2019)

Ducq et al. (2012) claim that when running system-of-systems, interoperability problems arise. Before elaborating with those, a system needs to be clearly defined. According to Ducq et al. (2019), a system is a limited set of elements that are related and have attributes, structured to achieve one or more objectives. A system is located within an environment of other elements, forming the environment. In companies, system-of-systems are often found partially integrated through an internal network. The main system in turn has a common objective which unites the subsystems.

3.3.2 Enhancing and measuring interoperability

According to Naudet et al. (2009), homogenization or bridging can solve interoperability issues between any two software. Homogenization means that models and systems are changed so that their interfaces match, requiring that both systems are possible to change, which is not often the case due to legacy problems. Transformations need to be made that can be syntactic or semantic. Naudet et al. (2009) claim that if homogenization is not possible, bridging can solve the problem. The bridge is also known as an adapter and is a system itself that acts as an intermediate between the two other systems. Bridges are often post-constructions when existing systems have interoperability issues. Due to their nature, bridges do not have the same performance in terms of security and robustness and require more extensive verification and validation to ensure that mappings are appropriate.

There are some ways in which interoperability can be measured, according to Ducq et al. (2012). Openness, both in terms of input and output measures how well the software can interact with its surroundings. Systems that only can send data forces other systems to be interoperable, thus the ability to handle input is an important part of system interoperability. The more stable the systems, the better the interoperability as behaviors are constant and predictable. Adaptability is equally

important, which means that the system can change according to environmental changes to be able to still fulfill its objectives.

Panetto (2007) presents four different interoperability maturity models for information systems, technical systems and organizations and shows how they correspond well to each other. Three models have five maturity levels while the fourth only has four. The models are combined and presented in a five-level interoperability scale in figure 10. At level zero the data is unstructured and hard for different systems to understand, all interoperability is in form of manual work, e.g., manually input of data in software or talking with a colleague. Level one includes documented and structured data, which is possible to share between systems, but manual work is still needed to make the information exchange. At level two the interoperability can be performed between systems on the local network. Each system is a black box with a standard interface allowing a simple exchange of data thanks to frameworks and logical data models. In level three the information sharing starts to become more advanced, here the data is located on a network which allows multiple systems to access it. The data transfer is also automated using common exchange models such as unified modeling languages. At level four all involved systems harmonize with each other and the exchange of information is seamless without any misinterpretations, even when different languages or formats are used.

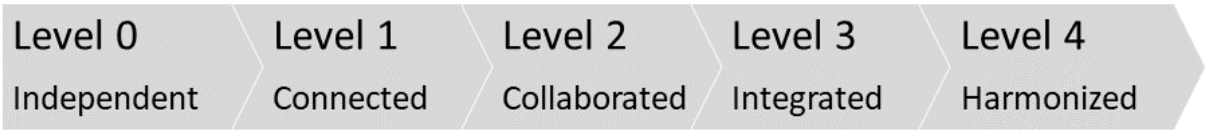


Figure 10. The maturity levels of interoperability (Panetto, 2007)

3.4 Motion and time data management

Apart from being able to generate instructions, the manikin simulations used will act as a reference to be able to set appropriate cycle times. Therefore, the concept of predetermined motion time systems and time data management will be introduced below.

3.4.1 Pre-determined motion time systems

Pre-determined Motion Time System (PMTS) is a standardized way of measuring work by observing the human motions in manual assembly (Zandin, 2001). The motions are categorized and correspond to different standardized times it will take for an average skilled worker working at a standard pace to perform it. A normal pace means that it should put the operator through a workload that is sustainable for an entire workday.

There are several standards available, each having various levels of precision. The higher the precision the more time-consuming the mapping of the motion time blocks will be (Zandin, 2001). Volvo GTO has an internal standard which is called MCMT4. The determination-speed ratio tells how much longer time it takes to set the planned time in comparison to the actual process time (Kuhlang et al. 2014). If using a detailed system such as MTM-1, it can be as high as 200 times longer. The accuracy should be in accordance with the time data usage area. For strategic decisions, accuracy can be lower compared to operational which require more detail.

Zandin (2001) declares three major advantages of PMTS. The first is that method problems and inefficiencies are brought forward as tasks are analyzed. Secondly, no performance rating of the operator is required from the analyst, making the standard objective and consistent. Thirdly, it can be used in the planning phase as a visualization of the task is enough to use motion blocks.

3.4.2 Time data management

According to Kuhlang et al. (2014), many companies do not have a well-functioning Time Data Management (TDM) system in place as they consider it costly and complex. TDM encompasses determination, pre-processing, application and administration of time data. Kuhlang et al. (2014) state the importance of having proper time data to plan and improve production processes.

Almström & Winroth (2010) investigated the reasons behind the gap between actual and registered times. According to them, wrongly set times will have the company make the wrong decisions. Prices are set on the wrong premises and investment calculations will not be accurate. Also, attempts to use support systems such as assembly execution systems will fail as they got incorrect data to base their decisions on. The reasons for this mismatch are many according to Almström & Winroth (2010). One is that wrong methods are used to set times, often due to lack of time or knowledge. Another is the fact that changes are made to methods and processes, but time data is not updated. The third issue is that additional time is added, allowance time to handle disturbances. Figure 11 shows their proposed data feedback cycle which takes the actual time data from production into account and then uses it for production planning and manufacturing control.

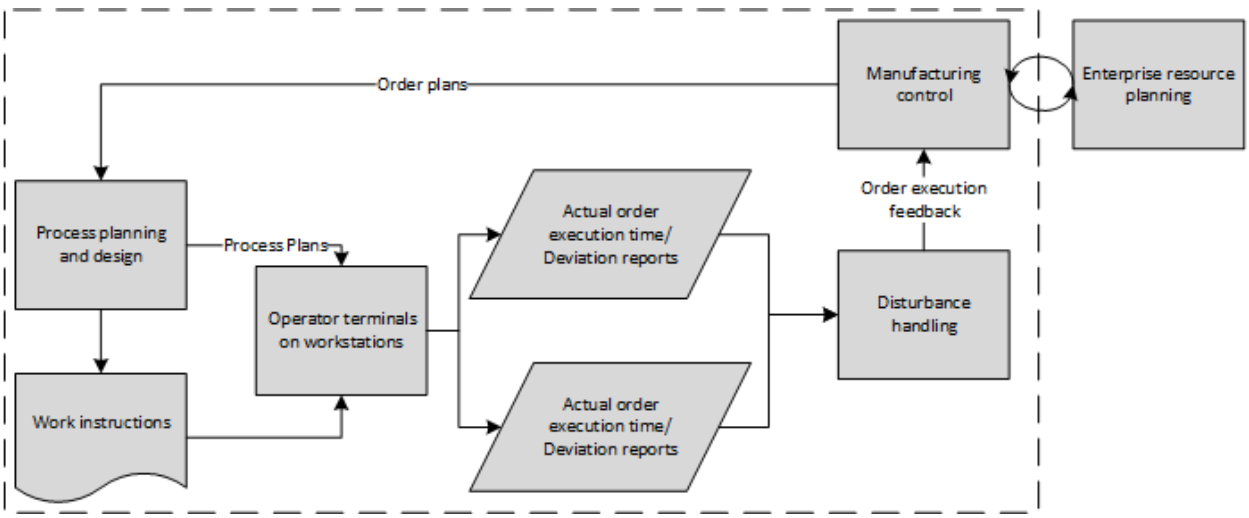


Figure 11. Description of TDM IT system (Almström & Winroth, 2010)

4 Information system landscape

To be able to auto-generate work instructions, multiple systems are included in the process to gather appropriate data. Below follow descriptions of their functionalities and interfaces which have been mapped throughout the project. IPS IMMA is a manikin simulation software designed to perform analysis of assembly ergonomics and manufacturability. AviX is designed to perform method analysis and line balancing. Data will be brought together in the ThingWorx IoT platform and its Operator Advisor assembly work instruction application. Figure 12 gives an overview of the information system landscape of this project.

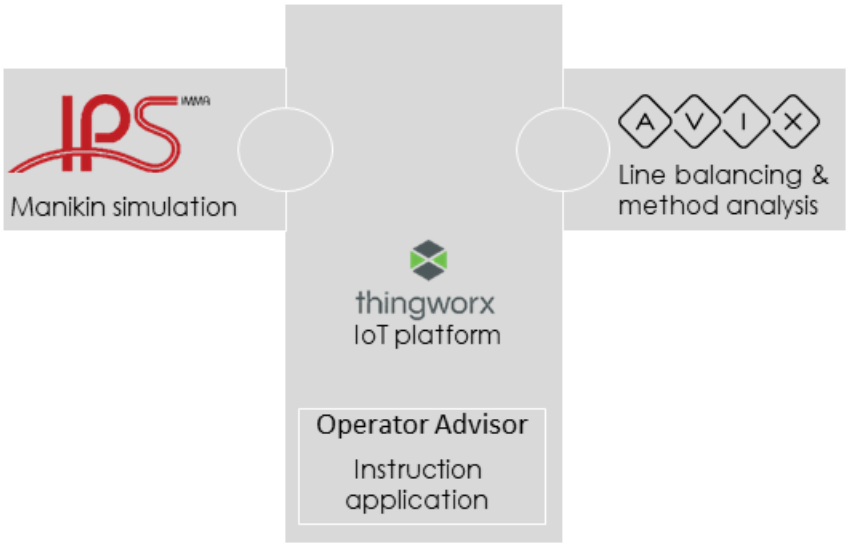


Figure 12. Overview of the system architecture and included modules

4.1 IPS IMMA

IPS is a software platform developed by Fraunhofer Chalmers Centre (FCC) to assist product design and manufacturing preparation processes. Initial functionality focused on robot path planning, to later be extended to spray painting simulation and assembly of flexible bodies. Intelligently Moving MAnikin (IMMA) is the latest addition to the IPS platform used to simulate human body movements, thus being a Digital Human Modeling (DHM) tool. The development of the IMMA module started after identifying that the assembly preparation process, as well as the product design process, could benefit from the possibility to virtually assemble products to ensure manufacturability and ergonomics. According to Högberg et al. (2016), other existing DHM tools focus on static postures and not on human movements. The development started and still takes place together with industrial partners, mainly from the vehicle industry, of which Volvo GTO is one. IMMA is an important part of the MOSIM project. A screenshot from the graphical user interface of the IMMA tool is shown in figure 13.

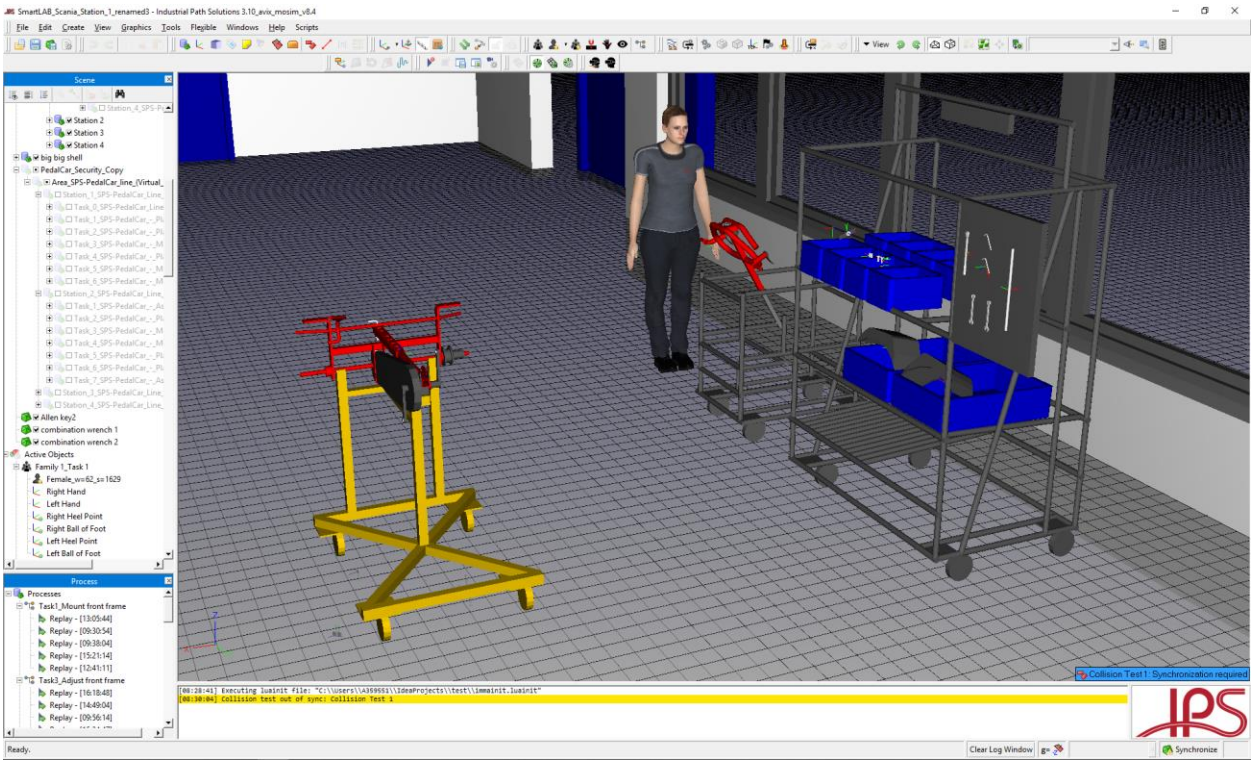


Figure 13. The graphical user interface of IPS IMMA

4.1.1 Functionality and workflow

The manikin is composed out of 162 joints and 82 segments to be able to simulate the human body (Hanson et al., 2019). The segments are assumed to have a mass and therefore center of gravity. A collision model is incorporated to avoid the manikin to crash into other objects in the virtual environment, as well as itself. According to Hanson et al., (2019) inverse kinematics are used to calculate the movements, supported by a so-called comfort function to penalize inappropriate body postures. Like in other DHM tools, manikin families can be created to be able to simulate manikins having different anthropometric measurements.

As IMMA uses mathematical modeling initially used for path planning, the body joint positions and angles are precise (Högberg et al., 2016), making it possible to extract lots of data to perform ergonomic evaluations of various kinds. Hanson et al. (2019) claim that many companies in Sweden have their own ergonomic evaluation methods. IMMA is compatible to use any of these or any standard evaluation method such as RULA or NIOSH. Ongoing research is trying to enhance the capabilities of IMMA, such as robot-human collaboration and vehicle driver simulation. As of today, IMMA does not allow handling of multiple parts at the same time or performing multiple tasks, such as looking and gripping at the same time. This means that the simulated time most probably will be longer than the actual.

According to Hanson et al. (2019), the aim has been to design software with a high-level programming language to enable engineers, ergonomists and others with or without programming knowledge to use the software. For example, commands such as “grasp hammer” should be

possible to use as they are similar to instructions given to operators. Högberg et al. (2019) claim that the instruction language used as input to the IMMA tool is inspired by PMTS-systems like MTM-SAM. Mårdberg et al. (2014) managed to map the high-level instruction language to SAM and MOST PMTS-systems automatically. By defining grammar, instructions could then be generated. However, the study emphasizes that mappings between the simulation software must be made specifically for each PMTS. As many companies use their own standard and the same goes for Volvo GTO, no effort was made to perform such mapping in this project.

4.1.2 IMMA information model

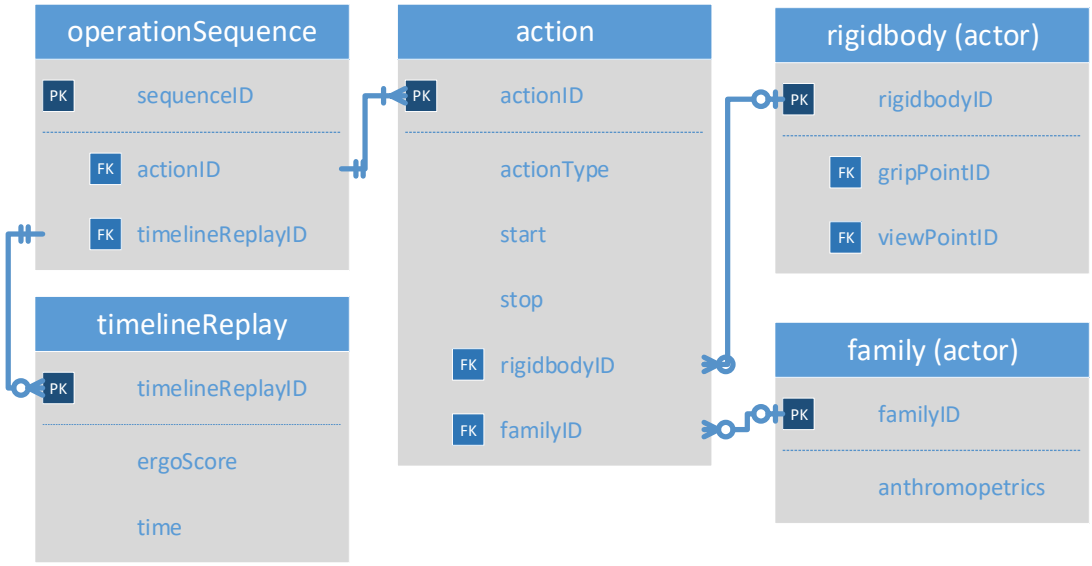


Figure 14. IMMA information model

In IMMA the simulation is described as an operation sequence. The operation sequences consist of multiple actions such as *Look at*, *Stop looking*, *Grasp*, *Release*, *Move to*, and *Follow*. These in turn are performed by actors, which are families and/or rigid bodies. Viewpoints and grip points are also defined on the rigid bodies and used to coordinate action movements, such as gripping or looking at a rigid body. In all cases except for follow, the actions belong to the family. In the follow action, the manikin is tied to the grip points of the rigid body and follows its motion. For a simplified information model of IMMA, see figure 14.

In the operation sequence, a table is defined showing which actions are to be performed and by what actor. When running the simulation, the timeline replay object is created which stores the ergoScore for the sequence as well as the time and path of rigid bodies and the manikin. The time is further specified on action level, please see figure 15 for graphical representations taken from task 1 of the pedal car case.

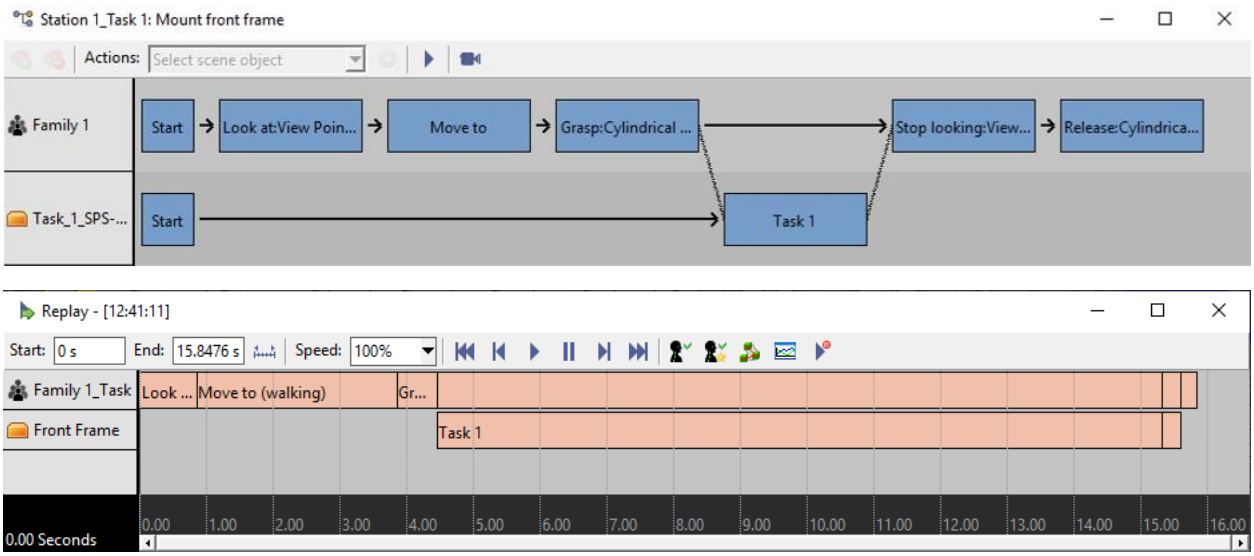


Figure 15. An operation sequence in IPS IMMA (above) and its timeline replay (below).

The actions available in IMMA currently are *Move to, Look at, Stop looking, Grasp, Release* and *Follow*. *Move to* is when the manikin moves to a position. *Look at* constrains the manikin to look at a certain point, while *Stop looking* removes this constraint. *Get* is when the manikin grasps one or more grip points of a rigid body and *Release* is when the manikin releases it. In figure 16, the hierarchy of the rigid body, grip points and viewpoints are exemplified along with a representation of the grip points and the manikin attached to these. The *follow* action is when the manikin follows a rigid body that it has grasped while moving it along a pre-calculated path.

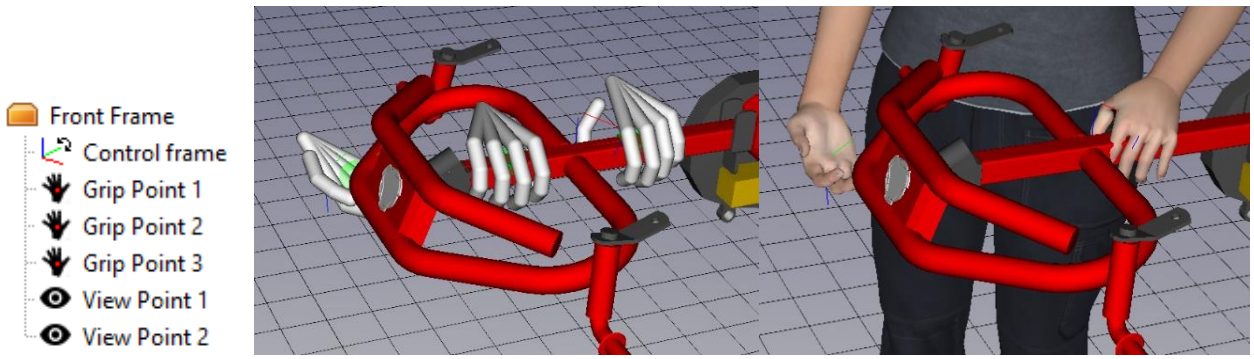


Figure 16. Snippets from IMMA scene showing grip points.

Grip & viewpoints are children of the rigid body (left), they are attached to the rigid body (mid) and the manikin in turn is attached to the grip points (right).

4.1.3 Input to IMMA

Hanson et al. (2014) describe the work sequence for performing a simulation in IMMA and thus the input needed to perform the simulations, see figure 17.

1. Geometries are imported to the software, using any CAD-format. This goes for both products to be worked on and the surrounding environment where assembly takes place such as storage racks and tools.

2. Parts to be assembled are defined and collision-free paths are calculated by defining start and end positions. Often, the final assembly is imported and those parts to be assembled are defined as rigid bodies, moved to their start positions and paths are then calculated.
3. Contact points are defined in the environment, that is grip points and viewpoints
4. Anthropometric values are defined so that a manikin family can be created. The manikin family can be predefined in a manikin library. The manikins are then linked to the contact points.
5. Simulation is performed along with visual verification.
6. Ergonomic assessment based on the chosen evaluation method is performed.

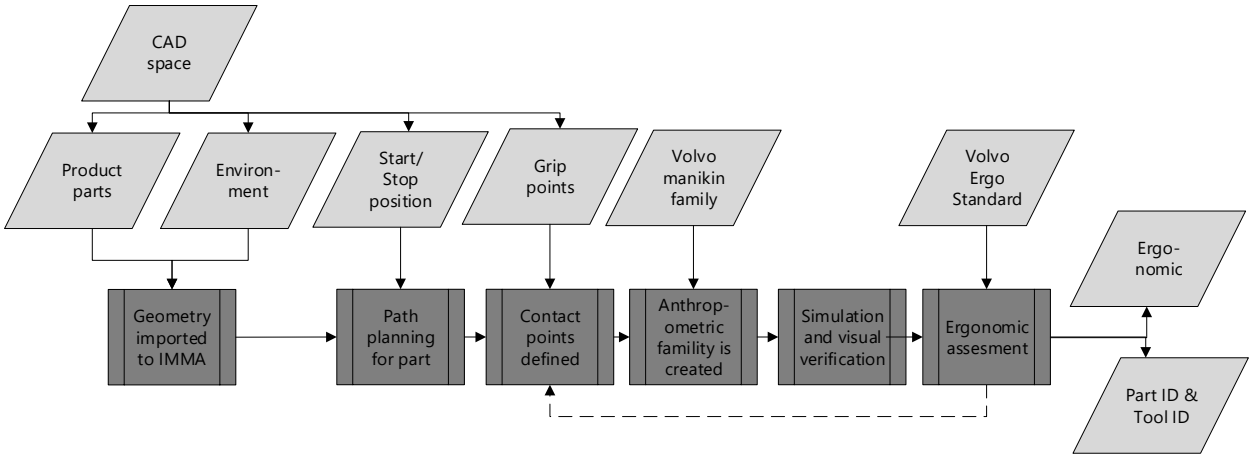


Figure 17. An overview of IMMA

4.1.4 Output from IMMA

IMMA has the possibility to export data using its Lua API (Application Program Interface). Since IMMA is a new module within the IPS suite, the API functionality related to the manikin simulation is limited. Rigid bodies and their path planning are part of the IPS base module, thus the related API functionality is more extensive. Only data suitable and valuable for work instruction generation will be presented here. The Lua script used to extract the data is presented in Appendix III.

- Item data: includes both parts and tools as they are modeled the same way as rigid bodies. By scanning the scene tree, grip and viewpoints can be found, and since they are attached to an item, the item itself can be found using predefined functions.
- Sequence data: name as well as the following subcategories
    - Actions instructed to be performed by the manikin such as: move to, look at and grasp. Related grip and viewpoints can be found by decoding the action name which is a composition of action type and any grip or viewpoints.
    - Ergonomic data: an average score for a task based on a chosen evaluation method such as RULA or REBA.
    - Time data: the time it takes for the manikin to perform the task. Note that the manikin cannot perform actions simultaneously, so the simulated time might be too long.

4.2 Solme AviX

AviX is a software currently used in manufacturing preparation to balance workstations (Enofe, 2017). AviX is developed by Solme AB in cooperation with industrial parties such as Volvo GTO. The main idea when designing AviX was to combine videos from assembly with PMTS evaluations to set cycle times appropriately. Using that data, the line can be balanced correctly and adjusted for any variant mix. As of today, additional functionality is available such as DFX, SMED and FMEA modules. The graphical user interface of AviX is shown in figure 18.

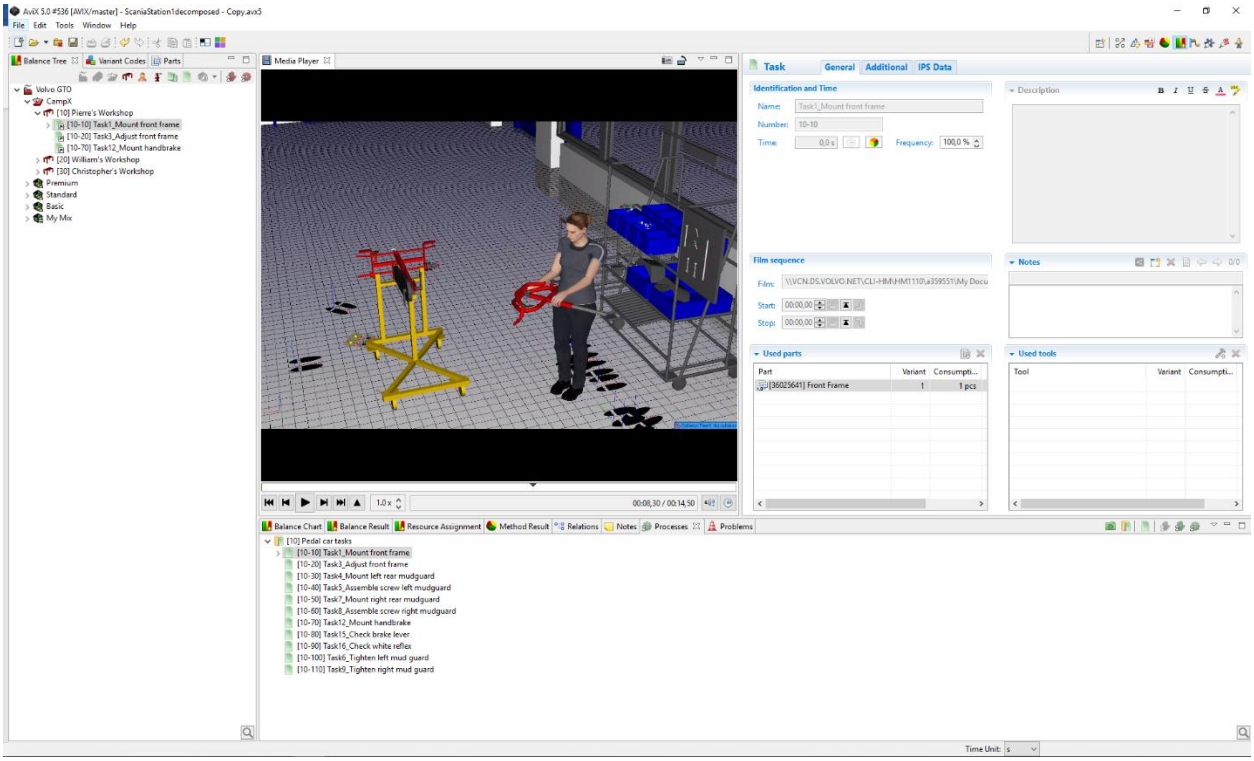


Figure 18. The graphical user interface of Solme AviX

4.2.1 AviX information model

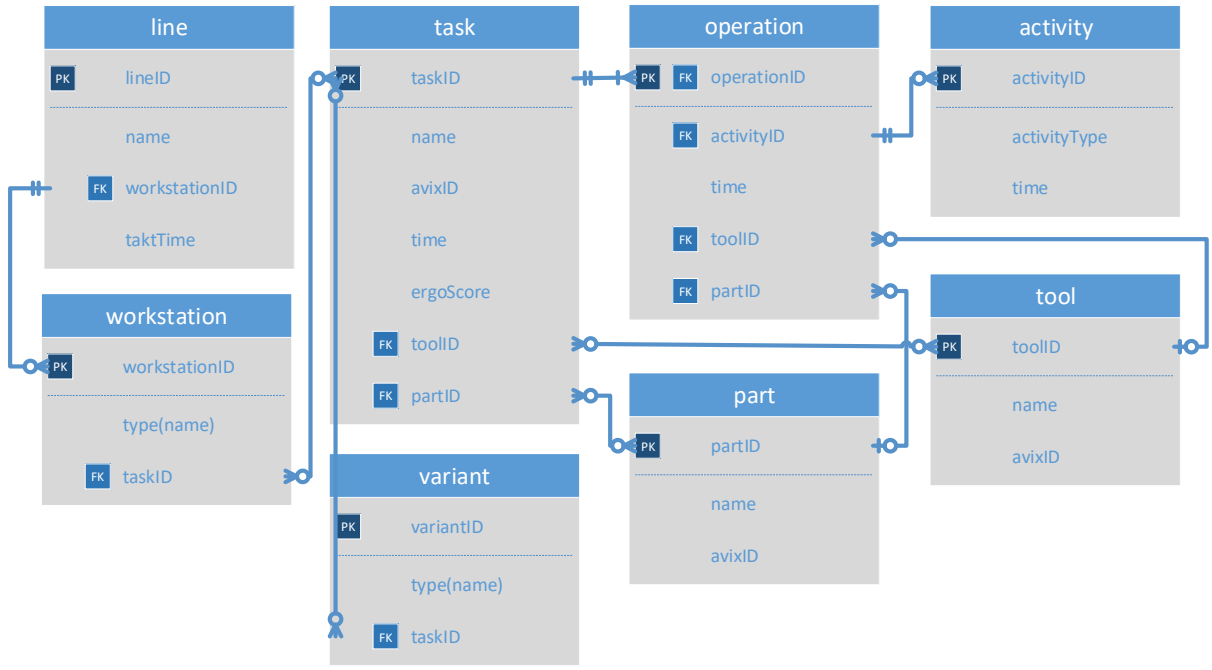


Figure 19. AviX information model

In the AviX information model, each factory is divided into lines, holding workstations which are assigned tasks, see figures 19 and 20. The tasks are assigned to different variants, so depending on which variants are to be made, different tasks will run. Each task can consist of one or more operations. The operation in turn can be divided into one or more activities for which thorough time analysis can be made. The parts and tools can either be set directly in the task or in each operation. In case of doing it in the operation, only one tool and one part can be assigned. The tool and part can then be found in the task as the need is derived via the operation.

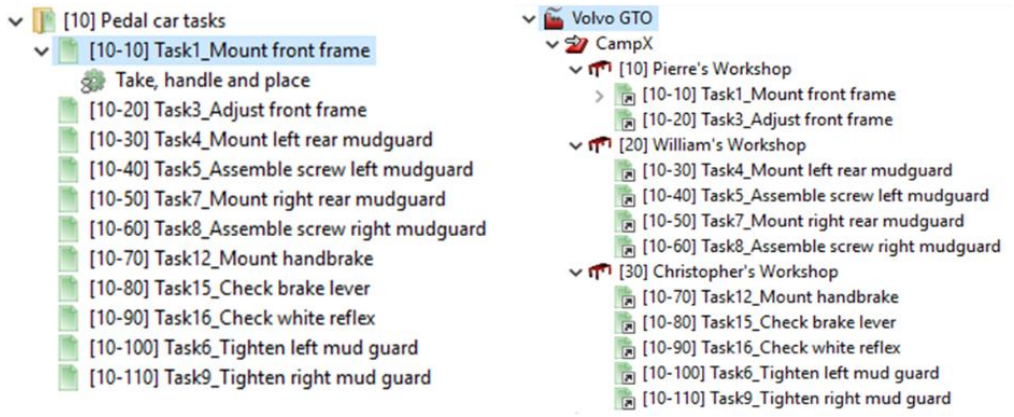


Figure 20. AviX process tasks (left) and work tasks (right)

4.2.2 Input to AviX

Data can be imported from PLM, ERP and other IT systems using the AviX Upstream Connector. The import module is often modified to the specific company, for example, the Volvo version ensures that data from the Sprint manufacturing preparation system can be mapped to the AviX task structure. In case data is not imported, or data is missing it must be added manually.

4.2.3 Output from AviX

AviX uses the Downstream Connector module to enable the export of task data. The data is supposed to be used in other systems, like MES to generate work instructions and plan processes. The export holds an XML file structure which is possible to modify by Solme, however not by the user of the software itself and contains the following:

- Task data; name, number and description as well as the following subcategories
    - Tools used: their name, number and description
    - Parts used: their name, number and description
    - Instruction string: the task instruction string is either entered manually into AviX or based on the activity division performed in the software, see figure 21. Note that activities themselves are not included in the export.

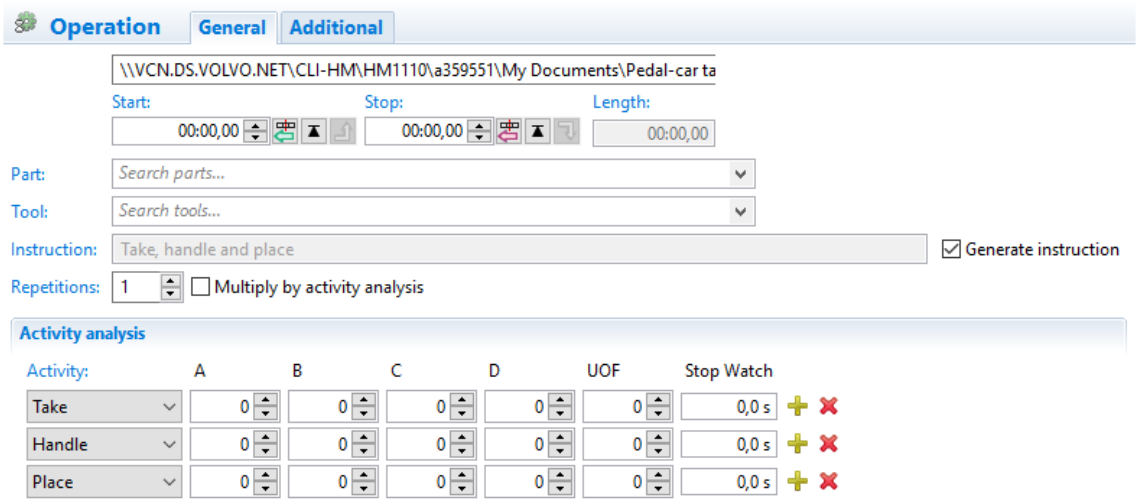


Figure 21. AviX operation and activity setup window

4.3 IMMA and AviX integration

The IMMA and AviX integration is currently under development and its functionality is presented below.

4.3.1 IMMA and AviX interoperability workflow

AviX has the possibility to export a JSON file of the task structure and related parts and tools with the purpose of being readable by IMMA. An import wizard, under development, helps the user to link tasks in AviX to operation sequences in IMMA. When running a simulation in IMMA,

ergonomic evaluation scores are saved. When running the export function to AviX, the same JSON structure that was put in is generated, with the addition of the ergonomic evaluation scores to each task. There is an ongoing project which aims to have the ergonomic scores mapped against the ergonomic evaluation tool built into AviX once imported. As this thesis aims at generating work instructions rather than ergonomic assessments, this functionality is not used.

4.3.2 Similarities in information models of IMMA and AviX

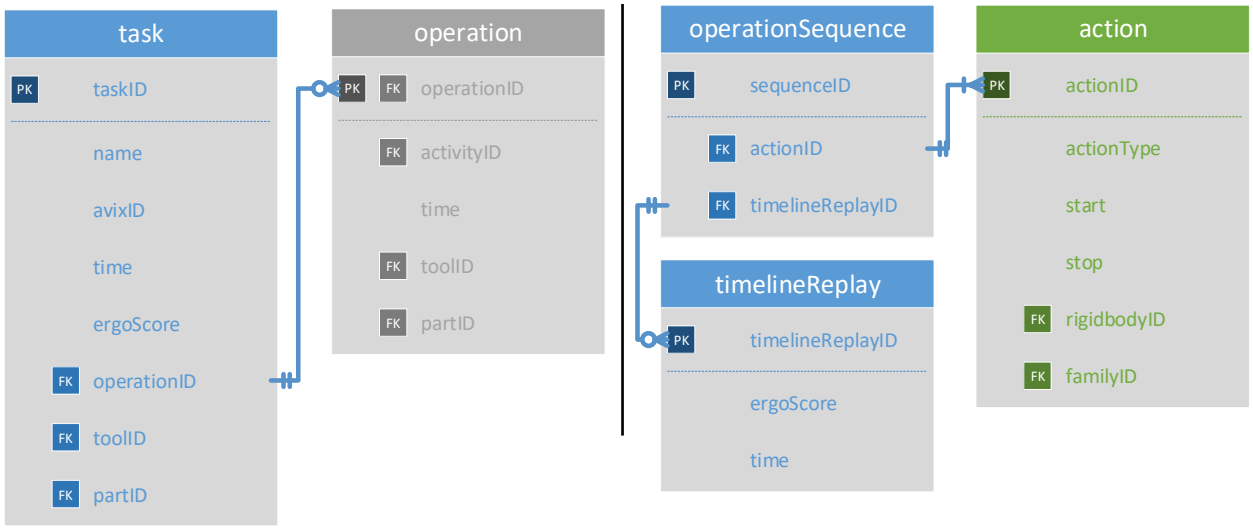


Figure 22. Task in AviX (left) correspond to operation sequence and timeline replay in IMMA (right)

The operation sequence in IMMA is similar to the task AviX, see figure 22. For example, they could reflect going to a material rack, picking up material and then assemble it on another part. When importing the AviX export into IMMA, the task is linked to the operation sequence. The operation object in AviX has a better resolution than the sequence in IMMA, however different from the action object. The activity in AviX rather corresponds to the action in IMMA as they both describe basic movements such as get/take, follow/handle, place/insert, see figure 23 for comparison. Note that the actions are not included in the export from AviX to IMMA and there is no plan to do so as the activities are claimed to be different from actions in IMMA. There exist more activities than actions, for example, *take, place, handle, affix* and *adjust*. The actions are more similar to instructing a human, while activities are more similar to PMTS-blocks.

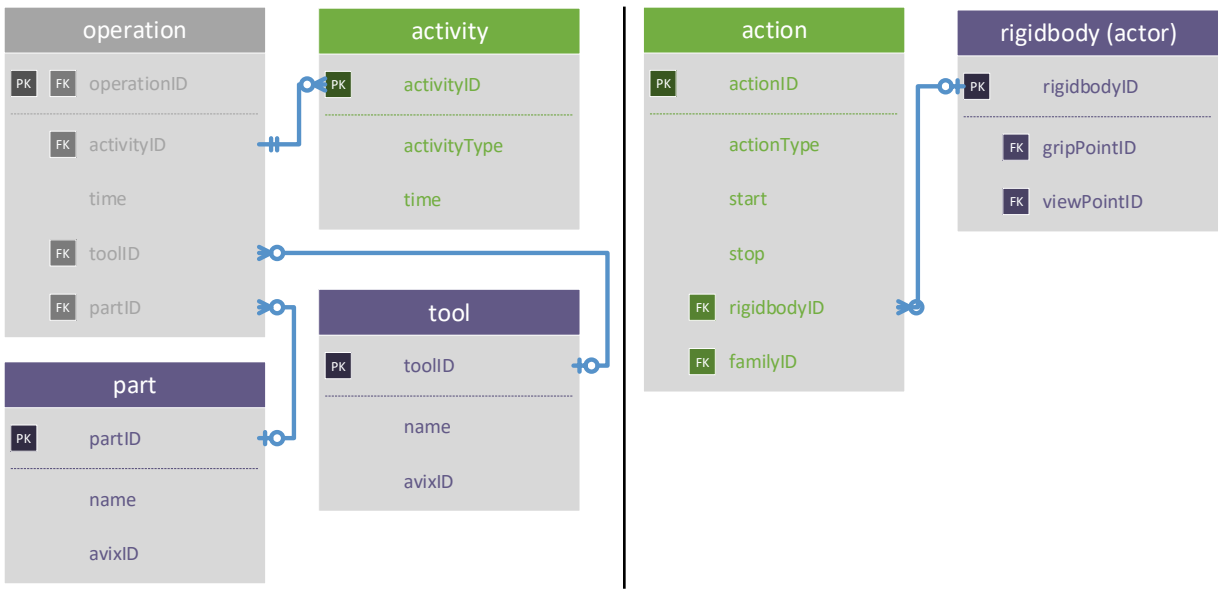


Figure 23. Parts and tools in AviX (left) correspond to rigid bodies in IMMA (right)

The tools and parts in AviX are linked to the task or operation. As the linkage is on a higher level in the hierarchy, it is not possible to say which activity interacts with the physical objects and in what way. The rigid body in IMMA corresponds to the tool or part in AviX, as it is the physical object that is interacting with the human in some way.

As AviX sets the initial file structure of the exports, its structure is similar to the information model of AviX itself. The ergoScore is added by synchronizing the ergoScore in the operation sequence to the task. In the export, the ergoScore of the last created timeline replay of the operation sequence is the one set.

4.4 ThingWorx

ThingWorx is an IoT platform which is planned to be used at CampX, which is hosting the Volvo Group Smart Factory arena. ThingWorx offers functionality such as device and sensor connectivity, data analytics tools and the possibility of having the user design apps themselves.

4.4.1 Developing applications in ThingWorx

ThingWorx uses an object-oriented approach where Things are like object instances in any object-oriented language. Things can be created as they are, but also by creating them as an instance of a ThingTemplate, which is like a class in Java. By taking this approach, multiple Things can be updated by just changing their common ThingTemplate. A Thing or a ThingTemplate can also implement one or many ThingShapes, which can be seen as elements of a Thing or ThingTemplate. ThingTemplates can inherit other ThingTemplates and the same goes for ThingShapes.

Things have properties, services and events which are defined directly in the Thing or by inheritance from its ThingTemplate or ThingShape. A property is an attribute such as name, ID, number, type, etc. Services are executed by calling them from services in other Things or when an

event occurs. An event happens when a property exceeds a value in some way and can be linked to a service in another Thing. Services are functions that a Thing can execute to perform calculations, data mappings, changing states, sending notices, etc.

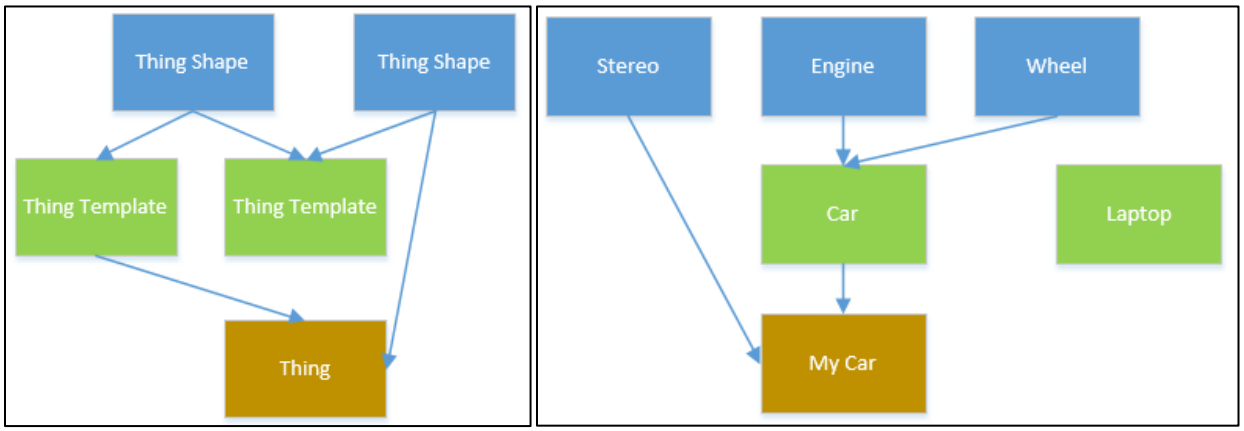


Figure 24. Generic creation of Thing (left) & specific creation of Thing (right)

To further emphasize the difference between the ThingShapes, ThingTemplates and Things a modeling example of a simple car has been made up, see figure 24. A certain car (My Car) is a Thing which is similar to other cars in many ways. The general functionality of a car is defined and inherited from the Car ThingTemplate. Any car in turn has certain elements that interact, have properties and behaviors such as an engine and four wheels, each described by individual ThingShapes inherited by the car ThingTemplate. My Car in particular has a stereo built-in, which is defined by a separate ThingShape. The car ThingTemplate does not inherit the stereo ThingShape as all cars do not have a stereo built-in. A Thing, in this case, My Car cannot inherit multiple ThingTemplates, meaning it cannot be a laptop at the same time being a car.

Data in ThingWorx other than properties of a Thing, like job and work order data, is stored in a database. Within it, data is stored in tables of different kinds. These are defined by DataShapes which describe what columns the table is made of. Using services in Things data can be read and written to the database.

4.4.2 Operator Advisor manufacturing application

The Operator Advisor is an app in ThingWorx that enables functionality to design and show digital assembly work instructions. The app is a package of Things, ThingTemplates, ThingShapes and DataShapes that has services, events and properties that interact to enable such functionality. The Operator Advisor data model is extensive and only those parts used for this project will be presented, please see the PTC help center for more extensive information.

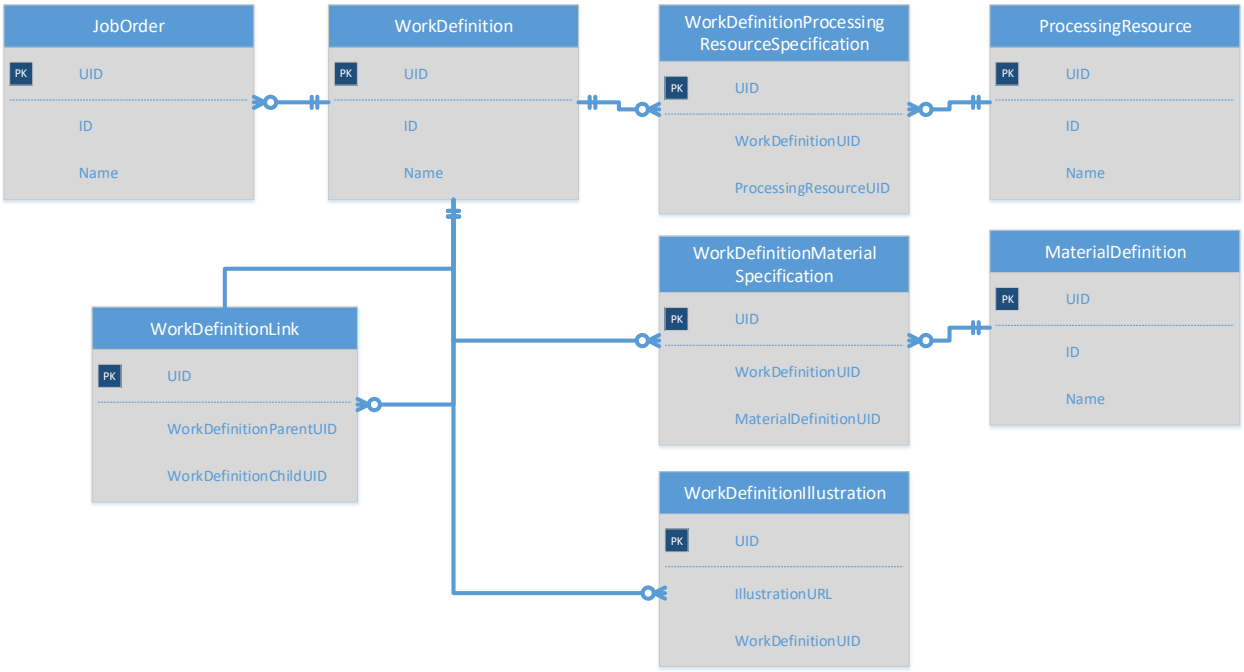


Figure 25. Used parts of the ThingWorx Operator Advisor information model

The WorkDefinition object is the centerpiece in the Operator Advisor information model, see figure 25. It holds information on the task such as instruction string, duration and notes. The WorkDefinition can have MaterialDefintions and ProcessingResources tied to it, which are representations of parts and tools respectively. They are not referenced directly from the WorkDefinition itself, but instead, WorkDefinitionProcessingResourceSpecifications and WorkDefinitionMaterialSpecifications can be created, acting as links by holding the UIDs of the objects it links together. In the link, the quantity can be defined. A link can only hold one WorkDefinition and one MaterialDefinition or ProcessingResource at the time, however, several links can exist for the same WorkDefinition, MaterialDefinition and ProcessingResource.

There is a possibility to link a WorkDefinition to another one, thus making a hierarchy of WorkDefinitions, using the WorkDefinitionLink object. In that way, tasks can be grouped which makes the steps required to assemble the product easier to overview. The WorkDefinitionIllustration object can hold references to pictures, animations or similar.

4.5 File formats

The file formats used in the demonstrators are presented shortly below. For more thorough explanations, please see relevant guides on the topic.

4.5.1 JSON file format

The JSON file format offers flexibility in terms of adding more data on the same hierarchy levels, as well as adding additional levels, and therefore suitable for the iterative development approach. When programming in Java and other languages there are several packages available for reading and parsing JSON format data. Most programmers are familiar with the file format, and if not, it is easy to learn and understand. Another advantage is that it is easily readable by humans due to its semantics. A snippet of the JSON format export used for the AviX and IMMA integration is shown in figure 26.


```
{
  "ipsavix": {
    "version": "0.2.0",
    "blockMarkers": [
      {
        "name": "Task1_Mount front frame",
        "number": "10-10",
        "avixID": "5fab9f04-2635-438a-beb9-1d3c233f546a",
        "parts": [
          {
            "name": "Front Frame",
            "number": "36025641",
            "avixID": "8705b43a-fee1-453f-933d-3d9cba0b651d"
          }
        ],
        "tools": []
      },
      {
        "name": "Task3_Adjust front frame",
        "number": "10-20",
        "avixID": "d221b6f9-4d6e-4f81-90a9-6b1eb61d9f4f",
        "parts": [],
        "tools": [
```


Figure 26. JSON file from the AviX/IMMA export

The data types which can be stored in the JSON format are numbers, strings, booleans, objects and arrays. Objects are collections of one or more key-value pairs, where the key is a string and the value can be of any data type. Arrays are ordered lists of values or other JSON objects.

4.5.2 XML file format

Like JSON, the XML format is designed with the purpose to be human-readable. The XML file format offers similar flexibility as the JSON format in terms of adding more data on a hierarchy level or extending the hierarchy itself. Programming applications use to have extensions to read the XML file format, and the same goes for ThingWorx. A snippet of the XML format export coming from the AviX Downstream Connector is shown in figure 27.

```
</line>
<workStations variantString="" versionName="" taktTime="60.0" totalTime="43.398" sequenceHint="1">
  <name>
    <strings locale="" value="Pierre's Workshop" root="true"/>
  </name>
  <properties name="se.solme.avix.custom.ips.rawdata">
    <values xsi:type="se.solme.avix.workinstructions.dataexchange.model:PrimitiveValue" null="true"/>
  </properties>
  <description/>
  <origin id="eaf8ff78-ac48-455f-b172-8d1a6171890f" source="AVIX"/>
  <resources variantString="" default="true">
    <name>
      <strings locale="" value="Pierre's Workshop" root="true"/>
    </name>
```

Figure 27. XML file from AviX Downstream Connector

An XML file is divided into elements, which starts and ends with so-called start and end tags. Between the tags is the content. The element can also hold attributes, which is a name-value pair that is stored within the start tag. Compared to the JSON format, the XML file is just a string of characters, thus it does not hold booleans and numbers as specific data types. Since the applications for the work instruction demonstrators only utilizes strings, it is not considered a problem. In most programming languages like Java and JavaScript, there exist functions that enable parsing strings consisting of only numbers into integers, doubles, or other data types that solely hold numbers.

5 Results

This chapter starts with a presentation of the information model developed followed by the design of its proof-of-concept. The latter one is developed in two versions, the first one which was developed in Java, and the second one which is implemented in ThingWorx and its Operator Advisor manufacturing application.

5.1 Work instruction information model

This section presents the information model and is divided into subsections, each one gradually expanding its scope. Please note that the division does not fully reflect the steps in which it was developed. In some cases, the definition of objects introduced early in the creation process had to be changed as the project proceeded. Crows foot notation has been used to relate the different objects to each other, explained in figure 28. The reason that one or more has been chosen over zero or more is that many objects should not exist if they do not belong to another object, e.g., a part that is never used.

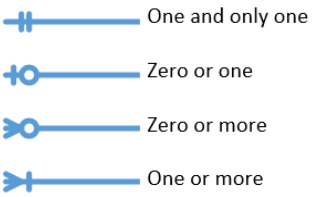


Figure 28. Crow's foot notations

5.1.1 Initial information model

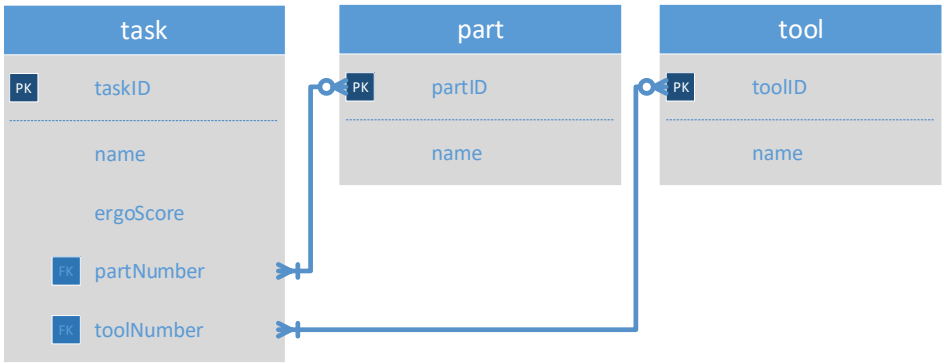


Figure 29. Initial information model

The information model in figure 29, where parts and tools are linked to each task has been used as a starting point in the development. This is a common way to assign parts and tools to tasks as it is simple and straight forward. AviX uses this structure as well as ThingWorx Operator Advisor, linking parts and tools to tasks, only the naming of the objects being different. Both systems can be used for task planning and work instruction generation, thus making this starting point reasonable. The primary keys got the format of *object-type + ID*, e.g., taskID and partID

5.1.2 Extension to partGroup and toolGroup concept

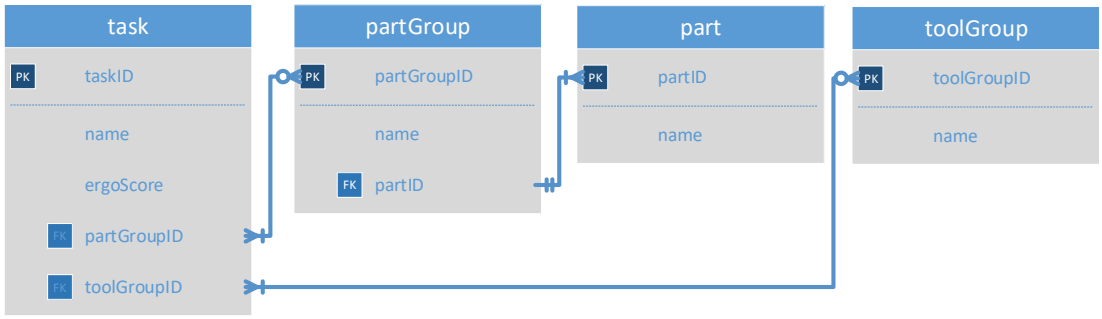


Figure 30. The extension to partGroup and toolGroup

The part and tool objects are extended to partGroup and toolGroup respectively, as in figure 30. The structure allows each partGroup to hold multiple unique parts, meaning that a task can be defined by what partGroup it involves, not a specific part. In a simulation, no matter if the part is black or blue, it can be considered the same task as long as it holds the same shape. Consequently, the same task simulation can be used to model multiple assemblies where the only difference might be the color of the part. The partGroup idea is demonstrated by grouping three mudguards together of different colors, see figure 31.

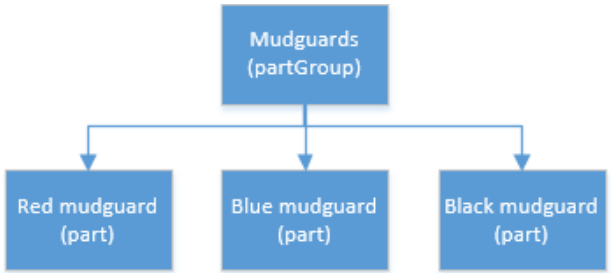


Figure 31. The partGroup concept exemplified

5.1.3 Adding sequences and actions

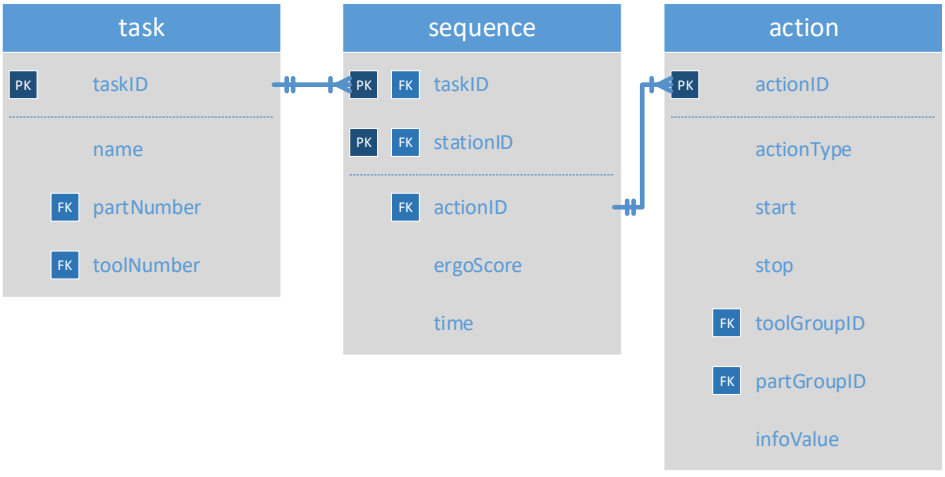


Figure 32. A task extended with sequence and action

The sequence and action extensions are better explained in two parts. First, there is the idea of a task representing one or more sequences. The difference between the two is that a sequence is unique for a specific station, therefore the stationID being an attribute and key in the sequence object, see figure 32. Since the activities to perform might be slightly different when the station design and environment are different, a new simulation will be required. Therefore, the ergoScore and time are moved from the task object to the sequence object. The sequence object has not a primary key, but instead a composite key of the taskID and stationID in combination due to its definition.

The second part of the extension is having the sequence in turn divided into multiple actions. Each action is a part of a task, an idea taken from the IMMA information model which breaks down operations sequences into actions. An action cannot alone compose a sequence, instead, it can be compared to a time block in a PMTS, e.g., MTM-SAM. Getting a part could be an action, or walking from position A to position B. The partGroup and toolGroup have been moved to the action object so that their context can be more specific. When performing a get action, it is interesting to know what object the operator should get. The action object has attributes such as stop and start to further specify the action. These positions are relevant for actions of type *follow* or *move to* as they involve movement from one position to another. The releaseID is set in the *release* action to tell what should happen to the item once released. A tool could be placed on a screw and used to enter it, a pin can be inserted into a hole or a cover could be placed on top of a container. Having releaseIDs such as *use, place* and *insert* is a first step in the direction of being able to provide more fine-grained instructions.

5.1.4 Adding location

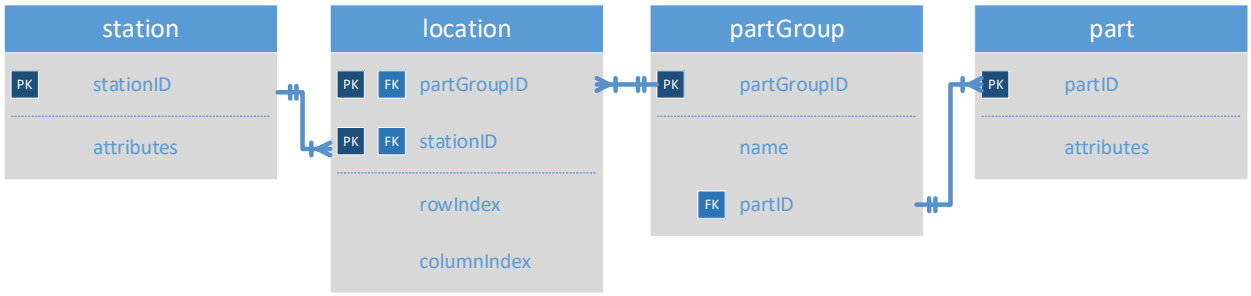


Figure 33. A location object connected to its two foreign keys

An important part of the instruction will be knowing where the part is located. It is assumed that the internal logistics system will deliver parts in sequence so that all parts within the same partGroup will have the same location. Instead of adding a location attribute to the part or partGroup, a location object is created with a composite key, see figure 33. The composite consists of partGroupID and stationID, which means that it will be unique for each unique combination of station and partGroup. Any location parameters, for example, row and column index on a shelf are stored as attributes within the location object.

5.1.5 Handling variants

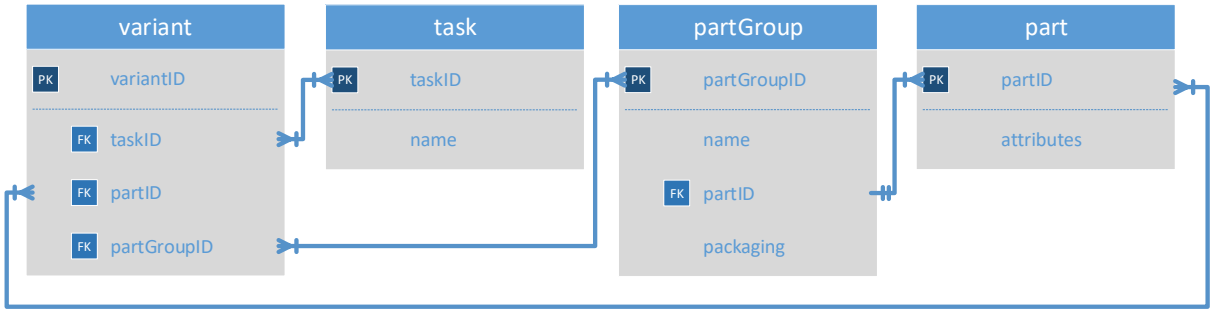


Figure 34. A variant object connected to its two foreign keys

To make a certain product, several tasks need to be performed. The tasks are further divided into sequences and actions. The list of all tasks is stored in the variant object, which is identified by its unique variantID. As the action holds a reference to what partGroup to handle, the variant object must know exactly what part within the partGroup should be used for this specific variant, see figure 34. As there might be multiple partGroup objects in one action, the part must be stored together with the partGroupID in the variant object. Since the parts are stored on task level in the variant object, it can act as a mBoM and thus cope with the challenge that a part might be handled in multiple actions in the same sequence although the quantity to use is only one.

5.1.6 Final model

By combining the concepts and extensions introduced a final information model can be presented in figure 35. Please note that the toolGroup has been left out to simplify the map, its structure is however similar to partGroup.

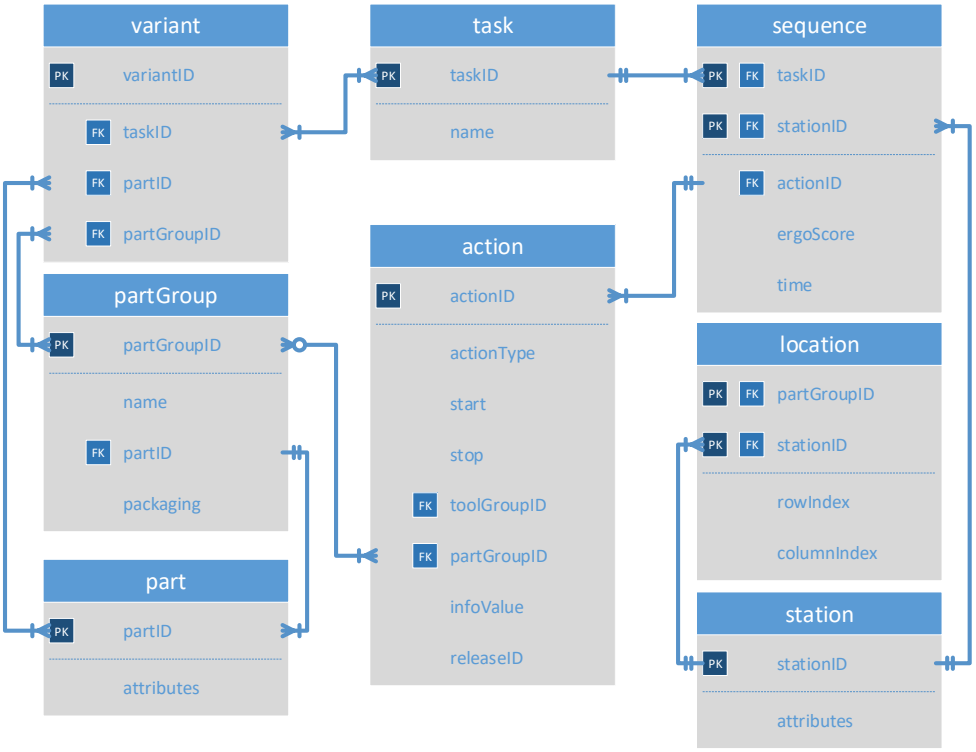


Figure 35. Final information model, note that toolGroup is missing

Variant

Each variant is identified by its ID, for the pedal car case it is made simple by having three variants namely basic, standard and premium. To make a variant, one or more tasks need to be completed.

What differs one variant from another might be just some detail in terms of parts used, for example, the standard pedal car might have blue mudguards whilst the premium variant has black ones. Therefore, the partGroupID and partIDs are stored as attributes in the variant object. The variant object can be seen as a mBoM, storing what parts are needed for a specific variant. By addressing part and partGroup together, once a partGroup is found via an action, the exact part can be identified in the variant object. A quantity attribute is not implemented, instead, the quantity is indicated by the number of times the same object appears for a certain task.

Sequence

The idea of having the assembly divided into tasks is incorporated in both AviX and IMMA, although the naming might be a bit different. In both cases, the task is assigned to a specific station, which makes it a sequence in this information model, since a task is considered to be a group of sequences. A sequence can only be assigned to a specific task, but a task can have several sequences assigned to it.

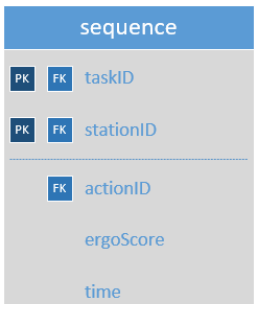


Figure 36. sequence object

A sequence is a task performed in a certain station environment. When both the task and environment is known, so is both placements of material and

tools and their movement, meaning that ergonomic and time evaluations can be stored in the sequence object. As the sequence is affected by the task to be performed and the station environment, it does not have a unique primary key but is composed of the stationID and taskID in combination. Each sequence holds one or more actions which together comprise the sequence.

As of today, the IMMA task simulation takes place at a certain station. AviX has the possibility to assign a task to a certain station as well. In other words, they generate sequence objects. Note that, in contrast to the AviX implementation today, the part and tool references have been moved in the information model to the action objects. To be able to generate specific instructions, the part and tools references must be specific to each action.

Task

The task object is a group of sequences where the operations performed on the product are the same, but the environment is different. When assembling a product, certain tasks need to be performed and it does not matter to the product where those tasks are performed. However, to the operator performing the tasks, the environment is important as different instructions may be needed and to make that distinction the sequence object was created.

A task starts with the operator having free hands and ends in the same state. Going to the material rack to pick up the mudguard and place it on the pedal car is considered a task. Each task is named to be easier identified apart from its ID, which is the primary key.

Action

An action is similar to a PMTS block, being just a simple movement and therefore a part of a sequence. As of today, IMMA uses action blocks such as *get*, *release*, *move to* and *look at*. By making modifications it is reasonable to assume that other divisions can be made. The action object holds many attributes to describe it as thoroughly as possible. It is identified by its ID, which is the primary key.

Figure 37. action object

The most important attribute is the actionType which describes which kind of movement the action refers to, e.g., get, walk, or release. Depending on the actionType, the attributes included are different. *Get* or *release* includes the toolGroupIDs and partGroupIDs of the tools/parts touched, if any. *Follow* actions includes the start and stop positions of the movement. The releaseID is assigned for *release* actions to define what should happen to the item released. A tool could be placed on a screw and used to enter it, a pin can be inserted into a hole or a cover could be placed on top of a container. Having releaseIDs such as *use*, *place* and *insert* is a first step in the direction of being able to provide more fine-grained instructions. Note that this functionality is not available in IMMA as of today. The same goes for infoValue which is a boolean telling if the action provides any valuable information to the operator or is just an action added in simulation added to be able to perform the task properly.

As of today, the action object exists in IMMA and can be exported, however with limited data content. The type is retrieved along with the name of the associated rigid body, if any. It is not possible to see the difference between parts and tools other than the name of the rigid body.

Part

A part is unique and can hold lots of attributes like supplier, color, weight, volume. The parts are grouped if similar in a partGroup object.

partGroup

A partGroup compromises one or more parts which are physically similar. An example is the left mudguard which in turn includes several parts such as a red, green and black left mudguard. In a simulation, a rigid body should be used which represents the partGroup and all its parts. This means that no extra simulation run will be needed for two variants which use mudguards of the same physical design, but with minor differences such as color or marking. The partGroup is defined by its primary key, the ID. It also holds a name to make identification easier. The partIDs of the parts included in the group are saved as attributes in the object.

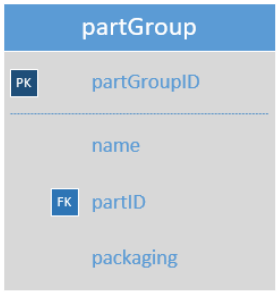


Figure 38. partGroup object

toolGroup

The toolGroup object is not further compromised of tool objects like for the partGroup, thus the naming convention might be misleading. The naming convention was kept as the opportunity to add such a feature should be possible later.

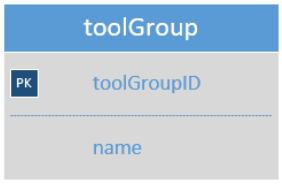


Figure 39. toolGroup object

Station

The station object does not have any suggested attributes except its keys. The stationID is used as a part of the composite key for the sequence and location objects.

Location

The location object tells where a certain partGroup is located if being at a certain station. Since it depends on station and partGroup, its primary key is a composite of the partGroupID and stationID. The reason partGroup was chosen and not partID is that parts are often sequenced at the station but delivered to the same physical position. For example, all mudguards, no matter the color are placed on the middle level of the material rack.

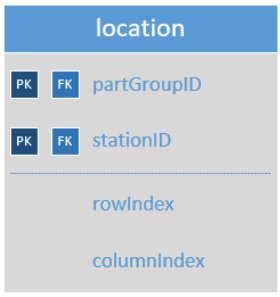


Figure 40. location object

Figure 41 summarizes how the different objects could be populated. IMMA is related to the manikin simulation, while the task allocation management is related to AviX. Note that this is not how the objects are populated as of today, just an idea of how it could look like. Logistics planning, order management and workstation design tools and systems are not included in this project.

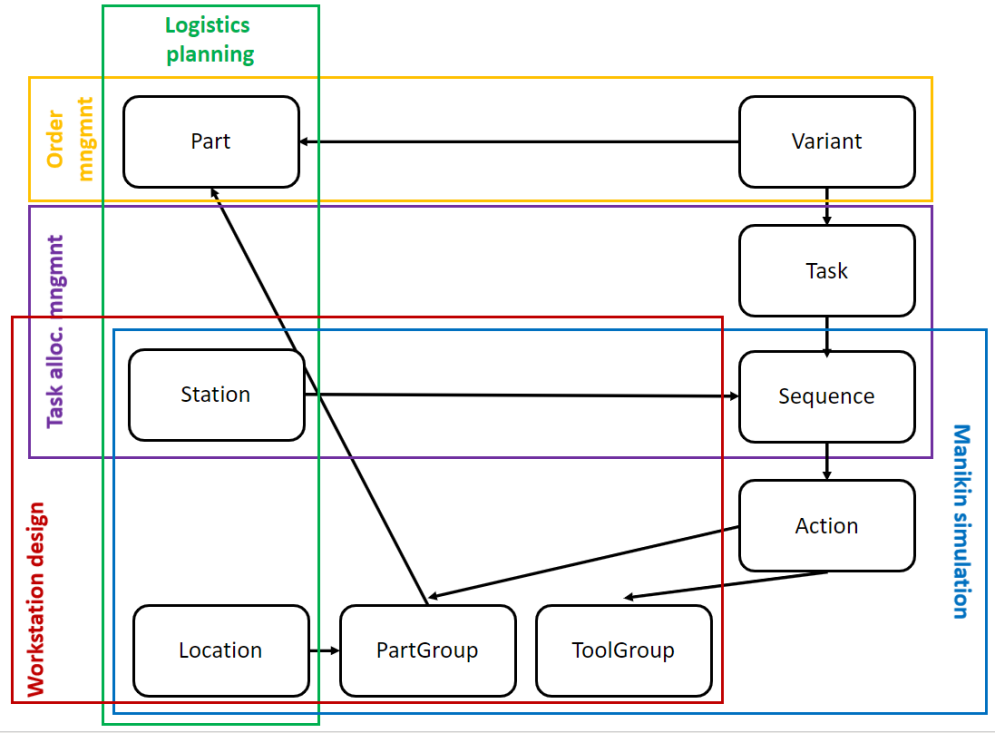


Figure 41. Summary of what processes can populate the information model objects

5.2 Required data to fully populate the information model

To populate the presented information model, the outputs from AviX and IMMA are useful. Below, data is listed that is necessary to populate the information model but not yet available in the exports as of today.

5.2.1 Output from IPS IMMA

- The rigid body should have the ability to be a part or tool. It could be achieved by adding a tag or metadata or split it into two different objects, parts and tools. The reason for needing such a difference is that the part tells what item should be processed, e.g., screw or clip. The tool on the other hand explains *how* an item should be processed, e.g., using a screwdriver or a wrench.
- Start and end positions of paths for parts and tools. Such positions are useful for telling *where* an item should be picked up, placed, put or inserted.

- Release type, which tells when the manikin releases the grip of an item what should happen to it. A part could be placed somewhere, entered or inserted, while a tool could be pressed, turned or used, gives some examples.
- Metadata ability, for rigid bodies and positions, are useful as those would enable IMMA to store data such as object ID numbers in different systems, not to be used in IMMA itself, however useful to any system taking advantage of any export data
- Quantities, especially for screws and nuts since operators often take more than one unit at a time. The quantity may be stored as metadata in the grasp action.

5.2.2 Output from AviX

To populate the information model to a greater extent, tool and part usage need to be assigned on the activity level instead of only the task level as of today. If that is possible, the information model would be more similar to that of IMMA. That enables the information model to be populated manually in cases it is not possible in the manikin simulation software.

5.3 Java demonstrator

The Java demonstrator has been developed as a part of the iterative design process and acts as a proof-of-concept on how data can be structured using the information model described earlier. In this section, its design both from a visual perspective as well as its functionality is described. The architecture and technicalities are described shortly, see Appendix I for full explanation and details. Note that all examples suggest that the information model can be fully populated. Some data is not possible to export from IMMA or AviX and is therefore made up. The ThingWorx implementation discussed in the next section considers those issues.

5.3.1 Demonstrator design and functionality

For the development of the Graphical User Interface (GUI), the Java Swing package has been used due to its simplicity which enables efficient development and keeps the code easy to understand for external parties with limited programming knowledge. See figure 42 showing an example for the pedal car case. As the thesis does not focus on the HMI and its design, its layout should not be considered as an operator interface.

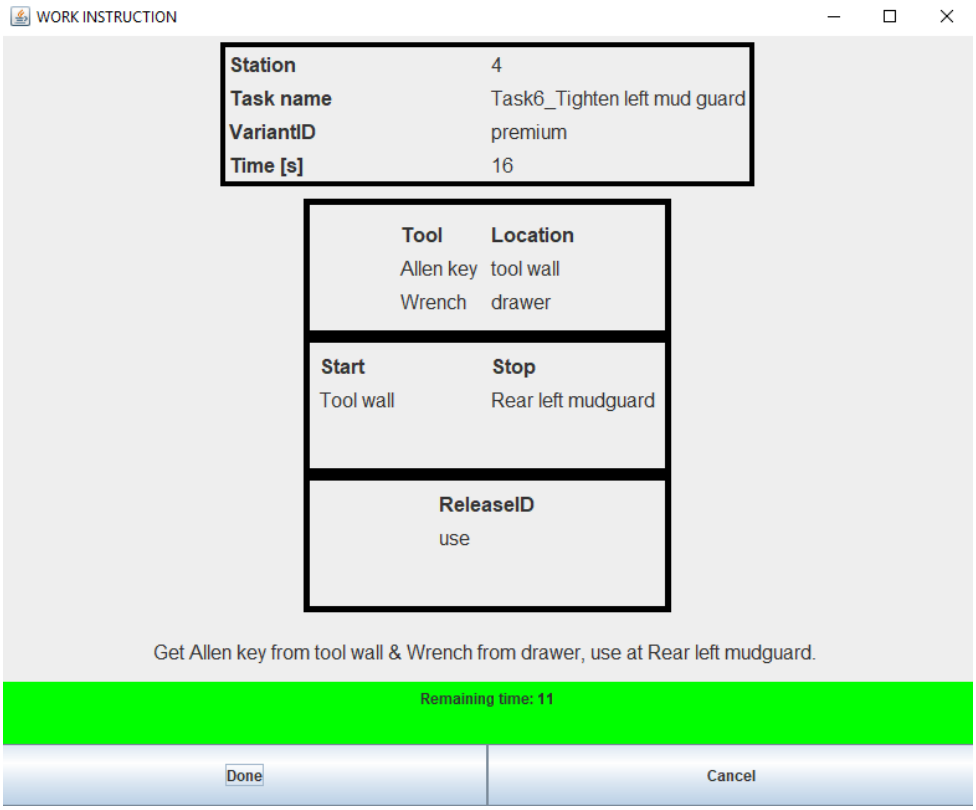


Figure 42. The demonstrator designed in Java to visualize the functionality of the information model

The Demonstrator is divided into three parts, using a Border Layout. The North sector is used to display station, task name, cycle time and variant information.

The middle sector is used to display all actions taking place in the task and should be read in the order top-down. Depending on the action type, different information is displayed. When a part or tool should be fetched, item name, ID, quantity, location and packaging are presented. When an action involves following an item from a position to another, the start and stop position of this path is presented.

The bottom sector is further divided into three sections. In the bottom, a control panel is found. The *Done* button loads the next task, while the *Cancel* button closes the instruction window. In the middle, a timer is found which shows the remaining task time and changes color according to the percentage of time left. At the top, a text string is generated which can be used to instruct the operator. It is generated using the same information as presented in the middle sector of the display.

According to programming best practice, the code has been split into different classes, see figure 43. The frame itself and its graphical constitutes are put together in a FrameUtils class. Methods exist within FrameUtils class that request information from the ApplicationUtils instance to populate the instruction window. The ApplicationUtils instance holds instances of other classes, such as StringUtils and PanelUtils used to build the instruction string and instruction tables. They, in turn, call the ServiceUtils instance which has methods to extract appropriate data from the JSON files where information on tasks and their actions, tools and parts are stored.

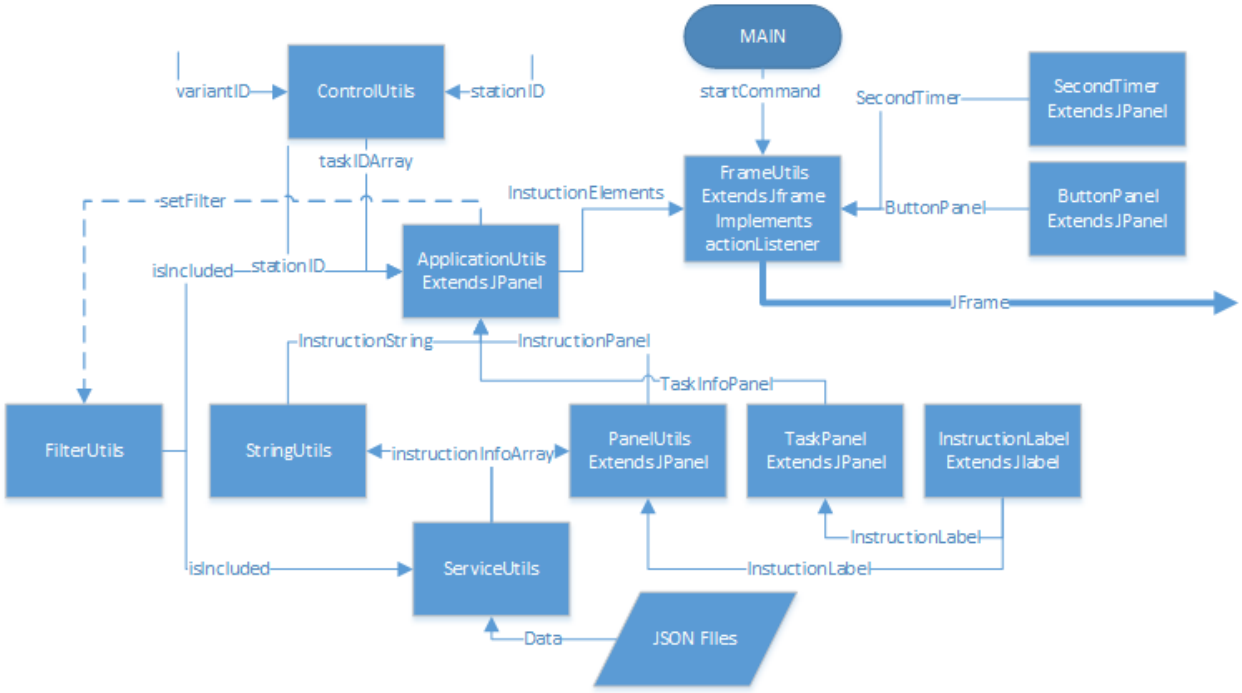


Figure 43. An overview of the Java program created to enable the visualization of the information model

The main method, apart from initiating the instruction window also sets filters and initiates the controller. The filters are located, set and read from a separate class, the FilterUtils class since they are used both in the ServiceUtils class used for getting data from the database (the JSON files) and when delivering the frame elements from ApplicationUtils to the FrameUtils instance. The controller is an instance of the ControlUtils class, in which stationID and variantID are set. Based on this, it is possible to deliver a list of tasks to be performed at a specific station to assemble a certain variant by reading the variant object of the information model. The ApplicationUtils instance uses the taskIDs when calling methods in the StringUtils and PanelUtils instances. The demonstrator classes are described more thoroughly in Appendix I.

5.3.2 Dynamically generated instructions

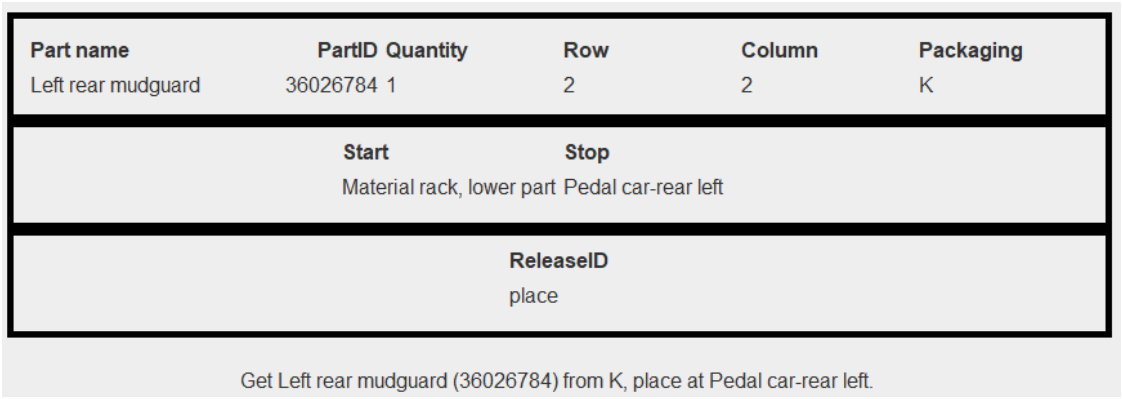


Figure 44. Action instruction list for mounting the left rear mudguard, no filters active

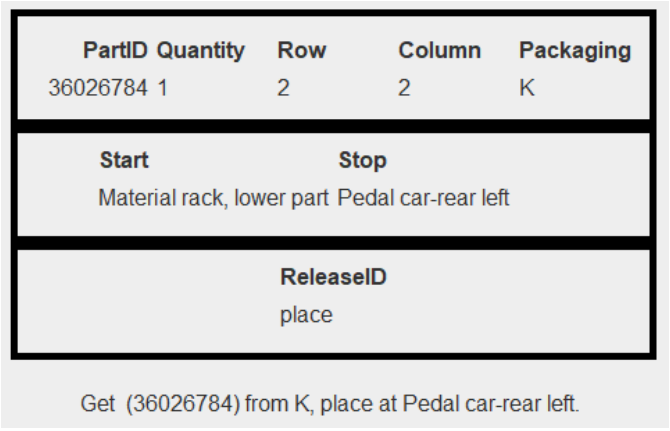


Figure 45. Action instruction lust for mounting left rear mudguard, part name is filtered out

The Java demonstrator has functionality that enables the user to set filters, what information is interesting to show in the tables and string respectively. Since data is stored in a structured way, just the data needed will be retrieved and put together. For example, regarding part information, the user can choose to include both part ID and number as in figure 44, or only partID as in figure 45. To take another example, the user can choose to include packaging information or not. In a real-world scenario, certain filter settings do not make sense, for example not including any part information, name nor number will make the instruction useless as the operator cannot understand what part to get.

5.3.3 Instructions possible to generate

Using the information model presented and populating it fully makes it possible to create the instructions presented in table 5. The strings generated from reading the database are a new way to combine the information presented in figures 44 and 45. Note that the quality control tasks: check brake lever and check white reflex did not generate any instructions as it was assessed that such operations require special content to be valuable, telling the operator what to check for, what to be extra aware of, and how to perform the control properly.

Table 5. Instruction strings generated for the pedal car assembly

Task	Instruction string
Mount front frame	Get Front Frame (36025641) from N/A, insert at Fully inserted
Adjust front frame	Get Measuring stick from toolbox, place at Tool wall
Mount left rear mudguard	Get Left rear mudguard (336026784) from K, place at Pedal car-rear left
Assemble screw left mudguard	Get Screw (985586) from L4 insert at Rear left mudguard. Get Nut (986589) from L4, thread at Rear left mudguard
Tighten left mudguard	Get Allen key from tool wall & Wrench from drawer, use at Rear left mudguard
Mount right rear mudguard	Get Right rear mudguard (36026785) from K, place at Pedal car-rear right.
Assemble screw right mudguard	Get Screw (985586) from L4 insert at Rear right mudguard. Get Nut (986589) from L4, thread at Rear right mudguard
Tighten right mudguard	Get Allen key from tool wall & Wrench from drawer, use at Rear right mudguard
Mount handbrake	Get Handbrake (36452586) from L3 and nut (964225) from L4, place at Behind wheel rear right
Check brake lever	N/A - Quality control
Check white reflex	N/A - Quality control

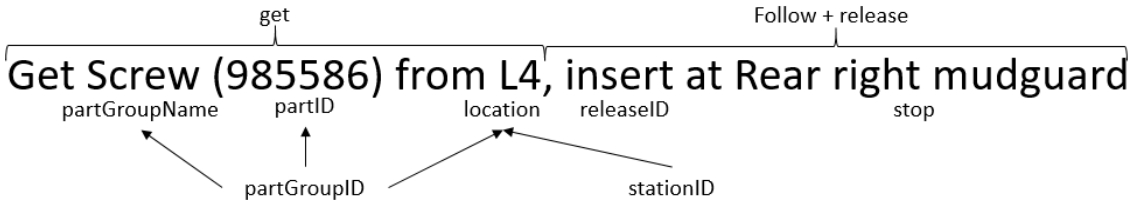


Figure 46. Breakdown of an instruction string and data pointers to the information model

To explain how an instruction string can be generated, the screw assemble on the right mudguard is chosen which is composed of three meaningful actions, see figure 46. The *get* action has a partGroupID stored in it, for which it can search the partGroup object to also find its name. By searching the variant object, the exact partID for this can be found. By combining the partGroupID and the stationID, the location object can be found, and thereby the location. In this case, L4 could be a compartment in a material rack. *Rear right mudguard* is the stop position of the motion, and the releaseID, *insert* tells that the part should be inserted at this position.

5.4 Implementation in ThingWorx

Using the data available as of today, the information model designed was implemented in ThingWorx and populated using data possible to export from IMMA and AviX. For that reason, the functionality will be different compared to the Java demonstrator presented in the previous section. This section first introduces the modifications made to the Operator Advisor information model, then presents the imports as well as instruction demonstrator. Specific DataShapes,

ThingShapes, ThingTemplates and Things designed to implement the information model are thoroughly presented in Appendix II.

5.4.1 ThingWorx Operator Advisor information model adaptations

The Operator Advisor by default uses WorkDefinition to describe a task. A WorkDefinition can be linked to other WorkDefinitions by using links and thus create hierarchies. Instruction strings can be stored in any WorkDefinition object on any level. To further specify a WorkDefinition, the Action object was formed, which has similar properties to a WorkDefinition. An option could have been to add another level of WorkDefinitions, but that would bring difficulties. When calling services to get the children of a parent, all children including the grandchildren will be included, which is undesirable. Creating the separate Action object also brings expansion opportunities as the Action can be modified differently from the WorkDefinition object. This way of extending the Operator Advisor information model also makes it clear what extra functionalities are added and what is Operator Advisor standard. The added DataShapes are marked green in figure 47.

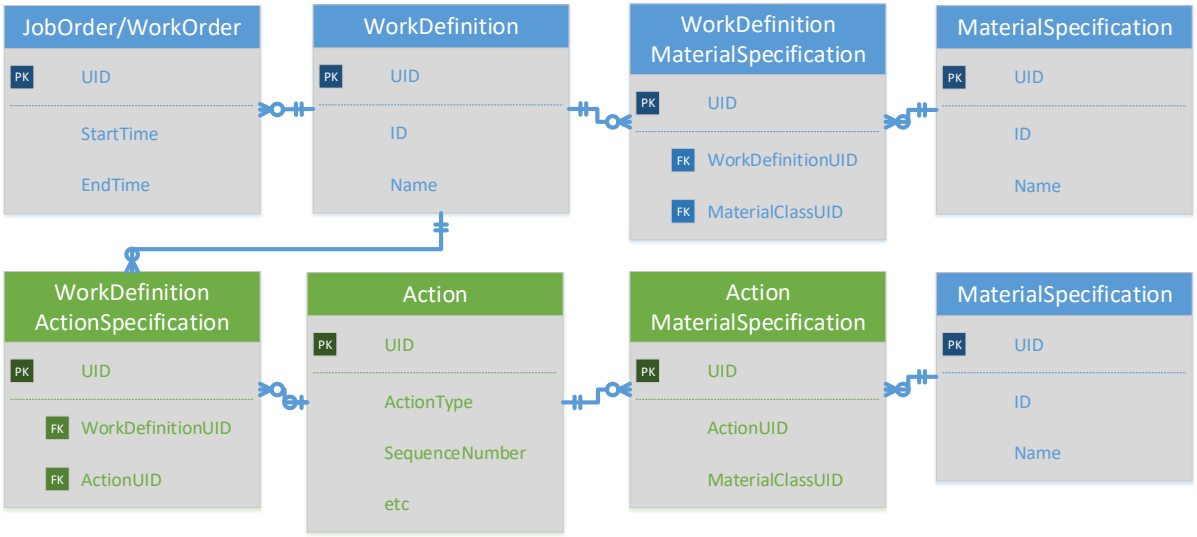


Figure 47. The information model of Operator Advisor with added functionality shown in the bottom row

Just as the WorkDefinition, the Action has ActionMaterialSpecifications and ActionProcessSpecifications to be able to link materials and parts to the object, only the material specification is seen in figure 48. The ordinary MaterialDefinition and ProcessingResource objects can be linked to the Action. It is possible to link MaterialDefinitions and ProcessingResources both to Actions and to the WorkDefinition they are found within at the same time. The quantity linked to the WorkDefinition acts as a BoM and clarifies the material usage. A part might be used in several Actions in the same task, but only added once to the actual assembly. This way of structuring the database ensures that the quantities used are correct.

5.4.2 Data importer

The Operator Advisor information model can be populated by integrating it with other software like PLM (Product Lifecycle Management) systems such as PTC Windchill. In this case, the project wants to showcase the technical interoperability between ThingWorx, IMMA and AviX.

For that purpose, services are designed that parse the data of these exports to the relevant ThingWorx DataShapes. A mashup seen in figure 48 has been designed which saves files to the repository and calls appropriate services to parse the data to the database.

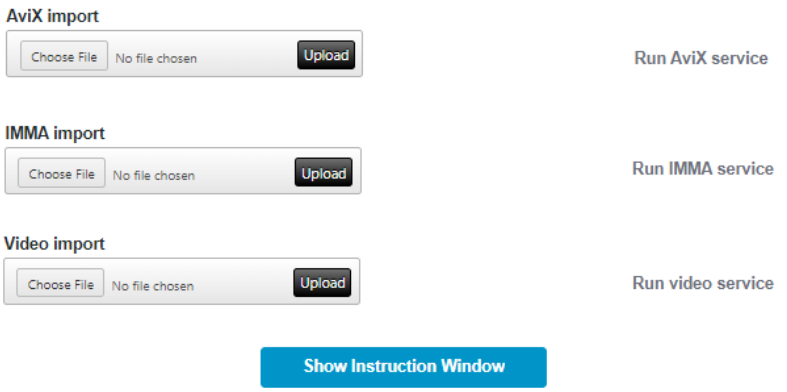


Figure 48. Data importer created for ThingWorx to make the import of data easier

The AviX import creates a WorkDefinition for each task, ProcessingResource for each tool and MaterialDefinition for each part. Note that the notion of sequence, being a task performed at a specific station is not implemented in ThingWorx as it is assumed that all tasks take place at the same station in this implementation. The concept of dividing parts into groups is also left out in this demonstrator as such functionality is not available in AviX and IMMA.

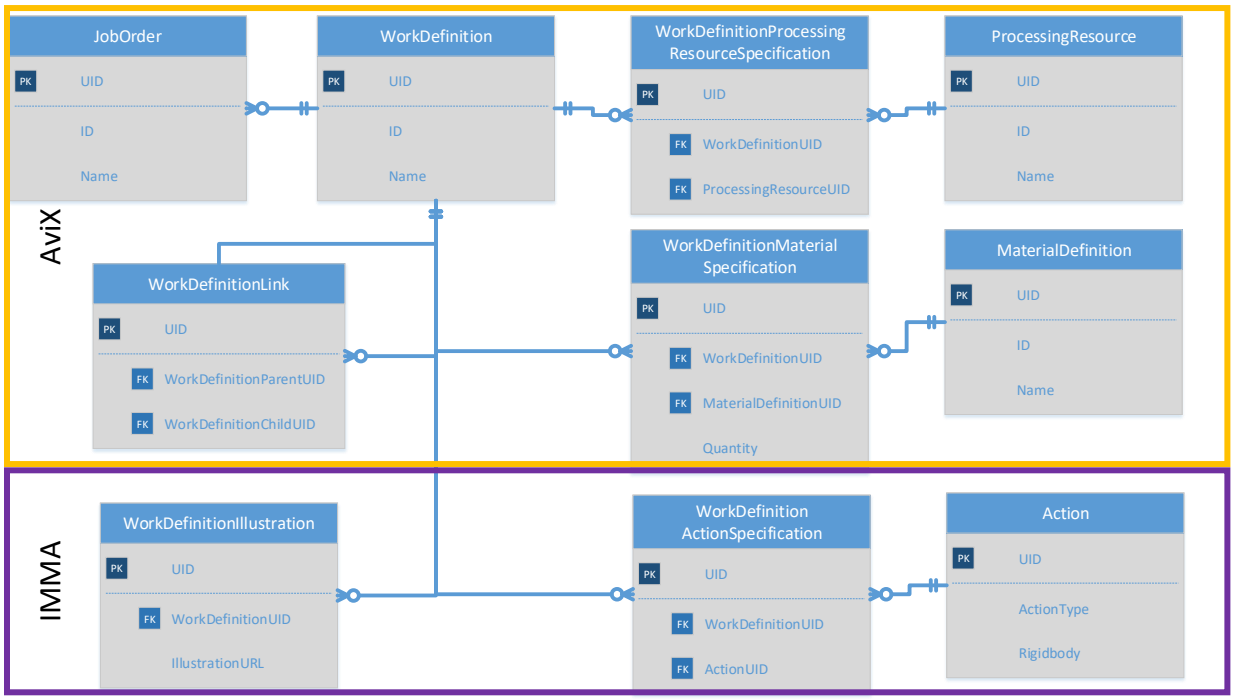


Figure 49. Extension of Operator Advisor information model clarifying what objects IMMA and AviX populates

In the AviX import service, WorkDefintionProcessingResourceSpecifications and WorkDefinitionMaterialSpecifications are created to tie the parts and tools to the task. The quantity of the part is stored in the specification object. Task, part and tool numbers in AviX are

mapped to the ID in ThingWorx, while names are mapped to names. All tasks to assemble the same product are then linked to the same JobOrder. The upper, yellow-marked part of figure 49 summarizes all DataShapes populated by the AviX import.

The IMMA import reads the name of all WorkDefintions and checks for any matching names, stored in the name attribute. In case there is a match, Actions are created which are tied to the WorkDefinition using WorkDefinitionActionSpecifications. The action type is mapped to the actionType attribute and the name of the rigid body it handles is stored as an attribute if applicable. Name is chosen to identify the part or tool in IMMA, but other identification could be used, such as ID number. The DataShapes populated by the IMMA import are marked purple in figure 50.

The video import first saves the animation to the ThingWorx repository. It then reads its name and searches for any matching WorkDefintion by name. If found, a WorkDefinitionIllustration object is created holding the URL of the animation.

5.4.3 Work instruction demonstrator

The work instruction demonstrator has a simple layout to show the user what data is available, see figure 50. The JobOrder can be chosen, so that belonging work definitions (tasks) can be displayed. When a task is chosen, lists of parts and tools appear, along with a list of all actions to the very right. If IMMA simulation animation is imported to the ThingWorx repository, it can be found in the instruction window too. Future work needs to better figure out how to present the information to the operator. Note that the filtering options are not available in the ThingWorx demonstrator as that is planned to be implemented at a later stage.

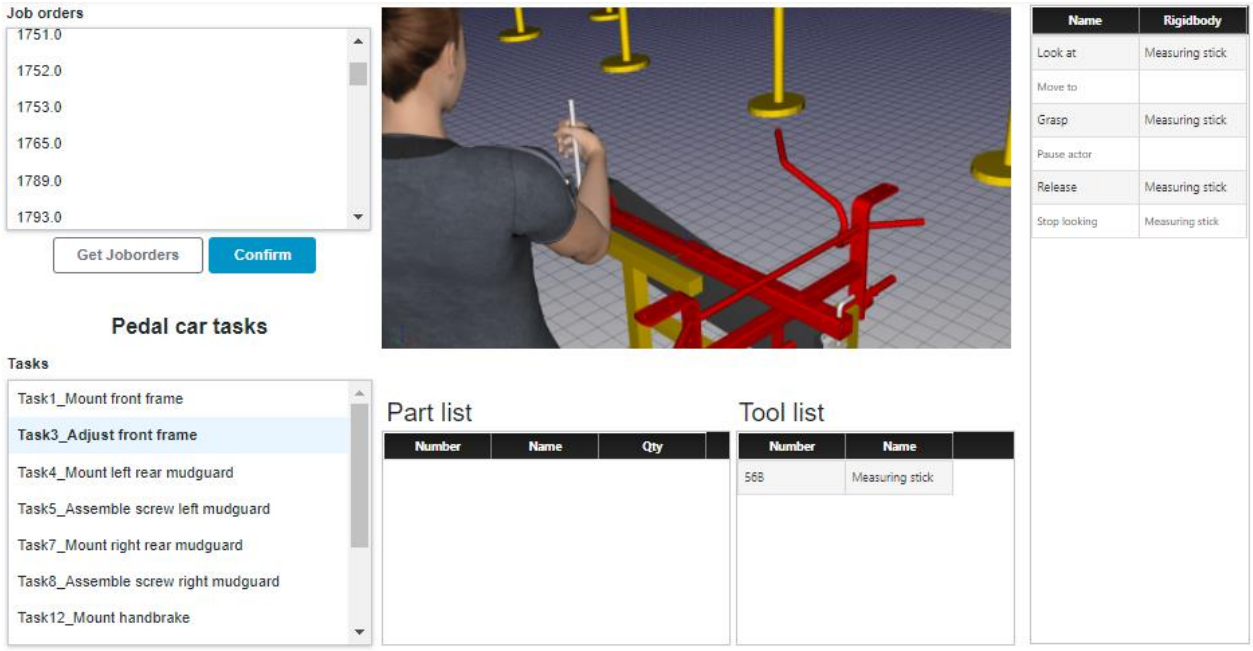


Figure 50. Demonstrator to visualize the functionality of the information model in the context of ThingWorx

5.4.4 Naming conventions

For the import services in ThingWorx to be able to assign actions to the correct WorkDefintion, the task name in AviX needs to match the name of the operation sequence in IMMA. All WorkDefinitons of the same name will get the same actions assigned, but with individual links. For the simulation video to be matched to the right WorkDefintion, the name of the video repository must be identical to the name of the WorkDefinition.

As of today, the name of the rigid body in IMMA, a tool or part is saved as a value in the Action object in ThingWorx. No link is established between the Action and any MaterialDefintion or ProcessingResource. The name of the rigid body could be matched to a name of a MaterialDefinition or ProcessingResource so that a link could be created from the action to the appropriate tool or part object. Such matching requires that the name of the rigid body exactly matches the name of a MaterialDefinition or ProcessingResource, which also need to have unique names.

6 Discussion

The discussion focuses on the information model designed and its contribution to the operator and system interoperability once implemented in a proof-of-concept. At the end of this chapter, the reader can find a method discussion where quality and generalizability dimensions are elaborated along with opportunities for future work and research.

6.1 Semantic interoperability

To achieve the semantic interoperability between cyber-world and physical-world systems as described by Panetto et al. (2019) the information presented to the worker is important. The demonstrators in Java and ThingWorx are examples of information systems that bring together data so that a context is created and thereby also valuable information to the worker. For the communication to work efficiently Panetto et al. (2019) emphasize that the information sent from the information system needs to be constructed in such a way that the receiver, in this case, a worker can make the correct interpretation. The information received should have the right content and be adapted to the individual needs of the operator as identified by Eriksson & Johansson (2017). In this section, the work instruction content using the designed information model is discussed.

6.1.1 Work instruction content

Table 6. Information needs in work instructions according to Hart (1996) and Eriksson & Johansson (2017).

Those identified in the demonstrator are checked, non-relevant are crossed out.

Work Instruction Information		
✓ Tool & parts (what)	~~Chassis #~~	✓ Sequence
~~Operators (who)~~	~~Truck product #~~	~~Real-time information~~
✓ Initial & end position (where)	~~Engine product#~~	Comments (special considerations)
✓ Order of operations (when)	~~Transmission product #~~	Quality reminders
Process hierarchy (why)	✓ Part #	Screw length
✓ How (e.g., screw, twist, etc.)	✓ Part name	General knowledge about a task
	✓ Quantity	Mobile information
	✓ Where to assemble	✓ Pictures
	✓ How to assemble	Feedback of work
	Change notes (product/balance/material)	Common problems and how to solve them

The work instructions which can be generated fulfills many needs described by Eriksson & Johansson (2017) and Hart (1996), see table 6. The crossed-out attributes have been considered not to be relevant for this proof-of-concept as they are related to the customer order and internal order systems within manufacturing, such as chassis, truck, engine and transmission product number. The data is still to be considered important, however not applicable in this laboratory

context. Some information content suggested such as quality reminders, special considerations and common problems are complex to convey. Certain information is brought to the surface after production starts and is feedback from operations. This kind of information tends to be more similar to the notion of tacit knowledge, rather than explicit knowledge, the latter one being the one addressed in this project. Quality control was one of the tasks in the pedal car assembly which conveyed that it is not feasible to auto-generate such instructions. Change notes, feedback and mobile information are equally important and to be considered when the technology matures and gets more widely implemented in a well-functioning HMI.

What makes the information model and its proof-of-concept differ from a BoM is the ability to describe not only *what* to assemble but also *when*, *where* and *how*, as suggested by Hart (1996) and again identified by Eriksson & Johansson (2017) in an HVLV assembly context. This possibility has been enabled by the division of tasks into actions and having tools and parts relate to the actions instead of task to get a more precise context. Using manikin simulation input can help to partly populate such information model as shown in the ThingWorx demonstrator.

It has been clarified that pictures and animations can be added easily if they can be generated in the first place. When deciding how to present the information Li et al. (2018) and Brolin et al. (2017) concluded that pictures in combination with text were better than providing plain text. Eriksson & Johansson (2017) emphasize that the graphical content must have the right quality to be valuable. The animation in the ThingWorx demonstrator does not necessarily fulfill this criterion but acts as an example of how such content easily can be embedded in the instructions if references can be created and used. As pictures and drawings are found in systems that might have interfaces to the platform used, it is important that these parts, tasks and other objects can find this data via references. The metadata ability of objects is therefore important not only in ThingWorx but also in other systems such as IMMA and AviX.

6.1.2 Work instruction quality

Haug's (2015) framework for instruction quality problems, see table 7, has been used for evaluation of the information model. The instructions created in the demonstrator show the capability of producing instructions which help overcome many intrinsic quality problems. The dynamic behavior shown in the Java demonstrator makes it possible to customize the information content to the receiver, identified important by Fässberg et al. (2011), Fast-Berglund et al. (2018) and Eriksson & Johansson (2017). In the proof-of-concept, information is read and stored in a database, and instructions are generated when needed by reading appropriate data. It allows more experienced workers to get less information and thus the sense that the instructions are repetitive and unneeded. At the same time, junior operators can be provided with more clear and precise instructions by having other filter settings. Enofe (2017), identified that the gap on what information is needed differs between engineering and assemblers, therefore letting the assemblers themselves decide could help bridge this gap.

Table 7. Instruction quality problem framework (Haug, 2015)

Instruction information quality problems				
Intrinsic problems	**Representational problems**	**Unmatched information**	**Questionable information**	**Inaccessible information**
Too repetitive	Difficult to Understand	Too complex content	Poor believability	Other accessibility barriers
Deficient	Verbose	Too large amount	Poor reputation	Security barriers
Ambiguous	Inconsistent	Untimely		
Unneeded				
Incorrect				

To overcome the problem of operators being questionable about instructions the instructions must be presented in such a way that the operators consider them trustworthy. Earlier research by Fasth & Stahre (2013), Fässberg et al. (2011) and Eriksson & Johansson (2017) have identified the importance of having correct and up-to-date information. The digitalization and auto-generation of the instructions make it possible to efficiently update the instructions by just updating relevant data and not entire text strings. The demonstrators have proven that digital instructions can be provided once the operator makes a request, thus the time dimension gap identified by Fast-Berglund et al. (2018) and Fässberg et al. (2011) can be bridged. Once graphical content is included in the instructions, it becomes more evident that the instructions are not updated, thus forcing manufacturing preparation to make updates. Many instruction information quality problems need to be tackled in the HMI as the representation is key in ensuring semantic interoperability between work instructions and the operator.

6.1.3 Cognitive support for Operator 4.0

Schuh et al. (2017) emphasize efficient communication with the operator in Industry 4.0, especially important due to the increasing complexity in production, as explained by Hu (2013). It is therefore important to be able to efficiently train workers to get them into the preferred operational assembly mode described by Mattson et al. (2020). With detailed instructions created in text combined with pictures and animations, it is possible to give the workers valuable insights into the *what, when, who, where, why, and how* of the task (Hart, 1996). This understanding makes it possible to support the intuition of the operator which is important to be able to work fast and efficiently according to Mattsson et al. (2020). The improvement of providing each operator the support it needs is described by Fast-Berglund et al. (2013) and Berlin & Adams (2017) as an effective way of improving both the productivity as well as the quality while decreasing the cost from defective products, of which the majority are due to human assembly errors according to Bäckstrand (2009).

6.1.4 RQ1: Characteristics of an information model for auto-generated work instructions

The most essential characteristic of an information model designed to orchestrate auto-generation of work instructions is the possibility to divide information on a hierarchy level below the task. Once populated, it has been proven in the proof-of-concept to be able to describe simpler assembly operations in the pedal car case. The dynamic behavior, only showing the information the operator asks for helps overcome instruction quality problems, mainly the information overflow. In the ThingWorx implementation, it has been proven that pictures and other graphical content can be added from the simulations to add information value. In work instructions, tacit knowledge and more complex descriptions are sometimes necessary, and for that manual work is still needed as auto-generated work instructions are better at describing *what* to assemble, *when, where* and *how,* but have a tough time describing how, as may be needed in quality control contexts and complex operations.

6.2 Technical interoperability

When implementing auto-generated work instructions, data from several sources need to be brought together. Interfaces of involved systems must be defined and work as planned, in other words, technical interoperability issues need to be solved. In this section, the interoperability of the system-of-systems designed in the second demonstrator is discussed, which consists of IMMA, AviX and ThingWorx.

6.2.1 Digital Human Modeling as a data source

The IMMA tool aims to simulate human movements and visualizing them to perform ergonomic evaluations and ensure manufacturability. As described by Högberg et al. (2016), the manikins in IMMA are instructed using a language similar to a PMTS-system. Such information is proven useful in the demonstrator, especially when those movements are put in relation to positions, parts and tools. That said, it is the input to IMMA which is useful for generating work instructions, and not the output. As the IMMA instruction language aims at describing how a manikin should move, it also seems reasonable that the same information is useful when provided to a human once slight adjustments on content are made.

Mårdberg et al. (2014) managed to map the IMMA instruction language to SAM and MOST systems and instruction strings. The results were fine-grained instructions on how to move the body, however, references to objects were not included. As identified by Eriksson & Johansson (2017), the information need of the operator is not how to move the body, rather what parts and tools to use, what to be especially aware of, etc. Therefore, the instructions provided to the manikin are too detailed to be used straight away for assembly work instructions. Information on movements can be left out, but relations to objects is interesting. The demonstrator has proven that such information can be retrieved, however, its content needs to be even more precise to be valuable to the operator. The possibility to add metadata to objects in IMMA or AviX could enable the possibility to gather related object data from other systems.

As Delin & Jansson (2015) found in their study, the assembly preparation process is not standardized in the organization. In most cases, manual work to make assembly instructions is needed by preparation engineers. The trend towards mass customization, described by Hu (2013) implies that the number of different tasks will increase. If having the possibility to use the data provided by IMMA and AviX to also make assembly instructions, time and resources could be spared in the assembly preparation process. As mentioned in the discussion on semantic interoperability, this would also make it simpler to update the instructions, thus increasing the value and usefulness.

In case no simulation is made, data will be missing on the action level so there must be other possibilities to input data to the database. There might be situations that are complex or even impossible to simulate. By designing a module in ThingWorx which can populate tasks with actions and relevant material and processing resource specifications, this obstacle can be overcome. Since no simulation is made, time data and ergoScore will for obvious reasons be missing. Another option could be to extend the activity objects in AviX, adding the possibility to link parts and tools to those so that the functionality is similar to that of IMMA. Tasks could then be populated with activities, parts, and tools and exported directly to ThingWorx.

6.2.2 Interoperability between IMMA and AviX

While IMMA is a tool that focuses on manikin simulations and ergonomic evaluations, AviX is a balancing tool for manual assembly (Ljung et al., 2020). As the purpose of the systems is different, information models are not compatible by default. The interoperability between the two is similar to the level one rating of Panetto (2007). In other words, it is possible to connect them as the data is well-structured, but manual work is needed so that the right links and interpretations are made. Recent efforts in the MOSIM project have enabled such functionality, using JSON format exchange of ergonomic data. Due to limited interoperability between the two systems, this thesis has focused on merging the data possible to extract externally, in this case in ThingWorx.

Since AviX and IMMA are systems designed for different purposes working on their own, they are considered two separate systems according to the definition of Ducq et al. (2012). When integrating these, alone or together with other systems, a system-of-systems forms. When having two different systems interoperate that are not initially designed for that purpose, two measures can be taken: homogenization or bridging, according to Naudet et al. (2009). Homogenization means that the systems are adapted so that their interfaces match, while bridging is an intermediate software that can match the two systems to be interconnected. Naudet et al. (2009) emphasize that adapting some systems might be difficult due to legacy. In this case, legacy is not the main issue however, as a modification of systems requires dedication in terms of resources like money and time, it has not been possible to push such changes. Bridging has instead been practiced where ThingWorx has been bridged to both IMMA and AviX successfully. As ThingWorx is different from ordinary systems, being an IoT platform and centerpiece within production, a core functionality is the ability to define and modify interfaces to external systems.

The bridging from IMMA has been slightly more flexible than for AviX as the Lua script has enabled the user to structure the export and define what data to include. Ducq et al. (2012) claim

that openness, which is the possibility to communicate with the system environment means better interoperability as it otherwise requires other software, in this case, ThingWorx to be interoperable. Letting the user set the structure of the export can be considered an efficient way to increase the openness and thus the interoperability of the system. As the IMMA module lacks some API functionality, those opportunities were still limited.

For the data to be merged appropriately in ThingWorx the work in the two different systems must be performed in a standardized way. This is in line with the claim of Naudet et al. (2009) who mentions that bridging tends to create more vulnerable connections than homogenization. In this case, certain naming conventions must be followed, such as identically writing the task name. This will become even more evident and fragile as the systems grow larger and handle more, complex data. This implementation also becomes fragile to future changes to any of the systems.

To further work on the interoperability, it is valuable to improve the export function from IMMA and AviX respectively. Today, the exports to ThingWorx are one-way streets which means that if the information is wrong, manual interpretation is needed or the import will fail. In the future, if ThingWorx can talk with AviX and IMMA, such problems can be overcome. By connecting both IMMA and AviX directly to ThingWorx and eventually let them communicate through the platform, the network complexity decreases since the number of interfaces to handle gets fewer.

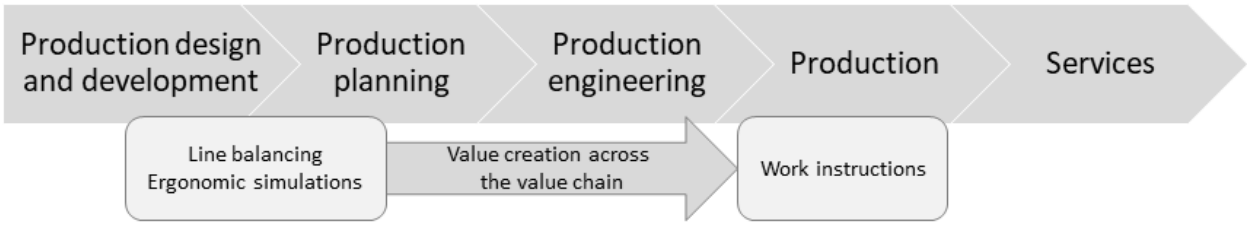


Figure 51. End to end integration across the value chain (Kagermann et al., 2013)

Increasing the interoperability between IMMA and AviX is well in line with the end-to-end integration idea of Kagermann et al. (2013), shown in figure 51. Connecting these systems makes it possible to enhance their contribution further up the value chain. Computer programs that previously were used for manufacturing preparation and product development now also have the capability to contribute with the necessary information for assembly work instruction generation.

6.2.3 Modular infrastructure

To increase the interoperability between systems a modular infrastructure as shown in figure 52 has been implemented, recommended by Åkerman et al. (2013). Basing the system on ThingWorx makes it possible to connect multiple subsystems such as AviX and IMMA. These can be handled as modules that can be exchanged if needed. Åkerman et al. (2013) claim that modularity allows to more easily connect new systems and add functionality. This opens the possibility to make more data available from a wide selection of systems. A PLM system such as Windchill could be added to provide more rich information on parts, or a logistics system providing info on parts location, which in turn can have other supporting systems. All data can be used in the IoT platform by combination so that new sets of more comprehensive data are created. As modules can be exchanged and new ones can be added, the system stays flexible and can handle new situations

efficiently. Fast-Berglund et al. (2014) claim that such modularity is important to increase competitiveness. This thesis has focused on the interfaces between ThingWorx, IMMA and AviX. The ThingWorx work instruction demonstrator works as a proof-of-concept that this architecture is reasonable. The final instruction application will look different, especially the HMI.

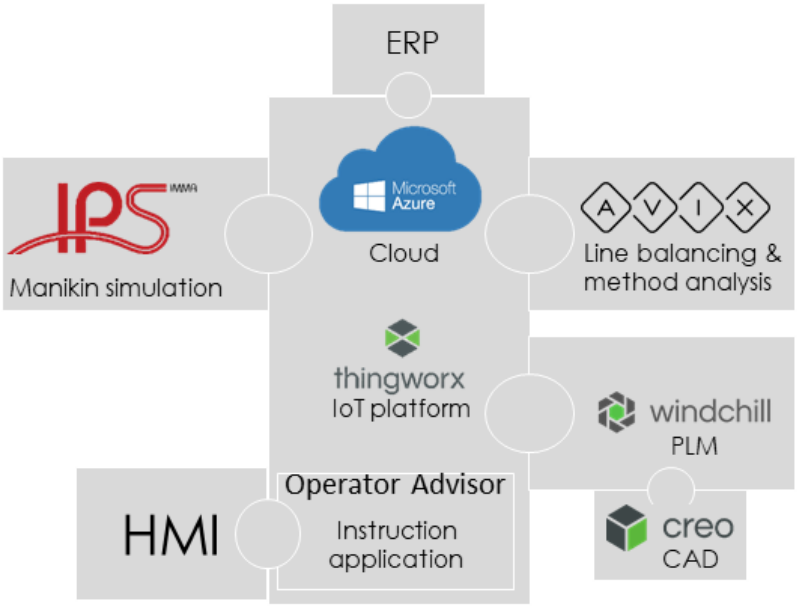


Figure 52. The modular design of the IT-infrastructure

As described, an ongoing project is developing the integration between AviX and IMMA to be able to better synchronize ergonomic evaluation data. The integration can be motivated by the fact that extensive data is needed from the simulations to appropriately map the ergonomic evaluation module in AviX. For the instruction application however, which needs several design iterations to reach a satisfactory result, integrating the software to ThingWorx separately instead of the AviX and IMMA integration ensures flexibility as lock-in effects to the design of their integration is unwanted.

6.2.4 RQ2: Using data from manufacturing preparation to create work instructions

To populate the information model with essential data, a modular system has been designed as suggested by Åkerman et al. (2013), having ThingWorx as a centerpiece to bring the data together. It has been possible to bridge interoperability issues between sub-systems without the need for homogenization by utilizing the flexible interface of ThingWorx. As IMMA and AviX have limited possibility to extract data, it has not been possible to completely populate the information model for the ThingWorx demonstrator. AviX has been providing the task, part and tool structure while IMMA input has enabled specification of movements within the tasks. To map data successfully standardized ways of working in AviX and IMMA, following certain naming conventions are necessary. Digital Human Modeling together with line balancing has the potential to become valuable information sources in auto-generated work instructions by enabling more rich output.

6.3 Scaling up

Scaling up the concept of auto-generating work instructions requires both that more complex operations can be simulated, as well as more operations. To handle the increased complexity, the computing and data storage architecture need to adapt. Assuming that cloud computing, as described by Zhang et al. (2010) will be the base in future architecture, the upscaling would be relatively simple from a technical point of view, as resources are located on remote servers and accessed through a web interface. Therefore, the focus in the discussion will be on the simulation and information models, and what value they can bring to the organization in terms of better assembly instructions and time data management.

6.3.1 Increasing the Technology Readiness Level

The Technology Readiness Level (TRL) shown in table 8 is created for evaluating technological systems in aerospace applications, but nowadays also used to evaluate other systems (NASA, 2012).

Table 8. Technology Readiness Level framework (NASA, 2012)

TRL	Description
9	Actual system "flight-proven" through successful mission operations
8	Actual system completed and "flight qualified" through test and demonstration
7	System prototype demonstration in a space environment
6	System/subsystem model or prototype demonstration in a relevant environment
5	Component and/or breadboard validation in a relevant environment
4	Component and/or breadboard validation in a laboratory environment
3	Analytical and experimental critical function and/or characteristic proof-of-concept
2	Technology concept and/or application formulated
1	Basic principles observed and reported

The system created has been successfully tested and has managed to deliver its key functionality in the pedal car case. The tasks within the pedal car case are simple compared to many tasks taking place on the shop floor within the case company as of today. The pedal car case is a laboratory environment, which puts the designed system in TRL 4 according to the definitions of NASA (2012). As the information model and the functionality of the proof-of-concept is not yet validated, one could argue that the system is only in TRL 3. Being able to show its functionality on a real production case will be a crucial step forward as it pushes the system closer to TRL 5.

More attributes may need to be added to objects in the information model and the information model itself be expanded with additional objects to continue increasing the TRL. The implementation made in ThingWorx shows that the software allows customization. As the information model in ThingWorx is based on the ISA-95 standard, which is widely implemented, the interoperability with other systems is high as well as the industry knowledge and understanding about it. Most probably, a slow scale-up, gradually adding more complexity is needed before auto-generated work instructions can be implemented in an operational environment.

6.3.2 Simulation of more operations

As a part of reaching higher technology maturity levels, the technology must be able to face situations closer to a real-world scenario. When scaling up, the number of tasks will increase as well as the part and tool list length. A better naming convention than the proposed one is therefore needed, just simply naming parts for what they are, e.g., screw and cover will not be sufficient. As of today, IMMA does not allow parts, paths or similar, to store metadata. To add such functionality would be valuable when scaling up, as it would make it possible to create references to external databases such as PDM systems to gather rich data. To reach higher technology readiness levels, the solution must be robust in a sense that it is not sensitive to minor syntax errors in the naming of tasks, parts and tools.

An alternative solution requiring less modification in IMMA could be to name parts by number instead of description to handle such couplings as of today, however, this way of working holds several issues. Firstly, the usability of the IMMA tool from the user perspective will decrease as a list of parts named by number says nothing about them. User-friendliness in the preparation process is one of the main reasons IMMA was created, according to Högberg et al. (2016). Secondly, parts undergo updates all the time and get replaced. In these cases, the part number may change but the part still has the same properties. Therefore a reference to a PDM system that is neutral to part version is needed.

As mentioned by Högberg et al. (2016), a key reason for creating the IMMA tool is to spare resources in the preparation processes. Therefore, all efforts to extend the use case and functionality of IMMA must hold the same goal. To optimize workflow in simulating assembly processes, there is a need to be able to use the same simulation if the tasks performed are the same, even though the involved parts are different. For example, data from the same simulation should be possible to use independently on what color the mudguard has. In the PDM system, mudguards of different colors hold different part numbers. This is the reason the partGroup concept was introduced in the presented information model, as it enables parts to be grouped, not in the sense of subassemblies but as siblings. By storing a reference to the group when performing the simulation, it can be used if one of the parts in the group is used.

6.3.3 In the context of Industry 4.0

Schuh et al. (2017) claim that an organization which has matured in their Industry 4.0 journey can adapt to current real-world situations. To be able to do that, the entire production system needs to become more flexible, and tying tasks to the station may then not be appropriate. Work instructions will continue to be an important part in assembly operations as humans will continue to be an integral part to perform certain tasks. According to Sony & Naik (2020), the aim of Industry 4.0 is not to replace humans but to assist them in becoming more productive. The interest in auto-generating work instructions becomes relevant to address this information need.

As products become more complex, and the variants many, in line with the mass customization concept (Hu, 2013), humans will probably perform other tasks in the future. More of the simpler routine tasks become automated, a trend that probably will continue with the introduction of collaborative robots. To summarize, humans will have to perform a wider range of tasks, which

also gets more complex. Once again, calling for well-designed work instructions that tackle the instruction quality dimensions defined by Haug (2015). Auto-generated work instructions are a way to be able to deliver appropriate support using a reasonable amount of resources, as manual preparation processes will become too tedious to generate such support. The instructions shown in the demonstrator holds for simple tasks, and refinement of the information model and its application is needed to handle more complex situations.

To populate the information model, it has been shown that input to manikin simulation software is valuable as it can give a satisfactory level of detail to the instructions. According to Hehenberger & Bradley (2016), access to a digital twin is beneficial when performing simulations as they then can be performed in an up-to-date environment. For automated work instructions to be possible at a greater scale and achieving a higher technology readiness level, the organization must first reach higher in terms of Industry 4.0 maturity so that tools, data and technology associated with the digital twin are available. Setting up the assembly and workplace for every simulation will not be resource-efficient once scaling up.

Kuhlang et al. (2014) as well as Almström & Winroth (2010) have identified that many companies do not have well-functioning time data management systems. Times are not set properly, and they are not updated when needed. If implementing work instructions using data from these simulations, there is a need to have up-to-date simulations, otherwise, the instructions will not make sense. That will in turn make sure that the assembly time data is updated when needed. The IMMA software, simulating body movements with the possibility to adjust walking speed (Högberg et al., 2019) makes sure that the time simulated is reasonable in comparison to a real-world scenario. Mårdberg et al. (2014) proved that it is possible to extend the IMMA tool to generate PMTS-based work instructions, specifying the time in TMUs. Currently, IMMA only has the possibility to perform one action at a time so the time is misleading. Precise time is needed if the data should be used for operations planning, according to Kuhlang et al. (2014). If the functionality for multiple actions unlocks, the time data output has the potential to become trustworthy. Proper time data management is necessary to be able to have a flexible and reconfigure manufacturing system, an important part of a mature Industry 4.0 organization that easily can adapt to changes.

6.4 Research quality

According to Bryman & Bell (2015), reliability, replication and validity are three important quality dimensions to take into consideration. Reliability is ensured by the iterative approach taken, as suggested by Hevner & Chatterjee (2010). Continuous updates with internal and external stakeholders have ensured both project relevance and valuable input to the design process of the information model and its proof-of-concept. Recommendations and guidelines from involved software companies have ensured that the solution benefits from available functionality and stays in line with software development plans.

Replicability has been achieved by thoroughly describing the models and programs generated in the design process. As discussed in the methodology chapter, the possibility to replicate the study when performing IT-system development is limited. Validity can be gained by testing the information model on other cases, such as a cross-member case taken from the Tuve manufacturing plant currently under development in AviX and IMMA.

This research project has followed the six-step framework of Peffers et al. (2007). As it is a part of a larger research project, the solution is just a small step in a long journey, indicated by the technology readiness level of 3-4. Earlier theses at the Volvo GTO have been able to identify the problem and objectives, therefore focus has been the design and development of the artifact. The HMI developed is just to show what data is accessible, and future work must design a better HMI. Therefore, performing an evaluation and user testing of the product as suggested by Peffers et al. (2007), has not been performed. Most probably, more iterations in the design cycle will be needed before considering user testing.

According to Deng & Ji (2018), information system development can be considered contributing to research when the solution is designed and structured in such a way that its generalizability is high. The generalizability has been affected as this is a case study performed at a certain company. Often, legacy in terms of IT-systems used limits the possibilities to create something new, according to Naudet et al. (2009). As this is a greenfield project at a lower technology readiness level, the project has not performed backward integration to existing IT-systems, thus limiting such difficulties. However, IMMA has been used for manikin simulation and AviX for line balancing, and that by nature affects the solution, especially as bridging between systems has been necessary to solve interoperability issues. As the pedal car case has been developed jointly with other industrial partners, it ensures the generalizability of the information model. Many tasks required to assemble the pedal car are typical for any assembly task in the industry.

6.5 Future work and research

This thesis aimed at finding an information model for auto-generation of work instructions and implementing it as a proof-of-concept. This project constitutes a small part of the journey towards higher maturity and technology readiness level of assembly work instructions. For that reason, several future work opportunities are suggested.

Validation of Information model

The information model presented acts as a first attempt to structure data in a way that enables auto-generation of assembly work instructions. It needs to be further validated by testing it in other scenarios other than those used within this project as its development was based on the pedal-car case, only being a laboratory environment. Most probably, attributes will have to be added to enable full functionality for more complex cases such as those found at Volvo GTO. For complete validation, the information model must be tested on other software than that used at the case company to also show that ThingWorx enables the flexibility proven in this case, and thus is a capable center point in the proposed modular architecture. Proving this would also allow the idea of auto-generated work instructions to mature and increase the technology readiness level.

Human-Machine Interface (HMI) for auto-generated work instructions

The thesis has focused on the structuring of information and enabling system interfaces to show that auto-generation of work instructions is possible in simple operations. According to the recommendation of Fast-Berglund et al. (2014) information system and content design has been separated from information carrier development. Fasth & Stahre (2013) emphasize that digital work instructions enable multiple information carriers, such as computer screens, tablets or smartwatches. Thorvald et al. (2014) have proven that by moving information closer to the operator, positive impacts can be seen on the production system. These questions need to be further handled and discussed in the HMI design. Understanding how to appropriately filter the data will be another prominent issue to increase the usefulness of the instructions (Fässberg et al., 2011; Fast-Berglund et al., 2018). Some of the information content that Eriksson & Johansson (2017) identified such as change notes, quality reminders and feedback have not been addressed by this proof-of-concept as they are functionality integrated into the HMI and requiring testing in more complex operations not available at this point.

Increasing interoperability in manufacturing preparation

In the model presented, some data is retrieved from manikin simulation software and some from line balancing software. Further work is required on how to increase interoperability with these systems to enable the extraction of more relevant data. Equally interesting is the backward integration to CAD-databases and other data sources. Recent research has among other things tried to perform manikin simulation in a point cloud environment to get the full picture of the station environment. Lindskog et al. (2012) claim that simulating in such an environment is more resource-efficient, as well as being easier to interpret for external parties. A more efficient setup of manikin simulations has been identified as important to be able to use it as a valuable information source for assembly work instructions in the future.

7 Conclusion

The operator will continue to play a key role in manual assembly and to improve the cognitive workload of operators and product quality, well-designed instructions are key. This thesis aimed at designing an information model suitable to handle the data needed for auto-generation of work instructions and then populate it using data from manufacturing preparation systems. The purpose of auto-generated instructions is to enable the generation of work instructions requiring little manual work. Being resource-efficient in the preparation phase is key as the industry currently faces increased product variety, a call for a short time to market and global cost competition.

An information model has been designed by compiling the information need of the operator. Its most important characteristic is the possibility to divide a task into smaller and simpler elements reflecting basic movements and further having the use of tools and parts relate to these. The instructions which can be generated give the possibility to describe not only *what* to assemble, but also *when, where* and *how* for simpler assembly tasks. Note that the model is not suitable when presenting more complex tasks like quality control and that it does not comprise the full information need that has been identified.

As of today, the information model can be partly populated using the data available in manufacturing preparation systems. Using AviX line balancing software has proven efficient to provide task, tool and part data. IPS IMMA manikin simulation software, especially the instruction language is useful also when creating assembly work instructions as it delivers more detailed data on the actions, the elements of the tasks. The two above mentioned systems are designed to assist in manufacturing preparation processes. By unlocking new usage areas of the accessible data, the time spent in these preparation tools can be better justified.

The interoperability between the involved systems has been achieved using an IoT platform as a centerpiece in a modular architecture. The flexible interface of ThingWorx has helped bridging interfaces of involved systems without the need for homogenization of software. It has enabled the orchestration of information sharing between the systems to populate the information model. The data exchange has been a one-way street, where data have been mapped to the information model once imported to the IoT platform, requiring that syntax and naming conventions are used properly. The possibility to add metadata and expand import and export functionality of other involved software would increase interoperability and enable upscaling.

Using data from manufacturing preparation processes has great potential in supporting the information need of Operator 4.0. Using an information model to structure the data from different systems ensures that instructions can be generated once needed and therefore adapted to the individual needs of the operator. Such characteristics maximize cognitive support and therefore improves the work environment as well as product quality. This is more important than ever as personalized production is key to be able to compete in the fast-paced global market.

References

TACO (insTruction innovAtion for Cognitive Optimisation) | Vinnova. (n.d.). Retrieved September 3, 2020, from https://www.vinnova.se/en/p/taco-instruction-innovation-for-cognitive-optimisation/

EUREKA ITEA3 MOSIM | Vinnova. (n.d.). Retrieved September 3, 2020, from https://www.vinnova.se/p/e-kluster-itea3-mosim/

Almström, P., & Winroth, M. (2010). Why is there a mismatch between operation times in the planning systems and the times in reality? *International Conference on Advances in Production Management Systems*.

Asklund, E., & Eriksson, R. (2018). *Digital Dynamic Work Instructions in a Variant Driven Industry An Investigation on the Effects of Dynamic Instructions on Operator User Satisfaction*. Chalmers University of Technology.

Berlin, C., & Adams, C. (2017). *Production Ergonomics: Designing Work Systems to Support Optimal Human Performance*. Ubiquity Press.

Brolin, A., Thorvald, P., & Case, K. (2017). Experimental study of cognitive aspects affecting human performance in manual assembly. *Production and Manufacturing Research*, *5*(1), 141–163. https://doi.org/10.1080/21693277.2017.1374893

Bryman, A., & Bell, E. (2015). *Business Research Methods*. Oxford University Press.

Bäckstrand, G. (2009). *Information flow and product quality in human based assembly*. Loughborough University.

Degerman, J., & Lindgren, S. (2010). *Information systems for long cycle time assembly : development of a visual support system for the case of the forklift manufacturer Atlet*. Chalmers University of Technology.

Delin, F., & Jansson, S. (2015). *Process for preparing work instructions-A multiple case study at Volvo Group Trucks Operations*. Linköping University.

Deng, Q., & Ji, S. (2018). A Review of Design Science Research in Information Systems: Concept, Process, Outcome, and Evaluation. *Pacific Asia Journal of the Association for Information Systems*, *10*(1), 1–36. https://doi.org/10.17705/1pais.10101

Ducq, Y., Chen, D., & Doumeingts, G. (2012). A contribution of system theory to sustainable enterprise interoperability science base. *Computers in Industry*, *63*(8), 844–857. https://doi.org/10.1016/j.compind.2012.08.005

Enofe, M. (2017). *Data Management in an Operational Context A study at Volvo Group Trucks Operations*. Lund University.

Eriksson, G., & Johansson, P. (2017). *Assessment of Information Needs in Manual Assembly*. Chalmers University of Technology.

Falck, A. C., & Rosenqvist, M. (2014). A model for calculation of the costs of poor assembly ergonomics. *International Journal of Industrial Ergonomics*, *44*(1), 140–147. https://doi.org/10.1016/j.ergon.2013.11.013

Fast-Berglund, Å., Fässberg, T., Hellman, F., Davidsson, A., & Stahre, J. (2013). Relations between complexity, quality and cognitive automation in mixed-model assembly. *Journal of Manufacturing Systems*, *32*(3), 449–455. https://doi.org/10.1016/j.jmsy.2013.04.011

Fast-Berglund, Å., Åkerman, M., Karlsson, M., Hernández, V. G., & Stahre, J. (2014). Cognitive automation strategie: Improving use-efficiency of carrier and content of information. *47th CIRP Conference on Manufacturing Systems*, *17*, 67–70. https://doi.org/10.1016/j.procir.2014.02.042

Fast-Berglund, Å., Li, D., & Åkerman, M. (2018). Creating Strategies to Improve the Use of IT- and IS-Systems in Final Assembly. *16th International Conference on Manufacturing Research*. https://doi.org/10.3233/978-1-61499-902-7-177

Fasth-Berglund, Å., & Stahre, J. (2013). Cognitive automation strategy for reconfigurable and sustainable assembly systems. Assembly Automation, 33(3), 294–303. https://doi.org/10.1108/AA-12-2013-036

Ford, H. (1926). Today and tomorrow. In *Doubleday, Page and Company*.

Fässberg, T., Fasth, Å., & Mattsson, S. (2011). Cognitive automation in assembly systems for mass customization. *4th Swedish Production Symposium*.

Hanson, L., Högberg, D., Carlson, J. S., Delfs, N., Brolin, E., Mårdberg, P., Spensieri, D., Björkenstam, S., Nyström, J., & Ore, F. (2019). Industrial path solutions - intelligently moving manikins. In *DHM and Posturography* (pp. 115–124). Academic Press. https://doi.org/10.1016/B978-0-12-816713-7.00011-8

Hanson, L., Högberg, D., Carlson, J. S., Delfs, N., Gustafsson, S., Keyvani, A., & Rhen, I.-M. (2014). IMMA - Intelligently Moving Manikins in Automotive Applications. *ISHS 2014, Third International Summit on Human Simulation*.

Hart, G. (1996). The five W's: an old tool for the new task of audience analysis. *Technical Communication*, *43*(2), 139–145.

Haug, A. (2015). Work instruction quality in industrial management. *International Journal of Industrial Ergonomics*, *50*, 170–177. https://doi.org/10.1016/j.ergon.2015.09.015

Hehenberger, P., & Bradley, D. (2016). *Mechatronic Futures*. Springer International Publishing Switzerland. https://doi.org/10.1007/978-3-319-32156-1

Hevner, A., & Chatterjee, S. (2010). Introduction to Design Science Research. In *Design Research in Information Systems*. Springer. https://doi.org/10.1007/978-1-4419-5653-8

Hu, S. J. (2013). Evolving paradigms of manufacturing: From mass production to mass customization and personalization. *Forty Sixth CIRP Conference on Manufacturing Systems 2013*, *7*, 3–8. https://doi.org/10.1016/j.procir.2013.05.002

Högberg, D., Hanson, L., Carlson, J. S., & Bohlin, R. (2016). Creating and shaping the DHM tool IMMA for ergonomic product and production design. In *International Journal of the Digital Human* (Vol. 1, Issue 2).

Kagermann, H., Wahlster, W., & Helbig, J. (2013). *Recommendations for implementing the strategic initiative INDUSTRIE 4.0*.

Kruchten, P. (2012). The frog and the octopus: a conceptual model of software development. In *Journal of Software and Systems*.

Kuhlang, P., Sihn, W., Erohin, O., Krebs, M., & Deuse, J. (2014). Morphology of time data management - Systematic design of time data management processes as fundamental challenge in industrial engineering. *International Journal of Industrial and Systems Engineering*, *16*(4), 415–432. https://doi.org/10.1504/IJISE.2014.060652

Li, D., Mattsson, S., Salunkhe, O., Fast-Berglund, Å., Skoogh, A., Broberg, J., Santana, A., Afonso, P., Zanin, A., & Wernke, R. (2018). Effects of Information Content in Work Instructions for Operator Performance. *8th Swedish Production Symposium*, *25*, 628–635. https://doi.org/10.1016/j.promfg.2018.06.092

Lindskog, E., Berglund, J., Vallhagen, J., Berlin, R., & Johansson, B. (2012). Combining point cloud technologies with discrete event simulation. *Proceedings of the 2012 Winter Simulation Conference*.

Ljung, O., Pascual, A. I., Högberg, D., Delfs, N., Forsberg, T., Johansson, P., Dahlvik, J., Sánchez, J. L. J., & Hanson, L. (2020). Integration of simulation and manufacturing engineering software-allowing work place optimization based on time and ergonomic parameters. *Proceedings of the 6th International Digital Human Modeling Symposium*, *11*, 342–347. https://doi.org/10.3233/ATDE200041

Mattsson, S., Fast-Berglund, Å., Li, D., & Thorvald, P. (2020). Forming a cognitive automation strategy for Operator 4.0 in complex assembly. *Computers and Industrial Engineering*, *139*, 105360. https://doi.org/10.1016/j.cie.2018.08.011

Mårdberg, P., Carlson, J. S., Bohlin, R., Delfs, N., Gustafsson, S., Högberg, D., & Hanson, L. (2014). Using a formal high-level language and an automated manikin to automatically

generate assembly instructions. *International Journal of Human Factors Modelling and Simulation*, *4*(4), 11–13.

NASA. (n.d.). *Technology Readiness Level | NASA*. Retrieved December 4, 2020, from https://www.nasa.gov/directorates/heo/scan/engineering/technology/txt_accordion1.html

Naudet, Y., Latour, T., Guedria, W., & Chen, D. (2010). Towards a systemic formalisation of interoperability. *Computers in Industry*, *61*(2), 176–185. https://doi.org/10.1016/j.compind.2009.10.014

Panetto, H. (2007). Towards a classification framework for interoperability of enterprise applications. *International Journal of Computer Integrated Manufacturing*, *20*(8), 727–740. https://doi.org/10.1080/09511920600996419

Panetto, H., Iung, B., Ivanov, D., Weichhart, G., & Wang, X. (2019). Challenges for the cyber-physical manufacturing enterprises of the future. *Annual Reviews in Control*, *47*, 200–213. https://doi.org/10.1016/j.arcontrol.2019.02.002

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Rezaei, R., Chiew, T. K., Lee, S. P., & Shams Aliee, Z. (2014). Interoperability evaluation models: A systematic review. *Computers in Industry*, *65*(1), 1–23. https://doi.org/10.1016/j.compind.2013.09.001

Schuh, G., Anderl, R., Gausemeier, J., Ten Hompel, M., & Wahlster, W. (2017). *acatech STUDY Industrie 4.0 Maturity Index Managing the Digital Transformation of Companies*.

Sony, M., & Naik, S. (2020). Industry 4.0 integration with socio-technical systems theory: A systematic review and proposed theoretical model. *Technology in Society*, *61*. https://doi.org/10.1016/j.techsoc.2020.101248

Thorvald, P., Högberg, D., & Case, K. (2014). The effect of information mobility on production quality. *International Journal of Computer Integrated Manufacturing*, *27*(2), 120–128. https://doi.org/10.1080/0951192X.2013.800236

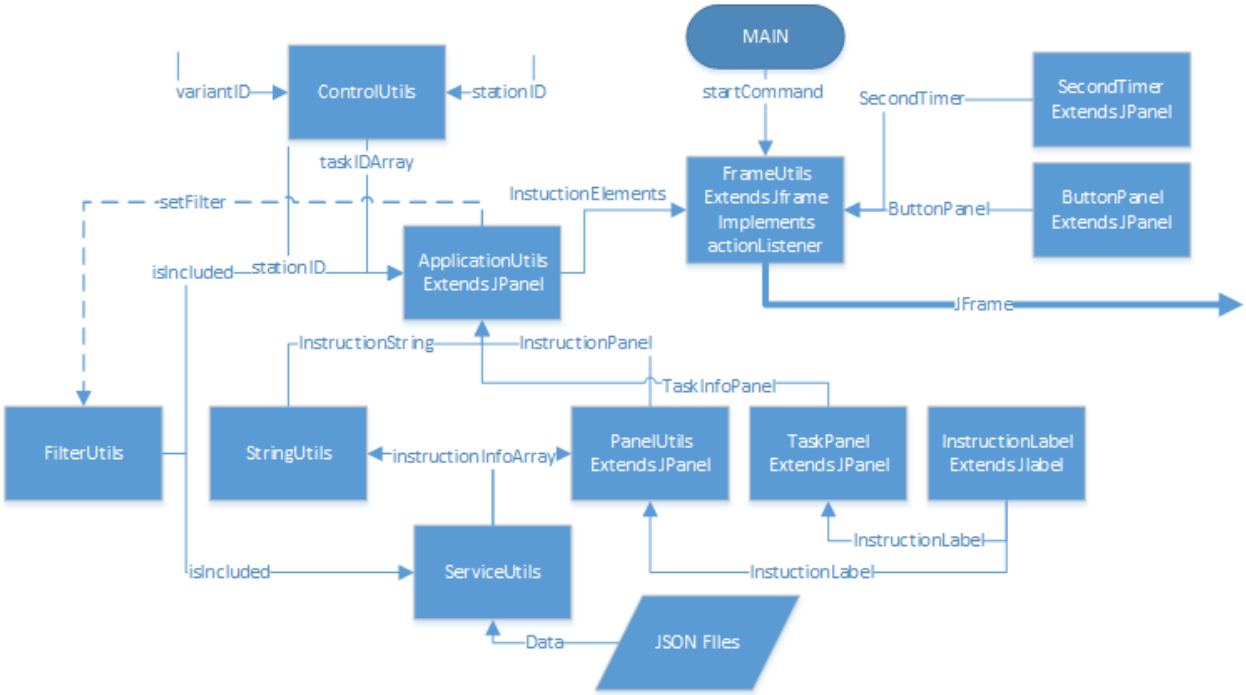
Zandin, K. B. (2001). *Maynard's Industrial Engineering Handbook, Fifth Edition*. McGraw-Hill Education.

Zhang, Q., Cheng, L., & Boutaba, R. (2010). Cloud computing: State-of-the-art and research challenges. *Journal of Internet Services and Applications*, *1*(1), 7–18. https://doi.org/10.1007/s13174-010-0007-6

Åkerman, M., Fast-Berglund, Å., Halvordsson, E., & Stahre, J. (2018). Modularized assembly system: A digital innovation hub for the Swedish Smart Industry. *Manufacturing Letters, 15*, 143–146. https://doi.org/10.1016/j.mfglet.2018.01.004

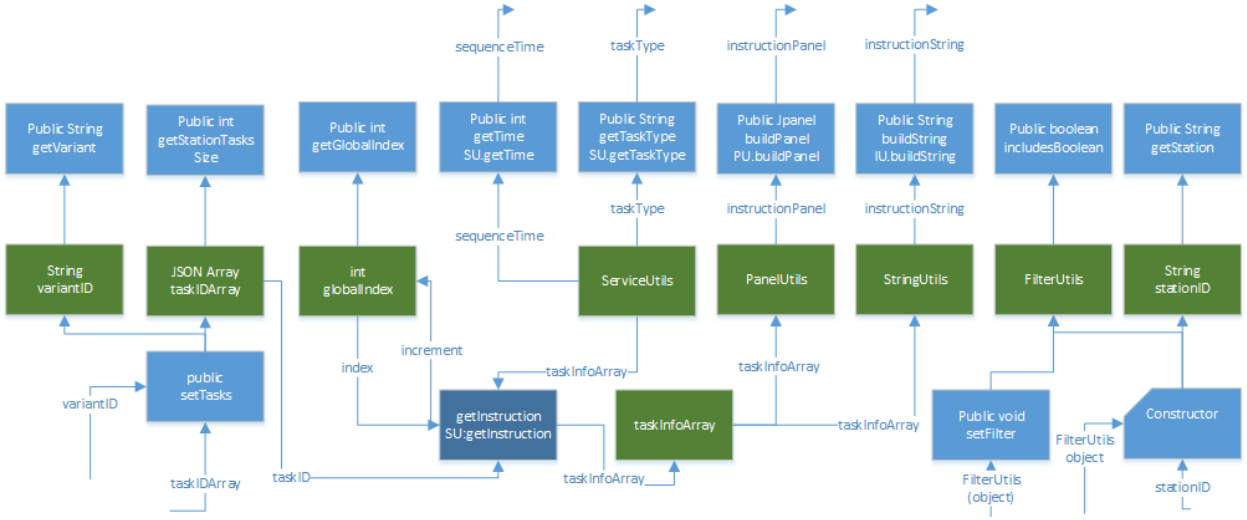
Appendices

Appendix I: Java demonstrator



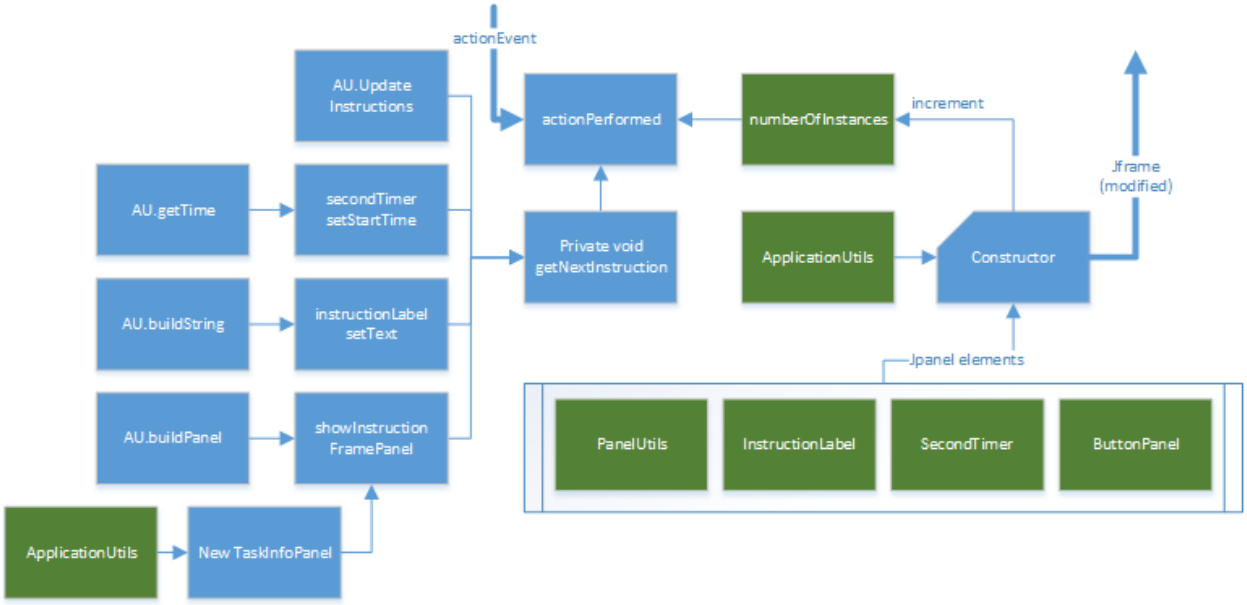
The Java demonstrator has been developed in an iterative process with input from the project provider and other stakeholders. During the development, the Java project was divided into classes which in turn were divided into methods, making each instance easier to understand and enables further development without increasing the complexity too rapidly. The ApplicationUtils class works as a hub to connect all the other classes and finally send the data to FrameUtils which show the instruction in Java swing components combined to a simple GUI.

Class ApplicationUtils



The ApplicationUtils class serves the purpose of acting as a coordinator. It holds instances of the classes used for information retrieval (ServiceUtils, PanelUtils, StringUtils and FilterUtils). There exists one instance of this class per station. It also holds the variantID, the stationID and the globalIndex. The latter one keeps track of the index of the taskIDArray currently at to be able to set the correct taskInfoArray when the operator asks for the next instruction in the assembly sequence.

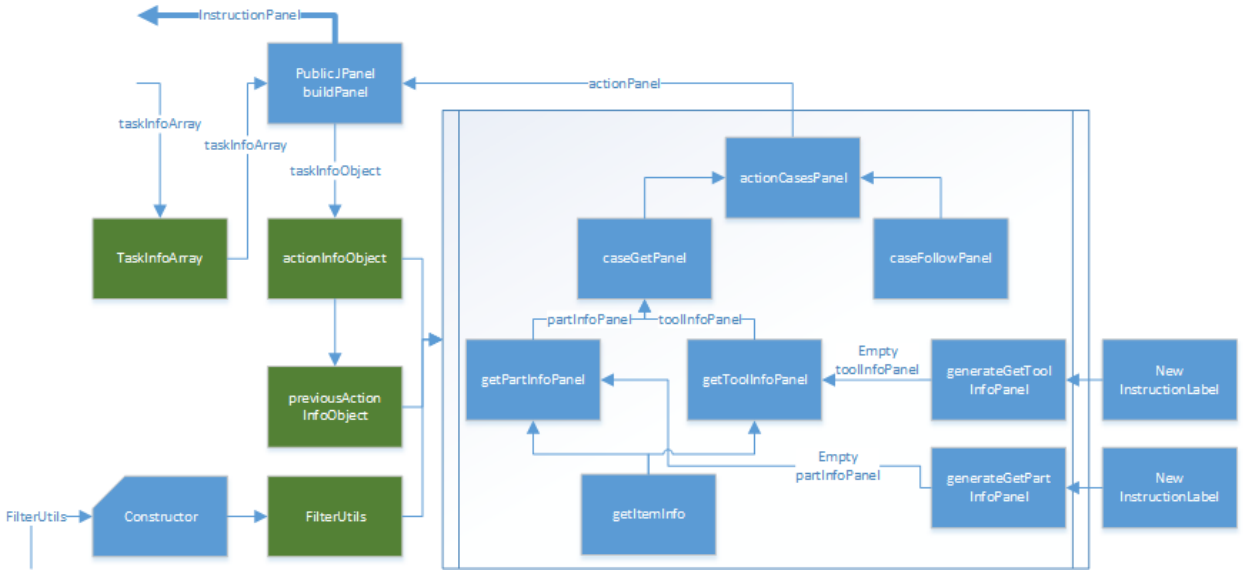
Class FrameUtils



The FrameUtils class extends the JFrame class. It holds instances of an InstructionPanel, InstructionLabel, SecondTimer and ButtonPanel. Together, they form the JFrame which shows all the instruction material available. The SecondTimer holds a countdown timer showing the remaining time for the task. The ButtonPanel holds control buttons such as *Exit* and *Next Instruction.*

When created an instance of the ApplicationUtils is needed as input, as the application asks this instance for all information needed and also for frame constituents. When clicking the *done* button, the getNextInstruction method is called. It asks the instance of the ApplicationUtils to update the taskInfoArray to the next task, which it retrieves from the ServiceUtils instance. It then asks for the task time to set the Timer in the ButtonPanel instance. The buildString and buildPanel methods are called in the ApplicationUtils instance to get the instructions in wished format. The InstructionPanel and InstructionLabel are cleared and filled with new input from the buildPanel and buildString methods.

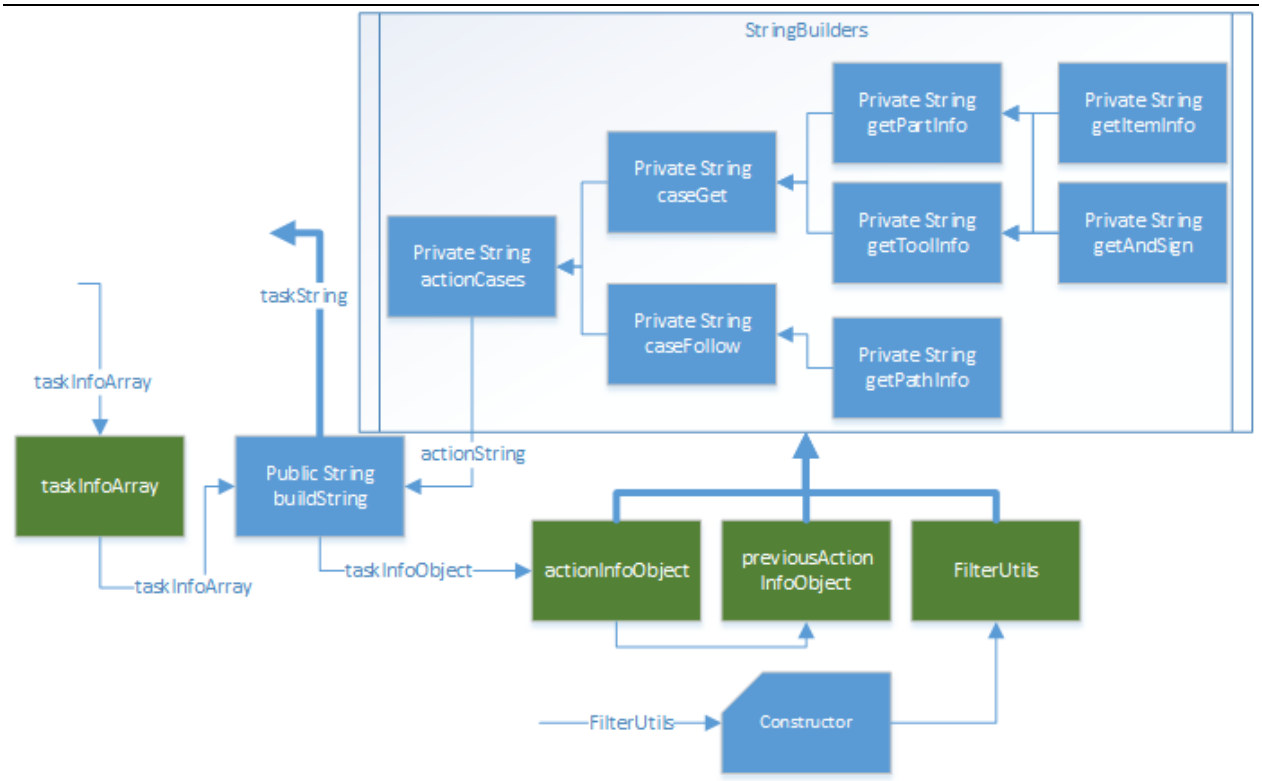
Class PanelUtils



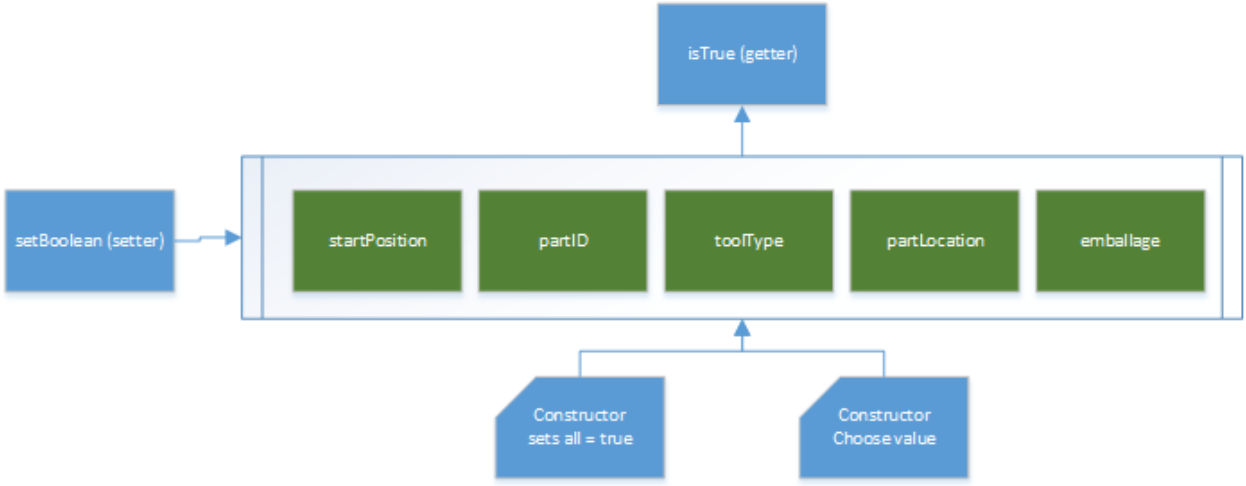
This class reads the data within the taskInfoArray and builds tables of the actions and associated data like part and tool info, according to the specification of the operator. The array consists of objects representing each action and relevant data and reads them one by one. Information need is specified in the FilterUtils object, which this class holds an instance of. An instance of the class will not work without any filter and therefore an instance of FilterUtils is set as an input argument in the constructor.

The lists are built using InstructionLabels, which is a subclass of the JLabel. Information is finally put in a JPanel, which means that this class is a subclass of JPanel, which is returned. As of today, the action types *Follow, Get* and *Release* will generate information in the panel returned.

Class StringUtils



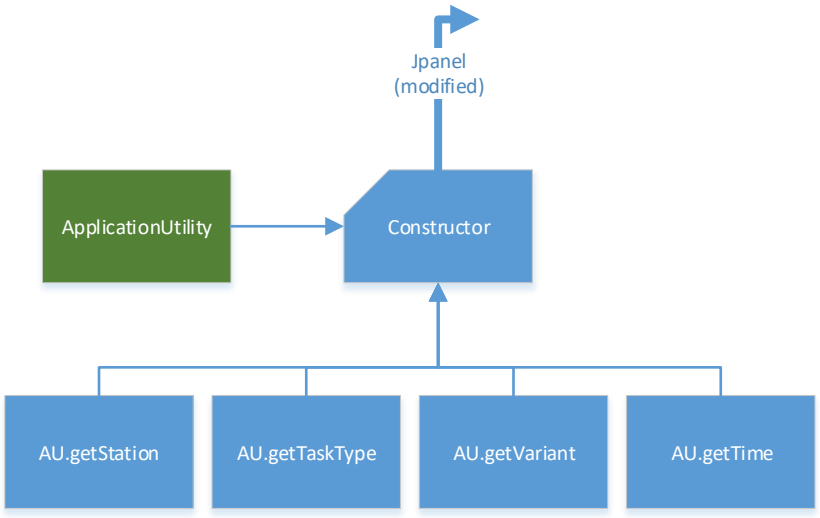
This class is similar to the PanelUtils class, it combines all information for all actions of a certain task to create a string. It uses the task info within the taskInfoArray as input. It holds an instance of the FilterUtils class to know what data to include in the text string. As this filter is necessary to have the instance functional, it is included as an input argument in the constructor.



The FilterUtils class holds the filter settings set by the operator. All settings are represented by Booleans as the information can be included or excluded in the instruction. As the filtering information is needed in several classes such as ServiceUtils, StringUtils and PanelUtils, they are put in a separate object so that these three instances can share one instance of FilterUtils. Also, this makes sure that the filter settings are coordinated between different instances.

The class has two constructors. The one having input arguments to set all the Booleans, the other one having no input arguments, setting all booleans to true. In the latter case, the operator will get all information available. The class also holds setter and getter functions to be able to change Boolean values once they are created.

Class TaskPanel

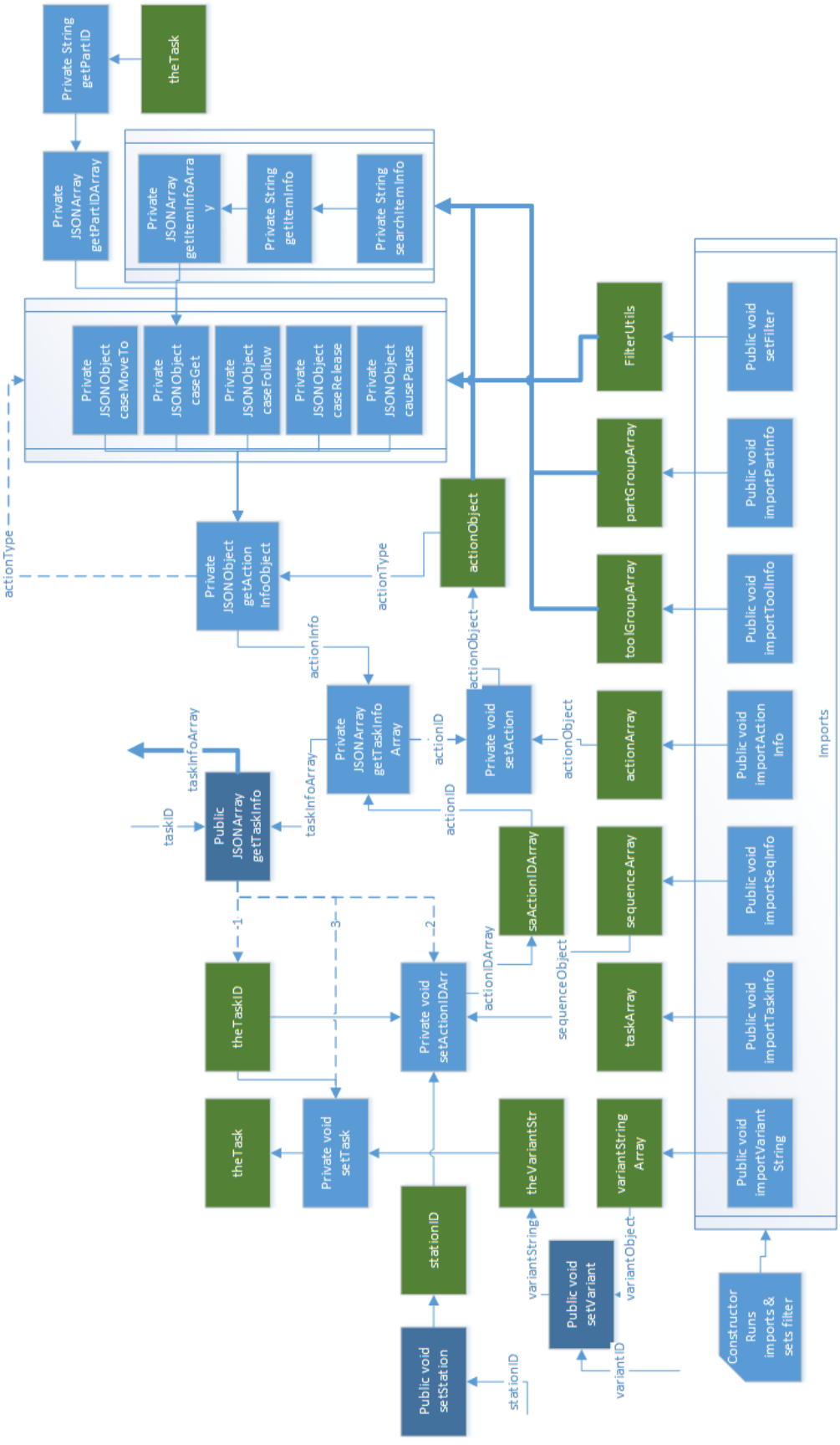


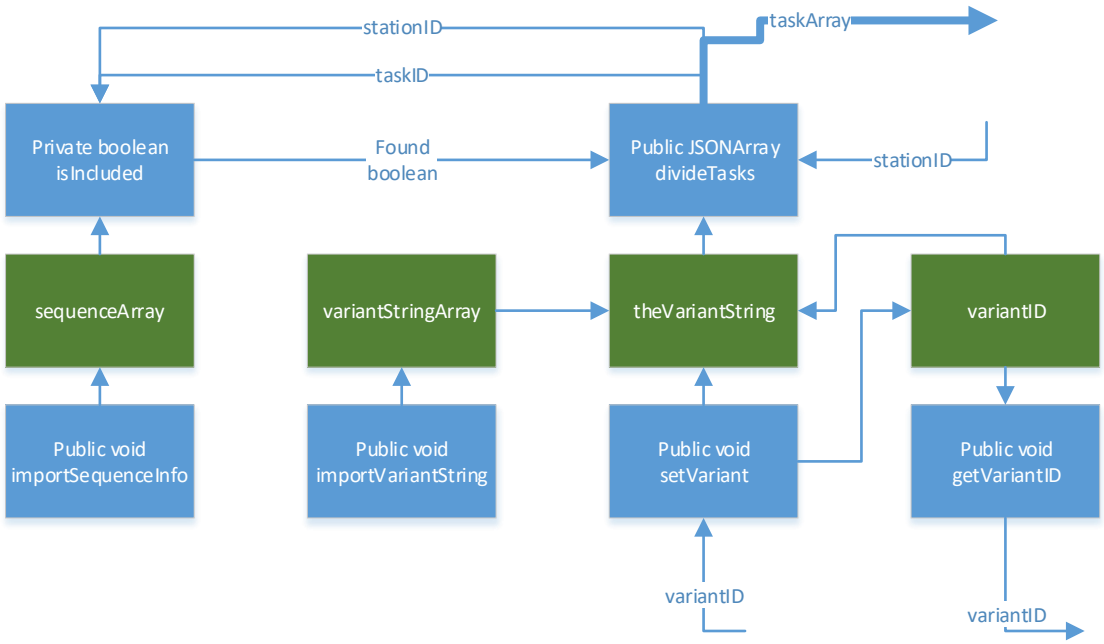
This class is an extension of the JPanel which fills the purpose of showing task-specific information such as variant ID, planned execution time and task name. It holds an instance of the ApplicationUtils class to gather relevant information. As this is needed, this instance is required as input in the constructor.

The ServiceUtils instance receives a taskID and returns a taskInfoArray, which is a JSONArray holding JSONObjects of actionInfoObjects. These are a combination of data that is set in the filter meaning that this class holds a FilterUtils instance which is read by calling getter-methods. The first key explains the actionType, and then other relevant information is listed such as start/stop, location, etc.

Since each station holds an instance of the ApplicationUtils class which in turn holds an instance of the ServiceUtils class, the stationID is also stored in the ServiceUtils instance. By having the taskID as input and the stationID stored, it has the primary key to the sequence object to be performed. Once the getTaskInfo method is called it loops through the actionIDs in the sequence one by one to find each action object and then fetch any information needed by searching relevant info on parts, tools and locations. Since the mBOM is stored in the variant string, the variantID is imported into the ServiceUtils instance so that the partID can be gathered if needed.

The data on parts, tools, actions, etc. is gathered from the database which in this case is made simple by storing all objects of the same type as JSONObjects in a JSONArray in a JSON file. The file is then imported when the ServiceUtils instance is created so that all arrays are stored in the instance.


Private String getPartID
theTask
Private JSONArray getPartIDArray
Private JSONArray getItemInfoArray
Private String getItemInfo
Private String searchItemInfo
Private JSONObject caseMoveTo
Private JSONObject caseGet
Private JSONObject caseFollow
Private JSONObject caseRelease
Private JSONObject causePause
FilterUtils
Public void setFilter
partGroupArray
Public void importPartInfo
toolGroupArray
Public void importToolInfo
actionObject
actionType
Private JSONObject getActionInfoObject
actionInfo
Private JSONArray getTaskInfoArray
actionID
Private void setAction
actionObject
actionArray
Public void importActionInfo
taskInfoArray
Public JSONArray getTaskInfo
taskID
-1
3
2
saActionIDArray
sequenceArray
Public void importSeqInfo
theTaskID
Private void setActionIDArr
actionIDArray
sequenceObject
taskArray
Public void importTaskInfo
theTask
Private void setTask
theVariantStr
variantString Array
Public void importVariantString
Imports
stationID
variantString
Public void setVariant
variantObject
Public void setStation
stationID
variantID
Constructor Runs imports & sets filter

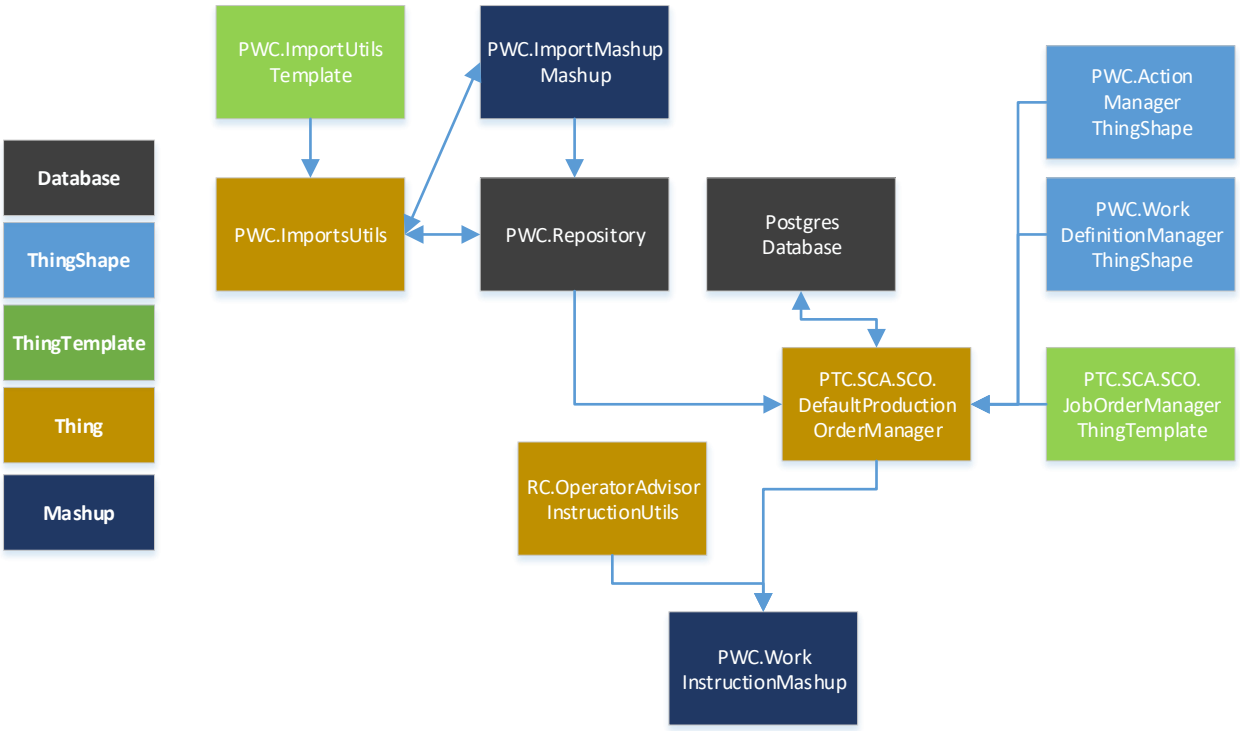


This class uses the variantID to extract the tasks needed to complete the product from the variantStringArray. The tasks are then split per station and taskIDs placed in separate taskIDArrays and sent to the instance of ApplicationUtils at each station. Since the sequence object has a primary key that is a composite key of the taskID and stationID respectively, knowing the stationID, the tasks can be assigned for each station. There is an assumption that a task can only be performed at one station. Otherwise, other methods to distribute the tasks between stations will be needed. In the full implementation, it should be possible to perform the same task at different stations, which means that the distribution is not definite but instead flexible.

Each application does not itself have the ability to gather information on what tasks to perform, the taskIDs must be provided. In a production environment, the instance of this class can be seen as an order manager. By centralizing the planning task, it can make sure that all tasks necessary to complete the product will be performed. Functionality that has the applications send assembly receipts could be implemented to mirror such system behavior.

Appendix II: ThingWorx demonstrator

PWC.WorkInstructionDemonstrator



Mashup: PWC.ImportMashup

This mashup uploads files to the repository, PWC.Repository. Simple functionality is also added so that it runs relevant services for each specific file imported, these services are in the PWC.ImportsUtilsThing.

Mashup: PWC.WorkInstructionMashup

The work instruction demonstrator is modeled in this mashup. It uses services from RC.OperatorAdvisorInstructionUtils and PTC.SCA.SCO.DefaultProductionOrderManager to gather info about tasks and associated parts, tools and actions. It also has the ability to show videos or animations imported from the IMMA simulation.

ThingTemplate: PWC.ImportUtilsTemplate

The PWC.ImportUtilsTemplate holds services to import data from IMMA and AviX and map them properly to Data Shapes and saving them in the Database by calling methods in the order manager of Operator Advisor, PTC.SCA.SCO.DefaultProductionOrderManager.

ImportAviX

Loops through the AviX Downstream connector export, per process group and calls the ImportTasks service. For each process group, it creates a WorkDefinition, acting as a parent to the WorkDefinitions created in the ImportTasks service later. It also creates a job order which has the address of the created WorkDefinition. For the ImportTasks service to

work properly, it needs the XML snippet of the process group and the WorkDefinition UID of the parent.

ImportTasks
Loops through the tasks in the process group and creates a WorkDefinition for each one of them. It also creates related WorkDefinitionLink which ties the WorkDefinition parent created in ImportAviX service. In each task, it checks if any associated parts or tools are to be found. If found, it calls the services ImportParts and ImportTools respectively. For them to work, XML-snippets and WorkDefinition UID are sent as input arguments.

ImportParts
Loops through the parts used in the task and creates a MaterialDefinition for each one of them. It also creates a WorkDefinitionMaterialSpecification to link the two together.

ImportTasks
Works similar to the ImportParts service. It loops through the tools used in the task and creates a ProcessingResource for each one of them. It also creates a WorkDefinitionProcessingResourceSpecification to link the two together.

ImportIMMA
Loops through the tasks in IMMA one by one. For each one, it takes the name and calls a function to get a table of all tasks with the same name. It then checks if any WorkDefinitionActionSpecifications exist for these WorkDefinition UIDs. In case they are missing, the JSON-snippet of the task is sent to the ImportActions service along with WorkDefinition UID. Due to this reason, the WorkDefinition presenting the task must already be existing, and therefore it is required that the AviX import is performed before the IMMA import.

ImportActions
Loops through the actions performed in the task. For each one of them, an action is created as well as a WorkDefinitionActionSpecification to link them together with the WorkDefinition.

ImportVideos
First gets a table of all WorkDefinition names. Loops through the list and calls the service ImportVideo to set the video paths based on the folder chosen. It requires that all videos be placed in the same folder and that they are named the same as the task, without any spaces. Only GIF-format files are accepted in the function called.

ImportVideo
It takes the name of the task as input and uses to call services to get a table of all WorkDefinition with matching names. For each WorkDefinition, it checks for associated WorkDefinitionIllustrations, the object used to specify the location of the video, picture or animation. If they exist, the service checks if any of them are videos. Those who represent a video illustration are updated. If not existing, a WorkDefinitionIllustration object is created with a path to the video, which is based on the task name itself.

SetRepository
The ImportUtilsTemplate holds the ThingName of the RepositoryThing for its services to call the right Repository. By default, its value is set to PWC.Repository.

GetRepository
This service can be called to set the value of the RepositoryThing where import files are stored if it is different from PWC.Repository, which is the default value.

Thing: PTC.SCA.SCO.DefaultProductionOrderManager

The Production Order Manager is a default thing within the Operator Advisor package which holds services to create, update, get and delete objects of the Operator Advisor information model. Most services are inherited from PTC.SCA.SCO.JobOrderManagerThingTemplate. To add the Action concept, it has been modified to also implement PWC.ActionManagerThingShape and PWC.WorkDefinitionMangagerThingShape.

ThingShape: PWC.ActionManagerThingShape

This ThingShape holds services to create, update, get and delete objects of the ActionShape, ActionMaterialSpecification and ActionProcessingResourceSpecification. It is implemented in the PTC.SCA.SCO.DefaultProductionOrderManager

ThingShape: PWC.WorkDefinitionManagerThingShape

This ThingShape has services that enable the user to create, update, get and delete WorkDefinitionActionSpecifications. It also has other services related to the WorkDefinition Shape, such as searching for them by Name. Those are needed for some newly implemented services, not included in PTC.SCA.SCO.DefaultProductionOrderManager as of today.

Thing: RC.OperatorAdvisorInstructionUtils

The thing has services which call services of PTC.SCA.SCO.DefaultProductionOrderManager and the Postgres Database, mainly to get information relevant for the work instruction to be presented in any mashup, e.g., the PWC.WorkInstructionMashup. By giving a Job Order ID or WorkDefinition UID, it can retrieve associated parts, tools, actions and illustrations, among other things. Some extra services have been added to be able to retrieve action information.

DataShapes

ActionShape: Represents the added Action object

WorkDefinitionActionSpecification: Links a WorkDefinition and an Action.

ActionMaterialDefinitionSpecification: Links a MaterialDefinition and an Action.

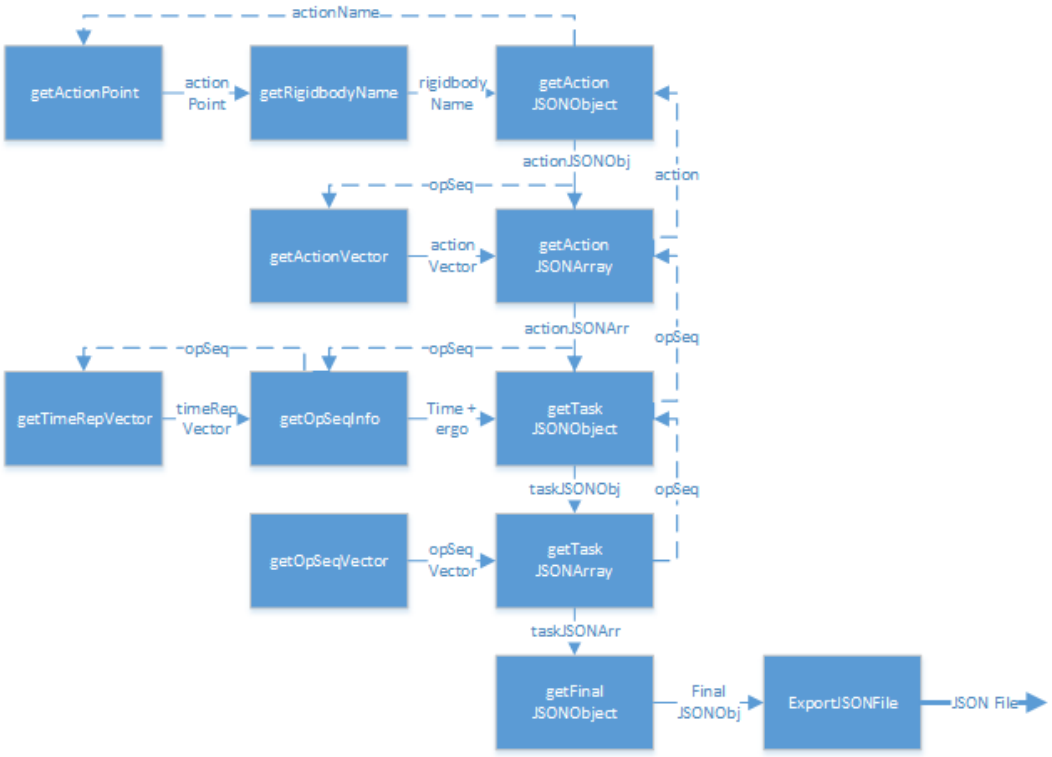
ActionProcessingResourceSpecification: Links an ActionProcessingResourceSpecification and an Action.

Appendix III: IMMA Lua script

To get data from IPS IMMA, a Lua based API is called. Since IMMA is a recently added module, its functionality is limited. Some functionality, such as searching the tree of active objects has more extensive functionality as it is a base of the platform, included in all modules.

Most data used to populate the information model is gathered via the OperationSequence objects, except the task time and ergonomic scores. That means that the OperationSequence holds instructions for how the manikin should move. It also means that the data used in the information model is input data to IMMA and not output.

To get the time and ergonomic scores, a simulation must be run using the instructions defined in the OperationSequence so that a TimelineReplay can be created which holds the simulation result data.



GetTaskJSONObject

Takes an OperationSequence and calls functions to get information to create the Action object. GetActionJSONArray is called to get the array of all Actions. GetOpSeqInfo is used to get the time and ergo data. Task information can be found in the OperationSequence itself.

GetTaskJSONArray

Calls the GetOpSeqVector to get a vector of all OperationSequences in the project, reads them one by one, and uses them to call the GetTaskJSONObject. The JSONObjects are put in a JSONArray which this function returns.

GetActionJSONObject

Takes an Action and calls supporting functions to get Action data. The GetActionPoint function takes the Action name and returns the name of the Grip or Viewpoint. Using this information, it calls the GetRigidbodyName function which searches the Active objects tree to find the Grip or Viewpoint and thereby its parent, which is the Rigidbody to which it belongs.

GetActionJSONArray

The function first calls GetActionVector to get the vector of all Actions that the manikin performs in the OperationSequence given. Using this information, it reads each Action one by one and calls the GetActionJSONObject function one by one to get Action JSONObjects.

GetFinalJSONObject

This function takes calls GetTaskJSONArray and encapsulates it in a JSONObject as required by ThingWorx.

GetOpSeqInfo

Takes an OperationSequence, calls the GetTimeRep function to get the TimelineReplay, and extracts time and ergo data. The ergo data available is extensive, however, this function just gives an average point as the ergonomic analysis is outside the scope of the project.

GetOpSeqVector

Retrieves the vector of OperationSequences in the project.

GetTimeRepVector

Takes an OperationSequence and returns a vector of all TimelineReplays.

GetActionVector

Takes an OperationSequence and gets a vector of all Actions the manikin performs.

GetActionPoint

Takes the name of the Action and extracts the name of the Grip or Viewpoint by decoding its semantics.

GetRigidbodyName

Takes the name of a Grip or Viewpoint and searches the active objects tree to find it and thereby its parent, which is the Rigidbody to which it belongs. This requires that all Grip and Viewpoints have unique names.

ExportJSONFile

Takes a JSONObject and saves it in a file.

DEPARTMENT OF INDUSTRIAL AND MATERIALS SCIENCE
DIVISION OF PRODUCTION SYSTEMS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY