Reconfigurable Equiplets Operating System

A Hybrid Architecture to Combine Flexibility and Performance for Manufacturing

Daniël Telgen*, Erik Puik*, Leo van Moergestel*, Tommas Bakker*, John-Jules Meyer†
*Research Centre Technology & Innovation
HU University of Applied Sciences Utrecht
Nijenoord 1, Utrecht, The Netherlands
Email: daniel.telgen@hu.nl
†Dept. of Information and Computing Sciences
Utrecht University
Utrecht, The Netherlands
Email: J.J.C.Meyer@uu.nl

***Abstract*—The growing importance and impact of new technologies are changing many industries. This effect is especially noticeable in the manufacturing industry. This paper explores a practical implementation of a hybrid architecture for the newest generation of manufacturing systems. The papers starts with a proposition that envisions reconfigurable systems that work together autonomously to create Manufacturing as a Service (MaaS). It introduces a number of problems in this area and shows the requirements for an architecture that can be the main research platform to solve a number of these problems, including the need for safe and flexible system behaviour and the ability to reconfigure with limited interference to other systems within the manufacturing environment. The paper highlights the infrastructure and architecture itself that can support the requirements to solve the mentioned problems in the future. A concept system named Grid Manufacturing is then introduced that shows both the hardware and software systems to handle the challenges. The paper then moves towards the design of the architecture and introduces all systems involved, including the specific hardware platforms that will be controlled by the software platform called REXOS (Reconfigurable EQuipletS Operating System). The design choices are provided that show why it has become a hybrid platform that uses Java Agent Development Framework (JADE) and Robot Operating System (ROS). Finally, to validate REXOS, the performance is measured and discussed, which shows that REXOS can be used as a practical basis for more specific research for robust autonomous reconfigurable systems and application in industry 4.0. This paper shows practical examples of how to successfully combine several technologies that are meant to lead to a faster adoption and a better business case for autonomous and reconfigurable systems in industry.**

Index Terms*— *Flexible Manufacturing Systems; Multi-agent systems; Autonomous agents; Reconfigurable architectures.

I. Introduction

Computers are continuously changing industry, and whereas in the past this has created opportunities for improved logistics and overview like SCADA (Supervisory Control and Data Acquisition) it is now starting to change the industry itself. This paper is based on our original work [1], which was presented at the INTELLI conference. Computers are not only supporting existing mechatronical systems, but are fundamentally changing the processes in how they are used. This is the basis for many changes in the field of manufacturing that are known under a variety of names, including 'Smart Industry', 'Agile Manufacturing', 'Industry 4.0', and Cyberphysical systems.

With the use of more computing power in manufacturing systems it is easier to integrate "intelligent", i.e., dynamic behaviour by using microsystems, i.e., sensors and actuators, together with advanced software to interpret the sensed data and act accordingly. This creates a robotic system that can dynamically interact and act with its environment. However, the dynamic behaviour of this systems can also increase complexity [2]. Therefore, it is important to create a balanced architecture that, on the one hand, has a high performance to control the hardware, i.e., dynamically interpret and interact with its environment in real-time and, on the other hand, is not so complex that it will be difficult to use and shows unexpected, i.e., unsafe or unwanted, behaviour.

From a hardware perspective there are also several changes that are occurring, as the costs for developing such complex systems have to be earned back and, therefore, it is necessary to reuse systems as often as possible. This has led to the creation of modular systems, where the modules can be used as building blocks that can be combined for a specific purpose. If the modular systems are standardized and well-documented they can also be easily reconfigured to provide a range of options. Since a well-defined module could be seen as a black box, it also lowers complexity of the overall system, since functionality can be abstracted on a higher level [2].

The structure of the paper is as follows: The next section will give further information for the motivation and how the research was conducted. Section III will provide more insight overview of current technologies and paradigms that are involved when researching smart technologies for manufacturing. After this overview the problem will be investigated more closely in Section IV, which will result in a number of specific research questions. Section V will then continue with the chosen approach and provide the basis for the requirements in Section VI. The requirements will then lead as input for an overview of several technologies and concepts in

Section VII. Section VIII will then describe the design choices that have been made. The design triggered the development of a software architecture and hardware platform thas been implemented and described in Section IX. Finally, the platform will be evaluated with several performance results in Section X. These results and critical points of the platform will be discussed and concluded in the final Section XI.

II. Research Motivation

Due to the obvious advantages of using smart systems, many companies and research groups are experimenting and conducting Research and Development with several projects. However, the success in industry itself has so far been limited [3]. This is due to a number of reasons, including the complexity of such systems with the high initial investment costs, the expertise needed to create such systems, and the difficulties to create a business case for dynamic systems. Schild and Bussmann show this in a case where a successful self-organized manufacturing system was setup at Daimler-Chrysler. They state that while the system was successfully implemented it was discontinued because a technical advantage is not always a measurable economical advantage [4]. When continuing this research in this field, therefore, it is important to take this aspect into account. In this research, this is done by taking into account several techniques and requirements that will lower the hardware costs, and also by targeting the manufacturing means at an even wider variety.

Besides industrial parts there is also a change in retail industry. Mass customization is slowly beginning to become a standard. Wind even introduces the concept of 'customerization'. This is a new business strategy that combines personal marketing strategies, with mass customization. Figure 1 shows that customerization comes through a combination of standardization, personalization and mass customization. Wind mentions that for successful customerization requires the integration of multiple processes, including operations and R&D [5]. He also states that increasing the digital content of everything the company does is one of two critical aspects that should be considered.

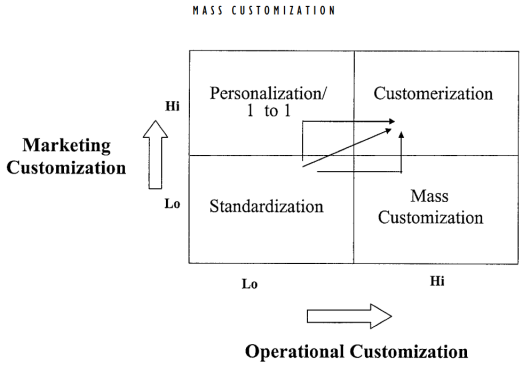

Figure 1. Combining personalisation and mass customization towards a new business strategy [5].

This paper will take a step further into the field of flexible reconfigurable manufacturing by laying the foundation for a flexible and reconfigurable architecture for high-mix, low-volume manufacturing that could enable customerization.

A. Research Approach

As observed in the introduction, technological advances and changes in industry are starting to have an increasing impact on the manufacturing industry. However, a true paradigm shift has not yet occurred, likely because the maturity and efficiency of these new technologies has yet to be proven. As Leitão states in the conclusion of his survey: 'The challenge is thus to develop innovative, agile and reconfigurable architectures for distributed manufacturing control systems, using emergent paradigms and technologies that can provide the answer to those requirements' [3]. Leitão also identifies some specific issues in this field, including:

1) The need for mature and proven technology - the majority uses laboratorial control applications without the need of physical devices [6].
2) Reconfigurability mechanisms - what architecture will support the society of distributed entities? [3].
3) Development-related aspects - current platforms have limited scalability and robustness [7].
4) Prediction in disturbance handling systems - the integration of prediction mechanisms with identification and recovery of disturbances that can prevent these problems [3].

The research in this paper responds to these factors by including the hardware systems and the practical implementation to show a feasible system that can be used for industry. Together with the hardware, several emulators and simulators will also be developed to test various aspects of the proposed system. Besides using the simulation for benchmarking, it will also be used to predict problems like collisions in the changing and possibly chaotic environment.

B. Hypothesis

The hypothesis is that new (software) technologies could be used to create a new flexible manufacturing paradigm that is (cost-) efficient and stable. The flexibility will have to provide for a much shorter time to market and an increased variety in products. The challenge for this hypothesis is to keep the complexity (and, therefore, the practical applicability) of the smart and flexible approach under control. This has to be proven by developing a proof of concept that shows the abilities, performance and stability of such an 'agile' architecture. To prove the feasibility and practical implementation the system will be fully built including low-cost hardware designed for this purpose.

C. Research Methodology

The study will be largely based on applied research. An architecture and several hardware and software systems will be developed to be used as a proof of concept. This system will be the basis of future research and will be combined with

several emulators to be tested in a large variety of cases. The study will combine quantitative and qualitative elements:

- performance: quantitative research based on empirical data using experiments from live proof of concepts and simulators.
- abilities: qualitative research - comparing designs and architecture performance based on correlation.
- processes: qualitative comparison based on cases with (partly) quantitative data.

The research project will start with a 'top-down' approach, i.e., deductive research, that will test the proposition set in the next subsection. However, the platform will be created 'bottom-up' and as such is hoped to be a basis for inductive research to create new insights in the future for the use of reconfigurable systems and application of self-organizing manufacturing platforms.

D. Propositions

To focus the research, a concept has been created for a manufacturing platform that is based on several basic principles:

1) A range of products can, in principle, be dynamically built on demand - i.e., the machines offer a (wide) range of services where any product that can be made with these services can be manufactured.
2) Each system (with its own purpose) is autonomous - i.e., both products as manufacturing systems have no strong dependencies and can act autonomously.
3) Hardware should be reconfigurable. - i.e., both hardware and software modules within a system should be able to be changed with limited downtime. In the case of the software, compiling code should be required for a reconfigure action.
4) Machines should be low-cost and single-purpose. - i.e., the flexibility that is offered should be done with limited investment costs to guarantee experimental use and a valid business case.
5) System behaviour should be transparent and safe. - The flexibility and dynamic behaviour of the system must be guaranteed not to lead to a high risk of use.

The idea is to create a range of products that can be built on demand, i.e., all products can be manufactured ad hoc so long as the parts and required services to assemble them are available in the manufacturing systems or 'grid'.

The principles also focus on limiting complexity by creating a minimum amount of interdependence between systems. This results in a concept that has been called 'grid manufacturing', where each manufacturing system delivers a service to a product. Since products and the manufacturing systems have their own purpose, i.e., the machine delivers a service, the product wishes to be produced, they are both autonomous and will work together dynamically. Hence, the system will not be a 'production line', since the need for services will depend on the specific product demand.

Since the products will be manufactured dynamically without any specific programming it is important that they are still produced safely and according to the specifications. The products will schedule themselves in negotiation with the manufacturing systems and, therefore, it is unknown which services are required. This makes it difficult to establish the demand, and therefore, the chances are that some manufacturing systems have a higher load than others.

III. General Research Overview

This section provides a theoretical framework and overview for the current research. It discusses a number of paradigms and technologies that could be applicable for the current study.

A. Manufacturing Paradigms

Many paradigms have emerged that have been of influence in the manufacturing industry, the most influential have been the three main paradigms, see Figure 2.

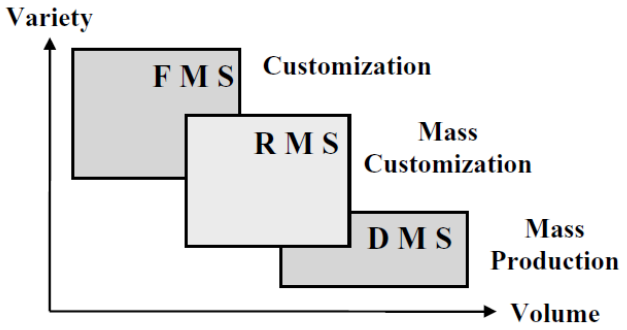

Figure 2. The three main manufacturing paradigms [8].

Dedicated Manufacturing Systems (DMS) are the classic way of mass-production. In this paradigm, all manufacturing systems are developed for a specific single-purpose goal with limited to no dynamic properties. This creates a cost-efficient system for producing high volumes of a single product over a longer timespan [9]. The requirements stay the same, therefore, DMS are known to have a high performance and limited initial costs.

Flexible Manufacturing Systems (FMS) offers dynamic behaviour, which it uses to react to changes. Usually, Flexible manufacturing systems offer a single purpose where the machine has the ability to perform one action. This ability is combined with sensors like a vision or dynamic routing system so it can adapt certain parameters, e.g., the position of a product. This creates more flexibility at the disadvantage of complexity and cost to the initial implementations.

Reconfigurable Manufacturing Systems (RMS) are unique in the perspective that the functionality of the system itself can be adapted [10]. As shown in Figure 2, they are positioned between FMS and DMS. However, since RMS provide a way to integrate change within a system it is expected that they will become more flexible over time [11]. ElMaraghy notes that the key characteristics of RMS include modularity, integrability, scalability, convertibility, and diagnosability [12]. These characteristics have to be taken

into account when defining the requirements for a flexible manufacturing architecture.

Besides the main three manufacturing paradigms, there are also many other paradigms and methods of interest.

Agile Manufacturing (AM) is characterised by the integration of customer and supplier for both product design, as manufacturing, marketing, and support services [13]. An agile manufacturing environment creates processes, tools, and a knowledge base to enable the organisation to respond quickly to the customer needs and market changes whilst still controlling costs and quality [14].

Manufacturing As a Service (MAAS) is a concept to deliver customizable and on-demand manufacturing. This is closely related to manufacturing clouds, where factories and their IT infrastructure are interconnected to create an infrastructure where ad-hoc products are being made [15].

Holonic Manufacturing is seen as an alternative to hierarchical management of manufacturing systems. It focuses on modularization and 'plug and play' capabilities when developing or using manufacturing systems [16]. Holonic manufacturing systems are often implemented using Multi Agent Systems [17], which will be discussed later in this paper.

Noteworthy are also a number of concepts that are closely related to this research:

Smart Industry and **Industry 4.0** are often used as a concept that combines industrial systems with properties like the 'internet of things' and other cloud related services. They depict the vision of smart factories that consist of **cyber-physical systems**. Cyber-physical systems (CPS) are collaborating computational (virtual) elements that control physical entities. i.e., basically a virtual entity with its virtual world image that uses its own world image to control and interpret the physical world and operate a physical counterpart in this environment. This virtual entity is commonly an embedded system within the system that it controls.

IV. Problem Description

While many paradigms and technologies show promise they are not yet considered mainstream in industry. As mentioned before, the initial investment costs and complexity are factors for this problem. However, while business is important, this particular paper will first focus on the technological aspects that are required to create a basic software platform as a basis for further research and test cases for industry. For this platform a number of main challenges have been identified:

1) Architectural performance / intelligence gap - The platform should both be able to show 'intelligent' behaviour and have real-time performance to control the hardware.
2) Abstract Services - To use manufacturing as a service and limit complexity the hardware can not be 'known' by the product.
3) Reconfigurable systems - It should be possible to quickly adapt the hardware and reconfigure the system by changing its hardware modules.
4) System Behaviour - While systems should be autonomous and modular and work in a dynamic 'chaotic' environment their behaviour should also be predictable and 'safe'.

The first problem will be the main focus of this paper. How can you create an architecture that combines the dynamic behaviour and flexibility for high-level functionality, i.e., understand its environment and cooperate with other systems in the grid, and low-level functionality, i.e., high performance hardware control and algorithms. These different functionalities are based on different behaviour and therefore have different requirements. High-level functionality is based on abstract cognitive processes that use networked data and slow heuristic processes. Low-level processes are based on strict rule based systems that have a direct impact on the actuators. As such they are usually written in native code using real-time systems. While native code could grant a higher performance it is also more difficult to develop. Additionally, it is important that the high-level functionality will have no performance impact on the low-level systems.

The next problems will be taken into account and seen as preconditions for the proposed architecture in this paper. Their impact on the architecture will be discussed. However, the solutions and research conducted to solve these specific problems are not within the scope of this paper itself and will be published in detail in future work.

The second problem focuses on the use of the manufacturing systems. Since the grid manufacturing concept asks for autonomous systems the product is not aware of which manufacturing system will produce it beforehand. As a result both product and the manufacturing system are not designed specifically for each other. Hence, to be able to use the service for a product they should be able to interface with and understand each other. This asks for an ontology that both product and manufacturing system can use. The architecture should take into account which services and limits it can provide and match these to the requirements of the product.

The third problem focuses on the reconfigurable aspect of the systems. Since demand can change it is important to adapt the systems to the (possibly new) demand. To create maximum flexibility the system should be easy to adapt and if possible automatically update its use and services so they can become unavailable to the products in the grid.

The last and fourth problem focuses on the system behaviour. Since products are unknown and manufacturing hardware can be reconfigured there is a large dynamic in a grid. Hence, it is difficult to define its exact behaviour. To be sure of the exact manufacturing specifications and safety aspects it is required to create specifications and procedures for action that a hardware module can perform. A system should be created that defines the behaviour and describes how it will act during diverse situations like starting up/shutting down or errors.

A. Research Questions

The research question will focus on the main problem of this paper, the creation and specification for a flexible software architecture that will provide both performance for low-level, and flexibility for high-level functionality.

1) What technologies are available for use in the proposed concept for grid manufacturing?
2) What requirements are necessary for a software architecture for smart industry?
3) How can low-level performance and high-level flexibility be combined?
4) Can such an architecture be scalable?
5) Is the proposed concept feasible for near future use in industry?

V. Context - The concept

The main concept is based on the philosophy of cyber-physical systems. A grid will consist of three main types of systems: several logistic services (including autonomous transport systems), autonomous manufacturing systems, and lastly the products that will also be cyberphysical systems. Figure 5 shows the concept of a grid with autonomous (cyberphysical) systems. The reconfigurable manufacturing systems are called 'equiplets'.

Classic manufacturing is based on a Line Cell Module Device (LCMD) model as shown in Figure 3. The model represents a modular manufacturing process based on 4 hierarchical levels. The line is literally a 'manufacturing line' that is made up of a number of cells where a specific job is performed. A Cell commonly uses multiple modules which perform specific actions, e.g., pick & place. The module can decomposed even further into devices, e.g., a pick & place module will likely consist of several sensors and actuators, each sensor and actuator can be seen as a device. The LCMD model is optimised for cost efficient manufacturing of products that are made in high quantities. Since this is a linear model where products are made in a line, any change at any level will influence the entire manufacturing process. Hence, the product as well as the manufactured process will have to be matured completely before actual mass manufacturing can start.

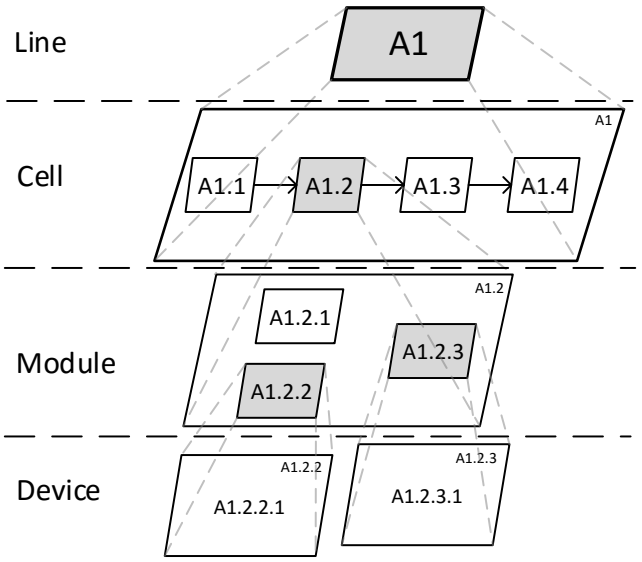

Figure 3. Classic Line Cell Module Device (LCMD) structure.

The concept of Grid manufacturing provides the opportunity to dynamically adapt both product and equipment at any level where the overall impact will be limited as much as possible. This is performed by autonomous reconfigurable systems that provide generic services that products can use. All systems in the grid should cooperate to become self-organizing. Because of the reconfigurable aspect of these systems they are named *equiplets*. Equiplets are not arranged in a line, but in a grid to emphasize that they can be used sequentially based on the current dynamic demand, i.e., dynamic in the sense that different products can be made at any time using equiplets in any order using (soft) real-time negotiation and scheduling to plan how a possibly unique product will be manufactured. Figure 4 shows a rendering of a grid with 12 equiplets, where every equiplet can have a different configuration to provide a variety of services that are required for the manufacturing process. Note that a grid does not require to have any specific form; depending on the demand they can be placed in any relative position based on the local logistic setup of the factory.

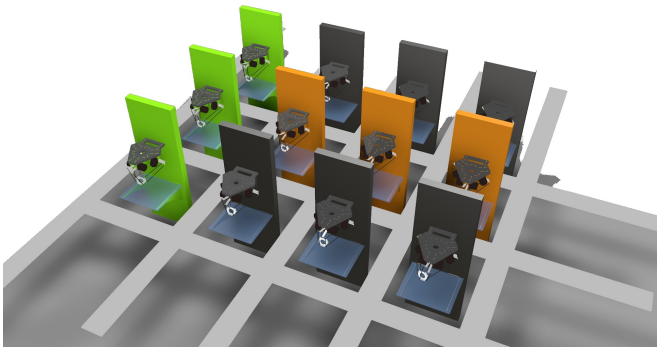

Figure 4. Example of a Grid Structure.

Equiplets provide capabilities that are based on the configuration of the specific modules that are installed. In this context, reconfiguration is defined as adding/removing/changing modules within the equiplet so as to change its capabilities. This includes both the physical change as the adaptation and configuration of the software to control the equiplet.

In contrast to LCMD, this architecture for Grid Manufacturing is called the Grid Equiplet Module (GEM) Architecture, as shown in Figure 6. With GEM the systems are loosely coupled, the Grid layer provides services to the autonomous equiplets. Modules are commonly designed as Components of the Shelf (COTS).

Besides delivering flexibility the concept also introduces a manner to bring the product designers and production experts closer together. In the past, lines were made specifically for one product and as such it would come at a high cost to take a working line off-line to create a prototype for a new product. Since grids can dynamically handle various products in parallel a product designer is able to use the same manufacturing equipment that is used for the final manufacturing to create prototypes and test the production phase. This shortens the time-to-market and lowers costs.

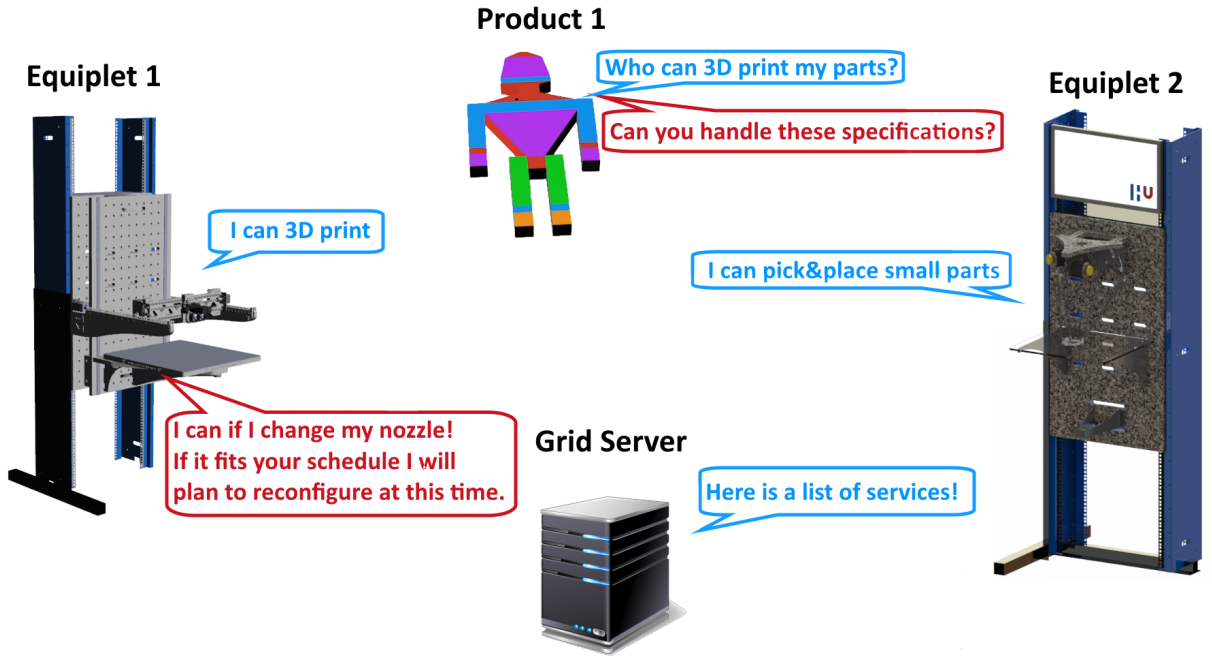

Figure 5. The simplified concept of grid manufacturing.

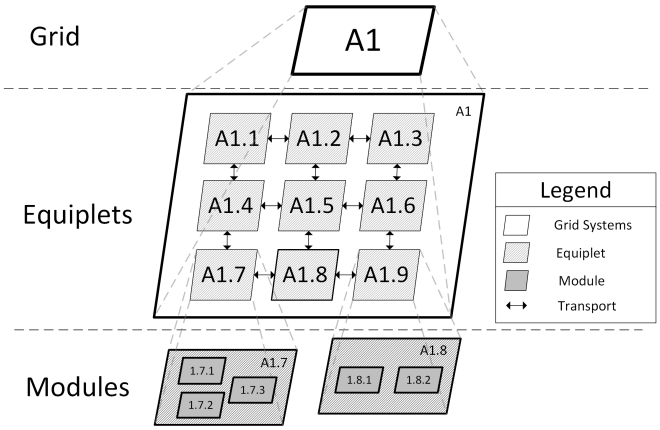

Figure 6. GEM Architecture.

To further work out the required platform and analyse the success of this concept the requirements should first be discussed.

VI. REQUIREMENTS

The requirements of grid manufacturing will be split into different parts. First, the preconditions and functional requirements of all levels within the GEM architecture. Then the products themselves, and finally the processes that are required to be active within the grid to fulfil its function.

Since the design of complex systems as used in grid manufacturing is challenging, the requirements are loosely based on the Axiomatic Design methodology developed by MIT [18]. Axiomatic design uses design principles or axioms, i.e., premises or starting points for reasoning. In Axiomatic design the characteristics needs are translated into four domains:

- Customer Domain - Customer Attributes (CA)
- Functional Domain - Functional Requirements (FR)
- Physical Domain - Design Parameters (DP)
- Process Domain - Process Variables (PV)

A. Functional Requirements

The functional requirements in Axiomatic Design are given by answering the question, 'what should the system do?'. This is placed in the scope of all software systems for multiple autonomous reconfigurable manufacturing machines. During the decomposition phase this clustered to a three level decomposition for the manufacturing system and a fourth for the product entity that represents the product:

- The Grid - A decentralised system where Equiplets and Products cooperate.
- Equiplet - An autonomous modular reconfigurable single-service low-cost manufacturing machine.
- Module - A hardware module that provides one specific function within an equiplet.
- Product - The cyberphysical entity that will represent the product.

B. Grid Level

The grid should be able to:

- GFR1 offer services to a variety of products.
- GFR2 validate and assess its own efficiency.
- GFR3 adapt (remove or add) services / equiplets with limited to no interference to other products.

- GFR4 provide for product transport dynamically between each equiplet.

C. Equiplet Level

An equiplet should be able to:

- EFR1 provide a specific service to a product.
- EFR2 be reconfigured (adding or removing of modules - to provide a different service).
- EFR3 work autonomous, i.e., it has no strict dependencies with other equiplets or create interferences for other equiplets.
- EFR4 automatically adapt its software when modules are added or removed.
  - EFR4A let its new service (capability) known to the grid.
  - EFR4B update its system behaviour and safety software.
- EFR5 translate abstract instruction from a product and translate it to instructions for its own specific hardware modules.
- EFR6 efficiently control the hardware in real-time.

D. Product representation entity

A product representation (in its manufacturing phase) should be able to:

- PFR1 coordinate its own production.
- PFR2 know of which parts it requires to be completed.
- PFR3 know which (abstract) services it requires to be assembled (production steps).
- PFR4 determine which services are available.
- PFR5 communicate with equiplets to determine if they can perform a production step.
- PFR6 create a (viable) schedule on how it will be produced.
- PFR5 log its production/assembly history.

E. Module Level

A module should be able to:

- MFR1 know its own characteristics.
- MFR2 accept and perform instructions from the equiplet.

VII. Technology comparison

To create a platform for the proposed concepts let us first investigate some current technology:

Agent Technology The word Agent comes from the Latin word *agere*, which means: to act. Software agents, as shown in Figure 7 are entities that have their own interpretation of their environment on which they act autonomously. Hence, we uphold the definition of Wooldrige and Jennings: 'An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives'[19].

Therefore, some consider agents as 'objects with an attitude', since unlike an object an agent has control over its own behaviour. Agents can also be seen as a higher abstraction of objects, which makes them ideally suited to deal within complex dynamic environments. Paolucci and Sacile noted that they can create a flexible, scalable and reliable production system [20].

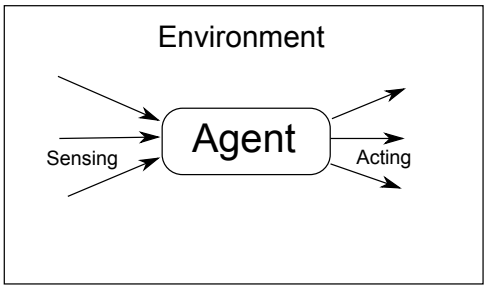

Figure 7. An autonomous agent in its environment.

Agents can be split in two main types:

1) Reactive agents
2) Reasoning agents

Figure 8 shows an example of a standard reactive agent cycle that perceives its environment through sensors, interprets it according to standard rules, and chooses an action accordingly and acts using actuators to change something in the environment.

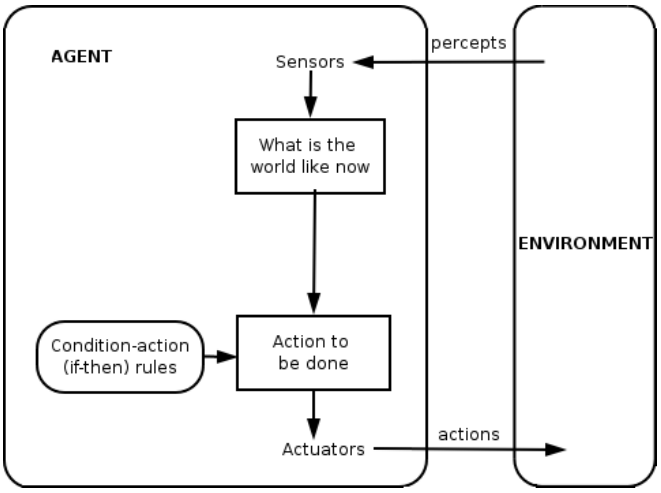

Figure 8. Simple reflex of an Intelligent Agent.

Reasoning agents exist in multiple types, the best known of which being the belief-desire-intention (BDI) agent, see Figure 9. The BDI agent uses the philosophy of Dennett and Bratman [21], [22]. The BDI agent uses its senses to build a set of beliefs, where its desires are a set of accomplishments that the agent wants to achieve. The BDI agent can choose desires that it wants to actively try to achieve, these are its goals. It then commits to a goal to make it into an intention, activating a plan that consist of actions that it will take to achieve its goal and thus satisfying its desire. Ideally, BDI agent uses the following sequence to achieve this [23]:

1) initialize-state
2) repeat
   a) options: option-generator(event-queue)

b) selected-options: deliberate(options)
c) update-intentions(selected-options)
d) execute()
e) get-new-external-events()
f) drop-unsuccessful-attitudes()
g) drop-impossible-attitudes()

3) end repeat.

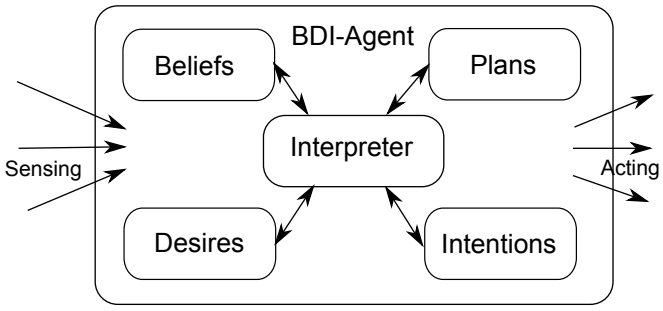

Figure 9. Beliefs Desire Intention Agent [24].

Figure 10 shows the concept of **Multi Agent Systems** (MAS), where multiple environments communicate and work within an environment. Agents interact and can cooperate or negotiate to achieve common goals. Either agent can have a specific role within a MAS and can interact with the other agents using specific permissions and responsibilities.

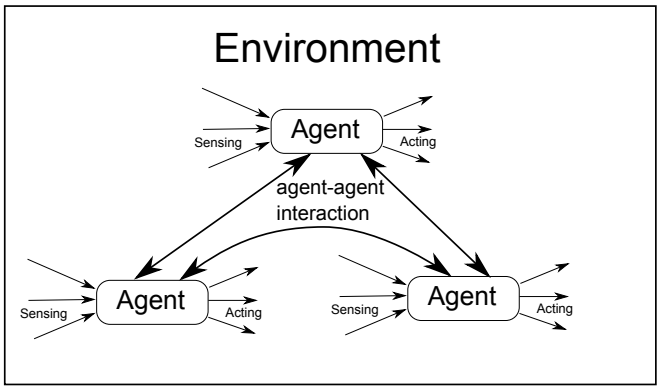

Figure 10. Multiple agents forming a Multi Agent System (MAS).

MAS can also be associated with **Environment Programming**. Environment Programming is seen as an abstraction where the environment is seen from the agent perspective. Objects in the environment that the agent interacts with are seen as programming models and are named 'artefacts'. This creates an extra abstraction where objects keep their abstraction layer and can be used effectively by the agents [25].

VIII. Design

The platform that has been developed is called REXOS, which stands for Reconfigurable EQuipletS Operating System. For the design it was necessary to consider an important rule of Axiomatic Design:

TABLE I. Design Matrix that shows the relationship between FRs and DPs

	Hardware Platform	Intelligent Platform
High Performance	x	
Intelligent behaviour		x

- Axiom 1: The independence Axiom - Maintain the independence of the functional requirements
- Axiom 2: the information Axiom - Minimise the information content of the design

These axioms show why cooperating autonomous systems like MAS lower the complexity of a design, since many functional systems can be isolated in a single entity. However, this is not true for equiplets. Hence, they require specific attention in the design.

The domains are represented as vectors that are interrelated by design matrices.

$$\{FR\} = [A] \cdot \{DP\} \tag{1}$$

Where {FR} is the vector of the Functional Requirements (What should it do), {DP} the vector for the Design Parameters (What can satisfy the FR), and [A] is the design matrix that hold the relationships between these two vectors:

$$\begin{bmatrix} FR1 \\ FR2 \end{bmatrix} = \begin{bmatrix} A11 & A12 \\ A21 & A22 \end{bmatrix} \times \begin{bmatrix} DP1 \\ DP2 \end{bmatrix} \tag{2}$$

As mentioned in the Problem Section, the architectural design has to be able to have high performance and be able to have intelligence behaviour. This is a common problem that is recognized in recent literature [26]. Based on the methodology of axiomatic design this urges us to think of how to decouple these properties.

Table I shows the relationship between the requirements and the solution. In this case, the requirements are decoupled through the creation of a *hybrid architecture* where multiple platforms are combined to potentially yield the best of two worlds [27]. This in contrast to a system where one platform is used where these requirements should be combined.

Hence, for the design, it is important to analyse a number of platforms and research how they can interface without becoming coupled. The next section will review a number of platforms and technologies that could become the basis of REXOS.

A. Choice of Technology

Grid Manufacturing can be seen as a complex system where many autonomous systems have to interact. This is one of the reasons why it is important to use autonomous entities to become as flexible as possible without creating too many interdependencies that increase the overall complexity of the system. As shown in Figure 11, a multi-agent system fits this requirement in that it offers a level of abstraction and limits the sphere of influence for an entity.

The use of a MAS seems a good option to choose as a basis for REXOS, if we investigate this further we can also look at the characteristics of a manufacturing environment [29]:

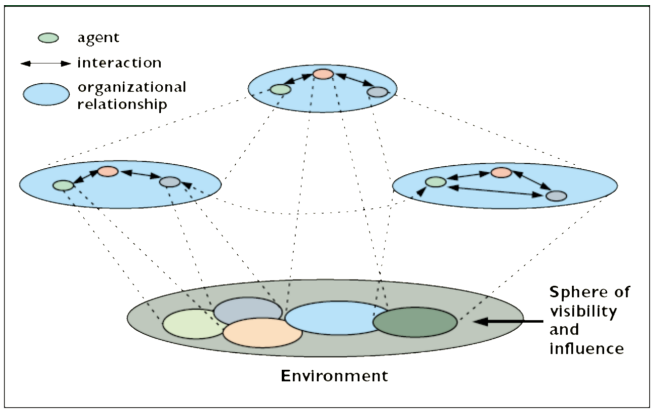

Figure 11. The sphere of influence within a MAS [28].

1) Autonomy
2) Cooperative
3) Communicating
4) Reactive
5) Pro-active

Together with the set requirements for equiplets and the grid, these fit perfectly into the concept of grid manufacturing. Besides the manufacturing processes themselves agents provide many more possibilities that are out of the scope for this paper, e.g., a product agent that stays with the product to analyse its behavior and offer problem solutions whenever possible [30]. The agent that can represent hardware could also be utilized to analyse efficiency and learn from the behaviour to optimize schedule times and other logistic matters.

B. Choice of platforms

Even though we choose to use MAS as a basis for REXOS, this does not fulfil all requirements that we require for grid manufacturing. MAS will provide a dynamic decision platform that will represent all systems. However, it is normally not suited for direct real-time control of hardware. Hence, it is important to investigate which platform could fulfil this requirement and to research how these platforms could be successfully combined. To approach this task lets first look at a number of agent platforms.

C. Agent platforms

The platform that is used for REXOS had to meet certain requirements as mentioned in the problem description. Several attributes also have to be satisfied, which are part of the Customer Domain:

1) The platform needs to be scalable.
2) For flexibility the platform needs to be able to change or add new agents during runtime.
3) The platform needs to be mature (for industrial application).
4) Performance needs to be sufficient to handle grid-wide logistics.
5) The platform should preferably be open source, but also applicable for industrial use with propriety sources.

Several agent platforms have been investigated:

- 2APL [31]
- JADE [32]
- Jadex [33]
- Madkit
- Jack
- Jason

The choice for the agent platform has become Java Agent Development Framework (JADE), since in JADE agents can migrate, terminate and start in runtime, also JADE has been widely adopted and has an active community. While JADE has no direct support for BDI agents it can be extended to add this when necessary in the future. Currently, the architecture does not force the use of the BDI. JADE is also compliant with the Interoperable intelligent multi-agent systems specifications standard FIPA. Which makes it possible to easily extend the MAS with other FIPA compliant systemsFoundation for Intelligent Physical Agents.

D. Diverse platforms

Besides the agent platform, there is a need to combine it with other platforms to control the hardware and satisfy the Customer Attributes and Functional requirements.

Robot Operating System (ROS) is a software framework that provides services, tools and libraries for robots [34]. The framework has extensive support for a variety of sensors and actuators and offers hardware abstraction and low-level device control. ROS is free and open-source and uses nodes as software modules that communicate with messages. Nodes can be started and stopped in runtime, making it possible to adapt software modules at any time. ROS has been created to create general purpose robot software that is robust.

Robot Operating System (ROS 2.0) is currently under development and is being created to overcome some limitations of ROS 1.0, including real-time requirements and use for multiple robots.

MongoDB Due to the diversity and flexibility of the grid it is difficult to define all schemas that relational databases use. MongoDB uses dynamic schemas, is cross-platform and has a document-oriented database. Hence, MongoDB can be used as a blackboard between platforms.

OpenCV Open Computer Vision can easily be integrated with ROS, it is released under the BSD license and can be used on multiple platforms. It has a focus on real-time applications and has been proven in many projects. Hence, it is logical to choose for the OpenCV library to integrated OpenCV in REXOS. The computer vision is used to identify and localise parts within the working space of the equiplet and is used for other logistic processes necessary for configuration and calibration of the systems, e.g., identification of a new gripper.

E. REXOS

REXOS has been developed using JADE and ROS 1. The main reason for using ROS is its proven use in many projects and the experience from other projects. The combination of

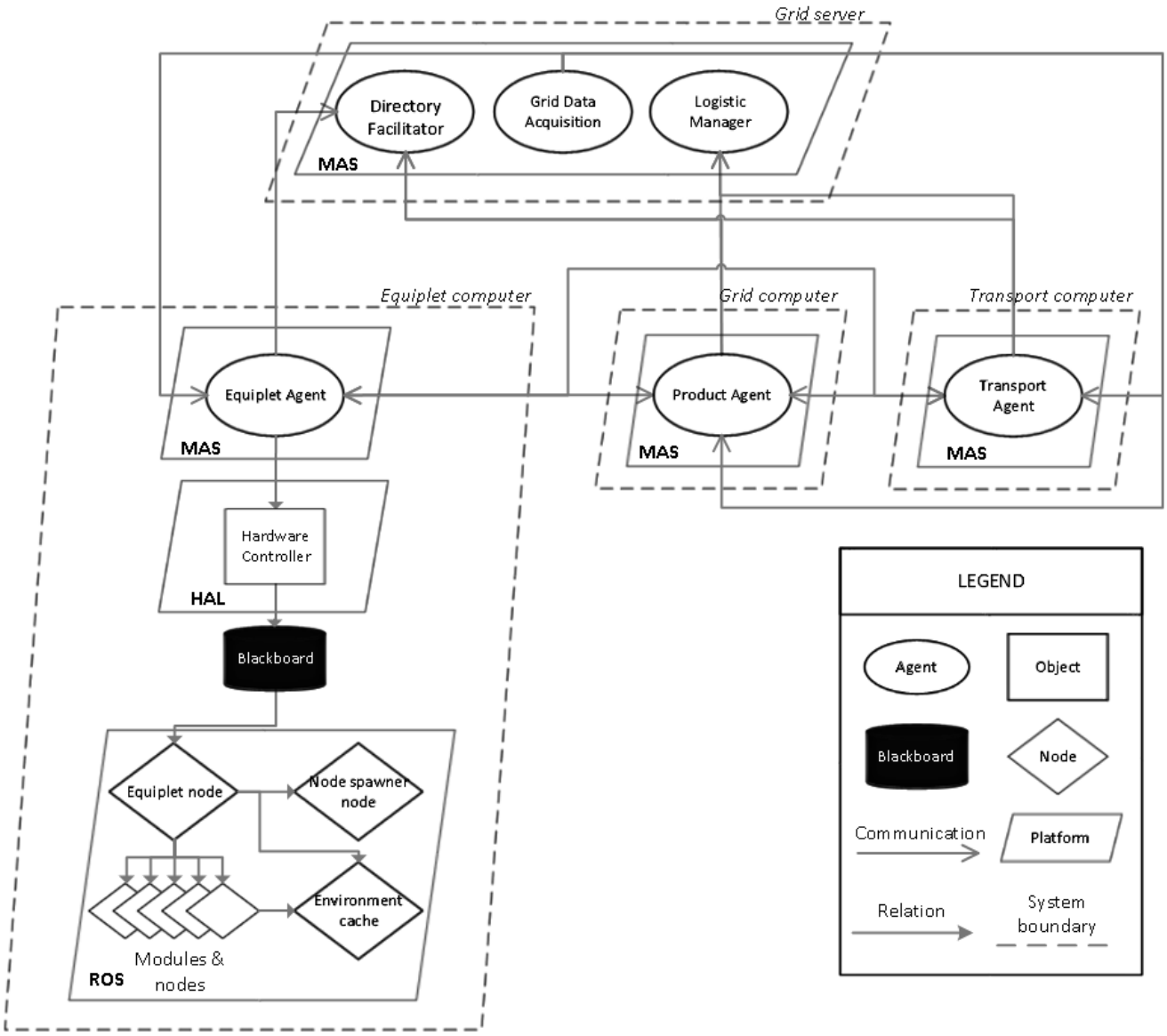

Figure 12. High-level software design of REXOS.

JADE and ROS results in the architecture as shown in Figure 12, the high-level design of REXOS.

REXOS is a distributed multi-platform system and as such will run on a number of computers. The basic logistics will run on a grid server that provides a Directory Facilitator (DF), Grid Data Acquisition System and Logistic manager. The DF can be seen as a yellow page service that knows which equiplets are active and what services they provide for the products. The Data acquisition will be used for statistical and Enterprise Resource Planning (ERP). The Logistic manager is mainly meant for transportation within the grid.

Products will be created dynamically, and when they are created they will usually be created by an application and then be moved to the grid server where it will be produced. If the product has an embedded computer the product agent will be moved to the product after it has been completed. However, it might also exist in the cloud. This way the product agent can be of value throughout the entire life-cycle of the product. This way it can provide a number of services for the owner and others who use it, e.g., manuals, repair or recycle information [35].

Transport agents are responsible for the transportation of the device. Depending on the implementation this could be done in a number of ways, including Autonomous Ground Vehicles (AGV) or with (multi-directional) conveyor belts.

The equiplet will be the main system in a grid and will have a number of main platforms that each consist of one or more entities:

1) The equiplet agent
2) The Hardware Abstraction Layer (HAL)
3) The ROS layer

All these platforms will commonly reside on one computer

and are embedded within the equiplet. The equiplet agent will represent the equiplet hardware and interact with the grid and the products. It will also deal with scheduling and determine its capabilities based on its configuration. When a product arrives on schedule to be manufactured it will send its product steps [29]to the equiplet agent that will forward it to the Hardware Abstraction Layer (HAL). The HAL is capable to interpret the steps and translate them to specific instruction known as 'hardware steps' that will be send to the ROS layer to be executed.

The ROS layer consists of an equiplet node and at least one node per module that represents the hardware module. It also consists of a spawner node that is able to start new nodes when modules are reconfigured. The equiplet node will receive instructions from the HAL. The ROS layer uses an environment cache that represents the physical dynamic environment. Information that the environment cache holds is, for example, the position of products that are perceived by a computer vision or external system.

The interface between the different platforms is essential for a successful hybrid architecture. While Figure 12 shows a blackboard, other implementations have been developed and will be discussed in the next section.

IX. Implementation

For this paper, the basic architecture of REXOS is being researched. Therefore, the main focus will be on the infrastructure and platforms that are required for grid manufacturing. As mentioned before in the introduction, the hardware is an important aspect to prove the feasibility of the concept. As such, this section will show the implementation of the software platforms, the interfaces, but also give an overview of the hardware that is used.

A. middleware

Figure 13 shows the implementation of an example of the JADE platform for grid manufacturing, which consists of two equiplets and a grid server. JADE uses a main container that can be connected to remote containers (which are in the other equiplets). The main container holds a container table (CT) and two special agents, named the Agent Management Service (AMS) and the Directory Facilitator (DF). JADE has the ability to replicate or restore the main container to remain fully operational in case of a failure.

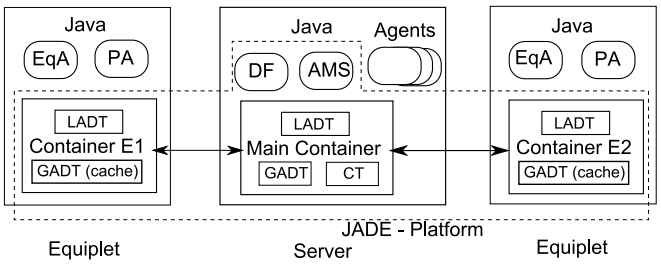

Figure 13. The Java Agent Development Platform.

In every container there is a Global agent descriptor table (GADT) that registers all the agents in the platform, including their status and location, and a local descriptor table (LADT). The GADT in the remote containers will be used for caching.

EqA and PA are the Equiplet and Product agent who will represent a specific product or equiplet.

B. Grid

The grid provides logistic functionality, which the autonomous equiplets can use. Based on the architecture shown in Figure 12 it is standard that the GRID services run on a separate server. However, since the software runs on a standard linux system and the JADE environment can be moved or distributed in any way, the GRID functionality can be run on any computer within the network. As such it is possible to start it on a computer within an equiplet. This creates the ability to quickly setup the functionality of an equiplet without requiring a complete infrastructure.

Transport [36], [37], [38] and other logistic systems like scheduling [39] are discussed in separate research and are considered out of scope for this paper.

C. Equiplets

The equiplet and its modules are specifically designed with grid manufacturing in mind. An equiplet consists of a rigid base with standard mounting points to attach modules. A standard equiplet is commonly used for assembly actions and as such typically uses 4 modules to be attached, a manipulator, gripper, vision system and a working plane. A standard equiplet stands on a rails to be easily moved and holds a standard on-board PC. Equiplets and a number of modules has been developed and tested, Figure 14 shows a demo setup of 2 equiplets configured with a pick and place setup.

Figure 14. Equiplet demo setup.

The REXOS architecture is based upon different technologies, the C++ based ROS and the JAVA based JADE platform. Therefore, the interface between these two is an important aspect for stability, performance and, therefore, scalability

issues. Due to the importance three different implementations have been developed:

1) Blackboard
2) ROS bridge
3) ROS Java

The *blackboard* implementation (see Figure 15) uses a MongoDB database server and multiple MongoDB database clients. The HAL and ROS components each have a client, connected to the server. By enabling the replication feature of MongoDB (usually used to keep the databases of multiple servers synchronised), the server generates an operation log, which logs all databases and collections on the server. The clients listen to this operation log using a tailable cursor. This enables the clients to communicate with each other via the server without having to periodically query the server.

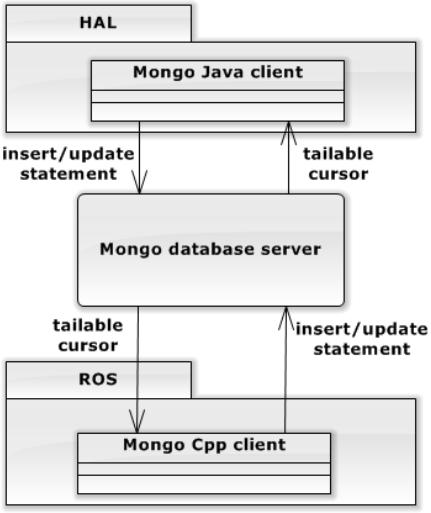

Figure 15. ROS HAL interface using a blackboard.

The *ROS bridge* implementation (see Figure 16) uses a ROS node, which acts as proxy between the HAL and ROS components. The bridge is written in Python and is designed for flexible integration of ROS in other non-C++ systems. The bridge acts as a websocket server to the outside (for REXOS this is the HAL component) and acts as a standard ROS node to the inside (for REXOS this is the ROS component). Because the bridge acts as a standard ROS node, the ROS component of REXOS can use standard ROS communication methods and messages, reducing the complexity of the interface.

The *ROS Java* implementation (see Figure 17) uses the rosjava_core library to communicate between the HAL and ROS components. ROS is currently available for C++ and Python. Rosjava_core is an attempt to make ROS available for Java. It implements the internal ROS infrastructure including time synchronisation, namespace resolving, topic and service advertising, and communication methods.

D. Modules

Modules are usually not designed specifically for a product, this enables the equiplets to offer generic services to a variety

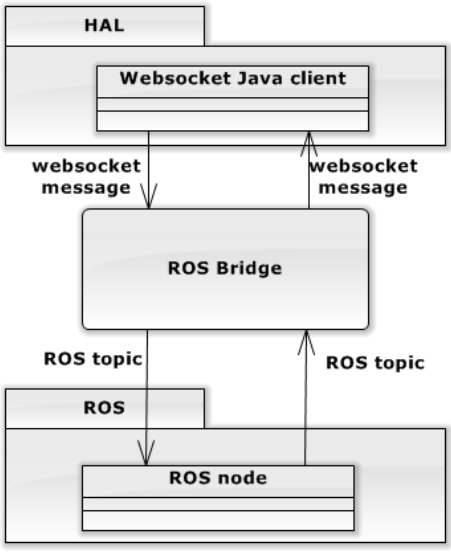

Figure 16. ROS-HAL interface using a ROS bridge.

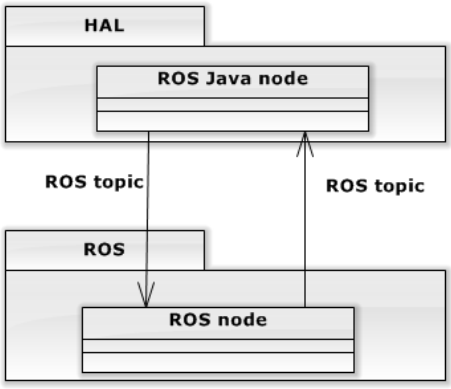

Figure 17. ROS HAL interface using a ROS JAVA node.

of products. For this purpose, a components off the shelf (COTS) strategy is adopted together with modules that are specifically designed for grid manufacturing using equiplets, and which are developed using product family engineering. As shown in Figure 18, Product Family Engineering uses common parts for as many different modules as possible.

Currently, a number of modules have been developed specifically for grid manufacturing:

Delta Robot The deltarobot is a parallel manipulator where three actuators are located on the base, and where arms made of light composite material are used to move parts. All moving parts have a small inertia, which allows for very high speed and accelerations.

Figure 19 shows the schematics of the deltarobot that is specifically designed to be used for equiplets. The end effector is designed in such a way that grippers can easily be changed using a precise clicking system with magnets. Many components of the delta robot are also manufactured using additive manufacturing, making it easy to customise or

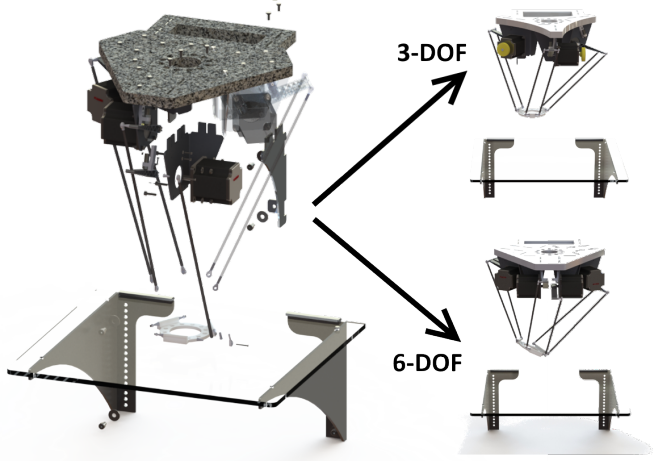

Figure 18. Two different parallel manipulators with 3 and 6 Degrees of Freedom using as many identical components as possible.

produce parts for the modules on demand.

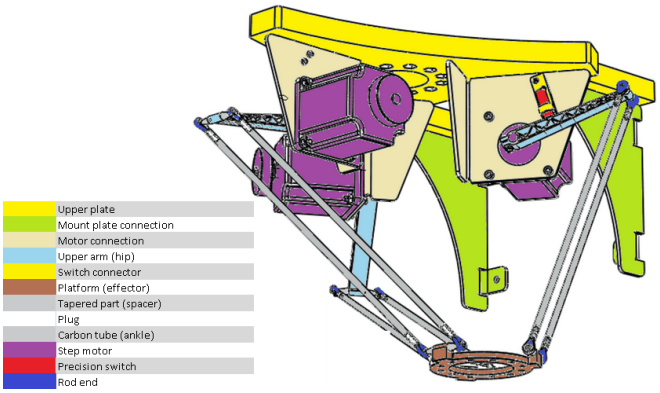

Figure 19. Delta robot Hardware Component Schematics.

The Delta Robot uses three actuators of the type Oriental Motors PK566PMB) that are controlled using motor controllers (Oriental Motors CRD514-KD). The controllers are directly accessed from the respective ROS module node. Figure 20 shows the steppermotor class, which uses a modbus interface. The modbus interface is offered by a generic InputOutput class that is implemented by the InputOutputModBusRtuController class. This class has been created for easy reuse throughout the system and is implemented by all RTU modbus implementations.

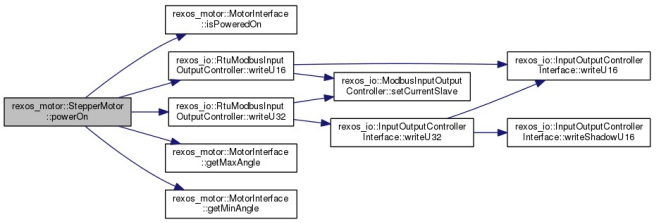

Figure 20. The call/inheritance graph for the motor controller.

Since equiplets are not specifically designed for a product it is important to have as much flexibility as possible to service a larger variety of products types. For this purpose, the Delta Robot design with 3 degrees of freedom was adapted to an inverted **Stewart Gough** platform, which uses 6 motors to be able to have a limited 6 degrees of freedom. Figure 21 shows this 6 DOF parallel manipulator module. This module is very useful since products can arrive at the equiplets in any orientation.

Figure 21. An adaptation to the deltarobot, which uses 6 motors as an inverted Stewart Gough Platform to create more flexibility.

This specific *Gripper* module is controlled using modbus over TCP. This is performed by an inline bus coupler (Phoenix contact IL ETH BK DI8 DO4) that is accessed from the respective gripper node. Most types of grippers that are currently used work with a pneumatic system to move an effector or create a vacuum to pick up small parts.

Figure 22. Gripper call / inheritance graph.

The *Vision Module* is of high importance within grid manufacturing. The product location will usually not be preprogrammed and as such must be detected dynamically in real-time. This is done by the Vision module that uses an OpenCV-based detection system to determine either the location of a product, or the location of a tray or other transport device that has the knowledge of the relative location of the product towards itself. The location data that will be found by the Vision Module will be delivered to the environment cache so that other modules can easily access it. The standard vision module holds a number of algorithms to deal with a variety of situation. It can automatically calibrate its lenses and has a built-in correction and balance system to deal with differences in lighting and distortions.

The *Work plane* is the area where a product, crate or vehicle that carries a product is located. Many equiplets use transparent working planes such that computer vision systems

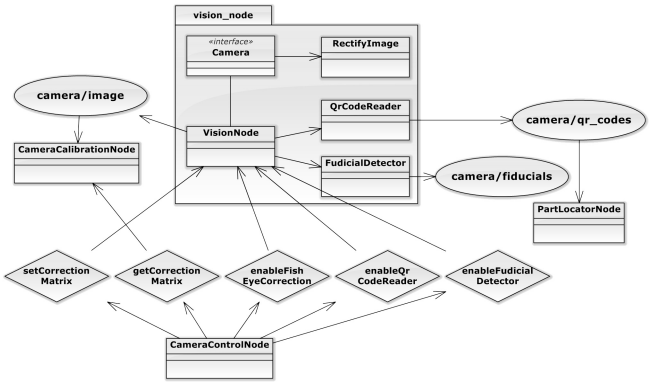

Figure 23. Vision system.

can be used to localise the specific parts. If a product is placed on a crate or cart the product agent can usually infer the location based on its own position as seen by the vision system. While the working plane has no actuators or sensors itself it is still seen as a module and has a ROS node that represents it. The ROS node is used to calculate its own position based on where it has been attached, its own known specifications and calibrations using vision and markers that can be placed on the working plane.

The *Additive Manufacturing module* can be used for a wide range of tasks. Figure 24 shows the 3D printer module that can print any object, i.e., casings or buttons for a unique customized internet radio. This module is an important asset for grid manufacturing since it makes it possible to create a variety of items not only for custom products, but also for the modules itself.

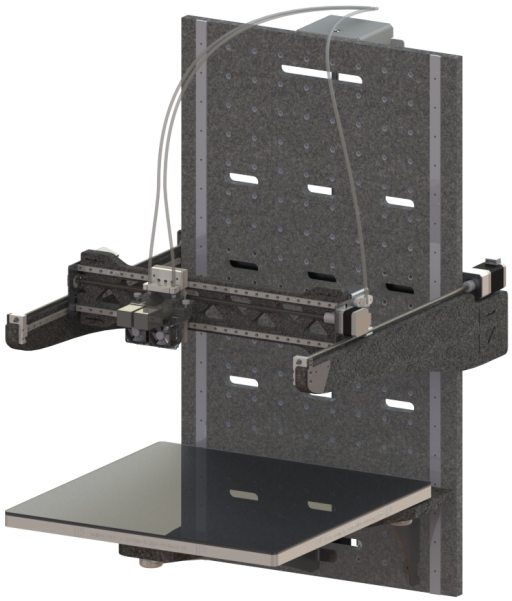

Figure 24. 3D printer module.

E. Basic operation

While not all services of a grid are relevant for this paper it is important to show the basic operation of a grid. More specific actions like reconfiguration and the implementation of the HAL will be discussed in other work. Figure 25 shows the normal operation when manufacturing. The sequence is implemented in the following way:

1) An equiplet agent is aware of the capabilities based on the modules it has configured.
2) The equiplet agent registers its service at the Directory Facilitator (DF) that acts as a 'Yellow Page' service for the Product agents.
3) When a product agent is initialised it has a number of product steps that describe how it needs to be manufactured, the product agent queries the DF to find the services that could potentially perform the steps.
4) The product agent uses the list of equiplets it has received from the DF to inquire the equiplets if the equiplet can match the specific schedule.
5) If the schedule can be met the product agent also inquires if it can meet its specific detailed criteria that the product may require for the product step to be performed adequately.
6) When all criteria are met and the product arrives on schedule at the equiplet it will send its instructions on how to perform the steps to the equiplet.
7) The equiplet will translate the steps to its specific hardware and send it to the equiplet node in ROS to control the hardware and perform the specific step.
8) When done the equiplet agent will inform the status to the product agent.

X. Evaluation and Performance

The next step will be to evaluate the performance and scalability by performing a number of benchmarks. These have been split in multiple types:

1) Synthetic benchmark - to test the individual systems and latencies during load [40].
2) Full testing in simulation mode - to test realistic cases using the entire architecture.

A. Synthetic benchmarking

First, a standard equiplet setup has been created that uses a ROS/JADE infrastructure connected by a MongoDB blackboard, see Figure 26.

Three tests were performed:

1) Node to Node communication over ROS.
2) Agent to Agent communication using JADE.
3) A pick and place case utilizing all layers.

For all three benchmarks 10,000 messages will be send, where full round time including a response message is measured. Results are shown in Figure 27, which uses a trend line over 5 periods. The message is an instruction that contains a JavaScript Object Notation (JSON) object that holds a target, ID, instruction data and parameters.

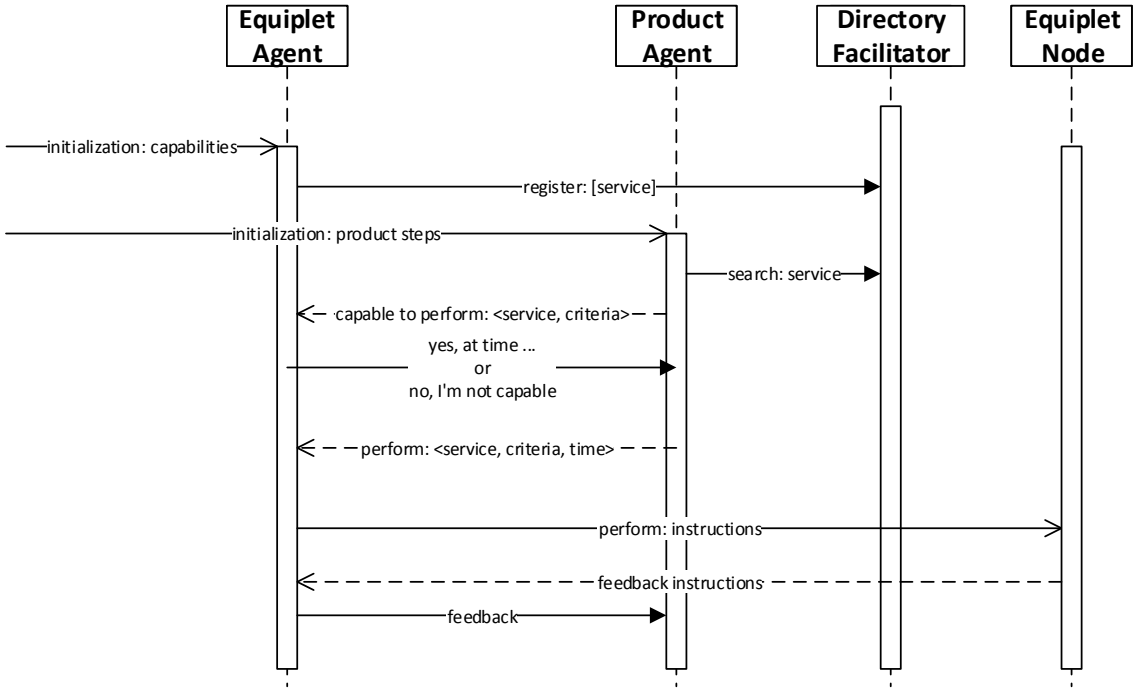

Figure 25. REXOS service.

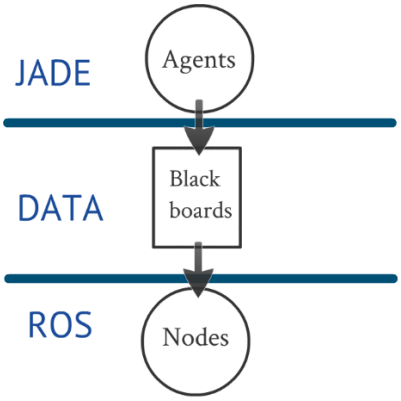

Figure 26. Synthetic test setup for benchmarking.

The results show that ROS to ROS and JADE to JADE performance is much better than when both are combined using a blackboard. Hence, different interfaces were required to be investigated to handle the communication between the ROS and agent layer. This was performed using the simulated benchmarking system.

B. Simulated benchmarking

This section evaluates the performance of the entire architecture using a full simulation of the system. The sources are identical to a real runtime situation, only the hardware responses are being simulated. The most important aspect of this test is the different interface implementations that connect the (C++ based) ROS and (JAVA based) JADE platform.

To determine the best implementation for the interface between HAL and ROS, every implementation was benchmarked. The benchmark has been performed using custom written software and measures the time required to communicate from node A to node B and back to A. Node B will respond immediately. Node A is always a ROS node, while node B is either a regular ROS node, a ROS Java node, or a Java ROS bridge listener. The only exception are the blackboard measurements. Because the blackboard implementation does not use the ROS infrastructure, measuring the latency using ROS is not an accurate measurement. Instead the time required to communicate from A to the MongoDB server and back to A is measured. Because this gives an unfair discrepancy in the measurement (the relevant case is to transmit a message from A to B and receive and response from B), the blackboard latencies have been multiplied by two to compensate that the message has to be send twice (first from A to MongoDB and then from A to B).

The idle equiplets are equiplets that have been started but are not performing any tasks. The busy EQs are equiplets executing a hardware step every 1 second and measuring data every 10ms. The measurements have been determined using 100,000 samples.

The average latency as seen in Figure 29 has been measured with 10 and 50 active equiplets. Other scenarios have also been measured, but produced less relevant data. The ROS C++ topic, service, and action servers used are native ROS communication methods and act as a reference. They are not actual implementations of the interface. The data clearly shows that the average latency for the ROS C++ topics is the lowest. The ROS C++ service and ROS C++ actionServer also have

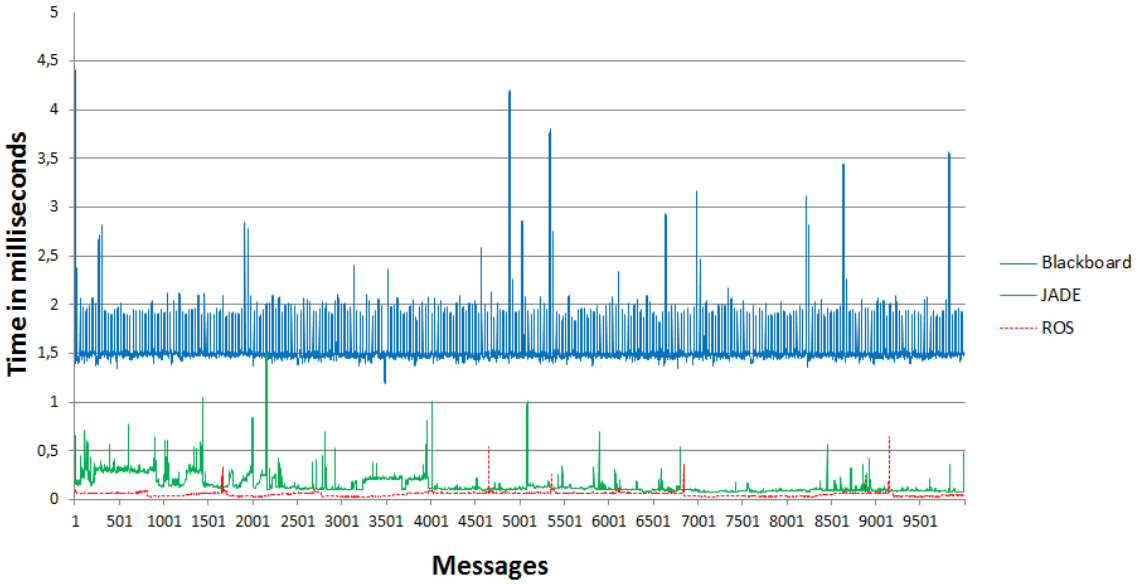

Figure 27. Synthetic benchmark of ROS to ROS communication (bottom), Agent to Agent communication (middle), and the pick and place benchmark (top).

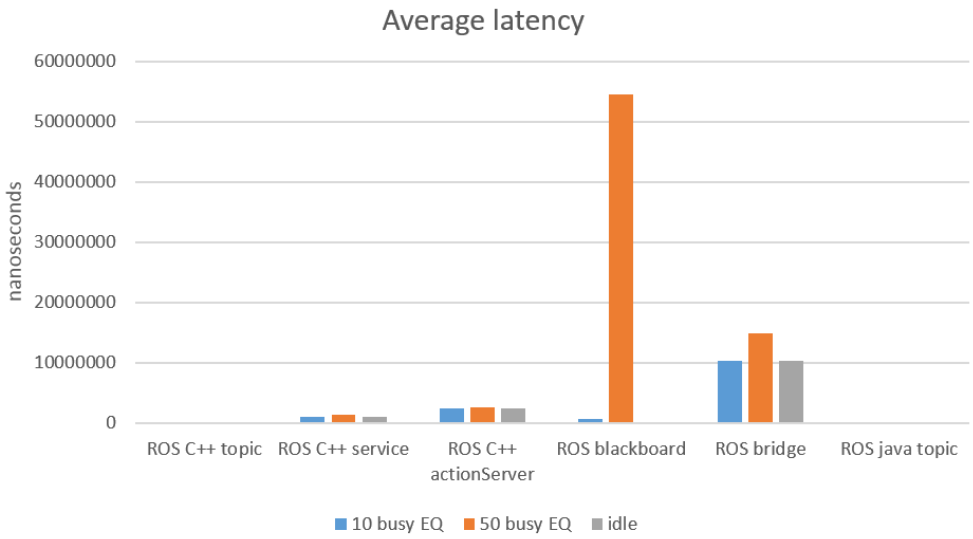

Figure 28. Average latency of the different interfaces that have been developed.

a low average latency. The blackboard implementation has a low base latency, but scales very poorly.

During the benchmarks it became clear that the blackboard implementation has a specific point after which the latency increases spectacularly. This might be caused by a connection pool in the MongoDB server running out, resulting in other connections having to wait. The ROS bridge has a very high base latency but scales much better. In all the other scenarios the base latency is also approximately 10,000,000 nanoseconds (equals 10 milliseconds). This suggests that the ROS bridge uses a periodical poll mechanism. The ROS Java topic has very low base latency and seems to scale excellently.

Figure 29 shows the consistency of the latency of an implementation of the interface. This shows how reliable the interface is when it comes to consistent behaviour. The average deviation matches the average latency in that once again the ROS C++ topic, ROS C++ service and ROS C++ action server perform very well, while the ROS blackboard scales poorly. The ROS bridge has a quite high, but steady deviation.

XI. Conclusion

The paper takes an all-embracing approach to self-organising, reconfigurable autonomous manufacturing systems. The goal behind this is to provide a basis for a practical implementation by combining new technologies as a staging ground for new manufacturing methodologies based on the industry 4.0 principles that will boost the adoption by industry. Hence, this includes the development of hardware, the use of system and software engineering principles and integration of

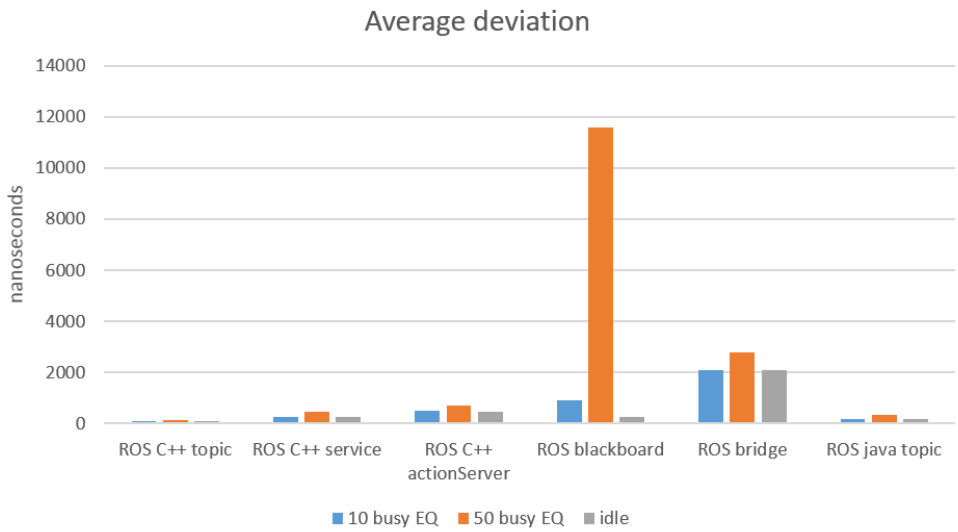

Figure 29. Average deviation of the latencies.

the newest hardware designing techniques. It also builds on current advances in software by using distributed systems and combining them in a hybrid architecture. The hope is that this leads to solutions that prove to industry that the newest technology is becoming more and more suitable for true mass adoption. This is done by investigating the current state of technology, analysing the requirements and new developments in smart industry and trying to encompass this in the concept of 'grid manufacturing' that consists of both a hardware platform, i.e., the equiplets and a software platform, i.e., REXOS.

The main research question of the paper was intended to show that grid manufacturing, based on the REXOS platform can combine low-level performance and flexibility using intelligent behaviour. The choice to combine two platforms are supported by the axiomatic design methodology, which strongly asks to decouple the requirements from the design parameters. The results give insights in how both JADE and ROS can best be interfaced using the JAVA ROS node that act as a wrapper for the messages from JADE towards the ROS platform. It also shows that the interface between these platforms are crucial to get a scalable platform, by demonstrating that the (originally developed) Blackboard interface was severely lowering the performance when 50 or more equiplets were used. However, the paper also introduces the entire concept, by introducing the GEM architecture and giving an introduction to the functionality that REXOS and the equiplets can provide.

The paper also evaluates the concept of grid manufacturing in general, taking design techniques and hardware into account. The equiplet platform in general and the modules specifically were designed using a low-cost strategy where equiplets can easily be reconfigured, providing a high utilization to a minimum cost. This was done by using combining product family engineering with the use of many standard components. When specific components have to be made they are commonly designed in such a way that equiplets can produce them themselves, for example by using 3D printed parts.

More generally, the paper makes it clear that it is essential to take an applied approach to solve these problems. The industry will require working proof of concepts that not only tackle the theory but also the practical problems that occur when working with complex systems such as the ones demonstrated in this paper. The development and testing of all these systems have required a large amount of work but also add to the validity, and therefore, usefulness for industry.

This research provides a number of insights:

1) ROS and MAS can be effectively combined - which decouples the performance and intelligence gap.
   a) The MAS provides the abstractness to deal with the dynamics that are required for self-organising systems.
   b) ROS gives the performance and tools to effectively develop a large range of control systems that can be reconfigured.
   c) The choice to specifically combine JADE and ROS seems to be effective.
2) The autonomous nature of both platforms makes it possible to adapt part of the systems during runtime, which is an important aspect when considering reconfigurability.
3) The use of autonomous systems makes it easier to lower interdependence between functionalities, creating a decoupled design, which lowers overal complexity.
4) Combining different platforms like MAS and ROS have a high potential for industry.

The combination of the requirements and propositions gives fuel to new research in more practical problems that are fundamental for smart industry; in future work both the aspects of (automatic) reconfiguration and dynamic (safety) system behaviour will also be discussed in more detail. However, the

proposition as mentioned in Section II-D seems feasible.

ACKNOWLEDGMENT

This work has been supported by the ARTEMIS R5-COP project and the HU University of Applied Sciences Utrecht, the Netherlands. Thanks go to all the computer engineering students that have helped with the implementations of REXOS, Maarten Dinkla for performing a language review, and Joost van Duijn for providing information about the hardware design process.

REFERENCES

[1] D. Telgen, L. van Moergestel, E. Puik, P. Muller, and J.-J. Meyer, "Requirements and matching software technologies for sustainable and agile manufacturing systems," in *INTELLI 2013, The Second International Conference on Intelligent Systems and Applications*, 2013, pp. 30–35.

[2] D. Telgen, L. van Moergestel, E. Puik, and J.-J. Meyer, "Agile manufacturing possibilities with agent technology," in *Proceedings of 22nd International Conference on Flexible Automation and Intelligent Manufacturing*. Tampere University of Technology, 2012, pp. 341–346.

[3] P. Leitão, "Agent-based distributed manufacturing control: A state-of-the-art survey," *Eng. Appl. Artif. Intell.*, vol. 22, no. 7, pp. 979–991, Oct. 2009.

[4] K. Schild and S. Bussmann, "Self-organization in manufacturing operations," *Commun. ACM*, vol. 50, no. 12, pp. 74–79, Dec. 2007.

[5] J. Wind and A. Rangaswamy, "Customerization: The next revolution in mass customization," *Journal of Interactive Marketing*, vol. 15, no. 1, pp. 13 – 32, 2001.

[6] K. H. Hall, R. J. Staron, and P. Vrba, "Experience with holonic and agent-based control systems and their adoption by industry," in *Holonic and Multi-Agent Systems for Manufacturing*, ser. Lecture Notes in Computer Science, V. Mak, R. William Brennan, and M. Pchouek, Eds. Springer Berlin Heidelberg, 2005, vol. 3593, pp. 1–10.

[7] V. Marik and D. McFarlane, "Industrial adoption of agent-based technologies," *IEEE Intelligent Systems*, vol. 20, no. 1, pp. 27–35, 2005.

[8] S. HU, "Paradigms of manufacturing - a panel discussion," *3rd Conference on Reconfigurable Manufacturing*, 2005.

[9] Y. Koren and M. Shpitalni, "Design of reconfigurable manufacturing systems," *Journal of Manufacturing Systems*, vol. 29, no. 4, pp. 130 – 141, 2010.

[10] M. Mehrabi, A. Ulsoy, and Y. Koren, "Reconfigurable manufacturing systems: Key to future manufacturing," *Journal of Intelligent Manufacturing*, vol. 11, no. 4, pp. 403–419, 2000.

[11] K. Stecke, "Flexibility is the future of reconfigurability. paradigms of manufacturinga panel discussion," in *3rd Conference on Reconfigurable Manufacturing*, 2005.

[12] H. ElMaraghy, "Flexible and reconfigurable manufacturing systems paradigms," *International Journal of Flexible Manufacturing Systems*, vol. 17, no. 4, pp. 261–276, 2005.

[13] A. Gunasekaran, "Agile manufacturing: a framework for research and development," *International journal of production economics*, vol. 62, no. 1, pp. 87–105, 1999.

[14] S. L. Koh and L. Wang, "Overview of enterprise networks and logistics for agile manufacturing," in *Enterprise Networks and Logistics for Agile Manufacturing*, L. Wang and S. L. Koh, Eds. Springer London, 2010, pp. 1–10.

[15] U. Rauschecker, D. Stock, M. Stöhr, and A. Verl, "Connecting factories and related it environments to manufacturing clouds," *International Journal of Manufacturing Research*, vol. 9, no. 4, pp. 389–407, 2014.

[16] D. C. Mcfarlane and S. Bussmann, "Developments in holonic production planning and control," *Production Planning & Control*, vol. 11, no. 6, pp. 522–536, 2000.

[17] A. Giret and V. Botti, "Engineering holonic manufacturing systems," *Computers in Industry*, vol. 60, no. 6, pp. 428 – 440, 2009, collaborative Engineering: from Concurrent Engineering to Enterprise Collaboration.

[18] N. P. Suh, *Axiomatic Design: Advances and Applications (The Oxford Series on Advanced Manufacturing)*. Oxford University Press, USA, 2001.

[19] M. Wooldridge and N. Jennings, "Intelligent agents: Theory and practice," *The Knowledge Engineering Review*, vol. 10, pp. 115–152, 1995.

[20] M. Paolucci and R. Sacile, *Agent-based manufacturing and control systems : new agile manufacturing solutions for achieving peak performance*. Boca Raton, Fla.: CRC Press, 2005.

[21] D. Dennett, *The Intentional Stance*. Cambridge, Mass: MIT Press, 1987.

[22] M. Bratman, *Intention, Plans, and Practical Reason*. Cambridge, Mass: Harvard University Press, 1987.

[23] A. S. Rao, M. P. Georgeff *et al.*, "Bdi agents: From theory to practice." in *ICMAS*, vol. 95, 1995, pp. 312–319.

[24] M. Wooldridge, *An Introduction to MultiAgent Systems, Second Edition*. Sussex, UK: Wiley, 2009.

[25] A. Ricci, M. Viroli, and M. Piunti, "Formalising the environment in mas programming: A formal model for artifact-based environments," in *Programming Multi-Agent Systems*, ser. Lecture Notes in Computer Science, L. Braubach, J.-P. Briot, and J. Thangarajah, Eds. Springer Berlin Heidelberg, 2010, vol. 5919, pp. 133–150.

[26] F. Heintz, P. Rudol, and P. Doherty, "Bridging the sense-reasoning gap using dyknow: A knowledge processing middleware framework," in *KI 2007: Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, J. Hertzberg, M. Beetz, and R. Englert, Eds. Springer Berlin Heidelberg, 2007, vol. 4667, pp. 460–463.

[27] R. C. Arkin, *Behavior-based robotics*. MIT press, 1998.

[28] N. R. Jennings, "An agent-based approach for building complex software systems," *Commun. ACM*, vol. 44, no. 4, pp. 35–41, apr 2001.

[29] L. v. Moergestel, J.-J. Meyer, E. Puik, and D. Telgen, "Decentralized autonomous-agent-based infrastructure for agile multiparallel manufacturing," *Proceedings of the International Symposium on Autonomous Distributed Systems (ISADS 2011) Kobe, Japan*, pp. 281–288, 2011.

[30] L. van Moergestel, E. Puik, D. Telgen, and J.-J. Meyer, "Embedded autonomous agents in products supporting repair and recycling," *Proceedings of the International Symposium on Autonomous Distributed Systems (ISADS 2013) Mexico City*, pp. 67–74, 2013.

[31] M. Dastani, "2apl: a practical agent programming language," *Autonomous Agents and Multi-Agent Systems*, vol. 16, no. 3, pp. 214–248, 2008.

[32] F. Bellifemine, C. G., and D. Greenwood, *Developing multi-agent systems with Jade*. John Wiley & Sons Ltd., 2007.

[33] L. Braubach, A. Pohkahr, and W. Lamersdorf, "Jadex: A short overview," *Proceedings of the Main Conference Net.ObjectDays*, 2004.

[34] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[35] L. v. Moergestel, J.-J. Meyer, E. Puik, and D. Telgen, "The role of agents in the lifecycle of a product," *CMD 2010 proceedings*, pp. 28–32, 2010.

[36] L. van Moergestel, E. Puik, D. Telgen, M. Kuijl, B. Alblas, J. Koelewijn, J.-J. Meyer *et al.*, "A simulation model for transport in a grid-based manufacturing system," in *Proc. of the Third International Conference on Intelligent Systems and Applications (INTELLI 2014)*. IARIA, 2014, pp. 1–7.

[37] L. v. Moergestel, E. Puik, D. Telgen, J.-J. Meyer *et al.*, "A study on transport and load in a grid-based manufacturing system," *International Journal on Advances in Software*, vol. 8, no. 1 & 2, pp. 27–37, 2015.

[38] L. v. Moergestel, J.-J. Meyer, E. Puik, and D. Telgen, "Agent-based manufacturing in a production grid: Adapting a production grid to the production paths," *Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART 2014)*, vol. 1, pp. 342–349, 2014.

[39] L. van Moergestel, E. Puik, D. Telgen, and J.-J. Meyer, "Production scheduling in an agile agent-based production grid," in *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2012 IEEE/WIC/ACM International Conferences on*, vol. 2, 2012, pp. 293–298.

[40] D. Telgen, L. van Moergestel, E. Puik, P. Muller, A. Groenewegen, D. van der Steen, D. Koole, P. de Wit, A. van Zanten, and J.-J. Meyer, "Combining performance and flexibility for rms with a hybrid architecture," in *Advances in Artificial Intelligence, 16th Portuguese Conference on Artificial Intelligence, local proceedings*. IEEE, 2013, pp. 388–399.