10th International Conference on Mass Customization and Personalization – Community of Europe (MCP - CE 2022)
Toward the Sustainable, User-Centric and Smart Industry 5.0
September 21-23, 2022, Novi Sad, Serbia

APPLICATION OF DECENTRALIZED AND SELF-REGULATING KNOWLEDGE BASES FOR ASSEMBLY DESIGN AUTOMATION

Stefan Plappert, Christian Becker, Paul Christoph Gembarski, Roland Lachmayer
Leibniz University of Hannover, Institute of Product Development, Hannover, Germany

Abstract: *During product development, changes to parts that are already built into assemblies usually lead to the need to check the function and consistency of the assembly. This procedure is very time-consuming and has to be performed again for each change. In this paper, an approach is presented in which the individual parts are represented as agents that adapt themselves to new conditions. The agents are combined in a multi-agent system (MAS) and interact via communication over messages. For this purpose, a methodical procedure for the development of the MAS and the implementation in a CAD development environment is presented. The validation of the MAS is carried out on the application example of a gearbox.*

Key Words: ***Multi-Agent System (MAS), Solution Space, Knowledge-based Engineering (KBE), Computer-Aided Design (CAD), Product Configuration***

1. INTRODUCTION

In order to handle the increasing demand for more customized products and systems, design automation systems are used in product development to support the designer in routine tasks by using design rules, dimensioning formulas, or automatic routines for geometry generation [1]. Most research considers the modeling of configurable products as an arrangement problem of a predefined set of components (solution spaces) into a valid product structure [2]. These large solution spaces pose the problem that a consistency check must be performed when the solution space is modified or extended. This consistency check can be very time-consuming if the entire solution space has to be checked, especially if the influence of a change cannot be limited to a specific range.

To overcome the problem of consistency checking, this paper presents an approach to divide the solution space into smaller decentralized solution spaces and perform exploration or consistency checking of the solution space by using communication of the decentralized components. For this purpose, multi-agent systems (MAS), which are based on the paradigm of distributed artificial intelligence, represent a promising approach, since real entities and their interactions can be directly represented by autonomous problem-solving agents [3]. Moreover, by partitioning into agents, multiple views of the product can be represented, such as a design or manufacturing perspective, which can help in the development of configurable products [2]. In order to support the designer in generating and evaluating solutions and to involve him in the decision-making process, the MAS has both communication capabilities and an interface to the CAD program, so that an automated adaptation of the CAD model can be made by varying the parameters.

The remainder of this paper is organized as follows: In section 2, the theoretical background and related works on multiagent systems are presented. Then, section 3 describes the methodological approach and the application example. The approach for decentralized solution space development with MAS is explained in section 4 and discussed in section 5. A summary and description of further research are given in section 6.

2. THEORETICAL BACKGROUND AND RELATED WORK

Agent-based technologies have their beginnings in the 1990s and are based on the paradigm of distributed artificial intelligence and program agents [4]. An intelligent agent consists of autonomous knowledge-based systems that have their own resources and expertise and can interact with others by perceiving, reasoning, adapting, learning, cooperating, and delegating in a dynamic environment [5]. This autonomous action is one of the main characteristics of agents in contrast to conventional software [4], as they can perceive the environment and make inferences in an intelligent manner. According to Weyns [6], agents have three services: the *perception service* enables the sensing of the environment and representation of it, the *action service* coordinates the operations of the agents, and the *communication service* manages the interactions among the agents.

In MAS, the common goals are achieved through collaboration since no single agent has all the knowledge required to solve a problem [7]. The basic idea of network agents is that they collaborate in solving problems, share knowledge, work asynchronously, are modular and robust through redundancy, and represent multiple viewpoints and expertise of experts [8].

One of the key components of MAS is communication. Agents must be able to communicate with users, with system resources, and with each other when cooperating, collaborating, negotiating, etc. [9]. Collaboration is a controlled process that takes place in a

collaborative environment where a group of agents works together towards a specific goal [10]. To ensure consistency of concurrent decision-making and integrity of information in product development, a coordination instance must be established. In addition, agents can negotiate to reach a mutually acceptable agreement on a particular course of action [11]. For structuring collaboration, a well-known organizational approach is to negotiate via a blackboard [12], where each agent has access to task-specific information.

Probably the best-known and most popular deliberative agent architecture is the Belief-Desire-Intention (BDI) architecture [9], [13]. This allows the agent to pursue explicit goals that are constantly compared with current beliefs to select appropriate implementation plans and provide them to the agent as intentions for execution. Here, agents draw on three sources of information: (1) perceptual information, where an agent acquires certain beliefs as a consequence of sensing the environment, (2) communication, where agents use message content as well as metadata, and (3) mental notes, where agents document their behavior and are able to remind themselves of things that happened in the past [14].

Three research directions can be identified in the literature on MAS in product development [15]. First, Chu et al. [16] use a MAS to enable a geographically distributed group of users to work synchronously and collaboratively in a 3D assembly over the Internet. Second, MAS are used as assistance systems for design engineers to support them in decision-making for DfX (Design for Excellence). This usually involves external intervention on the design, using different prior knowledge and expertise to evaluate a design [17]. Third, MAS can be built from agent-based functions, where CAD models can autonomously adapt to new situations by representing agents as intelligent features [18].

Design support systems based on MAS offer the possibility to identify, store, and process-specific domain knowledge in a decentralized manner so that a holistic view of the value-added process can be taken, and thereby engineers can be supported already in the product development phase [19], [20].

Ostrosi et al. [2] apply a MAS in the conceptual phase of the product development process to assist collaborative and distributed design and to consider the customer requirements using fuzzy set-based reasoning. For this purpose, they use four communities of agents: requirement community of agents, function community of agents, physical solution community of agents, and process constraint community of agents. In this way, the fuzzy relationship between product functions and alternative physical solutions could better represent the designer's continuous or approximate reasoning.

Bender et al. [21] define a Virtual Product Model Component (VPM-C) which is divided into four elements: a *part* represents a logistical data set and contains information such as color or material, a *geometry* is a 3D-CAD model, a *feature* describes the functionality and a *process* contains information about the production. The component agent is passive most of the time and monitors the actions of the designer in the CAD program and tracks the changes to the component. So far, the component agent is only a concept and has not yet been validated on an application example.

For the modeling of MAS frameworks, there are two approaches, either MAS frameworks are specifically programmed for specific areas of interest or they follow a holistic approach in the form of general-purpose platforms [22]. The best-known representative of the general-purpose platforms is JADE (Java Agent Development Framework), which is probably the most widely used MAS tool of the last two decades [23]. JADE is a FIPA-compliant platform that is built from agent containers distributed in a network [24]. Among the newer representatives of MAS is the platform SPADE 3.0 (Smart Python Agent Development Environment), which is programmed in Python and uses open instant messaging protocols to enable distributed communication between humans, agents, and third-party elements [22]. In addition, SPADE enables interaction between agents by exchanging information through predefined agent and behaviour classes. Furthermore, SPADE proposes the asynchronous programming paradigm to implement the agents' code internally.

Based on the theoretical background and related work on MAS, a possible architecture for MAS to represent decentralized solution spaces in a CAD program using the SPADE platform is investigated. The application domain here is product configuration and, in particular, the use of design automation. For this purpose, the following research questions have been identified:

- What is a methodical approach for the design and implementation of a configuration problem in which the individual components are represented as decentralized agents that communicate and interact with each other?
- How can an architecture for design automation be built with SPADE as MAS framework and an interface to a CAD program?

3. METHODICAL PROCEDURE

3.1 Multiagent Systems Engineering (MaSE)

For the function and subsequent maintenance of a MAS for design automation, a methodical and structured approach is decisive to resolve the trade-off between modularization through decentralized solution spaces and standardization through reusable components. One possible approach is the MaSE (Multiagent Systems Engineering) method by Deloach et al. [25], which has its origins in information systems engineering and starts from the initial system context and proceeds through an analysis and design phase (Fig. 1).

The analysis phase aims to define the system goals from functional requirements and subsequently derive roles that fulfill these goals. The process is divided into three steps: capturing goals, applying use cases, and refining roles [25], [26]:

- *Capturing Goals*: The first step extracts the goals based on the requirements and divides them into the overall system goal and subgoals.

- *Applying Use* Cases: In the second step, the goals are translated into use cases and the usage scenarios are described in detail.
- *Refining Roles*: In this step, roles are derived to implement the goals, since roles are typically goal-driven and map conveniently to agents.

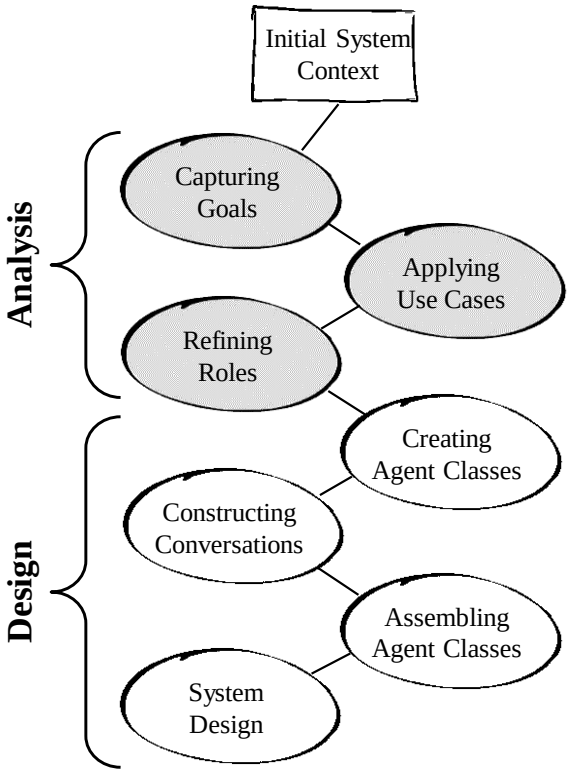


Fig.1. MaSE Phases (according to [25]*)*

The purpose of the design phase is to transform the goals and roles into a MAS in which agents are defined and their communication with each other is described. The MaSE design phase consists of four steps: creating agent classes, constructing conversations, assembling agent classes, and system design [25], [26]:

- *Creating Agent Classes*: In this step, the engineer assigns roles to the specific agents. An agent class is a template for a type of agent in MAS analogous to the object class in object-oriented programming.
- *Constructing Conversations*: Conversations model the communication between two agents, which are defined as coordination protocols. Here, the messages that are exchanged are defined as performatives that contain the purpose of the message as well as a set of parameters that represent the content.
- *Assembling Agent Classes*: In this step, the internal architecture of the agents is defined, which includes in particular the inference engine and the knowledge base.
- *System Design*: The last step involves selecting the current configuration of the system, such as the number of agents and agent types or the platform on which the MAS is deployed.

4. DECENTRALIZED SOLUTION SPACE DEVELOPMENT WITH MULTI-AGENT SYSTEMS

4.1 Application Example

Based on the methodical approach of the MaSe method, the extent to which this approach can be transferred to a design problem, which can be abstracted as a configuration problem, is investigated. For this purpose, a gearbox (Fig. 2) from engineering design is used as an application example, which consists of standard parts and is supplemented by individual manufacturing parts. In addition, there are several fixed interfaces between the components that influence each other, such as the diameter of the shaft and the inner diameters of the bearings or gear wheels. A special feature is an interface between the gears and the housing because here it has to be avoided that the components intersect and thus assembly would not be possible.

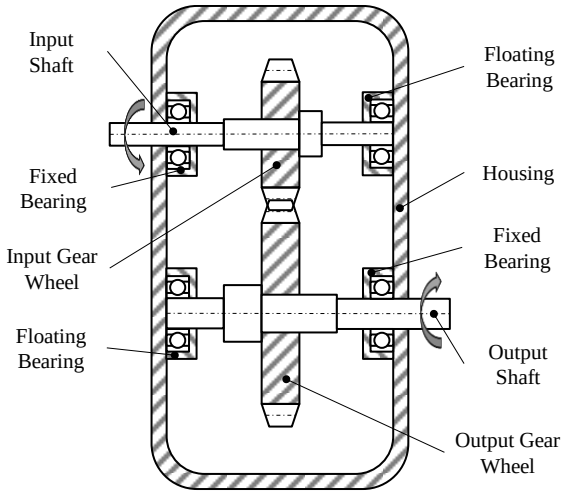


Fig. 2. A Gearbox as Application Example

The main function of the gearbox is to transmit or convert torque or speed. For this purpose, the gearbox is divided into several stages. In this example, there are two gear stages, one for the input torque or speed and one for the output torque or speed. Each of these stages contains a shaft to transmit rotation from outside the gearbox to the inside. On this shaft sits a gear wheel, which transmits the torque or speed. Here, the translation of the gearbox results from the ratio of the two gear wheels. To connect the gearbox with the environment, it is enclosed by the housing. Bearings serve as the interface between the shaft and housing, separating the dynamic and static components.

In the case of the application of the MaSE method in engineering design, the functions of the assembly are used to describe the goals, since the functional structure can be used to divide the assembly into modules.

4.2 Specification

After the goals or functions have been defined in the analysis phase, the next step is to consider the use cases. In this case, the configuration problem is viewed from a process engineering perspective by analyzing the subsequent usage of the system. For this purpose, a use-

case diagram (Fig. 3) is used, which divides the tasks of the human user or engineer and those of the MAS.

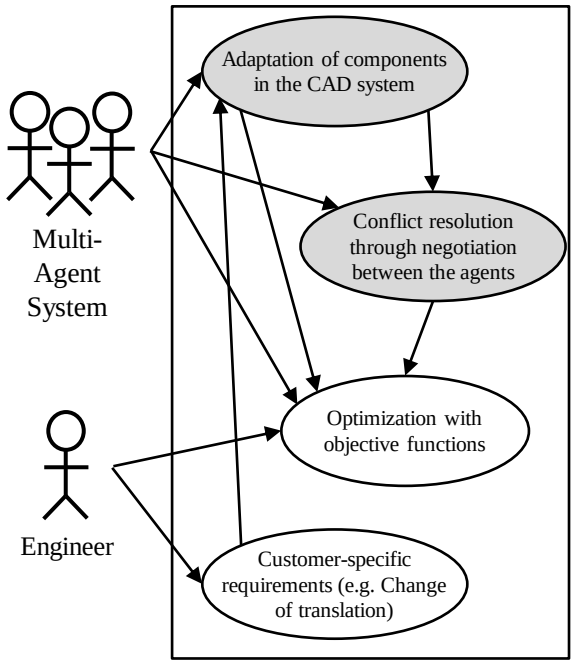


Fig. 3. Use-Case Diagram

In the case of the application example, the user provides requirements for a new configuration as input to the system. If necessary, he also specifies an optimization goal, e.g., the configuration should be as cheap or lightweight as possible. These optimization goals can be switched by changing behaviours within agents or distributed among multiple agents. Based on this information, the agents adjust the component assigned to them in the CAD system based on their knowledge and their own goals. In addition, they perceive the changes of the neighboring components and react to them as well. If a change leads to a conflict, e.g., the diameter of the shaft changes to a value for which there is no suitable bearing, this must be resolved. To do this, the agents negotiate with each other and try to come to an acceptable solution together. The agents also take into account the higher-level optimization goals specified by the user when coming to a conclusion or during the negotiation.

Based on the functions and the use cases, the roles are refined. In the case of the application example, the roles are derived on the basis of the functional entities or components. Liaison graphs can be used for this purpose because they allow the representation of an assembly as a graph [27]. The liaison graph for the gear application example is shown in Fig. 4.

The liaison graph consists of nodes and edges, where the nodes represent the components and the edges between the nodes represent the relationships between the different parts of the assembly [28]. Usually, the edges represent physical contact between the parts. By representing the assembly as a graph, the interfaces between the different components are clarified and the nodes can be understood as abstract placeholders, which are filled with content and knowledge in the further steps.

The liaison graph can be used on the one hand, to abstract the construction and/or configuration problem, and on the basis of it, the components are derived. On the other hand, the liaison graph can be derived directly from an existing CAD assembly [29] in order to parameterize it afterward. In addition, further information can be read out from the CAD model, such as geometry or material information.

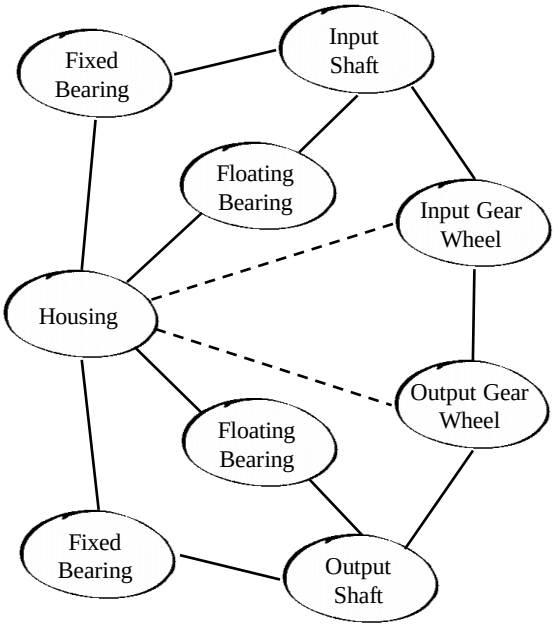


Fig. 4. Liaison Graph

With the definition of the roles and the components, the analysis phase of the MaSE method is completed. These first steps can be performed analogously for other graph-based methods, such as constraint satisfaction problems (CSP) or Bayesian networks (BN) [30].

4.3 Architecture

The design phase of the MaSE method takes the conceptual models of the analysis phase and implements them in an information system. For this purpose, agent classes are derived from the goals or functions and the components from the liaison graph in the first step. By using classes, the solution space for a component can be encapsulated and specific instances can be derived from the classes. Furthermore, the BDI architecture finds its application, since in this step the beliefs, by the interfaces to the neighboring components, the desires (Goals), by the specific goals of the agent class, and the intentions, by the parameters to be influenced as action set can be specified. The agent classes for the gearbox are divided into housing, bearing, shaft, and gear wheel.

One of the key functions of a MAS is communication because it can be used to link the decentralized solution spaces of the agents and to resolve conflicts or inconsistencies. Starting from the liaison graph, each edge represents a relationship between two components, which is realized in a MAS through communication. A widely used standard for messages between agents represents the FIPA (Foundation for Intelligent Physical Agents) communication model, where the context of multiple messages is provided by interaction protocols and conversation identifiers [24]. To allow each agent to

participate in the communication, the conversations defined in this step of the MaSE method are implemented in each agent.

To be able to meet the requirements described above, each agent class or agent must have basic functionalities, which are supplemented by individual properties. The internal structure of an agent class (Fig. 5) is based on the modules of the architecture from Plappert et al. [19]:

- *Perception & Action Module*: This module interfaces with the CAD model so that feature recognition can be used to perceive changes in the assembly. This is done by accessing the structure of the model, e.g., the feature tree, physical properties such as weight or material, or the faces, edges, and nodes of the B-Rep (boundary representation) structure. Furthermore, changes can be made to parameters so that the geometry of the model can be adapted.
- *Communication Module*: The communication of the agents in the communication module is done via an XMPP (eXtensible Messaging and Presence Protocol) server by sending and receiving messages. Besides the content, performative FIPA attributes like queries or informs can be given by metadata.
- *Knowledge Module*: The knowledge module contains specific domain knowledge, e.g., tables for standard parts or manufacturing restrictions. In addition, it is possible to use the knowledge from external databases.
- *Decision Module*: In the decision module, inferences are made as to which action the agent will perform next based on the communication with the other agents, the perceived changes in the CAD model, and the stored knowledge. For targeted communication, the agent also has an interface to the agent management via which it can query additional information about the other agents (similar to a yellow page service).

After the individual agents have been created based on the agent classes, the conversations defined and the required domain knowledge stored, the individual entities must be assembled into a complete system and made available for use. For this purpose, the overall system is divided into four layers (Fig. 6):

- *User Layer*: The user layer represents the interface to the user. The user can make entries via a graphical user interface (GUI). In addition, the coordination agent can involve the user in the decision-making process.
- *Management Layer*: The management layer manages the MAS in which the coordination agent passes on the user's information. In addition, the information about the agents, as well as agent classes, and their properties are archived in the directory facilitator (DF). Through the interface to the CAD assembly, higher-level changes can be detected, such as the addition of new components.
- *Agent Layer*: The agent layer contains the component agents from the liaison graph, which solve the configuration task through communication.
- *Parameter Layer*: The parameter layer contains the individual CAD components of the assembly, which are controlled via the various agents.

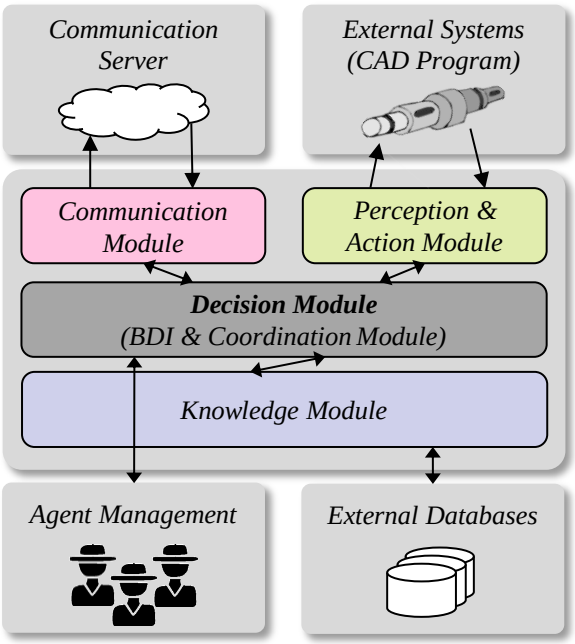


Fig. 5. Agent Architecture

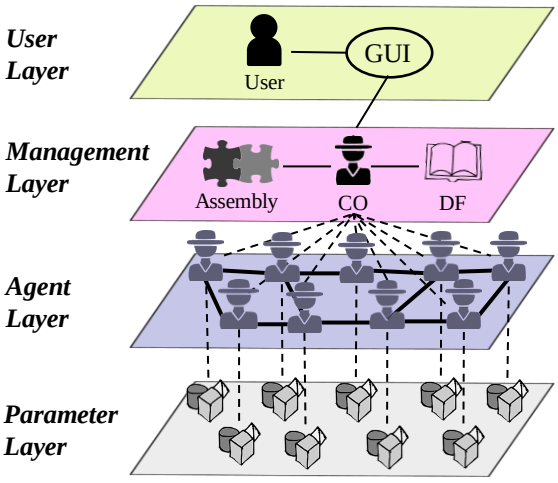


Fig. 6. Deployment Diagram

By dividing the system into layers, changes and maintenance can be carried out separately without having to adapt the entire system. In addition, this structure helps in setting up the overall system. Starting from the agent layer, the parameters of the CAD models can be linked and the interfaces to the coordination agent or the agent information can be stored in the management layer. The

interface to the user is only done via the coordination agent so changes only have to be made to one connection. Also, the exchange of the GUI is possible without much effort.

4.4 Implementation

After the structure of the overall system with the four layers has been presented, the implementation of the multi-agent system for a gearbox is explained in the following. Starting from the agent layer, the agent classes are defined and programmed in the MAS platform SPADE, which uses a modern and full-featured programming language such as Python, which is one of the most widely used programming languages for artificial intelligence (AI) applications [22] [31]. Another unique feature of SPADE is that it communicates via XMPP (Extensible Messaging and Presence Protocol), an open protocol for instant messaging and presence notifications that allows for natural human integration into the loop [22].

Building on the SPADE Framework, a template is built for each class, which contains the relevant variables and the included functions. In addition, it is defined how many instances are derived from the class and assigned to the individual components. Furthermore, it is defined which agent classes have interfaces with each other and must therefore also communicate via these.

For ensuring the documentation the coordination agent keeps log, so that can be reconstructed, how the decision-making was accomplished. For this purpose, the coordination agent is linked to all used agents via a communication interface and the information about the agents is stored in the directory facilitator. An overview of the classes used for the application example and their interfaces is shown in Fig. 7.

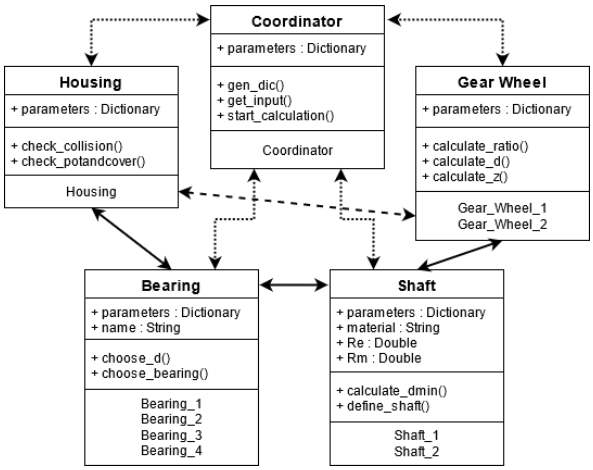


Fig. 7. Agent Classes

After the agents have been built and the functions have been stored, they are connected to the parameter layer in the next step. The parameter layer represents the highest level of detail, in which the parameters for controlling the CAD components are stored and the specific domain knowledge can be accessed. If an assembly or the CAD components already exist, the parameters can be derived directly from them.

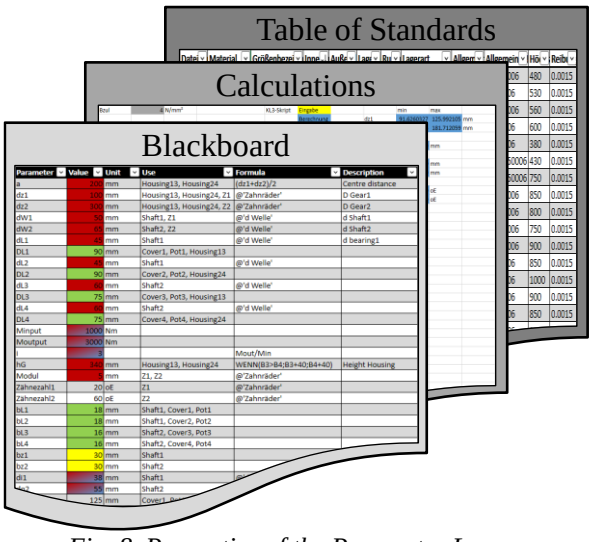


Fig. 8. Properties of the Parameter Layer

In the case of the gearbox, the information of the parameter layer is managed in an Excel file (Fig. 8). On the one hand, this has the advantage that the parameters can be transferred directly to the CAD program Autodesk Inventor to update the assembly. On the other hand, calculations such as the allowable torsional moment or the dimensioning of gear pairs are often performed in Excel tables and are therefore familiar to the designer. Furthermore, tables for standard parts, such as bearing catalogs, can be easily implemented and extended. With regard to the MAS, the Excel table also represents a blackboard that the agents can access and make changes to.

As soon as the agent layer and the parameter layer are linked and the interfaces to the management layer are established, the configuration of the gearbox can be started. For this purpose, the user makes changes to the torque or transmission parameters with the help of a GUI (Fig. 9).

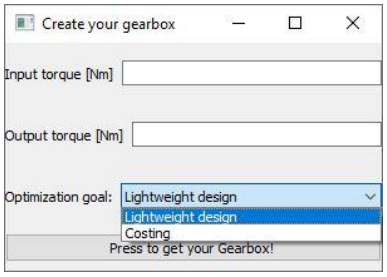
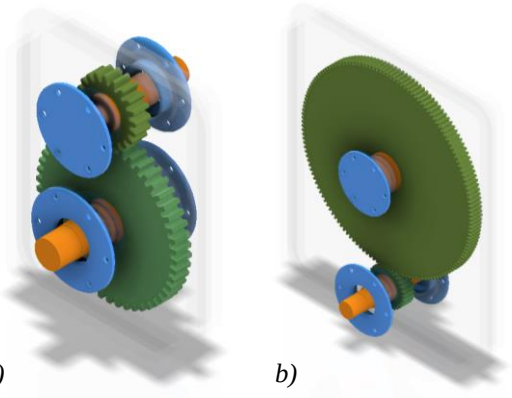


Fig. 9. GUI for the Gearbox Configuration

These changes are passed to the coordination agent, which in turn saves them in the blackboard and makes them available to the other agents. Fig. 10 shows two possible configurations of the gearbox, which differ in the doubling of the gear ratio.

Here, at first glance, the results of the assembly configuration are very similar to the configuration with known knowledge-based engineering systems. However, the reasoning follows a different principle, because the solution space is not completely described in the form of rules, models, or case bases, but is determined by the communication and collaboration between the agents. In this process, different perspectives and objectives can also

be stored in the agents. To give an insight into the decision-making process, the product configuration protocol is shown in Fig. 11.



a) *b)*

Fig. 10. A Gearbox as CAD Product Configuration

```
Start MAS...
...
manager: Generating Dictionary...
manager: Getting User input...
manager: Starting calculation...
manager: Sending message to shaft_1 ...
...
shaft_1: got a message from manager ...
shaft_1: Calculate dmin...
shaft_1: Sending message to bearing_1 ...
shaft_1: Sending message to bearing_2 ...
bearing_1: got a message from shaft_1 ...
bearing_1: Choosing d...
shaft_1: Sending message to gear_1 ...
bearing_1: Choosing bearing out of catalog...
gear_1: got a message from shaft_1 ...
bearing_1: Sending message to shaft_1 ...
gear_1: Calculate ratio...
gear_1: Calculate d and b...
shaft_1: got a message from bearing_1 ...
shaft_1: Define bearing seat of bearing_1 ...
gear_1: Calculate z...
gear_1: Sending message to manager ...
...
manager: Adding parameters gathered from gear_1 ...
manager: Adding parameters gathered from shaft_1 ...
manager: Sending message to housing ...
housing: Checking collision...
housing: Generating and checking pot and cover...
housing: No collision found...
housing: Sending message to manager ...
manager: Adding parameters gathered from housing ...
manager: Valid Calculation...
manager: Generating output...
MAS finished...
```

Fig. 11. Negotiation Protocol between the Agents

On the one hand, the protocol can be used to check the correctness of the assumptions so that the configuration represents a valid solution. On the other hand, negotiations can be traced and what influence the optimization goals have on them. An interesting communication has taken place between the shaft and the bearing agent. This is because the shaft agent changed the diameter at the bearing location so that the gear wheel could be mounted. The bearing agent then checks whether a suitable bearing is available in the database. Since this is not the case, it informs the shaft agent and suggests a different diameter to the shaft agent. The shaft agent checks the diameter and concludes that the increased diameter also increases weight. However, the user has specified as an optimization goal that the gearbox should be as light as possible. Since the optimization goal is a target requirement and the diameters of the bearing are fixed requirements, the shaft agent decides to accept the diameter change and adjusts the shaft diameter accordingly.

Compared to traditional product configurators, the MAS allows the solution space to be composed in the application phase. The individual agents are defined in advance, but only through the combination of the agents the entire solution space is completely defined and explored using communication between the agents. This has the advantage that the entire solution space does not have to be known during development, and if requirements change, only individual agents need to be adapted or replaced. In the case of the gearbox, for example, the bearing agent can be changed from a deep groove ball bearing to a roller bearing. For this purpose, an auction can be held to determine which bearing type is best suited for the given tasks. Furthermore, an extension of the system, e.g., by another gear stage, is easily possible, since only the class agents have to be transferred to the further components.

5. DISCUSSION

By implementing the configuration problem of a gearbox as a MAS, it could be shown that it is possible to divide a product configuration into different decentralized agents and arrive at a common solution through communication. In addition, the procedure for programming the system is very intuitive, since the individual components are first described and programmed before they are assembled into an overall system. As advantages of the MAS compared to central knowledge-based systems, the parallelization of processes through asynchronous programming, as well as the definition of agent classes that can be transferred to the specific agents for the individual components, were identified.

However, the parameters of the individual components are first determined by the agents and then coordinated through communication. This can lead to several iteration loops until an acceptable solution is found. In addition, each agent reacts separately to changes and adjusts its assigned component accordingly. Higher-level communication to use the same components within an agent class has not been implemented at this time. Further no multicriteria optimizations were accomplished, instead on the basis of the selected optimization goal the behaviors and computations within the agent were assigned.

6. CONCLUSION AND FURTHER RESEARCH

To overcome the problem of the lack of flexibility in the modification of CAD components, which are installed in different assemblies, and the central knowledge management or solution space development, this paper presents the development of a decentralized solution space using a MAS, in which the agents can be seen as self-customizing parts. By using agents and their reactive

behavior to changes, it is possible to estimate which interfaces and components have to be adapted, check the assembly's consistency after the change, and reflect possible inconsistencies directly back to the user.

The use of the MaSE method enables a structured approach to the development of a MAS, as it uses the analysis phase to determine the framework conditions and requirements for the MAS based on goals, use cases, and roles. The subsequent design phase supports the design and implementation of the MAS in a CAD development environment by defining agent classes, conversations, and the internal structure of the agent classes to subsequently embed them in an overall system.

In addition, the use of the SPADE framework proves practical, as the Python programming language enables the use of a variety of AI libraries, and communication via XMPP provides the ability to interact with other agents, humans, and even third-party tools.

The presented procedure offers a promising approach for decentralized knowledge management and automated adaptation of CAD components within an assembly. Furthermore, the agent architecture represents a kind of container, which has standardized interfaces, so that this container can be filled with specific knowledge and reasoning algorithms. By logging the exchanged messages, the decision-making process is documented and traceable.

The presented development is part of ongoing research, therefore the prototype has only been tested on a simple gearbox assembly so far. It is to be examined further how the system behaves with more extensive assemblies with more interfaces and dependencies. In addition, the provision of the infrastructure with the XMPP server means an initial additional effort, which is compensated only from a certain solution space size or with frequent changes.

For further research, the focus will be on the conclusion by means of communication and collaboration. On the one hand, the extent to which communication can be carried out in a more goal-oriented manner and, on the other hand, how negotiations can be linked-to optimization goals. Furthermore, we investigate how agents can learn by suggesting frequent configurations as initial design. Tools like case-based reasoning or probabilistic learning can be used for this purpose.

REFERENCES

[1] S. Vajna, C. Weber, K. Zeman, P. Hehenberger, D. Gerhard, and S. Wartzack, *CAx für Ingenieure*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2018. doi: 10.1007/978-3-662-54624-6.

[2] E. Ostrosi, A. J. Fougères, M. Ferney, and D. Klein, "A fuzzy configuration multi-agent approach for product family modelling in conceptual design," *Journal of Intelligent Manufacturing*, vol. 23, no. 6, pp. 2565–2586, 2012, doi: 10.1007/s10845-011-0541-5.

[3] N. R. Jennings and M. Wooldridge, "Applying agent technology," *Applied Artificial Intelligence*, vol. 9, no. 4, pp. 357–369, 1995, doi: 10.1080/08839519508945480.

[4] E. Dostatni, J. Diakun, D. Grajewski, R. Wichniarek, and A. Karwasz, "Multi-agent system to support decision-making process in design for recycling," *Soft Computing*, vol. 20, no. 11, pp. 4347–4361, 2016, doi: 10.1007/s00500-016-2302-z.

[5] Q. Liu, X. Cui, and X. Hu, "An agent-based intelligent CAD platform for collaborative design," *Communications in Computer and Information Science*, vol. 15, pp. 501–508, 2008, doi: 10.1007/978-3-540-85930-7_64.

[6] D. Weyns, *Architecture-Based Design of Multi-Agent Systems*, 1st ed. Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-01064-4.

[7] D. Ortiz-Arroyo and H. L. Larsen, "A Multi-Agent System Framework for Collaborative Decision Making Under Uncertainty," 2004.

[8] G. Weiss, *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press, 2000.

[9] F. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*, 1st ed. Chichester, UK: John Wiley & Sons, Ltd, 2007. doi: 10.1002/9780470058411.

[10] D. Weyns and F. Michel, "Agent Environments for Multi-agent Systems – A Research Roadmap," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9068, 2015, pp. 3–21. doi: 10.1007/978-3-319-23850-0_1.

[11] M. Ganzha and M. Paprzycki, "Implementing rule-based automated price negotiation in an agent system," *Journal of Universal Computer Science*, vol. 13, no. 2, pp. 244–266, 2007.

[12] W. Shen and J.-P. A. Barthès, "An experimental environment for exchanging engineering design knowledge by cognitive agents," *Knowledge Intensive CAD*, pp. 19–38, 1997, doi: 10.1007/978-0-387-35192-6_2.

[13] M. P. Georgeff and A. Rao, "An abstract architecture for rational agents," in *Proc. of the Third International Conference on Principles of Knowledge Representation and Reasoning*, 1992, pp. 439–449.

[14] R. H. Bordini, J. F. Hbner, and M. Wooldridge, *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, Ltd, 2007. doi: 10.1002/9780470061848.

[15] S. Plappert, P. C. Gembarski, and R. Lachmayer, "Multi-Agent Systems in Mechanical Engineering: A Review," in *Agents and Multi-Agent Systems: Technologies and Applications 2021*, vol. 241, G. Jezic, J. Chen-Burger, M. Kusek, R. Sperka, R. J. Howlett, and L. C. Jain, Eds. Singapore: Springer Singapore, 2021, pp. 193–203. doi: 10.1007/978-981-16-2994-5_16.

[16] C.-H. H. Chu, P.-H. H. Wu, and Y.-C. C. Hsu, "Multi-agent collaborative 3D design with geometric model at different levels of detail," *Robotics and Computer-Integrated*

Manufacturing, vol. 25, no. 2, pp. 334–347, Apr. 2009, doi: 10.1016/j.rcim.2007.01.005.

[17] P. C. Gembarski, "On the Conception of a Multi-agent Analysis and Optimization Tool for Mechanical Engineering Parts," in *Agents and Multi-Agent Systems: Technologies and Applications 2020*, vol. 186, G. Jezic, J. Chen-Burger, M. Kusek, R. Sperka, R. J. Howlett, and L. C. Jain, Eds. Singapore: John Wiley \& Sons, 2020, pp. 93–102. doi: 10.1007/978-981-15-5764-4_9.

[18] A. J. Fougères and E. Ostrosi, "Intelligent agents for feature modelling in computer aided design," *Journal of Computational Design and Engineering*, vol. 5, no. 1, pp. 19–40, 2018, doi: 10.1016/j.jcde.2017.11.001.

[19] S. Plappert, P. C. Gembarski, and R. Lachmayer, "Knowledge-Based Design Evaluation of Rotational CAD-Models with a Multi-Agent System," in *Advances in Systems Engineering*, L. Borzemski, H. Selvaraj, and J. Świkatek, Eds. Cham: Springer International Publishing, 2022, pp. 47–56. doi: 10.1007/978-3-030-92604-5_5.

[20] S. Plappert, C. Becker, P. C. Gembarski, and R. Lachmayer, "Feasibility Evaluation of Milling Designs Using Multi-Agent Systems," *Proceedings of the Design Society*, vol. 2, pp. 763–772, May 2022, doi: 10.1017/pds.2022.78.

[21] J. Bender, S. Kehl, and J. P. Müller, "A Comparison of Agent-Based Coordination Architecture Variants for Automotive Product Change Management," in *Multiagent System Technologies*, Springer International Publishing, 2015, pp. 249–267. doi: 10.1007/978-3-319-27343-3_14.

[22] J. Palanca, A. A. Terrasa, V. Julian, and C. Carrascosa, "Spade 3: Supporting the new generation of multi-agent systems," *IEEE Access*, vol. 8, pp. 182537–182549, 2020, doi: 10.1109/ACCESS.2020.3027357.

[23] F. Bergenti, G. Caire, S. Monica, and A. Poggi, "The first twenty years of agent-based software development with JADE," *Autonomous Agents and Multi-Agent Systems*, vol. 34, no. 2, p. 36, 2020, doi: 10.1007/s10458-020-09460-z.

[24] F. Bellifemine, A. Poggi, and G. Rimassa, "Developing Multi-agent Systems with JADE," in *Intelligent Agents VII Agent Theories Architectures and Languages*, Springer Berlin Heidelberg, 2001, pp. 89–103. doi: 10.1007/3-540-44631-1_7.

[25] S. A. Deloach, M. F. Wood, and C. H. Sparkman, "Multiagent systems engineering," *International Journal of Software Engineering and Knowledge Engineering*, vol. 11, no. 3, pp. 231–258, 2001, doi: 10.1142/S0218194001000542.

[26] S. A. DeLoach, "Multiagent systems engineering of organization-based multiagent systems," in *Proceedings of the fourth international workshop on Software engineering for large-scale multi-agent systems - SELMAS '05*, 2005, p. 1. doi: 10.1145/1082960.1082967.

[27] A. T. Mathew and C. S. P. Rao, "A Novel Method of Using API to Generate Liaison Relationships from an Assembly," *Journal of Software Engineering and Applications*, vol. 03, no. 02, pp. 167–175, 2010, doi: 10.4236/jsea.2010.32021.

[28] T. AlGeddawy and H. ElMaraghy, "Reactive design methodology for product family platforms, modularity and parts integration," *CIRP Journal of Manufacturing Science and Technology*, vol. 6, no. 1, pp. 34–43, 2013, doi: 10.1016/j.cirpj.2012.08.001.

[29] P. GU and X. YAN, "CAD-directed automatic assembly sequence planning," *International Journal of Production Research*, vol. 33, no. 11, pp. 3069–3100, 1995, doi: 10.1080/00207549508904862.

[30] S. Plappert, P. C. Gembarski, and R. Lachmayer, "Product Configuration with Bayesian Network," in *Proceedings of the 9thInternational Conference on Mass Customization and Personalization –Community of Europe (MCP-CE 2020)*, 2020, pp. 184–190.

[31] S. Cass, "The top programming languages: Our latest rankings put Python on top-again - [Careers]," *IEEE Spectrum*, vol. 57, no. 8, pp. 22–22, Aug. 2020, doi: 10.1109/MSPEC.2020.9150550.

CORRESPONDENCE

Stefan Plappert, M.Eng.
Institute of Product Development
Leibniz University of Hannover,
An der Universität 1,
30823 Garbsen, Germany
plappert@ipeg.uni-hannover.de

Christian Becker, B.Sc.
Institute of Product Development
Leibniz University of Hannover,
An der Universität 1,
30823 Garbsen, Germany
c.becker@stud.uni-hannover.de

Dr.-Ing. Paul Christoph Gembarski
Institute of Product Development,
Leibniz University of Hannover,
An der Universität 1,
30823 Garbsen, Germany
gembarski@ipeg.uni-hannover.de

Prof. Dr.-Ing. Roland Lachmayer
Institute of Product Development,
Leibniz University of Hannover,
An der Universität 1,
30823 Garbsen, Germany
lachmayer@ipeg.uni-hannover.de