Self-Organizing Assembly Systems

Regina Frei and Giovanna Di Marzo Serugendo, *Member, IEEE*

***Abstract*—This paper addresses a vision of future manufacturing systems, which are highly agile, user friendly, and increasingly based on autonomous components. Evolvable assembly systems (EASs) provide a solution for agile assembly, including a concept for reconfigurability with modularity at the mechanical as well as at the control level. It takes the multilateral relations among product, processes, and systems into account and allows the systems to evolve together with the requirements. Self-organizing assembly systems (SOASs) are a further development of EAS, allowing them to play an active role in layout design and production. This paper focuses on the self-organization mechanisms for the design of SOAS, as well as the system architecture, including agents and self-knowledge. Agentified modules participate in their own arrangement in the system layout and monitor themselves during production. Policies and metadata for self-management during production are described, and performance metrics for agility scenarios are indicated.**

***Index Terms*—Agile manufacturing, assembly systems, multiagent systems.**

I. Introduction

Agile manufacturing is one of the big challenges nowadays and will be an even bigger challenge in the future. As product life cycles shorten, market dynamics increase, and the economic situation deteriorates, the manufacturing industry needs to improve its competitiveness. It may help for automation to become sustainable and accessible to small and medium enterprises as well. Companies must be highly responsive to changing conditions, fluctuating demands, and individual customer wishes. Due to the well-known factors of high salaries in the Western Hemisphere, automation is the only alternative to offshoring entire industries. However, automation in its traditional form cannot cope with today's and tomorrow's challenges.

Dedicated production lines are optimized for a single specific product, or at the most for a specific product family, but can hardly be adapted to new and frequently changing requirements. *Flexible* manufacturing systems [1]–[3] incorporate a predefined set of capabilities that makes them both expensive and often overly sophisticated, which means that they are difficult to manage. The risk of paying for unused and wrong capabilities is high. *Reconfigurable* manufacturing systems (RMSs) [4]–[6] are already closer to coping with the need for agility at the shopfloor level because they are modular and allow the engineer to add/remove functionalities according to the needs. Recent efforts in the area of RMS do, however, often focus on reconfigurable machines [7] and the evolution of product characteristics [8]. A suitable control concept seems to be missing so far. Evolvable assembly systems (EASs) [9], [10] offer a solution, which includes finely granular modules with local intelligence and a multiagent control system. Product class characteristics are closely related with assembly processes and assembly systems. In the context of EAS, *evolvability* refers to a system's ability to continuously and dynamically undergo modifications of varying importance: from small adaptations to big changes.

Our proposal for self-organizing assembly systems (SOASs) [11]–[13] takes the EAS paradigm further by letting the systems play an active role and making systems more autonomous, and thus, more user friendly. The system is at the service of its user, similar to intelligent houses, which provide their inhabitants with all kinds of services due to home automation [14]. An SOAS actively participates in its own design at *creation time*: given a specified product order provided in input, the system modules select suitable partner modules and choose their own position in the assembly system layout. The appropriate assembly system and corresponding movements emerge from the self-organization process among the different modules in the assembly system. The self-organization (SO) and self-adaptation (SA) process does not stop at the layout formation. During *production time*, the modules self-manage: whenever a failure occurs in one or more of the modules, the SO process may lead to two different outcomes: the current modules adapt their behavior (change speed, force, task distribution, etc.) to cope with the current failure, potentially degrading performance but maintaining functionality, or may decide to trigger a reconfiguration leading to a repaired system. The actual decision will depend on the situation at hand and on specific production constraints (cost/speed/precision).

A. Contributions of this Paper

SOASs are situated in a very interdisciplinary field; research from manufacturing engineering, microengineering, mechanical, electrical and electronic engineering, computer science, distributed systems, SO, and formal methods comes together. This paper focuses on the creation of SOASs; it summarizes the general concept, the creation time mechanisms, and the production time architecture. This paper provides an overview of the previously published work and refers to suitable articles for details on specific topics: design [11], architecture [15], creation time mechanisms and execution of formal specifications in Maude [16], production time policies and metadata [17], partial implementation advances [12], [18], and overall view [13].

Manuscript received June 16, 2010; revised September 14, 2010; accepted November 21, 2010. This paper was recommended by Associate Editor J. Lazansky. This work was supported by the Portuguese Foundation for Science and Technology under Grant SFRH/BD/38608/2007 and by the European Union-funded Coordination Action PerAda (http://www.perada.org) for travel exchange between Portugal and the U.K. R. Frei currently receives a Fellowship for Prospective Researchers from the Swiss National Science Foundation.

R. Frei is with the Intelligent Systems and Networks Group, Department of Electrical and Electronic Engineering, Imperial College London, London SW7 2BT, U.K. (e-mail: work@reginafrei.ch).

G. Di Marzo Serugendo is with the University of Geneva, CH-1211 Geneva, Switzerland (e-mail: dimarzo@dcs.bbk.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCC.2010.2098027

B. *Organization of this Paper*

Related work is presented in Section II, and useful concepts for SOAS in Section III. Section IV gives an overview of SOAS, clarifies the requirements, and introduces the running example. The agents and self-knowledge are detailed in Section V. Mechanisms for SO and self-management are illustrated in Section VI. Section VII explains agility scenarios and performance parameters, and Section VIII concludes this paper.

II. Related Work

Holonic manufacturing systems [19], [20] concentrate on morphologic aspects (every item is a whole as well as part of a bigger whole). ADAptive holonic COntrol aRchitecture for distributed manufacturing systems (ADACOR) [21], [22] combines holonics with the idea of SO by using pheromones. At their inception, holonic systems were strongly biologically inspired; however, with time, the approaches have become mainly top–down, and thus, less suited for quick changes.

The architecture for agile assembly (AAA) [23] and its European equivalent, i.e., the *MiniProd* [24], [25], are probably two of the approaches that come closest to provide a solution for reconfigurability in assembly systems, but do not consider the mutual interrelations among product, processes, and systems, which are important for evolvability [9]. AAA and MiniProd are made for mini assembly and only suitable for a limited set of operations.

Realized by partially the same researchers as MiniProd, the project SIARAS [26] (short for *skill-based inspection and assembly of reconfigurable automation systems*) intends to automate part of the usually manual reconfiguration work in assembly systems. Similarly to SOAS, SIARAS is also ontology based. It analyzes the process requirements, specified as work flows, and makes suggestions for corresponding system configurations. CAD data of the parts and simulations help to answer geometrical questions, such as the reachability of certain points and potential collisions. The system can automatically cope with changes in the product, the processes, and the equipment. Different from SOAS, the SIARAS approach is centralized and does not consider local intelligence.

The software component in an agent-based automation architecture can be considered as a world model repository [27], containing a symbolic representation of the automation agent and its relations to its environment.

The use of ontologies in manufacturing has been thoroughly discussed and diverse proposals were made [28], [29]. An ontology-based reconfiguration agent [30] uses knowledge from an ontology to automate reconfiguration and avoid human intervention. The agent reasons about the current system as well as the new requirements and then decides if reconfiguration is necessary. However, in our proposal, there is no single agent, which reasons and decides; reasoning is distributed, and adaption happens locally.

The *Pabadis* [31] control architecture for *plug & participate* is based on distributed intelligence, a manufacturing ontology, a first embedded real-time agent platform for control, and a new generation of radio-frequency identifications (RFIDs). Having parallels with SOAS, Pabadis is focused on a higher level inside the manufacturing company, more at the level of production management [enterprise resource planning (ERP)].

Multiagent systems are increasingly used for projects in industry [32], [33], as their application to manufacturing has been studied for over two decades. Nevertheless, further efforts are still necessary to make them more accessible for industry and to link them with other industrial systems.

As for the *shop-floor design*, automatic procedures, to help the planning engineer define the shop-floor layout [34], [35], use CAD representations of robots and parts. The flow between generic workstations [36] determines their arrangement, which can be serial, parallel, or consist of any hybrid combination. However, workstations with varying skills were not taken into account. This approach considers that the assembly sequence can be chosen in function of the layout, whereas with SOAS, we assume that the assembly sequence is given.

Diverse types of shop-floor layouts (functional, cellular, distributed, hybrid, etc.) and corresponding performance metrics are compared in [37]. Their reconfigurable layout type, where the resources can be easily moved around in the shop floor, is quite inline with the distributed nature of EAS.

Reinforcement machine learning helps determine whether a shop-floor layout is feasible or not [38]. The suggested simulation model takes into account the arrival rate of the items, the cycle times, and the workstation capacities but does not consider the different types of processes, which may be required for different work steps.

Overlapping decomposition [39] is a system-theoretic approach to modeling and analyzing the performance of complex manufacturing systems. The layout is decomposed into serial production lines, which overlap at the beginning and the end of the line. Assembly, parallel, rework, feedforward, and scrap operations can modeled.

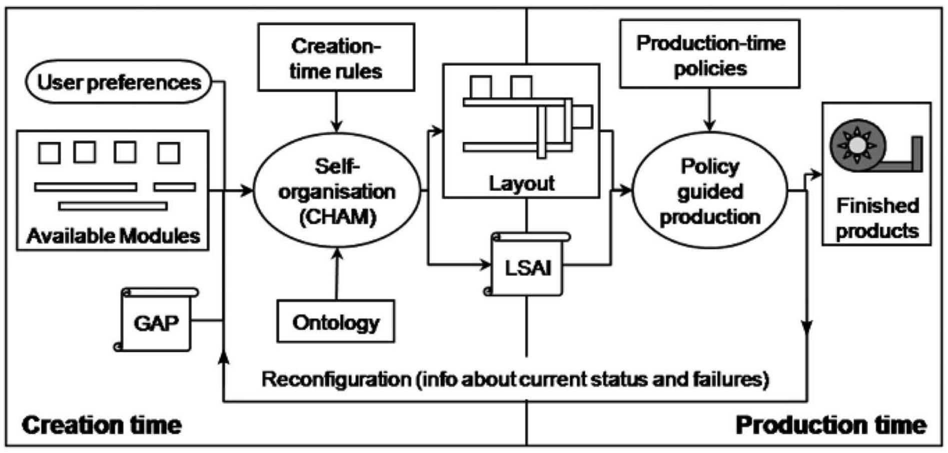
III. Concepts and Assumptions

A. *Assembly Terminology*

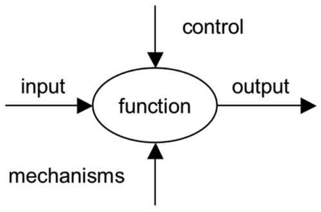
A *product* is composed of *parts* assembled in a specified way, either by human hands or by robots in an *assembly system* [11], which is an industrial installation able to receive parts and join them in a coherent way to form the final product. An assembly system consists of a set of equipment items (*modules*), such as conveyors, pallets, simple robotic axes for translation and rotation, as well as more sophisticated industrial robots, grippers, sensors of various types, etc.

B. *Agentified Modules and Services*

Both in EAS and SOAS, each module or manufacturing component is associated with a software agent, consisting of an abstraction of the component including its functionalities and interaction skills. In this way, every agentified module can offer its services to other agents and request theirs. Agents which dynamically collaborate form *coalitions* [18] and offer *composite skills*, based on the *simple skills*, which each module contributes. Agentified modules can be seen like the members of a swarm;



(a)



(b)

Fig. 1. SOAS. (a) Overview of the system including most relevant components. (b) SADT representation.

their coordination can be based on similar strategies, such as "pursue your local goals, keep your behavior in sync with your neighbors, and comply with the system's rules." Even though their mechanical properties are diverse, from a software point of view, they have similar or identical characteristics. They can spontaneously participate in a dynamic collaboration or withdraw from it, without disturbing the peers. Interaction is based on local rules.

C. Time Terminology

In the context of SOAS, we discern two phases in the life of an assembly system: At *creation time,*, the layout is being created by the agents as a virtual concept, and then, physically built by the user. At *production time*, an SOAS is assembling product items (software and mechanical modules are running). Creation time and production time may also overlap or occur concurrently in different parts of the same system; the concept of SOAS previews that the systems coevolve with the requirements, and thus, never become outdated.

D. Working Assumptions

We consider the following working assumptions reasonably realistic and useful for SOAS (for the full list, see [13]): System modules are versatile (can be used in different contexts) and work with *high precision and repeatability*. Each module disposes of various types of suitable *sensors* for diverse monitoring purposes. EAS and SOAS are *open systems*. New modules/agents may join at any instant, but there are no malicious attacks or undesired intruders that threaten our systems. The rules for SO and policies for self-management rely on *ontologies* [12], [40].

E. System Autonomy and User Friendliness

User-friendly systems should be easy to handle, also for non-specialized operators. This means that a certain degree of system autonomy is required. Manual (re)programming needs to be gradually reduced, as it requires specialist knowledge; selecting options proposed by the system is much more convenient and also feasible for usual operators. Nevertheless, system autonomy is a highly controversial issue in industry. At the same time, it is also very promising. As a general measure, it has to be ensured that the user never loses the ultimate control. The degree of autonomy needs to be variable [41] so that the user can modify it at wish (e.g., make the system to find solutions and ask the user for confirmation before execution).

F. Emergence in SOAS

Emergence, like autonomy, is a controversial topic in industry. Some of the emergent phenomena will be favorable to the accomplishment of the system's task and have considerable potential for advanced system behaviors, such as the emergence of complex capabilities out of simple ones. An example is the classical *pick & place* mechanism: There are many different ways of putting together a gripper with translation/rotation axes, but not all of them lead to the desired functionality. Favorable emergent phenomena could and should be exploited. Others may be less adapted, disturbing, or even harmful: e.g., system integration is supposed to function without unexpected symptoms. In nature, unsuccessful properties will be eliminated by natural selection. Such a mechanism is not viable in manufacturing environment: Harmful behavior cannot be allowed at any moment, and therefore, suitable control and safety mechanisms are needed. For further discussion, see [13].

IV. Self-Organizing Assembly Systems Overview and Running Example

A. Overview

Fig. 1(a) illustrates creation time, production time, and the relevant system components, using a *system analysis and design technique* (SADT) inspired representation, as shown in Fig. 1(b).

Creation time: At the core of the system is a self-organizing incremental and iterative process (modeled according to the chemical reaction model), *controlled* by creation time rules and *supported* by the ontology [12], [40].

Inputs to this creation time process are as follows.

1) The generic assembly plan (GAP), which specifies the assembly tasks in an abstract way, independent from which modules are going to execute the tasks. The GAP is explained in Section V-D1.
2) All the available modules as registered in the ontology. These modules will compose the layout. They offer skills, and those suitable for the tasks to be execute will react with the tasks and form coalitions with partner modules to offer composite skills according to the task requirements.
3) The EAS ontology, which contains all necessary knowledge about assembly concepts, processes, modules, parts and their relations.
4) User preferences expressed as policies, for instance, the preference for a linear layout, or the use of a minimal number of modules.

The *outputs* are 1) the layout (to be built by the user according to the results of the self-organized process and the user's selection, as explained in Section IV-C1) as well as a virtual representation of the layout and 2) the layout-specific assembly instructions (LSAI), explained in Section V-D2, which the modules derive from the GAP according to rules and taking into account their own characteristics.

Production time: The output of the creation time process is the *input* of the production time process. This process is *controlled* by production time policies (see Section VI-C), which are enforced with a reasoning engine,[1] and leads to the finished products at the end, which represent the *output*. If necessary, it also triggers system reconfiguration, which restarts the creation time process, taking as additional input the information about the current state of the system and its problems/failures.

B. SOAS Requirements

Requirements, which follow a self-* approach are called *self-* requirements* within the context of our paper. They are classified according to the life-cycle phase of the system.

1) Creation Time Self- Requirements:* When a product order arrives in the system, a suitable shop-floor layout needs to be built on the basis of the available system modules.

a) Creation of the layout (Requirement 1): Any new GAP triggers an SO process, leading to the self-selection of modules and their coalition formation. The modules and coalitions arrange themselves in the shop floor, and thus, a layout is created. If a layout already exists, it may be adapted parametrically, logically, or structurally [13], [40].

b) Transformation of the GAP into the LSAI (Requirement 2): Once the layout is formed, the GAP needs to be transformed into layout-specific assembly instructions, which the modules execute to assemble the product.

2) Production Time Self- Requirements:* During the production execution, the modules and coalitions coordinate their actions and take care of themselves and their neighbors.

a) Task coordination (Requirement 3): Tasks sequencing is done according to the LSAI. Modules coordinate their work according to the current status of the product being built. The results of each operation may be written on RFID (for questions of RFID survivability, see [42]), and the subsequent coalition can act accordingly.

Collision avoidance between the modules is also a fundamental part of the self-management process. Modules with overlapping workspace must maintain a minimum distance to each other while moving.

b) SA and self-management (Requirement 4): Whenever a failure occurs in one or more of the current modules of the system, the process may lead to three different outcomes, corresponding to the three types of adaptation: parametrical, logical, or structural. This often lead to forms of self-healing and self-optimization (for details, see [13]).

i) *Parametrical*: The current modules adapt their behavior (change speed, force, task distribution, etc.) in order to cope with the current failure, which maybe degrading performance but maintaining functionality.
ii) *Logical*: The current modules may be used in a different way, eventually requiring coalitions to be adapted accordingly. For instance, a robot may take over additional tasks, which were previously executed by other modules in order to equilibrate the work loads. The agents can do this autonomously by triggering reconfiguration process with minimal-effort preference.
iii) *Structural*: The layout may be changed at production time (addition/exchange/relocation of modules), and thus, trigger a new coalition, leading to a repaired system.

3) Classical Requirements: Some requirements are not directly related to a self-* approach, but still important for production or assembly systems. They include traceability, timely production, and safety. For more, see [13].

C. Running Example: Adhesive Tape Roller Dispenser

To illustrate the concept of SOAS, a simple product was chosen: an adhesive tape roller dispenser, short "tape roller," as shown in Fig. 2. It consists of two body parts ($Part_1$ and $Part_3$) locked by a screw ($Part_4$) and the tape roll ($Part_2$). The assembly is made on a workpiece carrier circulating on the conveyors.

For reasons of simplicity, the choice of system modules will for now be very limited: a *Z*-axis moving in a vertical direction, an *X*-axis working horizontally, and a feeder receiving screws in bulk. In reality, modules, which are available at other places in

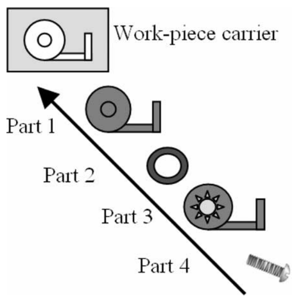


Fig. 2. Adhesive tape roller dispenser.

[1]For instance with JESS, http://www.jessrules.com.

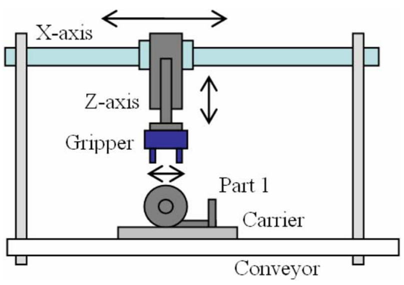


Fig. 3. Robot made of two axes and one gripper and $Part_1$ placed on a workpiece carrier circulating on the conveyor.

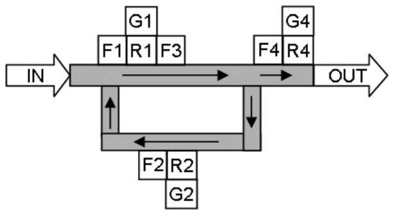


Fig. 4. Circular layout, where $Robot_1$ treats $Part_1$ and, later, $Part_3$. R: robot, G: gripper, and F: feeder.

the system, storage, or even in the system supplier's catalogue, can be considered for joining the coalitions on request. Fig. 3 shows a combination of the two robotic axes with a two-finger gripper mounted on the *Z*-axis. This configuration can be used for executing all the movements required to assemble the tape roller dispenser.

A more complete version of this running example, especially of the reactions between the different agents to form coalitions according to the GAP, is presented in [16].

1) Layout Formation: The SO process produces many different layouts and chooses one, depending on user preferences and available modules. For this running example, we consider that the layout shown in Fig. 4 is produced, with all the robots being identical and as shown in Fig. 3. Due to the layout's circular structure, workpiece carriers revisit $Robot_1$, which executes two different tasks. In this example, we assume that $Robot_1$ works faster than $Robot_2$ and $Robot_4$, and thus, assembles $Part_1$ as well as $Part_3$. *IN* represents the entry of the empty workpiece carriers and *OUT* means that the carrier with the finished product leaves the system. Feeders are placed next to robots, and robots are linked by a conveyor system.

2) Tape Roller System Scenarios: To define policies, we consider the following scenarios triggering an SO process in our assembly system. They are reconsidered in Sections VI-B, C1, and C2.

a) *New product order:* Carried by an order agent (OA), it triggers the building of the corresponding layout (actual user preferences considered: circular system and morphologically identical robots with one being faster than the others).
b) *Resilience during production:* There is a failure of some modules and reorganization of the remaining modules for building a repaired or alternative system. (Concrete incident considered: $Robot_2$ shows decreasing performance, and $Robot_1$ can take over until $Robot_2$ has been replaced.)

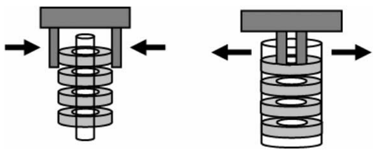


Fig. 5. Same gripper grabbing tape rolls: once fed from a stick (left) and once fed from a tube (right).

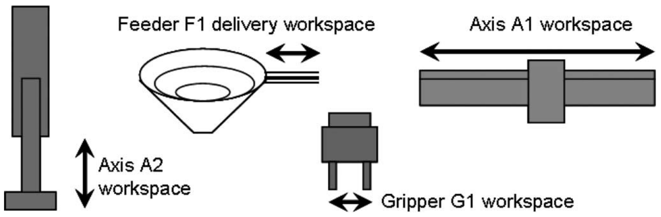


Fig. 6. Modules and workspaces (symbolic representation).

c) *Small change in tape roll conditioning:* Sometimes, little changes in conditioning or design do not need physical reconfigurations but only small modifications of the LSAI. (Concretely, instead of being delivered in a tube, the tape rolls are delivered on a stick. $Gripper_3$ will thus grab the tape roll from outside instead of inside, see Fig. 5.)
d) *Small change in product design followed by the reconfiguration of the layout:* Some modifications have bigger consequences and affect resource attribution. (Change considered: The screw will be eliminated and replaced by a snap-fit mechanism integrated in $Part_3$; as a consequence, $Robot_4$ is redundant and $Robot_1$ $(=Robot_3)$ needs a new gripper, which is able to apply a force F in the center of $Part_3$.)
e) *Optimization:* In case of difficult, long, or impossible handling paths, the system may propose layout changes based on a set of relatively simple policies.

V. Agents, Specifications, and Architecture

A. Agents and Modules

In comparison with the EAS and the corresponding multiagent software system *CoBASA* [10], SOAS require an enlarged set of agents. Each of these agents can be instantiated as often as required and according to the actual item to be represented. The agents in PROSA [43] served as an inspiration for this paper, but they were not a direct basis. For a detailed comparison, see [13].

The basic module types, which are needed for executing assembly operations are illustrated in Fig. 6: axes, grippers, and feeders. Most agents are embodied [e.g., manufacturing system modules and product parts (PartA)], but some do not have a physical body (e.g., OAs). All agents can request and/or provide services, which are based on their skills. The software part of the agents is implemented in Java Agent DEvelopment framework (JADE) [12], [18].

1) Order Agent: A product order comes into the system as an OA, asking for a certain number of instances of a specific product to be assembled within a certain deadline. An OA carries the *GAP* (see Section V-D1) given by the user, specifying in a general way how to assemble which parts. The GAP does not determine which module will execute which movement,[2] as this remains to be defined later. The GAP stays fixed, even in case of changes in the layout; only changes in the product design itself lead to changes in the GAP.

2) Product Agent: Product agents (PAs) represent the instances of the product to be made. They are launched by an OA and carry the *LSAI* (see Section V-D2), which is derived from the GAP according to the layout that has been created. The LSAI defines which module executes which movement in which order. Each PA exists until its product is finished. As the product assembly progresses along the assembly system, the PA requests the appropriate services from the various modules to perform the different steps on the assembly plan. PAs are associated with the RFID on the workpiece carriers.

3) Manufacturing Resource Agent: A manufacturing resource agent (MRA) is an agentified module that provides simple skills and has the capability to participate in coalitions. MRAs can incorporate the following items: A *robotic axis, A*, which is a module that can execute a movement along or around a certain direction (axis). Its workspace is thus linear or circular. If combined with another linear axis, their combined workspace can be square, or cylindrical in case of a rotational axis. More complex combinations have more complex workspaces: a *gripper G*, which is a device that is mounted on an axis and allows a part to be grabbed, an *industrial robot R*, such as a Delta, articulated arm robot, or Scara robot. All of them have elaborate workspaces. A *conveyor agent ConvA* transports material between locations in the system, mostly with the help of workpiece carrier agents (WPCAs) (see below). A conveyor is a typically linear transportation device. Other instances of conveyor modules are corner units and T-junctions. In addition, *cranes* may be included in this group. A *feeder agent FA* feeds parts into the system using any kind of device (vibration feeding, tubes, rails, pallets, or even manual delivery). A feeder receives the parts to be assembled and puts them at the disposal of the respective modules, which will treat them. For a *WPCA*, each carries a product on the conveyors along the assembly system and allows it to reach the MRAs.

4) Dynamic Coalition Leader Agent: As mentioned in Section III-B, MRAs can form meaningful coalitions to provide composite skills. Each coalition is represented by a dynamic coalition leader agent (DCLA). This agent does not emit commands, but it may play the role of a mediator or monitoring entity in case of difficulties.

5) Part Agents: Product parts, represented by PartAs, are delivered by the feeders and need to travel to their target position in the final assembly. There is one PartA per part type per product type (and not one agent per part).[3] PartAs collaborate with FAs to organize the delivery.

6) Ontology Agent: The ontology agent (OntA) provides controlled access to the ontology, among others offering the capability of filtering the list of possible agents that have a certain skill by the use of specific criteria. Depending on the implementation, there may be only one or several OntAs.

B. Self-Knowledge

Embodied intelligence [44] means that agents must know themselves, their physical bodies, their workspace, and their interactions with others. This implies specifying parts, modules, and assembly operations/skills in a computer-readable way. The agents carry XML files with the corresponding information, which complements the specifications given by the ontologies [12], [40].

1) Module Specifications: Modules (MRAs) need an *internal functional model* of themselves and their working space in the geometrical sense, as well as specifications of their pneumatic, electric, and electronic *interfaces*.

The services an MRA can provide are called *skills*; on the requirement side, they correspond to the processes needed to assemble a product. When modules come together, their workspaces merge. This may result both in an expansion or in a limitation; modules can give each other more freedom or constrain each other. Trajectories and workspace usage can be calculated with kinematic matrices, as taught in engineering courses, or using industrially available software. Doing this properly is crucial for collision avoidance by the means of the control software.

To facilitate the selection of the right coalition partners, each agent has a local database with *knowledge about partners*, which the agent has already worked with (successfully—*white list*—or with problems—*black list*). Similarly, modules will carry rules for forming complex skills in collaboration with their partners.

2) Part Specifications: Parts must be precisely specified, analogously to the module specifications. There is plenty of potentially useful information, including geometry, material, properties (insulating, conducting, magnetic, transparent, etc.), mass, rigidity (or stiffness), elasticity, the way of conditioning, and the way of gripping. In future, even a CAD file for every part might be included in the part specifications. This would allow graphically specifying the locations of assembly operations, gripping points, and other relevant spots.

C. Architecture

Fig. 7 illustrates the architecture of the system. The self-organizing and self-adaptive architecture that we use follows a service-oriented architecture, where components or agents provide services. It has been adapted from MetaSelf [45] and exploits dynamically collected *metadata* to support decision-making and adaptation based on the dynamic enforcement of explicitly expressed *rules* and *policies*. Metadata, rules, and policies are themselves managed by appropriate services, which are autonomous agents. The components, metadata, rules, and policies are all *decoupled* from each other and dynamically updated (or changed). Additional services to build the run-time infrastructure encompass the following: a registry/broker that handles the service descriptions and services requests supporting dynamic binding; an acquisition and monitoring service for the self-* related

[2]apart from the feeders, which are directly related to the parts.

[3]In case several product types using the same part are assembled at the same time, there is one PartA per product type, which uses the respective part.

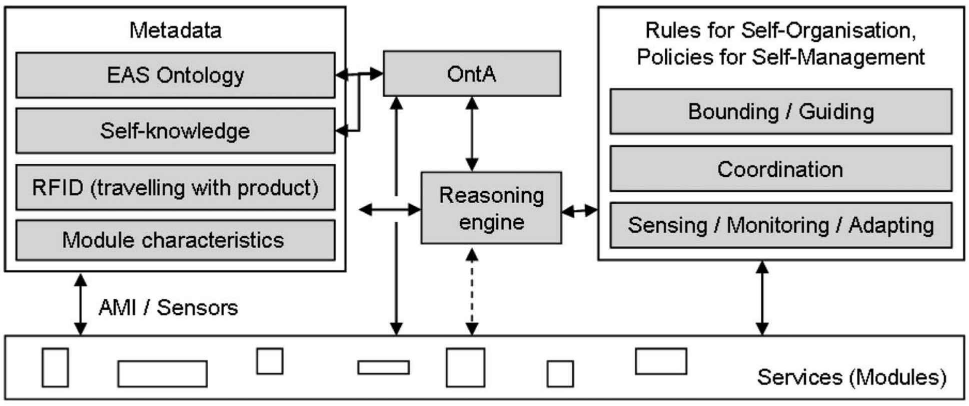


Fig. 7. System architecture.

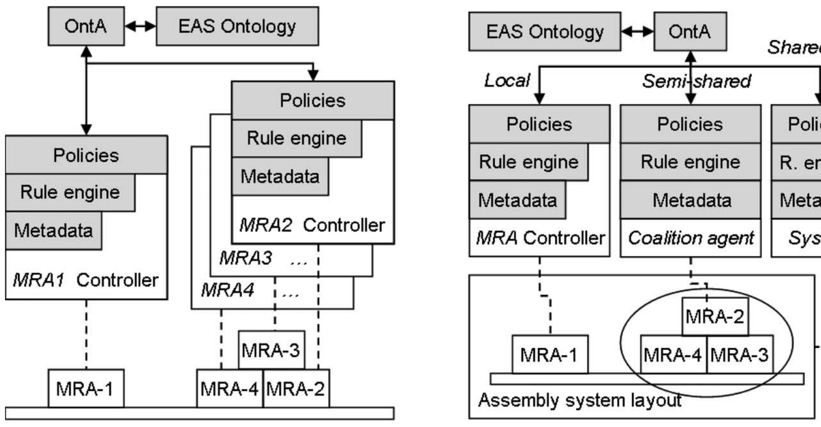


Fig. 8. (a) For every MRA, (b) for every coalition, for subsets of the system, and for the entire system.

metadata (e.g., performance); a registry that handles the policies; and a distributed reasoning tool that matches metadata values and policies and enforces the policies on the basis of metadata. Metadata is either directly modified by components or indirectly updated through monitoring. Metadata and policies cause the reasoning tool to determine whether or not an action must be taken.

All MRAs register their skills and constraints *(self-description metadata)* in the OntA. They have access to global or local *coordination metadata* (e.g., assembly status of current product item) and *resilience/self-* metadata* (e.g., current level of precision or speed of a module itself or of a partner module). Policies, to which the system as a whole has to adhere, are global to all components (for instance, "fulfill the GAP," "no MRA is allowed to move outside the allowed global workspace," or "the user favors a circular layout") or locally attached to individual components (such as "avoid collisions" or "adapt your own speed to the speed of your partner").

The local run-time software infrastructure is presented in Fig. 8(a). Besides representing the local arrangement (MRA-1), Fig. 8(b) depicts a similar organization for coalitions (semishared, in the middle of the picture, group MRA-2, MRA-3, MRA-4) and for the whole assembly system (shared, on the right-hand side). Coalitions do not have a controller (only modules have) but have metadata and policies shared by the agents in that coalition; the same applies for the system as a whole and subsystems thereof. Examples of metadata include the performance of the coalition as a whole, such as how many pieces a certain module has processed in the last 10 s. Notice that because coalitions are dynamically created in response to an incoming OA or a production condition, the corresponding policies are created or linked to the coalition on the fly. Finally, policies and metadata applying at the level of the whole system are shared by all agents.

D. Assembly Specifications

In the current proof of concept, the assembly specifications are written in Maude language [16]; when implemented, XML is likely to be chosen to express them.

1) Generic Assembly Plan: This specifies the way a product is to be assembled: It includes the assembly sequence of the different parts and the way they must be joined. Tasks are defined in the form of generic operations (equivalent to skills). The GAP does not provide information about what module to use and what movement to make. In other words, the GAP says *what* to do but not *how* and is thus independent from any layout. The goal of generically specifying the tasks is to describe operations independently from the concrete parts to be treated and independently from the resource (MRAs), which will execute

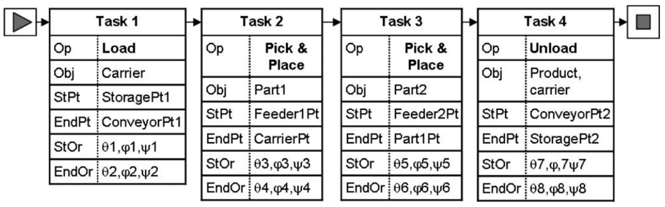


Fig. 9. Simplified tape roller GAP written as a workflow.

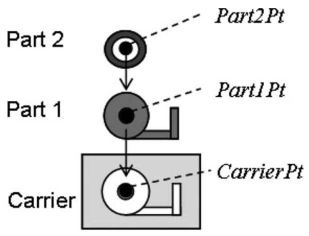


Fig. 10. Simplified running example.

the tasks. This gives the system a certain level of abstraction, and thus, the liberty to attribute any module with suitable skills to the operation in question.

Fig. 9 shows the example of a GAP for the simplified running example (assembling only parts 1 and 2, as illustrated in Fig. 10), represented as a workflow and written in XML. The four simple illustrated tasks each have an operation type (Op), an object to be handled (Obj), a start point (StPt), an end point (EndPt), as well as a start orientation (StOr) and an end orientation (EndOr), which we assume to be sufficient information at this stage of implementation. This GAP specifies that a carrier is loaded from the $Storage$ to $Conveyor_1$, then $Part_1$ is picked from $Feeder_1$ and placed on the $Carrier$, then $Part_2$ is picked from $Feeder_2$ and placed on top of $Part_1$, and finally, the $Carrier$ with the assembled product is unloaded to the $Storage$.

Notice that in the GAP, points refer to a general description of their type (for instance, a point on the feeder, which provides a certain type of part), which is then in the LSAI matched to a specific point (where the feeder is located in the layout).

2) Layout-Specific Assembly Instructions: For executing the assembly, the GAP needs to be transformed into the LSAI. This is done in collaboration between the OAs and the MRAs, and based on the created layout. The LSAI consists of executable programs for the robotic modules, which satisfy the GAP. The instructions are generated for a certain layout; if the layout is modified, these instructions must be changed. Fig. 11 shows the tape roller LSAI written as a workflow for the layout shown in the same figure. For instance, the second column of the LSAI specifies $Task_A$, which is a transport operation executed by $ConveyorCoalition_1$, to move the $Carrier$ from $ConveyorPt_1$ to $Robot_1Pt$. At the same time and in preparation of $Task_2$, $Task_{2-1}$ is a feeding operation, where $Feeder_1$ delivers $Part_1$ to $Feeder_1Pt$ with the orientations θ_3, φ_3, and ψ_3. Fig. 11 shows three inputs, represented by the flashes in the boxes on the left-hand side, and one output, represented by the square in the box on the right-hand side.

In comparison with the GAP, the LSAI contains additionally the following items: It instantiates the tasks of the GAP with concrete modules and movements and specifies *who* will execute the tasks and *how*. It contains transportation tasks in between the main tasks from the GAP. It includes feeding tasks, which provide the parts to be assembled.

VI. Self-* Mechanisms

To address the requirements detailed in Section IV-B, we propose the subsequently detailed self-* mechanisms. Section VI-A discusses the model for SO, which we used. Section VI-B explains the system at creation time and Section VI-C at production time.

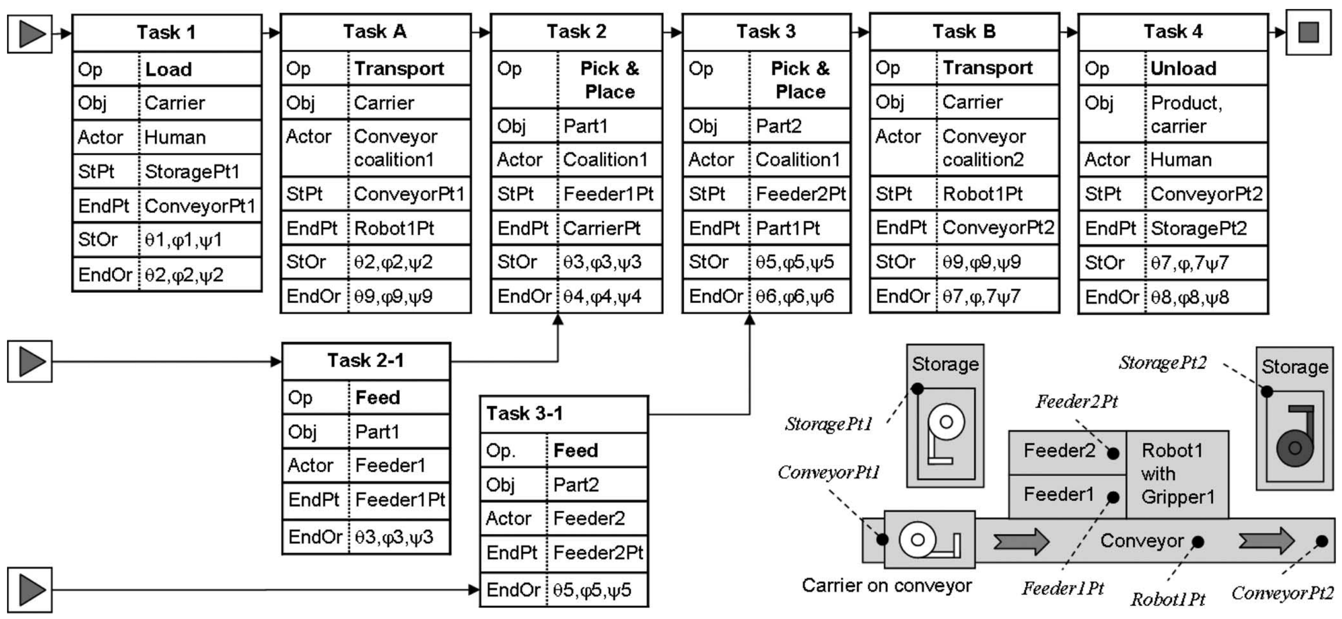
A. Model for SO in SOAS

SO refers to the ability of a system to undergo structural changes in order to cope with the ever-changing environment and the requirements resulting from it (in the context of assembly, this includes incoming assembly orders), without central command or important control from outside. It means that gradually the system will tend to acquire the ability to execute reconfigurations on its own, autonomously and automatically.[4] Physical changes need to be done by the user, but they can happen on the request of the system and according to its proper dynamics. This can be considered as a kind of *user-assisted self-assembly* on initiative of the modules, coalitions, or PAs; on the software and control side, the system can self-organize without the help of the user.

The chemical abstract machine (CHAM) [46], [47] is a design abstraction, which can be used for many different problems. It consists of a *molecule solution* and chemical reaction rules. The molecules spontaneously react with each other according to the rules; each time a rule fires, the molecules involved in the reaction are replaced (or *rewritten*) by their new composition (the outcome of the reaction). Afterward, other rules may apply, and the solution is rewritten again, and so forth, until no rule can be applied any more, and the system has converged to a stable state. The rules are applied in a concurrent and distributed way; in other words, the molecules self-assemble or self-organize.

In the case of SOAS, as illustrated in Fig. 12, the molecule solution is the set of all modules; rules are physical (possible) combinations of modules and their provided (simple or composite) skills matching requested tasks. The insertion of a product order into the solution triggers the reaction rules. As a result, modules form coalitions, according to their compatibility rules and composition pattern (compatible sizes and shapes, and combination of simple skills providing composite skills) and announce themselves for fulfilling some task specified in the product order. Just in the same way as molecules have properties of their own and potentially different ones when combined, modules also have properties of their own and potentially different ones in coalitions.

[4]A stepwise approach tending toward this goal is advisable.



Task 1	
Op	**Load**
Obj	Carrier
Actor	Human
StPt	StoragePt1
EndPt	ConveyorPt1
StOr	θ1,φ1,ψ1
EndOr	θ2,φ2,ψ2

Task A	
Op	**Transport**
Obj	Carrier
Actor	Conveyor coalition1
StPt	ConveyorPt1
EndPt	Robot1Pt
StOr	θ2,φ2,ψ2
EndOr	θ9,φ9,ψ9

Task 2	
Op	**Pick & Place**
Obj	Part1
Actor	Coalition1
StPt	Feeder1Pt
EndPt	CarrierPt
StOr	θ3,φ3,ψ3
EndOr	θ4,φ4,ψ4

Task 3	
Op	**Pick & Place**
Obj	Part2
Actor	Coalition1
StPt	Feeder2Pt
EndPt	Part1Pt
StOr	θ5,φ5,ψ5
EndOr	θ6,φ6,ψ6

Task B	
Op	**Transport**
Obj	Carrier
Actor	Conveyor coalition2
StPt	Robot1Pt
EndPt	ConveyorPt2
StOr	θ9,φ9,ψ9
EndOr	θ7,φ,7ψ7

Task 4	
Op	**Unload**
Obj	Product, carrier
Actor	Human
StPt	ConveyorPt2
EndPt	StoragePt2
StOr	θ7,φ,7ψ7
EndOr	θ8,φ8,ψ8

Task 2-1	
Op	**Feed**
Obj	Part1
Actor	Feeder1
EndPt	Feeder1Pt
EndOr	θ3,φ3,ψ3

Task 3-1	
Op.	**Feed**
Obj	Part2
Actor	Feeder2
EndPt	Feeder2Pt
EndOr	θ5,φ5,ψ5

Fig. 11. Simplified tape roller LSAI written as a workflow.

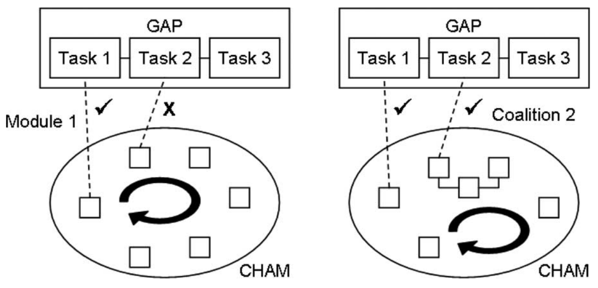


Fig. 12. Self-assembly of coalitions in CHAM.

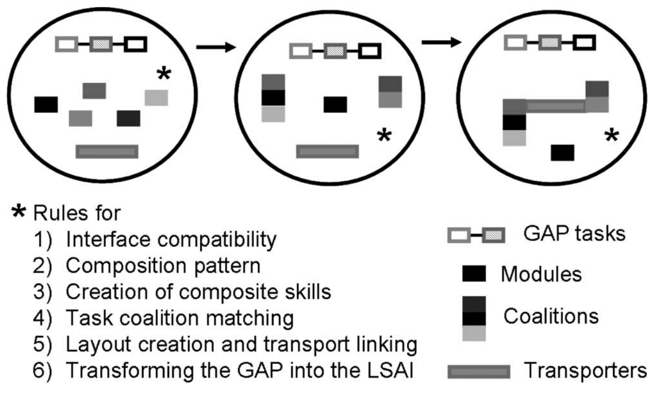

* Rules for
1) Interface compatibility
2) Composition pattern
3) Creation of composite skills
4) Task coalition matching
5) Layout creation and transport linking
6) Transforming the GAP into the LSAI
GAP tasks
Modules
Coalitions
Transporters

Fig. 13. Stepwise application of rules according to the chemical reaction model (schematic view).

B. Creation Time: Rules for SO

Reaction rules for SO apply to all modules at creation time. Fig. 13 graphically illustrates how modules progressively self-assemble to form coalitions and establish transport links in between. These coalitions are built according to a set of rules, as explained in the following, to progressively match with the tasks defined in the GAP, as shown in Fig. 12. All rules apply at all times; there is no explicitly specified order of application. However, implicitly, there is a logical order of application because of the conditions guarding the rules. Currently, these rules exist as a proof of concept in Maude [16] and lead to a valid solution. Implementation in a multiagent system is future work.

1) Rules: Addressing the Requirements 1 and 2 detailed in Section IV-B.

a) Interface compatibility: Modules match according to the types and characteristics of their physical interfaces. In reality, interfaces are often brand-specific and module-type-specific, and the compatibility between nonrelated modules is limited, thus avoiding combinatorial explosion. The number of interfaces a module has determines the maximal number of connections it may establish, e.g., due to physical limitations, a gripper is unable to connect to more than one axis at once.

b) Composition pattern: Typical basic combinations of modules are specified as composition pattern. For instance, the combination of three linear axes makes a Cartesian robot, and three rotational axes make an articulated arm robot, whereas a linear and a rotational axis build a robot with a cylindrical workspace. In cases, where many modules are available, such rules can help to reduce the search space by indicating which module combinations typically occur; corresponding coalitions would be formed with priority. This type of rule can be implemented either using module types or skills (in this paper, we chose the skills).

c) Creation of composite skills: Composite skills are combinations of simple skills, which come together when module coalitions are created. Most skill combinations lead to *immediate composite skills*, which are a direct combination (addition) of the simple skills. As an example, $\mathtt{move}(\mathtt{x})$ and $\mathtt{move}(\mathtt{y})$ can result in $\mathtt{move}(\mathtt{x}+\mathtt{y})$. However, some combinations also lead to the emergence of *additional composite skills*. For instance, a gripper has the skills $\mathtt{open}-\mathtt{close}$, and an axis has `move`. Together, they have $\mathtt{open}-\mathtt{close}$, `move`, and `pick&place`.

d) Task coalition matching: When a GAP is introduced, the modules' skills react with the operations requested by the tasks in the GAP. At the same time, the modules react with suitable partner modules, according to their own requirements (interface compatibility and necessary partners, like a gripper needs to be hold by an axis or a robot) as well as the corresponding composition pattern, previously explained. Once a coalition is formed, a corresponding *DCLA* (explained in Section V-A4) is inserted into the chemical solution. The coalition, again, when suitable reacts with other tasks and modules. This procedure continues until all the tasks have reacted with modules/coalitions, or until no more modules are available. In this case, the task will emit a user alert after a certain time because no satisfactory solution could be found. If a task reacts with more than one coalition (which will most often be the case), a selection may be made according to various criteria, such as a user preference or an optimization function. A coalition, which, in the process of forming itself, cannot find any suitable partner in the module pool, will be discarded. The probability of exactly the same (fruitless) coalition being built again is relatively low and gets smaller with an increasing number of available modules. Unsuccessful compositions may be recorded.

e) Layout creation and transport linking: Generic layout creation rules specify how MRAs and coalitions react with places in the layout and with conveyors to establish transportation links in between the robots. The coalition that was assigned to the first task chooses its place first (random or default position). Coalitions that come later place themselves at default distances from other coalitions, and ask for transportation skills to move the product from the previous robot to the current. Conveyors will react to such requests and provide the necessary services. A continuous path from IN to OUT must be formed.

f) Transforming the GAP into the LSAI (rewriting): The GAP now needs to be transformed into specific instructions for the selected modules/coalitions in their respective positions; this is the content of the LSAI. The CHAM rewriting technique will be exploited. Commercially available solutions for virtual engineering [48] allow the designer to simulate the workspace and verify if a certain point is reachable, to identify potential collisions and kinematic singularities, and to calculate trajectories, velocities, cycle times, and the stress on robot mechanics.

C. Production Time: Policies and Metadata for Self-Management

The following examples of production time policies and metadata refer to the running example. For more details about policy types and classes and metadata types, see [13].

1) Coordination at Production Time: Policies and dynamically updated metadata, which help the system to coordinate its actions at production time, addressing the Requirement 3 detailed in Section IV-B, are subsequently explained.

a) *Policies: Task sequencing policies* assure the well-ordered execution of the tasks specified in the GAP and the derived LSAI. A task can start when the WPCA (carrier) has reached its working position next to the robot. The WPCA can move on as soon as the robot has finished its operation. *Collision avoidance policies* help avoid crashes, e.g., "the distance MRAs must always be bigger than a safety threshold, unless they are members of the same coalition."
b) *Metadata: Task sequencing metadata* supports the application of task sequencing policies, e.g., for each PA: current status of LSAI execution—the result of any operation is always written into the product's RFID (success, failure, problems, etc.) and for each MRA: status/availability (idle, reserved, working, failure, and in storage). A log file stores the performed operations history (for traceability purposes). *Collision avoidance metadata* supports the application of collision avoidance policies, e.g., for all MRAs: occupancy of its workspace by other MRAs (e.g., feeders), including a list of possible undue items (e.g., a loose screw is lying on the workspace).
c) *Tape roller system scenario:* 1) *Task sequencing:* A WPCA circulates along the layout, and the PA notices that it passes twice next to $Robot_1$. According to its LSAI, it asks for the appropriate operation (adding $Part_1$ or $Part_3$). Similarly, $Robot_1$ accesses the RFID to read the previous and to write the result of the current task. 2) *Collision avoidance:* The workspaces of $Robot_1$, $Robot_2$, and $Robot_4$ do not interfere in this particular example; no danger of collision between them. Whenever a collision sensor detects something irregular in the workspace (a human hand or a loose screw), the operation is stopped until the intrusion has gone. If this takes more than a given time, the user is informed.

2) SA and Self-Management at Production Time: The metadata and policies required for SA at production time, addressing the Requiremen4 detailed in Section IV-B, are described here.

a) *Policies: Self-optimization policies* help MRAs improve their performance, e.g., "adapt the feeding speed to the part removal speed." (Feeders always need to deliver parts at their output. If the parts are taken away quickly, their feeding speed should be high.) *Self-healing policies* help to assure the good health of the system. It is necessary to report the state or, respectively, the result of collaboration with other MRAs. Every agent monitors its own state, especially with reference to critical parameters, and alerts the user in case of problems, e.g., MRAs need to monitor their own precision, and eventually also their neighbor's precision, in order to detect the effects of fatigue or other kinds of disturbances, and to take corresponding countermeasures. *System-level policies (bounding policies)*, e.g., "if one (or more) failure occur more than a certain number of times in a certain period of time, then alert the user."
b) *Metadata: Self-* metadata* supports performance parameters. For each MRA: maximal/optimal speed of operation, precision of movements on every MRA's own axes and its partners'/neighbors' axes, current queuing level of arriving products, and quality of assembled product (any measurable characteristic or feedback given by the user).
c) *Tape roller system scenario:* This was introduced in Section IV-C2. *SA:* $Robot_2$ experiences problems in reaching its target positions and asks for maintenance. As a

temporary solution, $Robot_1$ is asked to take over, and the user is asked to place $Feeder_2$ close to $Robot_1$.

d) *Reconfiguration triggered:* After taking over from $Robot_2$, $Robot_1$ experiences a high level of queuing. This leads $Robot_1$ to ask for a reconfiguration and, therefore, triggers the creation time mechanism.

VII. Agility Scenarios and Performance Metrics

As SOAS have not been fully implemented yet, it is not possible to measure their performance and compare it with the performance of other agile assembly systems. This is why, at this moment, we suggest a series of metrics, which can be used once the implementation has been completed.

It is difficult to express the performance of an SOAS in quantitative terms, as the goal is not to optimize the global throughput as it is for traditional assembly systems. Evolvability counts much more here, and again, is difficult to quantify. Furthermore, we are not yet able to realistically measure and compare SOAS performance with the performance of other approaches, as the project has so far not reached the stage, where implementation on a pseudoindustrial system can be tackled. Instead, we consider here which measurable factors will be considered in the future.

A successful assembly system is one, which correctly assembles the required products within the desired time span. Additionally, a successful *agile* assembly system is the one, which can assemble different types of products at the same time, which can evolve from one system form to another within a short time frame, which can cope with failures and perturbations, which requires minimal user interaction, and which uses resources in a sustainable way.

Within this context, sustainability includes 1) socioeconomic dimensions, i.e., keeping jobs and industries within Western Europe instead of losing them due to offshoring; 2) monetary dimensions, which means making investments in a way that they will not be wasted if the market situation changes; as well as 3) technological dimensions: Modules are reusable, legacy equipment can be integrated into newer system, in case of perturbations, production can be maintained in degraded modes, and the system takes care of itself.

A. Creation-Time-Related Metrics

To judge the validity of the SO approach suggested in this paper, it is necessary to define how exactly the effort for layout design and system configuration/reconfiguration may be measured. Is it how long a human operator works on it? Or how specialized that operator needs to be? Does mechanical work done by an operator with basic training count less than reprogramming by a specialist?

Besides these open issues, we suggest that for an SOAS, it is relevant to know: How often did a human need to intervene, while the system layout is being created? Were the agents able to find a suitable layout? Were the agents able to derive an LSAI, which correctly satisfies the GAP? When reconfiguration is required, how many different reconfigurations were proposed, and how many were accepted? How high or low is the "degree of external control" for system creation? How much global complete knowledge was needed?

B. Production-Time-Related Metrics

Most commonly available performance metrics concern the system at production time. For instance, Brueckner [49] suggests the following criteria for distributed manufacturing systems: load of processing stations and buffers, work in progress (should be minimized), local and global throughput values, communication and computation load; required human interference, effort for design and implementation, and effort for reconfiguration.

For an SOAS, it is relevant to additionally know: How often did a human need to intervene during production? How often did the system experience emergency stops? For how long was the system down? Were there any rashes or other catastrophes? How high or low is the "degree of external control" at production time? How much global complete knowledge was needed? How sustainable is the use of resources? (How well can legacy components be used, and how high is the utilization rate of the robots?) Could the system recover from failures on its own? To what failures is the system robust?

C. Cost of Reconfiguration

To evaluate the *cost of reconfiguration*, one may consider the following points: Judge the quality and complexity of a reconfiguration proposal. Decide about which system reconfiguration proposal to accept. Find out how easy it is to remove other modules, which would be in the way of the new modules. Take the new module from the storage or out of an old place in the layout. Arrange the new installation: geometric, electric, electronic/computational, pneumatic, etc., interfaces to be connected. Verify that the agents autonomously integrate the new module.

It is not within the scope of this paper to conclusively answer the questions of how much evolvability or SO cost or if it is really worth paying that much and, if yes, for producing which products exactly. Within the EUPASS project,[5] models for estimating cost of flexible microassembly were developed [50]. Adapting them to SOAS will be future work.

D. Agility Scenarios

To realistically compare the performance of an SOAS with the performance of other systems, certain aspects of specific agility scenarios should be compared.

Creation of a suitable layout to assemble a given product: Which steps does selecting suitable modules and composing a suitable layout to assemble a given product include? How much effort does this require? How many mistakes happen?

Programming of the modules/robotic cells/work stations: How are the modules controlled and coordinated? How work-intensive and error-prone is the programming? How easily can it be adapted to changes in the layout, the parts, or processes?

[5]http://www.hitech-projects.com/euprojects/eupass/index.htm

System management during production: How much human supervision is required and for which aspects? How is maintenance handled, and how many perturbations occur per time unit? How are they handled? How are movement accuracy and repeatability monitored and improved if necessary?

System recovery in case of module failures: How does the rest of the system react when one, two, or several modules experience partial or complete failures? Is the system able to find alternative ways of executing a task, or will it only stop?

Minor system reconfigurations: What is necessary to execute minor changes in the system, which may be due to a product design change or due to different feeding of a part? How much user interaction is required, and how complicated is it?

Major system reconfigurations: How does the system handle major reconfigurations? Is it easier/quicker to start afresh, or can an existing system be modified?

Further discussion about the nature and characteristics of performance metrics for distributed manufacturing systems as well as their mathematical expressions can be found in [21], together with further agility parameters to be considered.

VIII. Conclusion and Outlook

This paper discusses the concept of SOAS, their design rules, as well as their run-time architecture and policies. SOAS are composed of agentified modules, which carry thorough self-knowledge and engage in collaborations with peers to fulfill their tasks.

At *creation time*, the modules self-organize according to a set of rules. Each module offers simple skills; modules self-assemble to form coalitions and can thus offer composite skills to fulfill the assembly requirements specified in the GAP. The coalitions arrange themselves in the shop-floor layout, and the *LSAIs* are derived, specifying which module is going to execute what movement.

At *production time*, the modules are asked to assemble the product according to policies for SA and self-management. This includes monitoring of themselves and their peers and reconfigurations of minor scope. A need for major reconfiguration leads the system back to the creation phase.

Although implementation is still on-going, SOAS can already be identified as a promising approach. Besides providing a solution for tomorrow's agile manufacturing, SOAS may also be helpful for other applications of MetaSelf [45], the architecture for SO and SA, which we used. The MetaSelf architecture is now accompanied by a design method, which added valued to the architecture [51].

The implementation of the SOAS concepts and architecture presented in this paper are either already implemented (EAS ontology, OntA, agents, and dynamic coalitions) or currently being implemented (policies and metadata, reasoning with JESS, self-assembly in CHAM, and proofs of properties with Maude). The production time infrastructure will be addressed soon.

Future work: A mathematical model of the system already exists, whereas the formal proofs of system properties are still being considered. Furthermore, our next steps encompass the following: defining more sophisticated optimization policies, as the existing ones are very basic; investigating performance metrics for SOAS, including realistic metrics to compare SOAS with other approaches to agile manufacturing; considering how more complicated skills, such as *insert* and *press-fit/snap-fit*, can be treated. Strategies to deal with conflicting policies must be investigated. It still remains open how exactly the SA/self-management mechanisms at production time will work (monitoring, sensing, and coordination of workspaces). A prototype will be built.

References

[1] J. A. Buzacott and J. G. Shanthikumar, "Models for understanding flexible manufacturing systems," *Trans. Amer. Inst. Elect. Eng.*, vol. 12, no. 4, pp. 339–350, 1980.
[2] R. Kaula, "A modular approach toward flexible manufacturing," in *Integr. Manuf. Syst.*, West Yorkshire, U.K.: MCB Univ. Press, 1998, pp. 77–86.
[3] M. Onori and P. Groendahl, "MARK III: A new approach to high-variant, medium-volume flexible automatic assembly cells," *Robotica, Spec. Issue*, vol. 16, no. 3, pp. 357–368, 1998.
[4] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritchow, A. Ulsoy, and H. Van Brussel, "Reconfigurable manufacturing systems," *CIRP Ann.—Manuf. Technol.*, vol. 48, no. 2, pp. 6–12, 1999.
[5] M. Mehrabi, A. Ulsoy, and Y. Koren, "Reconfigurable manufacturing systems: Key to future manufacturing," *J. Intell. Manuf.*, vol. 11, pp. 403–419, 2000.
[6] H. ElMaraghy, "Flexible and reconfigurable manufacturing systems paradigms," *Int. J. Flexible Manuf. Syst.*, vol. 17, no. 4, pp. 261–276, 2006.
[7] R. Katz, "Design principles for reconfigurable machines," *Int. J. Adv. Manuf. Technol.*, vol. 34, pp. 430–439, 2007.
[8] H. ElMaraghy, T. AlGeddawy, and A. Azab, "Modelling evolution in manufacturing: a biological analogy," *CIRP Annals—Manuf. Technol.*, vol. 57, pp. 467–472, 2008.
[9] M. Onori, "Evolvable assembly systems—A new paradigm?," in *Proc. 33rd Int. Symp. Robot.*, Stockholm, Sweden, 2002, pp. 617–621.
[10] J. Barata, *Coalition Based Approach for Shop Floor Agility*. Amadora, Lisboa: Edições Orion, 2005.
[11] R. Frei, G. Di Marzo Serugendo, and J. Barata, "Designing self-organization for evolvable assembly systems," in *Proc. IEEE Int. Conf. Self-Adaptive Self-Organizing Syst.*, Venice, Italy, 2008, pp. 97–106.
[12] R. Frei, N. Pereira, J. Belo, J. Barata, and G. Di Marzo Serugendo, "Implementing self-organisation and self-management in evolvable assembly systems," in *Proc. IEEE Int. Symp. Ind. Electron.*, Bari, Italy, 2010, pp. 3527–3532.
[13] R. Frei, "Self-organisation in evolvable assembly systems" Ph.D. dissertation, Dept. Elect. Eng., Faculty Sci. Technol., Univ. Nova de Lisboa, Lisbon, Portugal, 2010.
[14] C. D. Nugent, D. D. Finlay, P. Fiorini, Y. Tsumaki, and E. Prassler, "Editorial, home automation as a means of independent living," *IEEE Trans. Autom. Sci. Eng.*, vol. 5, no. 1, pp. 1–9, Jan. 2008.
[15] R. Frei, B. Ferreira, G. Di Marzo Serugendo, and J. Barata, "An architecture for self-managing evolvable assembly systems," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, San Antonio, TX, 2009, pp. 2707–2712.
[16] R. Frei, G. Di Marzo Serugendo, and T. Serbanuta, "Ambient intelligence in self-organising assembly systems using the chemical reaction model," *J. Ambient Intell. Humanized Comput.*, vol. 1, no. 3, pp. 163–184, 2010.
[17] R. Frei, G. Di Marzo Serugendo, and J. Barata, "Designing self-organization for evolvable assembly systems," School Comput. Sci. Inf. Syst., Birkbeck College, London, U.K., Tech. Rep. BBKCS-09-04, 2009.
[18] R. Frei, B. Ferreira, and J. Barata, "Dynamic coalitions for self-organizing manufacturing systems," presented at the CIRP Int. Conf. Intell. Comput. Manuf. Eng., Naples, Italy, 2008.
[19] P. Valckenaers and H. Van Brussel, "Holonic manufacturing execution systems," *CIRP Ann.—Manuf. Technol.*, vol. 54, no. 1, pp. 427–432, 2005.
[20] V. Marik, V. Vyatkin, and A. Colombo, Eds., Holonic and Multi-Agent Systems for Manufacturing. Heidelberg, Germany: Springer-Verlag, 2007.
[21] P. Leitão, "An agile and adaptive holonic architecture for manufacturing control," Ph.D. dissertation, Dept. Elect. Eng., Polytechnic Inst. Bragança, Bragança, Portugal, 2004.

[22] P. Leitao and F. Restivo, "Implementation of a holonic control system in a flexible manufacturing system," *Trans. Syst., Man, Cybern. C: Appl. Rev.*, vol. 38, no. 5, pp. 699–709, 2008.

[23] A. Rizzi, J. Gowdy, and R. Hollis, "Agile assembly architecture: An agent based approach to modular precision assembly systems," in *Proc. Int. Conf. Robot. Autom.*, Albuquerque, NM, 1997, pp. 1511–1516.

[24] T. Gaugel, M. Bengel, and D. Malthan, "Building a mini-assembly system from a technology construction kit," *Assembly Autom.*, vol. 24, no. 1, pp. 43–48, 2004.

[25] C. Hanisch and G. Munz, "Evolvability and the intangibles," *Assembly Autom.*, vol. 28, no. 3, pp. 194–199, 2008.

[26] M. Pflueger and M. Bengel, "Automatisch rekonfiguriert," *Comput. Autom.*, vol. 1, pp. 38–41, 2009.

[27] M. Vallée, H. Kaindl, M. Merdan, W. Lepuschitz, E. Arnautovic, and P. Vrba, "An automation agent architecture with a reflective world model in manufacturing systems," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*. Piscataway, NJ: IEEE Press, 2009, pp. 305–310.

[28] S. Lemaignan, A. Siadat, J. Dantan, and A. Semenenko, "MASON: A proposal for an ontology of manufacturing domain," in *Proc. IEEE Workshop Distrib. Intell. Syst.: Collective Intell. Appl.*. Piscataway, NJ: IEEE Comput. Soc. Press, 2006, pp. 195–200.

[29] P. Vrba, M. Radakovič, M. Obitko, and V. Mařík, "Semantic extension of agent-based control: The packing cell case study," in *Proc. 4th Int. Conf. Ind. Appl. Holonic Multi-Agent Syst.*. Berlin/Heidelberg, Germany: Springer-Verlag, 2009, pp. 47–60.

[30] Y. Al-Safi and V. Vyatkin, "An ontology-based reconfiguration agent for intelligent mechatronic systems," in *Holonic and Multi-Agent Systems for Manufacturing* (Lecture Notes in Computer Science), V. Marik, A. Vyatkin, and V. and Colombo, Eds. Berlin/Heidelberg, Germany: Springer-Verlag, 2007, vol. 4659, pp. 114–126.

[31] A. Lueder, J. Peschke, T. Sauter, S. Deter, and D. Diep, "Distributed intelligence for plant automation based on multi-agent systems: the PABADIS approach," *Prod. Plann. Control*, vol. 15, no. 2, pp. 201–212, 2004.

[32] L. Monostori, J. Vancza, and S. Kumara, "Agent-based systems for manufacturing," *CIRP Ann.—Manuf. Technol.*, vol. 55, no. 2, pp. 697–720, 2006.

[33] W. Shen, Q. Hao, H. J. Yoon, and D. H. Norrie, "Applications of agent-based systems in intelligent manufacturing: an updated review," *Adv. Eng. Informat.*, vol. 20, pp. 415–431, 2006.

[34] U. Rossgoderer, C. Woenckhaus, G. Reinhart, and J. Milberg, "A concept for automatical layout generation," in *Proc. IEEE Int. Conf. Robot. Autom.*, Nagoya, Japan, 1995, pp. 800–805.

[35] Y. Tu and H. Holm, "Automatic determination of operation sequence for material handling and equipment set-up in one-of-a-kind production," *Int. J. Comput. Integr. Manuf.*, vol. 10, no. 6, pp. 435–445, 1997.

[36] R. Webbink and S. Hu, "Automated generation of assembly system-design solutions," *IEEE Trans. Autom. Sci. Eng.*, vol. 2, no. 1, pp. 32–39, Jan. 2005.

[37] S. Benjaafar, S. Heragu, and S. Irani, "Next generation factory layouts: Research challenges and recent progress," *Interfaces*, vol. 32, no. 6, pp. 58–77, 2002.

[38] C. Song, X. Guan, Q. Zhao, and Y.-C. Ho, "Machine learning approach for determining feasible plans of a remanufacturing system," *IEEE Trans. Autom. Sci. Eng.*, vol. 2, no. 3, pp. 262–275, Jul. 2005.

[39] J. Li, "Overlapping decomposition: A system-theoretic method for modeling and analysis of complex manufacturing systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 2, no. 1, pp. 40–53, Jan. 2005.

[40] N. Lohse, "Towards an ontology framework for the integrated design of modular assembly systems," Ph.D. dissertation, School Mech. Mater. Manuf. Eng., Faculty Eng., Univ. Nottingham, Nottingham, U.K., 2006.

[41] M. Goodrich, D. Olsen, J. Crandall, and T. Palmer, "Experiments in adjustable autonomy," in *Proc. IJCAI Workshop Autonomy, Delegation Control: Interact. Intell. Agents*, Seattle, WA, 2001, pp. 1624–1629.

[42] Y. Zuo, "Survivable RFID systems: Issues, challenges, and techniques," *IEEE Trans. Syst., Man, Cybern. C: Appl. Rev.*, vol. 40, no. 4, pp. 406–418, Jul. 2010.

[43] J. Wyns, "Reference architecture for holonic manufacture: The key to support evolution and reconfiguration," Ph.D. dissertation, Katholieke Univ. Leuven, Leuven, Belgium, 1999.

[44] R. Pfeifer and C. Scheier, *Understanding Intelligence*. Cambridge, MA: MIT Press, 1999.

[45] G. Di Marzo Serugendo, J. Fitzgerald, A. Romanovsky, and N. Guelfi, "Metaself: A framework for designing and controlling self-adaptive and self-organising systems," School Comput. Sci. Inf. Syst., Birkbeck College, London, U.K., Tech. Rep. BBKCS-08-08, 2008.

[46] G. Berry and G. Boudol, "The chemical abstract machine," *Theor. Comput. Sci.*, vol. 96, no. 1, pp. 217–248, 1998.

[47] J.-P. Banâtre, P. Fradet, and D. Le Métayer, "Gamma and the chemical reaction model: Fifteen years after," in *Proc. Workshop on Multiset Processing* (Lecture Notes in Computer Science). New York: Springer-Verlag, 2000, vol. 2235, pp. 17–44.

[48] M. Garstenauer, "Das virtuelle engineering," *Comput. Autom.*, vol. 9, pp. 24–26, 2009.

[49] S. Brueckner, "Return from the ant—Synthetic ecosystems for manufacturing control," Ph.D. dissertation, Inst. Comput. Sci., Humboldt-Univ., Berlin, Germany, 2000.

[50] M. Oulevey, P. Roduit, S. Koelemeijer Chollet, J. Jacot, P. Ryser, and K. Taferner, "A cost model for flexible high speed assembly lines," presented at the Int. Precision Assembly Semin., Bad Hofgastein, Austria, 2004.

[51] G. Di Marzo Serugendo, J. Fitzgerald, and A. Romanovsky, "Metaself—An architecture and development method for dependable self-* systems," in *Proc. Symp. Appl. Comput.*, Sion, Switzerland, 2010, pp. 457–461.

Regina Frei received the M.Sc. degree in microengineering from the Swiss Federal Institute of Technology in Lausanne, Lausanne, Switzerland, and the Ph.D. degree from the Department of Electrical Engineering, Faculty of Sciences and Technology, New University of Lisbon, Lisbon, Portugal.

She is currently a Postdoctoral Researcher with the Intelligent Systems and Networks Group, Department of Electrical and Electronic Engineering, Imperial College London, London, U.K. Her research interests include self-organizing assembly systems, self-* properties, and complexity engineering.

Giovanna Di Marzo Serugendo (M'07) received the M.Sc. degree in computer science and mathematics from the University of Geneva, Geneva, Switzerland, and the Ph.D. degree in software engineering from the Swiss Federal Institute of Technology in Lausanne, Lausanne, Switzerland.

She is currently a Professor and a Lecturer/Senior Researcher with the Object Systems Group, University of Geneva. From 2005 to 2010, she was a Lecturer with the School of Computer Science and Information Systems, Birkbeck College, University of London, London, U.K. Her research interests include self-assembly of software, self-organizing, and self-adapting systems realized using specification carrying code and reputation systems, mobile agents systems, and formal methods.

Dr. Serugendo was the Cofounder of the IEEE International Conference on Self-Adaptive and Self-Organising Systems. She is also the Editor-in-Chief of the Association for Computing Machinery's *Transactions on Autonomous and Adaptive Systems*.