Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

# An Architecture for Self-Managing Evolvable Assembly Systems

Regina Frei, Bruno Ferreira
Uninova
Caparica, Portugal
regina.frei@uninova.pt, bdf17288@fct.unl.pt

Giovanna Di Marzo Serugendo
Birkbeck College
London, United Kingdom
dimarzo@dcs.bbk.ac.uk

Jose Barata
Uninova
Caparica, Portugal
jab@uninova.pt

***Abstract*—Agile manufacturing requires high responsiveness at all levels of a company, but is especially challenging on the shop-floor level. Evolvable Assembly Systems (EAS) are a solution: agentified modules can be seamlessly integrated into existing systems, or removed at any instant. EAS offer a more flexible solution to automation production, but many system design and integration tasks are still done manually. Our goal is to make EAS increasingly self-managing: 1) to easily and quickly produce a new or re-configured assembly system each time a new product order arrives or each time a failure or weakness arises in the current assembly system and 2) to maintain production also under degraded conditions. This article describes an architecture for self-managing evolvable assembly systems. It involves on-the-fly self-assembly of robotic modules, dynamic coordination of tasks and self-adaptation to production conditions, mainly self-healing and self-optimisation. The architecture exploits self-description of modules, monitored modules behaviour and dynamic policies.**

## I. Challenges in agile manufacturing systems

Industrial suppliers of automation technology produce components such as electric drives, valves, grippers, or motors. Their customers from the automotive industry, pharmaceutical industry, food and package industry, etc. purchase these components for building **robotic assembly systems** in order to produce specific products for their own clients. Over the years, the needs of the suppliers' customers have evolved. In the era of *mass production*, customers produced large quantities of an identical product as fast and as cheap as possible. It was then worth paying the big investments for custom-made installations, which would be disposed of once the product was out of production. Today, the market tends increasingly towards *mass customization*, with customers clients individually selecting from various product options. Customers require **high responsiveness** and the ability to cope with a multitude of conditions. They increasingly need agile robotic assembly systems, able to cope with dynamic production conditions dictated by *frequent changes*, *low volumes* and *many variants*. There is an accrued awareness among industrial suppliers that industrial systems need to be quickly changeable, have to follow a Plug&Play approach, avoiding time- and work-intensive re-programming, and maintain productivity also under perturbations. Much attention has been given to the requirement for assembly systems to be easily reconfigurable or to seamlessly integrate new components. However, human involvement is still strongly required, in particular for designing the reconfiguration of the assembly system, for programming the individual devices, and for monitoring the production. Altogether, it is an expensive, error-prone, long and tedious process.

The **application of self-* principles** to the automation domain is a research area with the potential to attract a great industrial interest. It has not received much attention yet from the self-* community, currently more focused on P2P systems, mobile ad hoc networks, mobile robots, services or optimisation problems (e.g. shop floor scheduling). We envision that modules of an assembly system, selecting each other, re-programming themselves and working as an ecosystem, will provide reliable assembly systems that dynamically *self-organise* to process customised product orders and *self-adapt* to ensure production also in degraded modes, rather than stopping completely in case of perturbations. This will result in cost-efficient and quickly reconfigurable assembly systems requiring a minimum of human involvement.

Section II briefly describes the concept of self-managing evolvable assembly systems. Section III presents the architecture and design elements of these systems. Section IV provides implementation details. Section V discusses a case study of dynamic self-assembly of modules and some self-management scenarios. A discussion of the chosen approach follows in Section VI. Finally, Section VII presents related work.

## II. Self-management for assembly systems

An **assembly system** is an industrial installation that receives *parts* and joins them in a coherent way to form a final *product*. It consists of a set of equipment items (manufacturing resources or *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)** [1] is an assembly system which can co-evolve together with the product and the assembly process. Thanks to tiny controllers for local intelligence and software wrappers, every module is an embodied software 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*. Skills represent the functionalities that each module is able to perform. Skills offered by one module are called *simple*

978-1-4244-2794-9/09/$25.00 ©2009 IEEE

*skills*; those skills requiring the aggregation of more than one module (that is, module *coalitions*), are called *composite skills* [5]. A **Self-Managing Evolvable Assembly System** is an EAS with two additional characteristics: modules self-organise to produce an appropriate layout for the assembly and the assembly system as a whole self-adapts to production conditions. Recently, we developed a design process for self-* systems [2] and applied this design process to assembly systems [4]. This involves a service-oriented architecture, acquired and updated metadata (e.g. monitored behaviour of modules) and enforcement of executable policies.

### A. Self-* requirements and mechanisms

**Creation of the layout.** Any new product order triggers a self-organising process: the modules spontaneously create coalitions (self-assemble) with suitable partners and select their position in the assembly system layout.

**Task coordination at production time.** This includes *task sequencing* in which modules coordinate their work according to the current status of the product being built (indirect communication by storing the current advancement of the assembly in a shared place - an RFID attached to the carrier of the product being assembled); and *collision avoidance* where modules with overlapping workspace must maintain a minimum distance to each other while moving.

**Self-adaptation at production time.** 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 three different reconfiguration types [7]. *Parametrical*: the current modules adapt their behaviour (change speed, force, task distribution, etc.) in order to cope with the current failure, possibly degrading performance but maintaining functionality. *Logical*: the current modules may be used in a different way, possibly forming new coalitions. For instance, a robot may take over additional tasks, which were previously executed by other modules in order to equilibrate the work-loads. *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.

Resilience to failures and collision avoidance is performed by modules monitoring their own behaviour or their neighbours' behaviour. If a module fails during production, or if there is a change in the product design, coalitions can dynamically change or be (re-)built at any instant, replacing/changing modules in order to address the failure or the design change.

## III. Architecture and design

Figure 1 illustrates the architecture of our system. The services are provided by autonomous 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. Agents, metadata and policies are *decoupled* from each other and can be dynamically updated or changed.

Additional services to build the 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 (e.g. performance); 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 and policies cause the reasoning tool to determine whether or not an action must be taken.

### A. Agents

An assembly system consists of a group of distributed agents. **MRA** (Manufacturing Resource Agent): an agentified module that provides simple skills and has the capability to participate in coalitions. **DCA** (Dynamic Coalition Agent): provides composite skills obtained by the aggregation (self-assembly) of multiple MRAs. A provided simple or composite skill is called a *service*. **OA** (Order Agent): specifies the number of products of the same type to produce as well as the generic assembly plan to follow in order to create these products. **PA** (Product Agent): specifies the detailed assembly plan of the product that is currently being assembled. As the product assembly progresses along the assembly system, the PA requests the appropriate services from the various MRAs in order to perform the different steps on the assembly plan. **WPCA** (Work-piece Carrier Agent): transports the product along the assembly system and allows it to reach the MRAs executing the desired operations.

### B. Metadata

Metadata is data about functionality and performance characteristics, as opposed to data which is treated directly by the agents. Metadata is stored, published and updated at run-time (monitoring of the modules) or by the agents themselves (sensing/acting). Metadata types include: (1) self-description of the modules: skills, interfaces information, location, conditioning, etc. (see section IV-A for more details); (2) environment related metadata: e.g. coordination information stored in the RFID of the product; (3) self-* properties metadata: performance level of individual modules or coalitions such as speed, precision, etc., captured by sensors attached to manufacturing resources. The Agent Machine Interface (AMI) is a specific software which links the sensors and actuators to the MRAs and updates metadata according to the sensors readings.

### C. Services Registry and Broker

The EAS ontology[11] has been developed, using Protégé-OWL[1]. It specifies classes of modules (such as transport or drilling modules) and their related skills. It also describes additional shared knowledge such as the nature of the exchanged messages, skill composition rules, device constraints and capabilities.

The ontology agent *OntA* agent provides controlled access to the EAS ontology and supports service registration and retrieval. It acts as a service registry for the MRAs to register their skills, and as a broker for retrieving requested services.

[1]http://protege.stanford.edu/

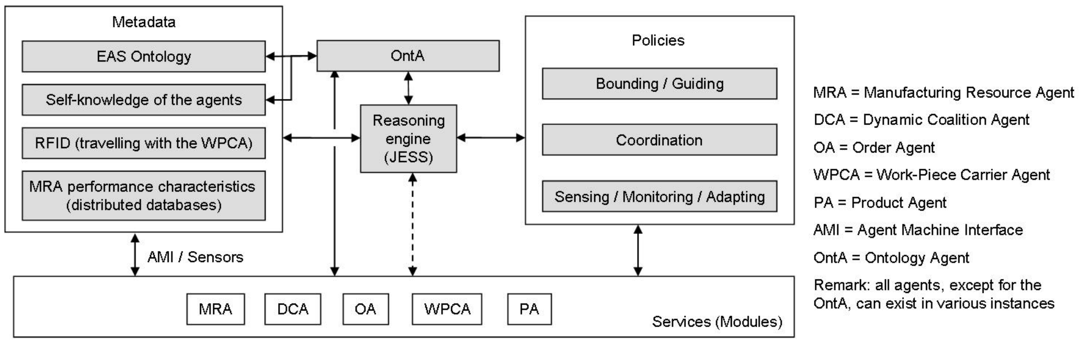
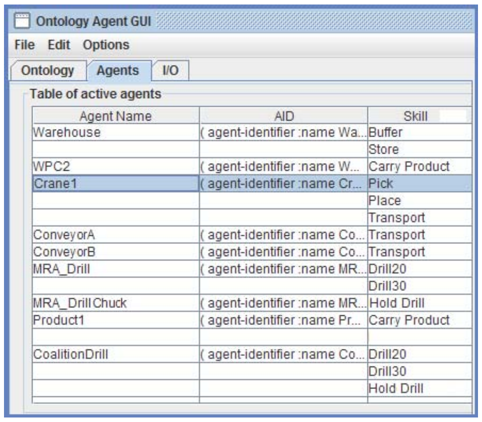

Metadata
EAS Ontology
Self-knowledge of the agents
RFID (travelling with the WPCA)
MRA performance characteristics (distributed databases)
OntA
Reasoning engine (JESS)
Policies
Bounding / Guiding
Coordination
Sensing / Monitoring / Adapting
AMI / Sensors
MRA
DCA
OA
WPCA
PA
Services (Modules)
MRA = Manufacturing Resource Agent
DCA = Dynamic Coalition Agent
OA = Order Agent
WPCA = Work-Piece Carrier Agent
PA = Product Agent
AMI = Agent Machine Interface
OntA = Ontology Agent
Remark: all agents, except for the OntA, can exist in various instances

Fig. 1. Architecture for Self-Managing Evolvable Assembly Systems

Once a specific skill is requested, OntA returns the current list of agents providing the requested skill together with additional information (interfaces, performance characteristics, etc.).



| Agent Name | AID | Skill |
|---|---|---|
| Warehouse | ( agent-identifier :name Wa... | Buffer |
| | | Store |
| WPC2 | ( agent-identifier :name W... | Carry Product |
| Crane1 | ( agent-identifier :name Cr... | Pick |
| | | Place |
| | | Transport |
| ConveyorA | ( agent-identifier :name Co... | Transport |
| ConveyorB | ( agent-identifier :name Co... | Transport |
| MRA_Drill | ( agent-identifier :name MR... | Drill20 |
| | | Drill30 |
| MRA_DrillChuck | ( agent-identifier :name MR... | Hold Drill |
| Product1 | ( agent-identifier :name Pr... | Carry Product |
| | | |
| CoalitionDrill | ( agent-identifier :name Co... | Drill20 |
| | | Drill30 |
| | | Hold Drill |

Fig. 2. OntA interface showing some registered skills

Figure 2 shows an example of MRAs and coalitions registered with OntA. The Crane1 agent with three registered skills (the three skills of the actual underlying Crane module): *Pick* for being able to seize parts; *Place* for being able to release previously seized parts; *Transport* for being able to move seized parts. The 'CoalitionDrill' agent includes two modules: an agent called *MRA Drill* which can drill holes of 20mm and 30mm diameter, and another one called *MRA Drill Chuck* which holds the drill. The three skills registered for this coalition correspond to the union of skills provided by the drill (*Drill20*, *Drill30*: drilling holes of 20 or 30 mm) and the drill chuck (*Hold Drill*).

## D. Policies

Policies come in different categories and apply to different levels: system-level policies vs. agent level policies (Figure 1). **Guiding policies** stand for both high-level goals the system as a whole has to reach (e.g. drilling holes into 30 pieces) and for individual components goals (e.g do not get blocked). **Bounding policies** prevent the system from going beyond its limits, which are set by the designer (maximum speed, minimum quality of products or too many failures), as well as imposed by the environment (e.g. a particular type of modules cannot be placed on the lower left corner of the assembly plan). **Coordination policies** refer to task coordination (e.g. scheduling of tasks or overlapping of work spaces). Finally, **sensing/monitoring policies** are lower level policies attached to individual components specifying how to react to on-going activities in the system. 'If the target position after a movement has not been reached correctly, take corrective measure (advance more or less, ask for maintenance). For more see [4].

## E. Enforcement of Policies

The run-time infrastructure is equipped with specific services responsible for enforcing higher-level policies, such as guiding and bounding policies by directly acting on the agents (e.g. replacing, reconfiguring modules), the metadata values or the policies. The reasoning engine serves two purposes: firstly, it allows the system to infer new composition rules whenever a new type of module is inserted into the ontology; secondly, it allows the system to infer the action to be performed whenever a policy applies. Policies are generic, e.g.: 'Replace slow module by an equivalent module having a higher performance.' By accessing metadata about current performance of the modules, the reasoning engine can determine which of the available modules can actually replace the failing one.

## IV. Implementation

Figure 3(a) shows how the architecture is being implemented. Each MRA is linked to an individual computer[2] which hosts: a *controller* that pilots the actual movements of its associated module; metadata local to the module, such as performance or precision of the module or self-description (see below); local (low-level) policies; and the reasoning engine JESS[3] for enforcing the policies through the controller, on the

[2]this is current practice in automation research
[3]http://herzberg.ca.sandia.gov/

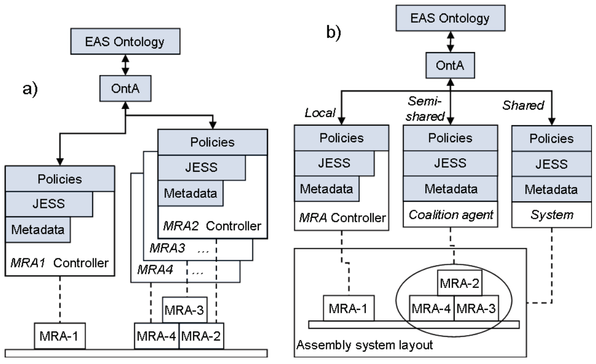


Fig. 3. a) for every MRA, b) for every coalition, for sub-sets of the system, and for the entire system

basis of metadata. The language used for writing policies is OWL, which assures the compatibility with the EAS Ontology.

Figure 3(b) depicts a similar organisation for coalitions (group MRA-2, MRA-3, MRA-4) and for the whole assembly system. Coalitions do not have a controller, but have metadata and policies shared only by the agents in that coalition. Examples of metadata include performance of the coalition as a whole, such as how many pieces the CoalitionDrill has processed in the last 10 seconds. An example of a coalition policy is: If queuing level of pieces is high, increase coordinated speed. Notice that as coalitions are dynamically created in response to an incoming Order Agent or a production condition, the corresponding policies are created or linked to the coalition on-the-fly. Policies and metadata applying at the level of the whole system are shared by all agents.

## A. *Self-describing services*

Every module carries an XML-file which contains diverse information about the module characteristics, limitations and requirements, as well as some variables such as the location and the workload. This is crucial to support dynamic coalitions [5]. The XML-file is saved locally on the MRA's controller. It allows the module's agent to know the module itself, which is useful when searching for suitable partner modules and when answering requests from another module, such as compatible interfaces and dimensions. This limits considerably the viable combinations of modules and thus prevents from a combinatorial explosion during the creation of the coalitions.

Figure 4 is the first part of such an XML-file, which is part of the metadata stored in the computer attached to the module (see Figure 3(a)). The file shows the details of the MRA *Drill Chuck*. Each module has a unique ID. The *ModuleName* describes the type of the module, it is the same for every module of the same type. The *Description* in natural language is relevant for the human user only. The *SerialNumber* is uniquely assigned to the module by the supplier and follows the supplier's format. *Skill* contains the skill name and various items which depend on the skill type in question. For the drill chuck, it includes the diameter of the drill it can hold, the maximal depth, the maximal rotations / minute, the workplace (i.e. the absolute position where the module is placed). The *Requirements* specify the need for additional skills (such as *Hold Drill Chuck*, *Rotate 2000* and *Move vertical*) which have to be provided by a partner module, as well as the minimal time necessary to execute the skill. The second part of the XML-file is generic for all modules and assembly parts and defines characteristics such as the physical size of the module and its centre of gravity.

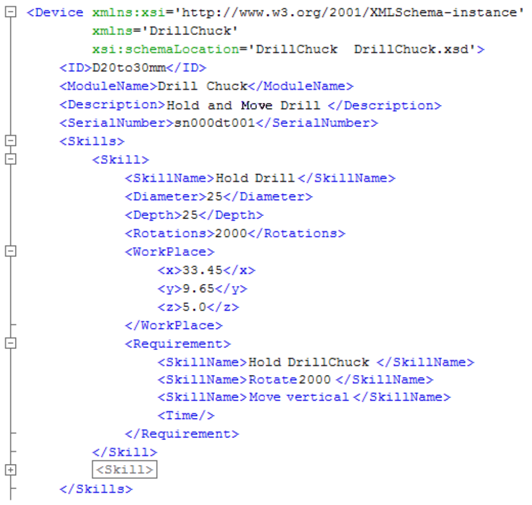

```
<Device xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
        xmlns='DrillChuck'
        xsi:schemaLocation='DrillChuck  DrillChuck.xsd'>
    <ID>D20to30mm</ID>
    <ModuleName>Drill Chuck</ModuleName>
    <Description>Hold and Move Drill </Description>
    <SerialNumber>sn000dt001</SerialNumber>
    <Skills>
        <Skill>
            <SkillName>Hold Drill </SkillName>
            <Diameter>25</Diameter>
            <Depth>25</Depth>
            <Rotations>2000</Rotations>
            <WorkPlace>
                <x>33.45</x>
                <y>9.65</y>
                <z>5.0</z>
            </WorkPlace>
            <Requirement>
                <SkillName>Hold DrillChuck </SkillName>
                <SkillName>Rotate2000 </SkillName>
                <SkillName>Move vertical </SkillName>
                <Time/>
            </Requirement>
        </Skill>
        <Skill>
    </Skills>
```

Fig. 4. Part of an XML-file showing examples of module specifications.

## V. Case Study

The MOFA [5] system at Uninova, Portugal, an industrial-like system used for educational purposes, was used for this study. It consists of various modules that can be plugged and unplugged: robot stations, conveyors, tools, buffers, magnetic crane, etc. A small case study illustrates how a dynamic coalition to drill holes is formed. Notice that we are not concerned by performance measurement during production time; our interest is on the dynamic reconfiguration of the assembly system.

## A. *Drilling Service Composition*

An *Order Agent* requires a hole to be drilled and thus requests this service. For this purpose, a coalition is formed. Figure 5 shows the UML-sequence corresponding to the case study, where the OA sends a request message to *OntA* asking for the skill *Drill*. The OntA answers that *MRA Drill* can provide this, and the OA requests this agent to participate. MRA Drill analyses if there is some further requirement for the requested action. If not, the agent would communicate that it can perform the action and send the location where it works. In our case, there are some further requirements: MRA Drill needs to be hold by a drill chuck. MRA drill therefore launches a *Coalition Agent* and provides it with the relevant information.

The newly created coalition initially contains one agent only: the agent at the origin of the coalition (i.e. the one really able to perform the requested skill, with the help of the other agents). The CA then analyses the requirements and asks OntA who can perform those skills. According to the answer from OntA, the coalition asks the indicated agents to join (and eventually asks the user to bring the required module to the corresponding location, joining them physically). The *MRA drill chuck*, in Figure 5 called 'MRA Hold', as this is its main functionality, then informs the coalition about its location, requirements, limitations, and other available skills (all the potentially relevant information), just as the MRA Drill did before.

The coalition agent repeats the step of analysing if there is any open requirement. In our case, the drill chuck needs to be rotated and moved vertically in order to drill a hole. The procedure of calling further MRAs to join the coalition continues as explained before. Once all the coalitions members have joined and all the requirements are satisfied, the final steps in the self-assembly of services are the skill registration in the ontology and sending an *inform* message to the product, indicating the coalitions's workplace (the place where the MRA drill performs its action). The coalition is now ready to work. During production, when the product arrives at the workplace of the coalition, the *Product Agent* requests the service to be executed and the coalition agent starts the working process.

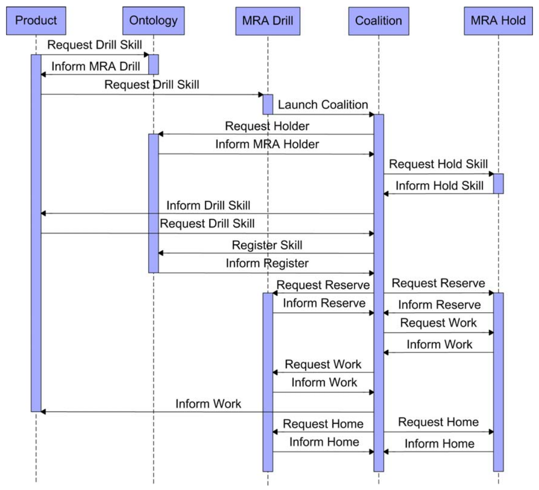


Fig. 5. UML sequence showing how services self-assemble.

## B. Rules for self-assembly

The requirements for additional partners' skills, which every module carries, include also specifications concerning the interfaces, general compatibility, etc. Still, several modules may suit the requirements and be available. In this case, how to choose the next module which will join the chain?

At the moment, a rule about lower cost is applied for this purpose (e.g lower power consumption, minimum time to perform operation). Other rules can be added in the same way. Rules could for instance tell the agent to choose the module which has already been used successfully before, or choose the newest one. Such rules can be generally valid, or only for specific cases, or only certain modules.

Whenever a module joins the coalition, it may bring further requirements, and the coalition grows. Depending on the choices made, a chain may get shorter or longer, be more or less complicated to realise physically. Some module combinations may lead to a dead end because they need modules which are not available any more. In other cases, a module asked to join may refuse - on grounds of another reservation, a planned maintenance stop, or a previous problematic collaboration. As a consequence, the last service in the chain will announce a failure, and the concerned assembly options are abandoned in favour of others.

During production, in case one of the services experiences a failure (for instance, a module is blocked), the most important criterion for choosing a suitable replacement could be minimising the effort of change.

**Self-(re-)configuration:** At production time, a new layout must be created whenever a new product order with different assembly operations arrives. Sometimes, an existing layout can be adapted to fit the new order. The layout formation is progressively reached through a series of service requests including the needed 3D movements (x/y/z dimensions micro-instructions). This follows the procedure described in subsection V-A. Endless reconfiguration iterations are avoided by a time-out policy which alerts the user of a problem when no solution could be found within a certain time range.

**Self-repair:** When a robot experiences problems and asks for maintenance, as a temporary solution, the robot agent or the coalition agent it belongs to asks another robot to take over, which means that it takes the previous robot's place in the self-assembled chain. The new robot must make sure that it fits, and will otherwise refuse to collaborate. In such a case, the coalition agent continues the search for suitable replacement, and if there is none, it will make a request for re-configuration of the layout.

**Self-adaptation:** In case of a small product design change, which only lightly touches the assembly process, there is no need for complete layout re-configuration. Consider the case of a previous Order Agent requiring the skill *Drill30*, while the new design requires a hole of 20mm only. If one of the MRAs in the layout has two skills both Drill30 and Drill20, there is no need for physical change. During production time, PAs will request the service Drill20 instead of Drill30. If no MRA can offer the Drill20 service, the coalition offering Drill30 will adapt to the new requirement. It will request a Drill20 tool and integrate this new tool into the existing coalition. The guiding policies for choosing the options with less changes / less effort apply.

## VI. Discussion

### A. No global knowledge

None of the elements involved in the self-managing assembly systems as described in this paper have global or complete knowledge of the system's goals and the ways to achieve it. Every MRA knows itself and some of its peers. It can communicate with other agents, but does not know what the system is supposed to achieve. The EAS Ontology does not know this, either. It only contains terms and their relations to one another. OntA knows which agents are in the system, and knows which MRA can perform which skills. It has no knowledge of the product to build. Each Product Agent knows what assembly actions should be executed, and is able to request the necessary services from other agents, but does not know anything further. Order agents know how to assemble the parts in a generic way, only (e.g. part 1 screwed with part 2), but do not know the actual details of the assembly (which robot will execute which movement).

### B. Characteristics of self-managing assembly systems

The Order Agent drives the dynamic organisation of the manufacturing resources; there is no centralised planning or allocation of resources. The generic assembly plan is provided by the OA; it does not dependent on actual layout details. Manufacturing resources spontaneously self-assemble in response to the incoming product order and progressively fill a detailed product assembly description. The assembly system that will ultimately produce the requested products is dynamically designed (dynamic positioning and selection of interacting resources) through self-organisation. The assembly system can absorb unanticipated product designs and unanticipated changes of resources. Thanks to the agentification of modules and software wrappers, manufacturing resources are seamlessly integrated or removed. Production occurs with decentralised and distributed control. Self-knowledge and self-monitoring of the MRAs keeps the system in good conditions at production time. Agent communicate with their neighbours to fulfil the assembly tasks.

## VII. Related work

Other concepts which currently address the needs of Mass Customization industry include the following: Flexible and Reconfigurable Manufacturing Systems [3] mainly focus on machining tasks, not on assembly. Holonic Manufacturing Systems [12] concentrate on morphologic aspects (every item is a whole as well as part of a bigger whole). The Architecture for Agile Assembly and its programming model [6] is probably one of the approaches which comes closest to providing a solution for reconfigurability in assembly systems. Dynamic coalitions in manufacturing MAS are based on central databases where agent relations are stored [10]. Heterogeneous multi-robot coalitions can be built on the basis of a schema approach [9]. A schema defines the function of a component, the inputs it takes, and the outputs it provides. By joining a schema with suitable other schemas, entire networks can be generated. 'Plug'n'Produce' can be achieved thanks to process-oriented programming [8] instead of device-level programming. This includes forming composite skills out of simple ones, describing devices in detail, using Ontologies and facilitating user interaction.

## VIII. Conclusions

This article presented an architecture for providing self-management in Evolvable Assembly Systems. Manufacturing resources self-organise to create the assembly system that will process product orders and self-adapt during production for maintaining acceptable levels of production or quality. The work described in this article is being implemented at Uninova, Portugal. Videos showing the demonstrator in operation are online[4]. Further steps include definition of an EAS policy language, the integration of policies and metadata for reinforcing the self-organising and self-adapting aspect of EAS, as proposed in [4].

## Acknowledgments

The authors thank Antonio Amado, Jose Belo and Nuno Pereira, whose M.Sc. projects contribute to the implementation of the EAS Ontology and the research related to policies and metadata in EAS. Regina Frei is supported by the Portuguese Foundation for Science and Technology (PhD grant SFRH / BD / 38608 / 2007). The authors also thank the EU-funded coordination action Perada[5] for financially supporting travel exchange between Uninova and Birkbeck College.

## References

[1] J. Barata. Coalition based approach for shopfloor agility. Edições Orion, Amadora - Lisboa, 2005.

[2] G. Di Marzo Serugendo, J. Fitzgerald, A. Romanovsky, and N. Guelfi. Metaself - a framework for designing and controlling self-adaptive and self-organising systems. Technical report, Computer Science and Information Systems, Birkbeck College, 2008.

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

[4] R. Frei, G. Di Marzo Serugendo, and J. Barata. Designing self-organisation for evolvable assembly systems. In *IEEE Int. Conf. on Self-Adaptive and Self-Organising Systems*, Venice, Italy, 2008.

[5] R. Frei, B. Ferreira, and J. Barata. Dynamic coalitions for self-organising manufacturing systems. In *CIRP Int. Conf. on Intelligent Computation in Manufacturing Engineering*, Naples, Italy, 2008.

[6] R. Hollis, A. A. Rizzi, H. B. Brown, A. E. Quaid, and Z. J. Butler. Toward a second-generation minifactory for precision assembly. In *Int. Advances Robotics Program Workshop on Microrobots, Micromachines and Microsystems*, Moscow, Russia, 2003.

[7] N. Lohse. *Towards an ontology framework for the integrated design of modular assembly systems*. PhD thesis, University of Nottingham, 2006.

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

[9] L. E. Parker and F. Tang. Buidling multirobot coalitions through automated task solution synthesis. *Proc. IEEE*, 94(7):1289–1305, 2006.

[10] M. Pechoucek, V. Marik, and O. Stepankova. Coalition formation in manufacturing multi-agent systems. In *Int. Conf. on Database and Expert Systems Application*, London, UK, 2000.

[11] L. Ribeiro, J. Barata, M. Onori, and A. Amado. Owl ontology to support evolvable assembly systems. In *9th IFAC Workshop on Intelligent Manufacturing Systems*, Szczecin, Poland, 2008.

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

[4]http://www.archive.org/details/DynamicCoalitions-Video

[5]http://www.perada.org