Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems

Designing Self-Organization for Evolvable Assembly Systems

Regina Frei
New University of Lisbon
Portugal
regina.frei@uninova.pt

Giovanna Di Marzo Serugendo
Birkbeck College
University of London, UK
dimarzo@dcs.bbk.ac.uk

Jose Barata
New University of Lisbon
Portugal
jab@uninova.pt

Abstract

Current solutions for industrial manufacturing assembly systems do not suit the needs of Mass Customization industry, which is facing low production volumes, many variants and rapidly changing conditions. This paper proposes the concept of Self-Organizing Evolvable Assembly Systems, where assembly system modules and product parts to be assembled self-organize and self-adapt (among others, choose their coalition partners, their location and monitor themselves) in order to easily and quickly produce a new or reconfigured assembly system each time a new product order arrives or each time a failure or weakness arises in the current assembly system. This paper presents the design and partial implementation of such a system following an architecture for self-organizing and self-adaptive systems based on policies enforced at run-time on the basis of collected and updated metadata. As a case study, the assembly of a Adhesive Tape Roller Dispenser is considered.

1. Challenges in Manufacturing

In the era of Mass Production, companies needed optimal solutions to produce large quantities of an identical product as fast and as cheap as possible; it was worth paying the big investments for custom-made installations, which would be disposed of once the product is out of production. In the best case some of the equipment could be reused but needed "manual" re-programming. Any change (even small) in the product design or any failure of a module implied halting the production and eventually re-programming the system, which is a work- and time-intensive as well as error-prone procedure.

Nowadays, the market tends increasingly towards Mass Customization, meaning that clients like to individually select from various product options. Companies require high responsiveness and the ability to cope with a multitude of conditions. Especially assembly is a critical area: due to the high salaries in the Western Hemisphere, companies often have to off-shore manual work to low-wage countries and run the risk of not only losing jobs but also knowledge and finally the entire business. The only alternative is automation. Robotic assembly systems have to become able to cope with such dynamic production conditions: *frequent changes*, *low volumes* and *many variants*. Industrial systems need to be quickly reconfigurable, following a Plug&Play approach and avoiding time- and work-intensive re-programming. They need to perform according to specifications and have to maintain productivity also under perturbations.

Evolvable Assembly Systems (EAS) [8] specifically target the challenges encountered in Mass Customization industry. So far, EAS were based on a multi-agent control solution called CoBASA [1], operational at Uninova, Portugal. EAS are not yet self-organizing - configuration of the agents, their positioning into an appropriate assembly system layout, monitoring and re-configuration in case of failure are still done manually. Applying an architecture for Self-Organization and Self-Adaptation is a fundamental step towards realizing true self-organizing evolvable assembly systems. A generic architecture for self-organizing and self-adaptive systems has been proposed in [5]. This architecture specifically addresses systems working under dynamically changing conditions. It encompasses coordination of work among autonomous components, seamless addition or removal of components at any time, and failures of various types, while still maintaining a predictable level of dependability. These are exactly the characteristics manufacturing systems need to cope with in dynamic production situations. The focus of this paper is on describing a design accommodating *both* self-organizing and self-adapting issues in an industrial assembly system.

Section 2 and 3 discuss respectively the notion of Evolvable Assembly Systems and our case study: the assembly of an Adhesive Tape Roller Dispenser. Section 4 briefly presents the generic self-organizing architecture. Section 5 explains the design principles used for creating Self-Organization in EAS. Section 6 reports implementation aspects. Finally, Section 7 discusses related works.

978-0-7695-3404-6/08 $25.00 © 2008 IEEE
DOI 10.1109/SASO.2008.20

2. Evolvable Assembly Systems

An **assembly system** is an industrial installation able to receive parts and join them in a coherent way to form the final product. It 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. An **Evolvable Assembly System (EAS)** is an assembly system which can co-evolve together with the product and the assembly process; it can easily undergo small and big changes in product design and seamlessly integrates new modules independently from their brand or model. Modules carry tiny controllers for local intelligence. Thanks to wrappers, every module is an agent, forming a homogeneous society with the others, despite their original heterogeneity (nature, type and vendor). Each module is carrying self-knowledge information about its physical reality, especially its workspace (the portion of the space that the module uses when in action / that is accessible by the module), its interfaces and its skills (the capabilities of the module). Several modules together can dynamically form a coalition in order to offer complex skills.

Basic module types which are needed for executing assembly operations are as follows (see Figure 1): An **axis** is a module which can execute a movement along or around a certain direction (axis). Its workspace is thus linear or circular. If combined with other linear axis, their combined workspace can be square, or cylindrical in case of a rotational axis. A **gripper** is a device which is mounted on an axis and allows to grab a part, either with its fingers (mostly 2-3 of them), by aspirating it or by activating an electric magnet. A **feeder** is a device which receives the parts to be assembled and puts them at disposal of the respective modules which will treat them. For instance, tape rolls are contained in a tube and pushed upwards, where a robot can grip them. In case of screws, they are put into a vibrating bowl (see Fig 1), which - due to the vibrations - delivers them well-aligned on a rail, where a robot can pick them. A **conveyor** is a typically linear transportation device consisting of several modules which can be arranged to move work-piece carriers, or sometimes loose parts. Other instances of conveyor modules are corner units and T-junctions.

The modules, even though actively transporting parts while assembling product parts, cannot physically move themselves without human assistance from one location to another across the assembly system. Modules ask a human operator to move them.

2.1. EAS and Self-Organisation

Our goal is to develop *self-organizing evolvable assembly systems*, i.e. given a specified product order provided in input, the system's modules spontaneously select each other (preferred partners) and their position in the assembly system layout. They also program themselves (micro-instructions for robots movements). The result of this self-organizing process is a new or reconfigured assembly system that will assemble the ordered product. In the self-organizing jargon, the appropriate assembly system *emerges* from the self-organization process going on among the different modules of the assembly system. Any new product order (seen as a global goal) triggers the self-organizing process, which eventually leads to a new appropriate system - there is no central entity, modules progressively aggregate to each other in order to fulfil the product order. This automated process does not stop at the layout formation. During production time, whenever a failure or weakness occurs in one or more of the current elements of the system, the process may lead to two different outcomes: the current modules adapt their behavior (change speed, force, task distribution, etc) in order to cope with the current failure, eventually 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).

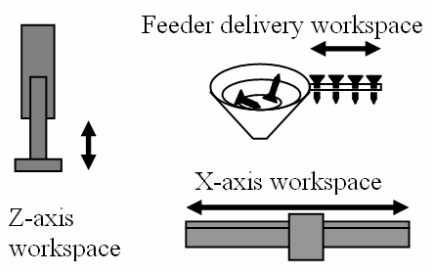

Figure 1. Selection of modules

A **Self-Organizing Evolvable Assembly System** is thus an EAS with two additional characteristics: 1) modules *self-organize* to produce an appropriate layout for the assembly and 2) the assembly system as a whole *self-adapts* to production conditions.

2.2. Agents and infrastructure

EAS consist of a set of different agent types explained in this section; each of them can be instantiated as often as required and according to the actual item to be represented.

Manufacturing system modules are agentified and we thus speak about **manufacturing resource agents**. This category of agents encompasses among others robotic modules, conveyors, work-piece carriers, feeders and axes.

A product order comes into the system as an **order**

agent, asking for a certain number of instances of a specific product to be assembled within a certain deadline. An order agent carries the assembly plan instructions, called **Generic Assembly Plan**, specifying in a general way how to assemble which parts. The generic assembly plan says *what* to do but not *how* and is thus independent from any layout; only changes in the product design itself lead to changes in the general assembly plan.

In order to build the actual assembly, the generic assembly plan needs to be transformed into **Layout-Specific Assembly Instructions**, which are in fact executable programs for the robotic modules, also called "micro-instructions". Using the Dynamic Map (see below) as a support, the different agents - involved in the layout - transform the generic assembly plan into layout-specific assembly instructions. These instructions specify which module executes what movement in which order; they are generated for a certain layout - if the layout is modified, these instructions must be changed.

Product agents represent the instances of the product to be made and are associated with an RFID on the work-piece carriers. Product agents carry the micro-instructions, and each of them exists until the product item it refers to is finished.

Part agents represent the product parts which are delivered by the feeders and need to travel to their target position in the final assembly. There is one part agent per part type (not one per part). Part agents collaborate with feeder agents to organise the delivery of the parts.

The **Dynamic Map** (DMap) is a registry where modules currently being used in the layout register and update their availability. The DMap is thus, order-specific and dynamic - reflects the current status of the layout; each time the layout changes or a module becomes unavailable, the DMap is updated (the corresponding modules are unregistered from the DMap).

The **Directory Facilitator** (DF) is a registry where *all* the agents (including those that are not part of the layout, i.e. those that are no in the DMap) register and may retrieve services from other agents.

Here is an example of generic assembly instructions: *Pick part 1 and place it on the work-piece carrier, then pick part 2 from its feeder and insert it into the hole in part 1 by applying a force x.* The generic assembly plan does not provide information about what module to use and what movement to make. It only provides information about how the different product parts must be assembled and in which order. More details are included in the layout-specific assembly instructions, which, for the generic assembly instructions given above, could look like this: *Robot R1 with gripper G1 is at position P1 = (x1/y1/z1) and moves to position P2 (above pallet), opens the gripper (if it was closed before), moves down to P3, closes the gripper, then moves to P4 (location on the work-piece carrier), opens the gripper. Then R1 moves on to P5 (feeder), closes the gripper, moves to P6 (location of insertion) and moves down towards P7 with a force x, then opens the gripper and moves back to P1 and closes the gripper.*

3. The Adhesive Tape Roller Dispenser Assembly System

To illustrate the concept, we have chosen a simple product: an adhesive tape roller dispenser (see Figure 2) consisting of two body parts (Parts 1 and 3) locked by a screw (Part 4) and the tape roll (Part 2). In the rest of this paper we will refer to this assembly system as the Tape Roller Dispenser. The assembly is made on top of a work-piece carrier circulating on the conveyors.

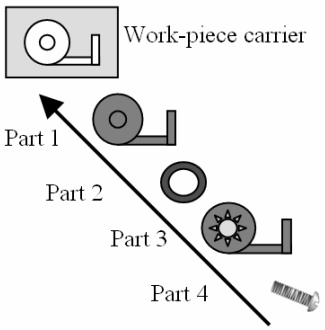

Figure 2. The Adhesive Tape Roller Dispenser

For reasons of simplicity, the choice of system modules at hand will for now be very limited (examples shown in Figure 1): a Z-axis moving in a vertical direction; an X-axis working horizontally; a feeder receiving screws in bulk. In reality, modules available at other places in the system, in the storage or even in the system supplier's catalogue can be considered for joining the coalitions on request. For illustration, Figure 3 shows a combination of the two robotic axes introduced in Figure 1, with additionally a 2-finger gripper mounted on the Z-axis. This configuration can be used for executing all the movements required to assemble the Tape Roller Dispenser.

Layout. The Self-Organization process could produce many different layouts and choose one depending on user preferences and available modules. For this case study, we consider that a circular layout, such as the one shown on Figure 4, is produced, with all the robots being identical and as shown on Figure 3.

This layout has some interesting characteristics: being circular, work-piece carriers re-visit Robot R1, which executes two different work-steps. In this example, Robot R1

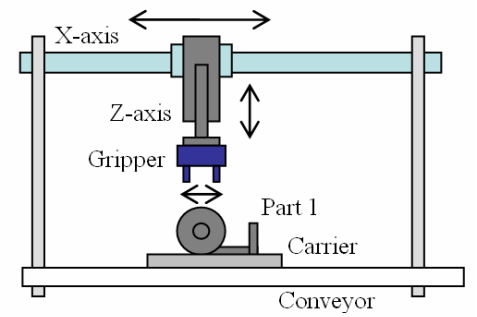

Figure 3. A simple robot made of 2 axes and 1 gripper and Part 1 placed on a work-piece carrier circulating on the conveyor

works faster than Robots R2 and R4, and thus assembles Part 1 as well as Part 3. The letters A, B, C and D on Figure 4 refer to the respective feeding of the Parts 1, 2, 3 and 4, while "IN" represents the entry of the empty work-piece carriers and "OUT" means that the carrier with the finished product leaves the system. The screw (Part 4) is fed by a feeder such as on Figure 1 and picked up / transported by a magnetic screw driver. The tape roll is fed from a tube, while the body parts are conditioned on pallets.

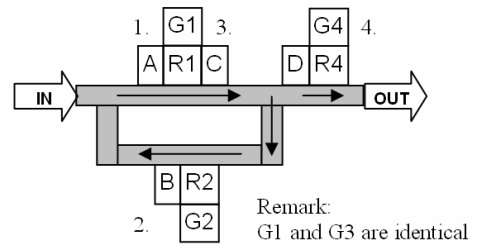

Figure 4. Circular layout where Robot 1 treats Part 1 and later also Part 3

Agents. The agents in this case study are: the *order agent* - carries the generic assembly plan for Tape Roller Dispensers; *product agents* - carry the layout-specific assembly instructions, which the visited resource modules will be asked to execute; *manufacturing resource agents* - 5 linear conveyor units, 2 corner units and 1 T-junction unit; Robots R1, R2 and R4: each needing 1 X-axis, 1 Z-axis and 1 gripper (G1, G2, G4) with 2 fingers; 2 Feeders (tape rolls and screws - B and D); 2 Pallet feeders (1st and 2nd body part - A and D); *part agents* - Part 1 to Part 4, work-piece carriers.

Scenarios. We will consider the following scenarios triggering a self-organization process in our assembly system:

New product order: carried by an Order Agent in input to the system, it triggers the building of the corresponding layout as a result of a self-organizing process. (User preferences considered: circular system, morphologically identical robots with one being faster than the others.)

Resilience during production: fatigue / failure of some modules and re-organization of the remaining modules for building a repaired / alternative system. (Concrete incident considered: robot R2 shows decreasing performance; robot R1 can take over until R2 has been replaced.)

Small change in product design followed by the reconfiguration of the layout: some modifications have consequences which affect resource attribution. (Change considered: the screw will be eliminated and replaced by a snap-fit mechanism integrated on Part 3; as a consequence, robot R4 is redundant and R1 (=R3) needs a new gripper which is able to apply a force F in the center of Part 3.)

4. Generic Self-Organizing Architecture

The self-organizing and self-adaptive framework [5] that we apply follows a service-oriented architecture (see Figure 5 for its run-time infrastructure), where the services are provided by components or agents. The architecture exploits *metadata* to support decision-making and adaptation, based on the dynamic enforcement of explicitly expressed *policies*. Metadata and policies are themselves managed by appropriate services. The main point is that the components, the metadata and the policies are all *decoupled* from each other and can be dynamically updated (or changed).

The run-time infrastructure is not necessarily centralized, the different services providing access to the description of components, or monitoring and acquisition of metadata can reside at different locations and work autonomously. Metadata and policies have either a local or global scope, and can be locally attached to a component. The actual implementation depends on the application.

Components are the different agents of the system (product agents, order agents, manufacturing resource agents, etc.)

Metadata is data about functionality and performance characteristics, as opposed to data which is treated directly by components. Metadata is stored, published and updated at run-time by the run-time infrastructure (monitoring activities) or by the components themselves (sensing/acting). Different types of metadata are available: self-description of the components (possibly including interfaces information, location, conditioning, etc.), environment related metadata (possibly supporting coordination, e.g. through stigmergy), self-* properties metadata refer to resilience and non-functional metadata related to either in-

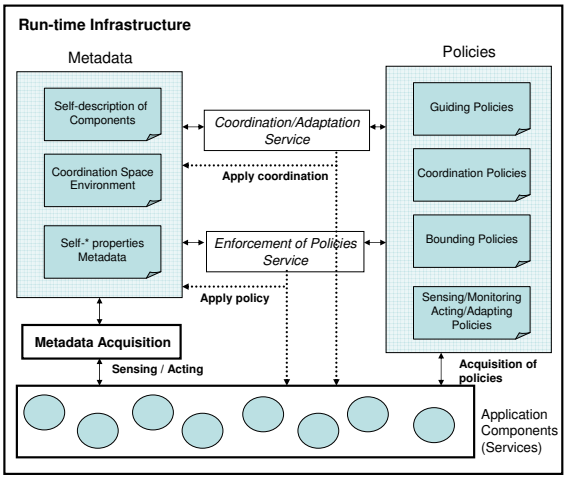

Figure 5. Run-time infrastructure

dividual components or to groups of components (such as level of "fatigue" of a component of the assembly system).

Policies are also available at run-time to both the run-time infrastructure and the components themselves. Policies can be subject to dynamic changes. They come in different categories, and apply to different levels (system-level policies vs. component level policies). Guiding policies stand for both high-level goals the system as a whole has to reach and for individual components goals. Coordination policies refer to rules which components have to adhere to when coordinating their different tasks (for instance scheduling of tasks or overlapping of work spaces). Bounding policies are intended to prevent the system going beyond its limit: these are limits set by the designer to avoid the system going out of control (too many failures), as well as limits imposed by the environment on the system (e.g. a particular type of modules cannot be placed on the lower left corner of the assembly plan). Finally, sensing/monitoring policies are lower level policies, where individual components react to on-going activities in the system. There is no restriction on the policies, they can be action-, goal- or utility-based policies [10].

Enforcement of Policies. Low-level policies attached to specific components trigger a reaction in those components based on the corresponding metadata value (for instance, monitoring metadata about position allows to take corrective measures if necessary). The run-time infrastructure itself is equipped with specific services responsible to enforce the policies (given the current metadata values) by directly acting on the components (e.g. replacing, reconfiguring), the metadata values or the policies. It provides different tasks related to the processing of metadata stored in the metadata registry, such as comparison/matching of metadata, determination of equivalent metadata information, composition of metadata; it encompasses automated reasoning over the policies and the metadata.

Coordination / Adaptation. This service implements the self-adaptive and/or self-organization pattern chosen for the particular application (at design-time). For instance, it is responsible to support digital pheromone (concentration, volatility) if this is the chosen adaptation mechanism.

Generic services. Additionally necessary services to build such a run-time infrastructure encompass: a registry/broker that handles the service descriptions and services requests supporting dynamic binding; an acquisition and monitoring service for the self-* related metadata; a registry that handles the policies; a 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, together with the policies, cause the reasoning tool to determine whether or not an action must be taken. The Enforcement of Policies services act on both components and metadata, impacting components both directly and indirectly.

Self-Organisation and Self-Adaptation. For the specific case of Evolvable Assembly Systems, components are the different actors of the assembly system: order agents, product agents, resource agents (including those in store) and part agents. Components register their services, skills and constraints *(self-description metadata)*. They have access to global or local *coordination metadata* (e.g. assembly status of current product item) and *resilience metadata* (e.g. current level of precision or speed of a module itself or of a partner module). Policies that the system as a whole has to adhere to are global to all components (for instance "'fulfill the generic assembly plan"' or "'no manufacturing resource agent 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"').

Self-organization is mainly related to the creation of a new layout (a new organization of the resource agents) when a new generic assembly plan is given or when a failure has occurred. This happens bottom-up, following the Tiles model [15] (detailed later). *Self-adaptation* is related to the production time (adaptation of speeds, tasks coordination and collision avoidance) and leads to self-organization when it triggers a new re-configuration.

See [5] for a more detailed discussion of this framework, in particular how it supports the design phase, how it allows control to be inserted in the system, and how it unifies self-adaptation (top-down: evaluate the global system be-

haviour and adapt the agents' behaviour accordingly) and self-organization (bottom-up: local rules lead to a global result) designs.

5. Design of Self-Organizing Evolvable Assembly Systems

The architecture described in Section 4, when applied to Evolvable Assembly Systems, requires (at design-time) the determination of components, self-* requirements, a self-organization and/or self-adaptation mechanism and the choice of a coordination mechanism. On the basis of these choices, corresponding metadata and policies are derived.

5.1. Components

As said above the components are: order agents, product agents, manufacturing resource agents (including those in store) and product parts. See Section 3 for more details about these components.

5.2. Self-* Requirements

Layout Formation. Any new assembly plan triggers a self-organization process leading to a new configuration (layout). This encompasses self-selection of modules and their partners, and establishment of the layout-specific assembly instructions (micro-instructions).

Task coordination at production time. *Tasks sequencing* is done according to the specific assembly instructions. Modules coordinate their work according to the current status of the product being built. *Collision avoidance* is also part of the self-organizing process. Modules with overlapping workspace must maintain a minimum distance to each other while moving.

Self-Adaptation. *Self-healing*: During production, whenever a module failure is detected, by the module itself or one of its partners, either the modules can solve the problem by themselves (software restart, open and close a gripper, etc) or they alert the user. "Module failure" can mean many types of perturbations, for instance a blocked gripper, a software problem, a lost part or anything else. *Self-optimization*: During production, the speed of processing the product parts is adjusted to the queuing level.

5.3. Mechanisms

Self-organization mechanism (1). The overall mechanism for letting the different agents perform the appropriate tasks at the different phases of the production (from product order to finished product) is based on *stigmergy by work-in-progress*, also called *qualitative stigmergy* [2]. The different *configuration states* of the system trigger different responses from the agents; they represent the different phases of production. Configuration states are: C1 - New generic assembly plan in input; C2 - Production; C3 - Production completed (stop); C4 - Re-configuration requested.

Self-organization mechanism (2). The mechanism chosen for letting the different modules and product parts reorganize whenever a new assembly plan arrives, or when a change in one of the modules is requested, is inspired by the tiles self-assembly model of crystal growth. In this model, tiles progressively attach to each other following matching rules and current configuration of the structure to build [15]. This is used in Configurations C1 and C4. In [3], the tiles self-assembly model is used to calculate the result (a solution) of a mathematical function; analogously, in EAS, the tiles (agents) self-assemble to provide a solution (a layout - DMap), to the given function (Generic Assembly Plan).

Self-adaptation mechanism. Self-adaptation, e.g. resilience to failures, to the effects of fatigue and collision avoidance, is performed by modules monitoring their own behaviour or their neighbor's behaviour. This is used in configuration C2.

Coordination mechanism. During production, the coordination of tasks for each individual product item is done through indirect communication by storing the current advancement of the assembly in a shared place - a RFID attached to the product item. This is used in configuration C2.

Metadata and policies. We concentrate here on describing the metadata and policies related to the three categories of self-* requirements described above. Figure 6 shows how the policies and metadata relate to each other and to the three use cases above. They are further detailed in Subsections 5.4 to 5.6.

5.4. Layout Formation (C1/C4)

5.4.1. Policies

Guiding Policies. These are high-level policies that mostly take the form of a goal-based policy or a utility-function policy. The generic assembly plan acts as a high-level goal and triggers C1.

System-level policies. **(P1)**

- Build a continuous path from IN to OUT .
- Favor a layout with / without a circle .
- Minimize changes in the layout when re-configuring due to failure or change in product design.
- When creating the path through the layout, the workspaces of different module coalitions may not interfere as this might lead to collisions (except for conveyors); inside a coalition however, workspace interference is necessary.
- If Layout is completed, change configuration to C2.

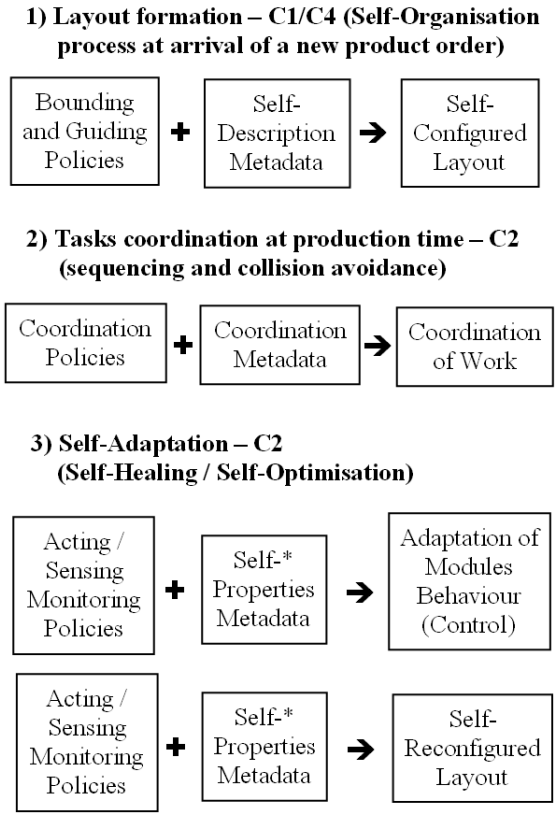

Figure 6. Relations between Metadata and Policies

Goals of order agents. **(P2)**

- Fulfill the generic assembly plan.
- Produce the specified number of products.
- If production finished, trigger configuration C3.
- Minimize travel from feeders to target positions on assembly.

Goals of manufacturing resource agents. **(P3)**

- Find a compatible partner (matching skills, interfaces, etc.) to work with in the given generic assembly plan. For instance, a robotic axis needs to find a suitable gripper.

Bounding Policies. (P4)

- Every module must always stay inside the allowed area, defined as the general workspace.
- Grippers cannot be placed at any location except on the axis holding it or in the tool warehouse. This policy can be gripper-specific or general for the whole layout.
- Respect speed and force limits.

5.4.2. Metadata

Self-description metadata.

- For each manufacturing resource agent: all registered skills, interfaces, workspace and constraints (e.g. preferred ways of combination with other modules, list of preferred partners);
- For each product part: its location and the way it is conditioned and gripped.

5.4.3. Tape Roller System Scenario

Establishment of a Circular layout (new product order). The layout formation is progressively reached through a series of service requests including needed 3D movements (x/y/z dimensions - micro-instructions). The first request is the one from the order agent requesting to fulfil the generic assembly plan **(P2)**. Requests are progressively fulfilled by resource agents **(P3)**. Self-description metadata is matched against the requests taking into account skills, constraints and any system **(P1)** / bounding **(P4)** policy, and current achieved configuration. This corresponds to the matching rules of the tile self-assembly model. In the particular scenario of the Tape Roller System, the set of robots at disposal causes Robot R1 to be selected twice for two different tasks favoring thus a circular layout. This happens to be also the primary choice from the system-level policy **(P1)**.
From screw to snap-fit (small design change). The screw is not needed any more because a snap-fit is now integrated with Part 3. The minimum change policy **(P1)** in case of reconfiguration drives the selection of an additional gripper G3 mounted on R1 able to apply the appropriate force on Part 3. Robot R4 and its gripper are not needed anymore.

5.5. Coordination (C2)

5.5.1. Policies

Task Sequencing Policies. A task can start when the workpiece carrier has reached its position next to the robotic axis. The carrier can move on as soon as the robot has finished its operation.

Product agent goals. **(P5)**

- Fully satisfy own list of (specific) assembly instructions.
- If current operation finished, move to next position in the layout.

Individual parts agent goals. **(P6)**

- Travel from feeder to target position in the assembly.
- Communicate preferential and forbidden gripping positions.

Collision Avoidance Policies.

Resource agents policies. **(P7)**

- Distance with any other resource agent must always be bigger than safety threshold.

- Always stay inside global workspace.
- If an undue object is detected inside the workspace then stop. If the object remains there for more than 3 seconds, alert the user. Remark: some other modules as well as the parts being treated are allowed inside the respective workspaces of the active modules.

5.5.2. Metadata

Task Sequencing Metadata.

- For each product agent: current status of product agent's assembly - the result of any operation is always written into the product's RFID.
- For each resource agent: Modules' log file - store history of operations performed (for traceability purposes).
- For each work-piece carrier: Work-piece carrier position. Sensors provide exact position of the different work-piece carriers (approaching robot, leaving robot, etc.).
- For each resource agent: status / availability of resource agent (idle, reserved, working, failure, in storage).

Collision Avoidance Metadata.

- For all resource agents: occupancy of its workspace, including position of the modules in the workspace and list of undue items (e.g. a loose screw is lying on the workspace).

5.5.3. Tape Roller System Scenario

Task sequencing. A work-piece carrier circulates along the layout. The product agent notices it passes twice on Robot R1. According to its assembly plan it asks **(P5)** for the appropriate assembly (Part 1 or Part 3) **(P6)**. Similarly for Robot R1, it accesses the RFID information to determine which task to do next on the product agent on the carrier that just arrived.

Collision avoidance. The workspaces of R1, R2 and R4 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 root operation is stopped until the intrusion has gone **(P7)**.

5.6. Self-Adaptation (C2)

5.6.1. Policies

Self-optimization policies.

Feeder policies. **(P8)**

- Piece delivery speed is adapted to piece removal speed. (Feeders always need to deliver pieces at their output. If the pieces are taken away quickly, their delivery speed should be high.)

Conveyor policies. **(P9)**

- Achieve requested throughput (1).
- Respect specified speed limits (2).
- (2) has priority over (1).

Resource agent policies. **(P10)**

- If queue of product agents is above the queuing threshold, increase speed of operation.
- If quality decreases, decrease speed of operation in order to increase quality.

Self-healing Policies. It is fundamental to report the state or respectively the result of collaboration with other agents. Every agent monitors its own state, especially with reference to critical parameters, and eventually alerts the user in case of problems. E.g., modules 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.

Axes and conveyor policies. **(P11)**

- If the target position after a movement has not been reached correctly, take corrective measure (advance more / less, ask for maintenance, etc).

Resource agents' policies. **(P12)**

- In case of malfunctioning, request a replacement from a coalition partner.
- If no replacement found, ask for a re-configuration of the layout.
- If queuing level is too high and speed is at maximum, ask for a re-configuration of the layout.
- If gripper blocked then 1) open / close gripper again, 2) restart software, 3) reset power, 4) replace gripper or 5) call user.

System-level policy (bounding policy). **(P13)**

- If one (or more) failure occur more than a certain number of times in a certain period of time, then alert user and trigger configuration C4.

5.6.2. Metadata

Self-* Properties Metadata. For each resource agent:

- Maximal / optimal speed of operation
- Own precision - movements on every axis
- Precision of partners (neighbors)
- Queuing level of product agents (in input to that resource agent)
- Quality of assembled product

5.6.3. Tape Roller System Scenario

Self-adaptation. Robot R1 takes over from Robot R2: Robot R2 experiences problems in reaching its target positions and asks for maintenance **(P11)**. As a temporary solution, R1 is asked to take over and the user is asked to place feeder B close to R1 **(P12)**.

Re-configuration triggered. After taking over from Robot R2, Robot R1 experiences a high level of queuing. This leads Robot R1 to ask a re-configuration **(P1)** (and triggers configuration C4).

5.7. Design analysis and predictability of dependability properties

Policies (further refined and enforced at run-time) turn to be a useful tool for analyzing the correctness of the design and determining (proving) properties using temporal logic formulae. A first step is to prove that a layout will be found. As another example, consider the safety property: "Collisions never happen"; the liveness property: "The specified number of products has eventually been assembled"; or the invariant: "At all times, the quality of assembled products is above a specified threshold". The safety property is guaranteed by the collision avoidance policies **(P7)** and the selected layout; the liveness property is a policy by itself (goal of order agent - **P2**), while the invariant is provided by the self-optimization/self-healing policies (decrease speed / increase movement precision - **P11**). Formal proofs of these properties need formal specifications of the system, the properties, the policies (and their potential conflicts) and will be investigated.

6. Implementation

EAS are based on CoBASA [1], which has been continuously growing during the last years. The currently implemented system encompasses: a directory registration service; dynamic coalitions of modules to provide complex services; easy reconfiguration in case of failures or little changes in process or product design. It does not include self-organised layout creation from scratch yet.

Services Directory and Ontology: Modules and product parts are all wrapped as agents and register their services (self-description metadata) to a Directory Facilitator (DF) storing relevant global knowledge such as the nature of the exchanged messages, skills and skills' composition rules. The DF is an active entity incorporating an EAS Ontology, which is currently being refined. It responds to requests by returning the list of appropriate modules (skills, speed, etc.). The ontology also serves as a support for the automatic generation of complex skills out of simple skills offered by single modules. Generic rules for their creation are being elaborated.

Dynamic coalitions: In initial CoBASA, coalitions were formed by the human user, and they were thus static. The current new approach, enabling the system to execute self-organization, is based on dynamic coalitions which the agents form themselves and which can change at any time without the user having to reconfigure the coalitions. Knowledge about useful or problematic coalitions is kept (instead of lost at coalition resolution) and can be re-used any time the same complex skills are required again - complex coalitions also register their services to the Dynamic Map. Coalitions are formed through service requests and matching provided by the Directory Facilator described above and by exploiting the defined Ontology.

Layout formation and micro-instructions: the self-organised layout formation will be based on the notion of dynamic coalitions. It will be realized using a series of service requests (incorporating the required moves - x/y/z values of the micro-instructions) in order to fulfill the given generic assembly plan. If the generic assembly plan cannot be fully realized, the current layout is abandoned and a new one is re-built. The layout formation and the micro-instructions are then progressively built together. These trial and error constructions of layouts are performed in a virtual space. Modules are not actually asking to be placed on a layout before a complete satisfying solution is reached. To this end, we are currently implementing a 3D simulation of the assembly systems, respectively the modules and their ways of aggregation.

Tasks coordination: a product item being assembled moves along the layout on a work-piece carrier. An RFID associated to the specific product being assembled stores all historical data related to assembly. Based on this information, the product requests the service of the next module as established in the specific assembly plan and the work-piece carrier then moves to the respective module.

Collision avoidance: the first level of safety is calculating the trajectories of the robots and making sure that they never intersect at the same instant. The second level are collision sensors which are being checked at operation time.

Self-adaptation: Self-diagnose of EAS is being investigated; each module is responsible for diagnosing itself and taking corrective measures if required (self-healing).

7. Related work

There are other concepts which currently address the needs of Mass Customization industry. Flexible and Reconfigurable Manufacturing Systems [6] focus on machining and not on assembly. Holonic Manufacturing Systems [14] concentrate on morphologic aspects (every item is a whole as well as part of a bigger whole). The Architecture for Agile Assembly [13] is probably one of the approaches which come closest to providing a solution for reconfigurability in assembly systems, but does not consider the mutual interrelations between product, processes and systems.

Considering the applications of MAS in industry [11], the evolution of agent technologies and manufacturing will probably proceed hand in hand. Also Service-Oriented Architectures [4] enjoy increasing popularity.

"Plug'n'Produce" can be achieved thanks to process-oriented programming [12] instead of device-level programming. This includes forming complex skills out of simple ones, describing devices in detail, using Ontologies and facilitating user interaction.

The Pabadis'Promise project [7] has parallels with EAS but is focused on a higher level inside the manufacturing company, more at the level of production management (Enterprise Resource Planning), while EAS work mainly at shop-floor level.

Autonomy is fundamental for EAS, but it is also a highly controversial topic in industry. The degree of autonomy needs to be adjustable [9], and that the human always keeps the control over the degree of autonomy of the robots.

8. Conclusions and Outlook

The work presented in this article still needs further development and full implementation to be entirely validated, but already at this stage, it shows a high potential for tackling the challenges of modern manufacturing. Regarding self-adaptation and self-organizing issues, this paper proposes an innovative architecture for combining these two concepts (top-down and bottom-up) when designing a system. It shows the relevance of such an architecture in a real-world industrial scenario, but does not (yet) provide a ready-to-use recipe for building self-organising assembly systems.

The current version of the designed system is self-organizing and supports evolvability, but does not consider a way of measuring the quality of the obtained solution: Is it really the solution with maximum precision and minimum resources that has been selected? How long does it take to find a solution, if any? This could be further achieved by inserting more utility functions into the policies.

There are several possibilities to use learning in layout formation: once a solution is found, it could be remembered, and further self-organization processes could take profit of it. It also remains to be investigated what to do in case of conflicting policies - consulting the human user is always an option.

Future work encompasses: development of the run-time infrastructure (Figure 5) supporting dynamic use of policies and metadata; creation of the layout according to the tiles self-assembly model; as well as formal study of the design and proof of dependability properties.

9. Acknowledgement

Regina Frei thanks the Portuguese Foundation for Science and Technology for the financial support (PhD grant SFRH / BD / 38608 / 2007).

References

[1] J. Barata. *Coalition Based Approach For ShopFloor Agility*. Edições Orion, Amadora - Lisboa, 2005.

[2] E. Bonabeau, M. Dorigo, and G. Théraulaz. *Swarm Intelligence*. Oxford University Press, Santa Fé Institute, 1999.

[3] Y. Brun and N. Medvidovic. An architectural style for solving computationally intensive problems on large networks. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, Minneapolis, MN, USA, 2007.

[4] A. Colombo, F. Jammes, H. Smit, R. Harrison, J. Lastra, and I. Delamer. Service-oriented architectures for collaborative automation. In *Ann. Conf. of the IEEE Industrial Electronics Society (IECON)*, pages 2649–2654, Raleigh, North Carolina, USA, 2005.

[5] G. Di Marzo Serugendo, J. Fitzgerald, A. Romanovsky, and N. Guelfi. A Generic Framework for the Engineering of Self-Adaptive and Self-Organising Systems. Technical Report CS-TR-1018, School of Computing Science, University of Newcastle, 2007.

[6] H. A. ElMaraghy. Flexible and reconfigurable manufacturing systems paradigms. *International Journal of Flexible Manufacturing Systems*, 17(4):261–276, 2006.

[7] L. Ferrarini, C. Veber, A. Lüder, J. Peschke, A. Kalogeras, J. Gialelis, J. Rode, D. Wunsch, and V. Chapurlat. Control Architecture for Reconfigurable Manufacturing Systems: the PABADIS'PROMISE approach. In *IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, pages 545–552, Prague, Czech Republic, 2006.

[8] R. Frei, L. Ribeiro, J. Barata, and D. Semere. Evolvable Assembly Systems: Towards User Friendly Manufacturing. In *IEEE International Symposium on Assembly and Manufacturing (ISAM)*, pages 288 – 293, Ann Harbor, Michigan, USA, 2007.

[9] M. A. Goodrich, D. R. Olsen, J. Crandall, and T. J. Palmer. Experiments in adjustable autonomy. In *IJCAI Workshop on Autonomy, Delegation and Control: Interacting with Intelligent Agents*, 2001.

[10] J. O. Kephart and R. Das. Achieving Self-Management via Utility Functions. *IEEE Internet Computing*, 11(1):40–48, 2007.

[11] L. Monostori, J. Vancza, and S. Kumara. Agent-Based Systems for Manufacturing. *CIRP Annals - Manufacturing Technology*, 55(2):697–720, 2006.

[12] M. Naumann, K. Wegener, and R. Schraft. Control architecture for robot cells to enable Plug'n'Produce. In *Int. Conf. on Robotics and Automation (ICRA)*, pages 287–292, Roma, Italy, 2007. New Orleans: Omnipress.

[13] A. Rizzi, J. Gowdy, and R. Hollis. Agile Assembly Architecture: an Agent Based Approach to Modular Precision Assembly Systems. In *Int. Conf. on Robotics and Automation (ICRA)*, pages 1511–1516, Albuquerque, New Mexico, 1997.

[14] P. Valckenaers and H. Van Brussel. Holonic Manufacturing Execution Systems. *CIRP Annals - Manufacturing Technology*, 54(1):427–432, 2005.

[15] E. Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, 1998.