Chemically Inspired Rules for Self-Healing Assembly Systems

Giovanna Di Marzo Serugendo[1], Regina Frei[2], and Jose Luis Fernandez-Marquez[1]

[1]Institute of Services Science, University of Geneva, Rte de Drize 7, CH-1227 Carouge, Switzerland
[2]EPSRC Centre in Through-life Engineering Services, Cranfield University, Bedfordshire MK43 0AL
giovanna.dimarzo@unige.ch, work@reginafrei.ch, joseluis.fernandez@unige.ch

Abstract
Self-organising assembly systems (SOASs) are advanced assembly systems that play an active role in their own design and during production. Agentified modules participate in their own arrangement in the system layout, monitor themselves and self-adapt to production conditions. In previous works, we addressed the design phase of the assembly system. We defined a self-organising process based on specific rules, inspired by chemical reactions, such that given a specified product order provided as an input to the system, the system modules select suitable partner modules and choose their own position in the assembly system layout. This paper, specifically tackles the second aspect: self-management of the SOAS to production conditions. We define here a series of rules, similarly inspired by chemical reactions, supporting the SOAS during production. Whenever a failure occurs in one or more of the modules, the current modules adapt their behavior (change speed, force, task distribution, etc.) to cope with the current failure, potentially degrading performance but maintaining functionality. They may also decide to replace a faulty module or trigger a reconfiguration, thus leading to a repaired system. In addition to the rules addressing self-healing and self-management during production, this paper also discusses a specific framework to support the execution of the rules.
Keywords:
Industrial robots, self-management, chemical reactions.

1 INTRODUCTION

Industrial assembly systems are often rather rigid in their structure and limited in the flexibility of their functionalities. Adapting them to necessary changes as well as coping with failures is often a work-intensive and error-prone process, given that assembly systems are complex systems composed of a multitude of heterogeneous units that need to collaborate to successfully assemble the desired product. In this situation, it is highly beneficial to allow assembly systems to increasingly take care of themselves, i.e., to adopt self-* properties.

In previous works, we explained ways for assembly systems to become more agile, adaptive and gradually more autonomous. We proposed the notion of Self-Organising Assembly Systems (SOAS) [10, 11], which become self-organising in their creation phase (spontaneous (re-)organisation of the modules without external control) and self-managing in their production (autonomous adherence of the system to high-level policies). Both phases are based on a model of transitions exploiting the chemical rules metaphor. We described the self-organising phase in previous works [12]. This paper focuses exclusively on self-managing and self-healing aspects during production.

A difficulty consists in defining the right policies and rules for the assembly systems to exhibit the desired self-* properties at run-time. This paper describes a set of chemical rules allowing us to deal with self-managing issues such as: collision avoidance, self-optimisation or dynamic replacement of a faulty robot during production. We express the rules through the SAPERE [9, 28] framework, specifically meant for developing self-* systems (both self-organising and self-managing) with chemical rules.

Organisation of this paper: Section 2 reviews related work in approaches to self-* properties in robotics as well as self-management in software systems. Section 3 presents the essence of Self-Organising Assembly Systems (SOAS). Section 4 details the product used as a case study. Section 5 illustrates the software architecture. Section 6 introduces the SAPERE approach, and Section 7 explains how self-management of industrial robots is achieved in SAPERE. Finally, section 8 draws conclusions and indicates future work.

2 RELATED WORKS

2.1 Self-* Properties in Robotics

Most robotic systems that implement some kind of self-* properties do so through modularity and redundancy; a failing module or robot will be replaced by a

functional one. Usually, the robots or modules used in these approaches are physically and functionally homogeneous, which means that they all have the same physical and functional characteristics. The robotic modules can connect with each other through some mechanism. Often, they exhibit some degree of self-adaptivity and self-organisation. A wide range of self-reconfigurable robots exists [26]. Self-reconfigurable robots are sometimes also referred to as being *polymorphic* because of their ability to change their shape and move in various ways.

Under the title of 'self-repairing mechanical systems', the principles used in reconfigurable robotics - namely self-assembly and self-repair through module replacement - were discussed [22]. This requires component redundancy and functional redundancy, meaning that the system includes several of the same modules, and that a function can be achieved through various module combinations[1]. The idea is that faulty components are excluded from the system and replaced by peers.

As for agility and work heading towards self-* properties in industrial robots, *Holonic Manufacturing Systems* [27, 20] concentrate on morphologic aspects (every item is a whole as well as part of a bigger whole). ADACOR [18, 19] combines holonics with the idea of self-organisation 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.

Reconfigurable manufacturing systems (RMS) [17, 21, 8] are modular and allow the engineer to add / remove functionalities according to the needs. Evolvable Assembly Systems (EAS) [23, 4] offer a solution which includes finely granular modules with local intelligence and a multi-agent 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. SOAS [10] take the EAS paradigm further by letting the systems play an active role and making systems more autonomous and thus more user-friendly.

An architecture for self-adapting and self-managing robotics systems is presented in [7]. It uses distinct meta-level components for sensing, computation and control, which in turn are monitored, managed and adapted by other (higher layer) meta-level components. The components' behaviours are guided by changeable adaptation policies which may be used to activate a set of recovery mechanisms.

To account for the self-adaptability required in robotic systems, a policy and architecture based approach is presented in [14]. The architecture consists of several layers, including a reactive connector topped by a sequencing connector and then a deliberative connector. Successful applications were made with self-adaptive Robocode and Mindstorms robots.

The SHAGE (Self-Healing, Adaptive, and Growing SoftwarE) framework [16] was made to be dynamically adapted to behavioral, situational and environmental changes in complex systems, in particular intelligent service robots. SHAGE consists of [16]:

- a situation monitor to identify internal and external conditions of a software system,
- ontology-based models to describe architecture and components,
- brokers to find appropriate architectural reconfiguration patterns and components for a given situation,
- a reconfigurator to change the architecture based on the selected reconfiguration pattern and components,
- a decision maker able to learn to find the optimal solution of reconfiguring software architecture for a situation,
- and repositories to effectively manage and share architectural reconfiguration patterns, components, and problem solving strategies.

The work described in this paper is different from the related works in various ways; for one, takes agile manufacturing systems a step beyond reconfigurability, as it makes the system participate in its own design and redesign; it includes both self-organisation and self-management and uses in both cases mechanisms modelled by chemical reactions.

2.2 Approaches to Self-Management

Recently, many initiatives related to the developement of self-managing systems have been proposed, mainly: Autonomic Computing [15] (IBM), Adaptive Infrastructure (HP[2]), N1[3] (Sun), Dynamic Systems Initiative[4] (Microsoft), Adaptive Network Care (Cisco), Proactive Computing (Intel), Organic Computing [24] (Fujitsu) (from [25])

The main idea behind autonomic computing [5] is to address complexity by using technology to manage technology. The self-management process is divided in 4 functional areas: self-configuration, self-healing, self-optimisation, and self-protection. In autonomic computing, IBM proposed autonomic managers - " A component that manages other software or hardware components using a control loop". The control loop of the autonomic manager includes monitor, analyze, plan, and execute functions [5]. Mainly, the monitor function provides mechanims for collecting, aggregating, filtering, and reporting details (such as metrics

[1] The principle of degeneracy as discussed in [13] additionally includes structural and functional plasticity, meaning that a component can fulfil more than one function.

[2] http://h20000.www2.hp.com/bc/docs/support/SupportManual/c01755823/c01755823.pdf

[3] www.e-business.com/software/software-pdf/n1.pdf

[4] http://download.microsoft.com/download/e/5/6/e5656886-ad18-4afd-945f-3680278dfd58/DSI%20overview.doc

[5] http://www.research.ibm.com/autonomic/overview

and topologies) from a managed source. The analyze function provides the mechanisms that correlate and model complex situations, allowing the autonomic manager to learn about the environment, and predicting future situations. The plan function provides mechanisms for constructing the actions needed to achieve goals and objectives. The execute function provides mechanisms for controling the execution of a plan [5].

Policy Management for Autonomic Computing (PMAC) is a generic policy middleware platform that can be used to manage multiple aspects of a large scale distributed system such as QoS, configuration. In [1], authors provide an overview of the Policy Management for Autonomic Computing (PMAC) platform, and shows how it can be used for the management of networked systems. A policy is a rule that contains four components: (1) conditions, that determine if the rule is applicable, (2) actions, that are executed if the conditions are true, (3) priority, define which rule is going to be applied when there is a set of rules which conditions are true, and finally (4) role, define which agents are going to be subject to the rules. For example, a policy defined for mail servers, may have the role “mail-server”. Analogously to autonomic computing, the Agent Manager is a policy-based manager, which monitors, analyzes, and plans according to the policies defined for the resources managed by the Agent Manager.

In [3] show that the chemical reaction paradigm approach naturally models self-organisation and self-management and propose a high-order extension of the Gamma chemical programming model [2], for the modelling and description of autonomic systems.

Analogously to Autonomic Computing, Organic Computing [24] focuses on self-* properties, in order to control the global behaviour that emerge from the local interactions. However, while autonomous computing is mainly focused on server architectures that can be managed without active human interference, Organic computing is focused on large collections of intelligence devices providing services to humans adapted to the current requirements of their execution environment. Thus, beside self-organisation, man-machine interaction is an essential part of Organic Computing.

Deriving from principles similar [3], the SAPERE approach, used in this paper, is a chemical reaction based approach supporting in a unified and homogeneous way both self-organisation and self-management of eco-systems of services. The SAPERE infrastructure runs specific middleware executing chemical reactions among mobile nodes.

3 SOAS Overview

Self-organising assembly systems (SOAS) are a specific category of assembly systems where first, a self-organisation process supports the modules in creating the design of the assembly system: in response to an incoming product order, modules (such as grippers, robots, axis or feeders) progressively select their own partners and their location in a layout; and second, the so obtained assembly system self-manages during production time (Figure 1). The self-* processes follow the chemical abstract machine (CHAM) principles.

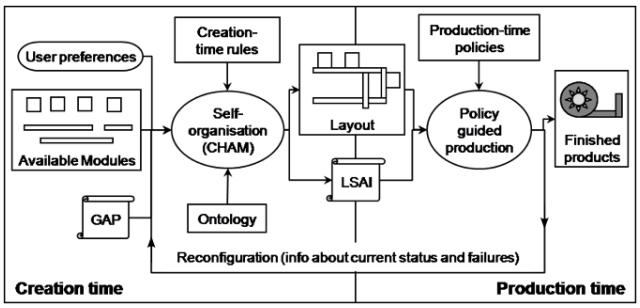

Figure 1: SOAS: creation and production time

In previous research we focused first on the description of SOAS model [11]. In this model, each individual *module* (e.g. gripper, axis, etc.) is "agentified", i.e. it is represented by a software agent that acts on its behalf and provides the capability or the skill of the module under the form of a service to the other modules (e.g. a gripper agents provides the service "open" and "close", while an axis agents provides the service "move"). We denote by *Module Resource Agent* (MRA) such software agents. When two or more modules are combined together, the resulting *robot* is called a coalition and provides a composite service (e.g. a gripper and a linear axis together provide the "pick and place" service) (Figure 2).

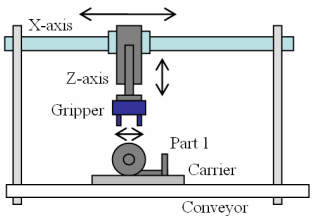

Figure 2: SOAS modules and services

Second, we established the self-organising process that guides the modules to design the assembly system in response of an incoming product order. The approach was inspired by the chemical abstract machine paradigm where "molecules" (e.g. description of tasks to be fulfilled in relation with the product order, services provided by modules) are inserted into a common set or "soup of molecules", where a series of "chemical reactions" (matching tasks and services, linking modules according to their physical characteristics, matching fulfilled tasks with positions in a layout, providing precise movements to assemble the product) progressively helped the modules in designing the assembly system [12] (Figure 3).

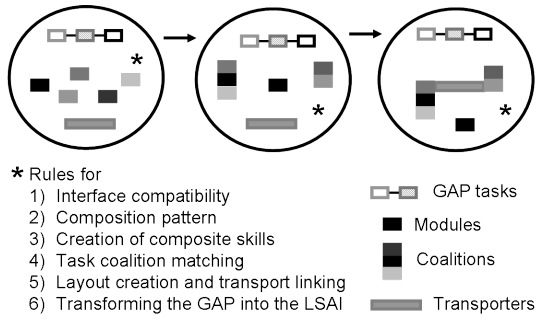

Figure 3: Chemical reactions guiding the design of the assembly System

This paper focuses on the remaining part, i.e. the self-management of the obtained assembly system during production time. Here too, we consider a chemically-inspired approach. Indeed, the SOAS model considers a framework where self-management and self-organisation could be both accommodated in a homogeneous way.

3.1 Production Time Self-* Requirements

During production, modules and compositions of modules coordinate their actions and take care of themselves and their neighbours. Self-* requirements in relation with production are as follows:

- Task sequencing: modules coordinate their work according to the current status of the product being built. The results of each operation is written on a radio-frequency identification (RFID) tag carried by the product being assembled. The next composition of modules in the assembly system will read the RFID and apply the appropriate next steps in the assembly operation.

- Collision avoidance: modules must maintain a minimum distance among each other and the outside environment (e.g. walls, people) while moving.

- Self-optimisation/adaptation to production condition: modules adapt their behavior (change speed, force, task distribution, etc.) in order to cope with the current performance of neighbours or to correct their own precision;

- Self-healing (a): in case of failure of a module or a composition of modules, other modules or robots can overtake tasks of the failing modules.

- Self-healing (b): in case of failure that cannot be overtaken by the current assembly of modules, the layout is changed at production time (addition/exchange/relocation of modules), and thus, the system triggers a self-organisation process to produce a new composition of module, leading to a repaired assembly system.

4 CASE STUDY

For illustration purposes, we assume the following simple product to be assembled: an adhesive tape roller dispenser, consisting of two body case parts ($Part1$ and $Part3$), a tape roll ($Part2$) and a screw ($Part4$), assembled on top of a carrier, as shown in Figure 4. The assembly is made on a workpiece carrier circulating on the conveyors. For more details see [10].

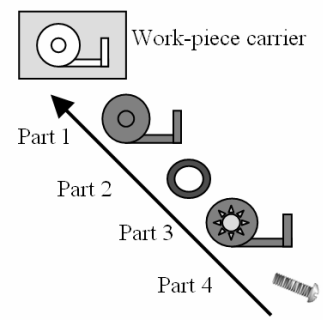

Figure 4: The adhesive tape roller dispenser

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 (Figure 5).

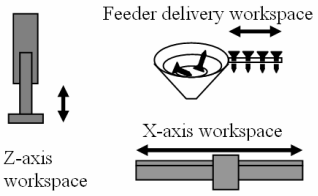

Figure 5: Modules and their workspace

Figure 2 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.

We assume also that the assembly system obtained as a result of the self-organising phase is as shown in Figure 6. All the robots are identical and as shown in Figure 2. Due to the layout's circular structure, workpiece carriers revisit robot $R1$, which executes two different tasks. In this example, we assume that $R1$ works faster than robot $R2$ and robot $R4$, and thus, assembles $Part1$ as well as $Part3$. IN represents the entry of the empty workpiece carriers and OUT means that the carrier with the finished product leaves the system. Feeders $F1$ to $F4$ are placed next to robots and deliver $Part1$ to $Part4$ respectively. Robots are linked by a conveyor system.

We consider the following coordination and self-management cases happening during production: task sequencing, changes in parts delivering (Figure 9), collision avoidance, adaptation to production

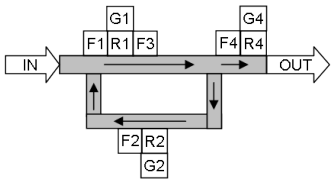

Figure 6: Assembly System Designed through the Self-Organising Process: Grippers (Gi), Robots (Ri) and Feeders (Fi) modules are positioned in the layout

conditions, and self-healing. These cases are developed and explained in Section 7.

5 Architecture

The SOAS architecture follows a service-oriented architecture where MRAs (agents), on behalf of robotic modules, provide services. The architecture integrates both *rules for self-organising mechanisms* and *policies for self-management*. It has been adapted from MetaSelf [6] and exploits dynamically collected and maintained *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 (e.g. agents of the underlying infrastructure).

The MRAs (providing services), 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-* 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). 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“).

Figure 7 shows the architecture distributed at the level of individual modules (MRA-1) and groups of modules (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). Compositions of modules do not have a controller (only modules have) but have metadata and policies shared by the agents in that composition; 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 10s. Notice that because coalitions are dynamically created in response to an incoming production order 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.

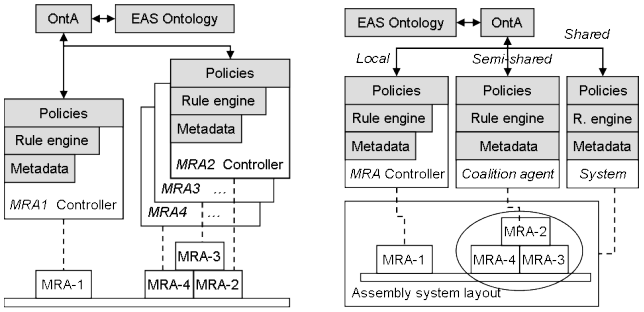

Figure 7: Architecture: component view

6 SAPERE Overview

We summarise here the SAPERE framework and how SOAS are mapped to it. For a more extensive discussion of the SAPERE framework, readers are referred to [9, 28]. The main concepts of a SAPERE system are as follows (see also Figure 8):

“Live Semantic Annotation” (LSA): An LSA is a tuple that represents any information about an agent or service. LSAs are injected in the LSA’s space representing the (updated) state of its associated component. Relating to SOAS, LSAs are carrying all information MRAs want to exchange with each other: data such as MRAs services requests or MRAs services descriptions, coordination data such as assembly status of the current product being assembled, metadata such as monitored performance, and policy’s descriptions.

LSA’s Space: a distributed, shared space where context and information are provided by a set of LSAs stored at given locations (in specific computing nodes). To simplify the design, we consider here only one computing node and one LSA space. Individual MRAs, coalition or the whole assembly system access/update their respective LSAs information through the LSA space. Not all MRAs or coalitions have access to all LSAs. They have access only to those LSAs they have produced themselves (i.e. information about themselves - description, own perfor-

mance, etc.) and to information to which they register or are bound do (i.e. performance of neighbours).

ECO-LAWS: Eco-laws are chemical rules that act on the LSAs stored in each LSA's Space by deleting, updating, creating or moving LSAs between LSA's Spaces. Eco-laws reside in the LSA's Space and are invoked following an implicit invocation pattern. In SOAS, two types of eco-laws are considered: those that define the chemical rules that support the self-organising process of a SOAS creation (design of the assembly system); and those that support self-management. This paper is concerned only with the second type. Eco-laws supporting self-management work as follows: based on an LSA describing a policy attached to an MRA or a coalition, and an LSA inserted by an MRA requesting some action from on another MRA, the self-managing eco-law seamlessly grants or not the request.

LSA Bonding: An LSA bond acts as a reference to another LSA and provides fine-tuned control of what is visible or modifiable to each Agent and what is not. If an LSA of a given Agent includes a bond to an LSA of another Agent, the former Agent can inspect the state and interface of that other Agent (through the bond) and act accordingly. In SOAS, MRAs of a coalition are linked with bonds to the LSAs of the different MRA part of the coalition, describing their current status, as well as to an LSA describing the status of the coalition as a whole. In addition, individual MRAs or coalitions are bond to additional LSAs specifying policies.

Agents: Agents execute in hosts and are able to locally access the LSA's Space available in that host. In SOAS, the Agents are the MRAs and the different coalition agents.

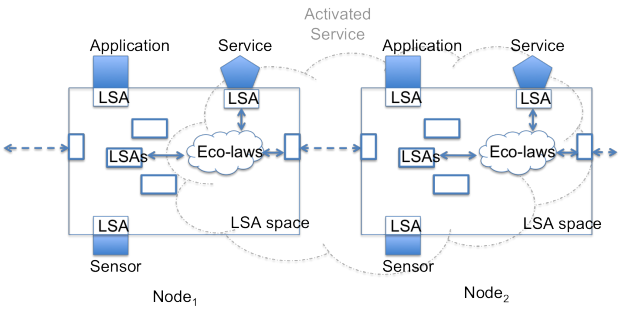

Figure 8: SAPERE Framework

7 SELF-MANAGEMENT IN THE SAPERE FRAMEWORK

This section describes an instantiation of the SOAS architecture and the handling of policies and self-managing cases in the SAPERE infrastructure for the case study discussed in Section 4 and show how they are handled in the SAPERE framework.

7.1 Case Study Revisited

Task sequencing: a carrier containing the product being assembled circulates along the layout carrying an RFID tag logging the tasks applied on the product. According to the current status reported in the RFID, $R1$ that sees each product twice, applies the appropriate operation (i.e. adding $Part1$ or $Part3$).This is a coordination activity. Here we have the LSA (corresponding to the RFID tag) of the product being built that describes its current assembly status. In this case, this LSA stores the following information: $\langle \mathtt{product_{ID}}, \mathtt{Part_1}, \mathtt{Part_2} \rangle$: it specifies that product with id $\langle \mathtt{product_{ID}} \rangle$ is currently made of $Part_1$ and $Part_2$, thus the MRAs coalition R_1 by accessing the information stored in this LSA, knows that it has to add $Part_3$. Once $Part_3$ is successfully added to the product, the product agent updates the LSA, which will now read as: $\langle \mathtt{product_{ID}}, \mathtt{Part_1}, \mathtt{Part_2}, \mathtt{Part_3} \rangle$.

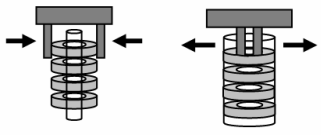

Figure 9: Stick Tube: change in the delivery

Changes in tape roll conditioning: instead of being delivered in a tube, the tape rolls are delivered on a stick. Gripper $G2$ thus grabs the tape roll from the outside instead of from the inside, see Figure 9.This is also a coordination activity. Feeder of $Part_2$ provides and updates an LSA that describes the way parts are delivered: in a tube $\langle \mathtt{Feeder_2}, \mathtt{tube} \rangle$, or updates this information when they are delivered on a stick: $\langle \mathtt{Feeder_2}, \mathtt{stick} \rangle$. By accessing the LSA of feeder F_2, the MRA for Gripper G_2 adapts its way of handling the pieces.

Collision avoidance (a): the distance between MRAs must always be bigger than a safety threshold, unless they are members of the same composition of modules. Robots $R1$, $R2$, $R3$ must not interfere with each other. This case needs a bounding environmental policy, stating the minimum distance between two robots coalitions is above a certain threshold, or that workspaces never overlap: $\langle \mathtt{Policy_{ENV}}, \mathtt{NO\_OVERLAPPING\_SPACES} \rangle$. Each time one of the robots R_i moves, the LSA representing the status of the corresponding coalition is updated with a value representing the current 3D workspace occupied or by the robot: $\langle \mathtt{R_i}, \mathtt{workspace_i} \rangle$. An avoidance collision eco-law then considers the environmental policy and the current workspaces and injects or not a specific LSA $\langle \mathtt{ID}, \mathtt{STOP} \rangle$ for stopping the MRAs work:

$$
\begin{aligned}
&avoidCollisionEcolaw \ :: \\
&\langle \mathtt{Policy_{ENV}}, \mathtt{NO\_OVERLAPPING\_SPACES} \rangle, \\
&\langle \mathtt{R_1}, \mathtt{workspace_1} \rangle, \langle \mathtt{R_2}, \mathtt{workspace_2} \rangle, \langle \mathtt{R_4}, \mathtt{workspace_4} \rangle \\
&\mapsto \\
&\langle \mathtt{Policy_{ENV}}, \mathtt{NO\_OVERLAPPING\_SPACES} \rangle, \\
&\langle \mathtt{R_1}, \mathtt{workspace_1} \rangle, \langle \mathtt{R_2}, \mathtt{workspace_2} \rangle, \langle \mathtt{R_4}, \mathtt{workspace_4} \rangle \\
&\langle \mathtt{ID}, \mathtt{STOP} \rangle, if \ workspaces \ overlap
\end{aligned}
\tag{1}
$$

Robots and MRAs permanently check for the stopping LSA is in the LSA space, and stop moving if it is present. In real case scenarios, priorities among policies need to be considered for ensuring the system stops immediately.

Collision avoidance (b): 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.The sensor updates its corresponding LSA $\langle \texttt{sensor}_{\texttt{ID}}, \texttt{IRREGULARITY} \rangle$, to inform the system about an irregularity. As soon as the sensor LSA with the information about the irregularity is injected in the LSA space, a safety eco-law produces the stopping LSA $\langle \texttt{ID}, \texttt{STOP} \rangle$:

$$\begin{array}{l} safetyEcolaw \ :: \\ \langle \texttt{sensor}_{\texttt{ID}}, \texttt{IRREGULARITY} \rangle \\ \mapsto \\ \langle \texttt{sensor}_{\texttt{ID}}, \texttt{IRREGULARITY} \rangle \\ \langle \texttt{ID}, \texttt{STOP} \rangle \end{array} \quad (2)$$

Self-optimisation/adaptation to production conditions: feeders $F1$ to $F4$ adapt the feeding speed to the part removal speed provided by the joint work of robots and the grippers $G1$ to $G4$. Feeders always need to deliver parts at their output. If the parts are taken away quicker, their feeding speed should be higher. This is a coordination task, where Gripper Agents update regularly their corresponding LSA with their speed $\langle \texttt{Gripper}_{\texttt{i}}, \texttt{speed}_{\texttt{i}} \rangle$. Feeder Agents, with bonds to Gripper Agents, read the Grippers LSA and adapt their speed accordingly.

Self-healing (a): $R2$ performance decreases (e.g. decrease in speed or too many broken pieces). Either R_1 or R_2 itself notices the slowing down performance reflected in the performance metadata associated with $R2$. $R1$ takes over from $R2$ until the latter is replaced. LSA of Robot $R2$ indicates the current performance $\langle \texttt{R}_2, \texttt{speed} \rangle$. A self-healing takeover eco-law modifies the LSA of R_1 indicating that it should take over from R2:

$$\begin{array}{l} takeOver \ :: \\ \langle \texttt{R}_2, \texttt{speed} \rangle \\ \mapsto \\ \langle \texttt{R}_2, \texttt{speed} \rangle, \langle \texttt{R}_1, \texttt{TakeOverFromR2} \rangle, \\ if \ speed < threshold \end{array} \quad (3)$$

Self-healing (b): After overtaking from $R2$, $R1$ experiences a high level of queuing. This leads the system to ask for a reconfiguration and, therefore, triggers the self-organising process to provide a new design of the assembly system. This is performed with the following eco-law:

$$\begin{array}{l} newDesign \ :: \\ \langle \texttt{R}_1, \texttt{TakeOverFromR2}, \texttt{queueLevel} \rangle \\ \mapsto \\ \langle \texttt{R}_1, \texttt{TakeOverFromR2}, \texttt{queueLevel} \rangle, \\ \langle \texttt{ID}, \texttt{STOP} \rangle, \langle \texttt{system}_{\texttt{ID}}, \texttt{REDESIGN} \rangle, \\ if \ queueLevel > threshold \end{array} \quad (4)$$

The main distinction between the SAPERE approach and conventional approaches lies in the fact that control and management occurs locally and in a decentralised manner. There is no central control entity that supervises all the modules and triggers repairs or halts when necessary. Instead the different modules go ahead in their work, it is only when their actual respective configurations (such as speed, positions reflected in the LSAs) fires a chemical reaction that the latter, by inserting the appropriate LSA, then triggers the expected behaviour in the modules.

8 CONCLUSION

This paper describes how using a chemically-inspired technique, we can perform coordination and self-managing tasks within industrial robots at production time. The specific examples have been instantiated in the SAPERE framework, an EU project defining a chemically-inspired framework for self-organising and self-managing context-aware systems. Future work will concern the development of more self-managing cases for SOAS and the development of corresponding simulations.

ACKNOWLEDGEMENT

This work is partially supported by the EU-FP7-FET Proactive project SAPERE Self-aware Pervasive Service Ecosystems, under contract no.256873.

REFERENCES

[1] D. Agrawal, Kang-Wong Lee, and J. Lobo. Policy-based management of networked computing systems. *IEEE Commun. Mag.*, 43(10):69–75, October 2005.

[2] J.-P. Banâtre, P. Fradet, and D. Le Métayer. Gamma and the chemical reaction model: Fifteen years after. In *WMP*, volume 2235 of *LNCS*, pages 17–44. Springer, 2000.

[3] J.-P. Banâtre, Y. Radenac, and P. Fradet. Chemical specification of autonomic systems. In *In: Proc 13th International Conference on Intelligent and Adaptive Systems and Software Engineering*, pages 72–79, 2004.

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

[5] IBM Corporation. An Architectural Blueprint for Autonomic Computing - White Paper. June 2006.

[6] G. Di Marzo Serugendo, J. Fitzgerald, and A. Romanovsky. MetaSelf - an architecture and development method for dependable self-* systems. In *Symp. on Applied Computing (SAC)*, pages 457–461, Sion, Switzerland, 2010.

[7] G. Edwards, J. Garcia, H. Tajalli, D. Popescu, N. Medvidovic, G. Sukhatme, and B. Petrus. Architecture-driven self-adaptation and self-management in robotics systems. *Int. Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 142–151, 2009.

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

[9] F. Zambonelli et al. Self-aware pervasive service ecosystems. *Procedia Computer Science*, 7:197–199, 2001.

[10] R. Frei. *Self-organisation in Evolvable Assembly Systems*. PhD thesis, Department of Electrical Engineering, Faculty of Science and Technology, Universidade Nova de Lisboa, Portugal, 2010.

[11] R. Frei and G. Di Marzo Serugendo. Self-organising assembly systems. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, 41(6):885–897, November 2011.

[12] R. Frei, G. Di Marzo Serugendo, and T.F. Serbanuta. Ambient intelligence in self-organising assembly systems using the chemical reaction model. *J. of Ambient Intelligence and Humanized Computing*, 1(3):163–184, 2010.

[13] R. Frei and J. Whitacre. Degeneracy and networked buffering: principles for supporting emergent evolvability in agile manufacturing systems. *Accepted for publication in Journal of Natural Computing - Special issue on Engineering Emergence*, 2011.

[14] J.C. Georgas and R.N. Taylor. Policy-based self-adaptive architectures: a feasibility study in the robotics domain. In *Int. Workshop on Self-Adaptation and Self-Managing Systems (SEAMS)*, pages 105–112, New York, NY, USA, 2008. ACM.

[15] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, January 2003.

[16] D. Kim, S. Park, Y. Jin, H. Chang, Y.S. Park, I.Y. Ko, K. Lee, J. Lee, Y.C. Park, and S. Lee. SHAGE: a framework for self-managed robot software. In *Int. Workshop on Self-Adaptation and Self-Managing Systems (SEAMS)*, pages 79–85, New York, NY, USA, 2006. ACM.

[17] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritchow, A.G. Ulsoy, and H. Van Brussel. Reconfigurable manufacturing systems. *CIRP Annals - Manufacturing Technology*, 48(2):6–12, 1999.

[18] P. Leitão. *An agile and adaptive holonic architecture for manufacturing control*. PhD thesis, Department of Electrical Engineering, Polytechnic Institute of Bragança, Portugal, 2004.

[19] P. Leitao and F.J. Restivo. Implementation of a holonic control system in a flexible manufacturing system. *Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(5):699–709, 2008.

[20] V. Marik, V. Vyatkin, and A.W. Colombo, editors. *Holonic and Multi-Agent Systems for Manufacturing*. Springer, Heidelberg, 2007.

[21] M.G. Mehrabi, A.G. Ulsoy, and Y. Koren. Reconfigurable manufacturing systems: Key to future manufacturing. *Journal of Intelligent Manufacturing*, 11:403–419, 2000.

[22] S. Murata, E. Yoshida, H. Kurokawa, K. Tomita, and S. Kokaji. Self-repairing mechanical systems. *Autonomous Robots*, 10:7–21, 2001.

[23] M. Onori. Evolvable assembly systems - a new paradigm? In *33rd Int. Symposium on Robotics (ISR)*, pages 617–621, Stockholm, Sweden, 2002.

[24] H. Schmeck. Organic computing - a new vision for distributed embedded systems. *Object-Oriented Real-Time Distributed Computing, IEEE International Symposium on*, 0:201–203, 2005.

[25] R. Sterritt and M. Hinchey. SPAACE IV: Self-Properties for an Autonomous & Autonomic Computing Environment - Part IV A Newish Hope. In *Proceedings of the 2010 Seventh IEEE International Conference and Workshops on Engineering of Autonomic and Autonomous Systems*, EASE '10, pages 119–125, Washington, DC, USA, 2010. IEEE Computer Society.

[26] K. Stoy, D.J. Christensen, and D. Brandt. *Self-Reconfigurable Robots: An Introduction*. MIT Press, 2010.

[27] P. Valckenaers and H. Van Brussel. Holonic manufacturing execution systems. *CIRP Annals - Manufacturing Technology*, 54(1):427–432, 2005.

[28] M. Viroli, D. Pianini, S. Montagna, and G. Stevenson. Pervasive ecosystems: a coordination model based on semantic chemistry. In Sascha Ossowski, Paola Lecca, Chih-Cheng Hung, and Jiman Hong, editors, *27th ACM Symp. on Applied Computing (SAC 2012)*, Riva del Garda, TN, Italy, 26-30 March 2012. ACM.