A Sample Application of ADELFE Focusing on Analysis and Design the Mechanical Synthesis Problem

Davy Capera, Gauthier Picard, Marie-Pierre Gleizes, and Pierre Glize

IRIT, Université Paul Sabatier,
F-31062 Toulouse Cedex, France
{picard, capera, gleizes, glize}@irit.fr
http://www.irit.fr/SMAC

Abstract. This paper aims at explaining how to follow an agent-oriented process to develop a multi-agent mechanism design system. ADELFE methodology is devoted to adaptive multi-agent systems in which adaptation is enabled by cooperative self-organization. Two main works are emphasized. First, the analysis leads to the agent identification by studying the interactions both between the system and its environment and within the system itself. Second, the different modules of the agents and their cooperative attitude are modeled during the design phase. Such an approach is promising, but raises some difficulties considering the notion of cooperation, which is discussed before concluding.

1 Introduction

In spite of the ever increasing number of developed agent-oriented applications, Multi-Agent Systems have still not manage to break through to the industrial side. This lack of acknowledgment may have two main reasons. First, to transmit any technology to industrial level, academics must provide guidelines, formalisms and tools to manipulate their technologies. MAS community has done a large effort in this methodological domain. DESIRE [1] or GAIA [2] are examples of early agent-oriented methodologies embedded in industrial projects that provide rich formalisms. Moreover, for a few years, a real work has been made in normalizing agent-oriented concepts, as in the AUML community within the OMG [3]. Some methodologies, such as ADELFE[1] [4] has been enriched by pedagogical tutorials –significant examples of how to develop an application with a given method– and tools to enlarge the multi-agent developers community, in the same way than MESSAGE/INGENIAS [5] or agentTool [6]. The second reason may be the lack of real applications for everyday industrial use. MAS community has

[1] ADELFE a French RNTL-funded project which partners are ARTAL Technologies and TNI-Valiosys from industry and IRIT and L3I from academia.

M.-P. Gleizes, A. Omicini, and F. Zambonelli (Eds.): ESAW 2004, LNAI 3451, pp. 231–244, 2005.

to exhibit proofs of the agent technology efficiency and to tackle unsolved or mis-solved problems. For instance, the time table (or scheduling) problem has not been really solved by agent-technology, even if some works seem promising [7]. But real comparisons to already used technology, such as CSP-based or evolutionary solvers, has not yet exhibited agents' relevance. In this article, as an example of this kind of application, we will expound a problem which has been proposed by the aeronautics industry: the mechanism design.

Mechanism design consists in assembling mechanical components such as links (rigid bodies, bars) and joints (hinges, cams, etc.), in order to build a mechanical system which performs a specific function such as following a precise trajectory. Our objective is to develop an automated tool – based on the adaptive multi-agent systems paradigm – which is able to synthesize a mechanism from a given set of goals and constraints through a self-assembling process. ADELFE method is devoted to develop applications in which components cooperatively self-organise to reach an adequate and adapted functionnality, and thus ADELFE is relevant to tackle the mechanical synthesis problem. Currently, there is no software, used by industrial mechanism designers, that includes any tool to support the synthesis phase of the mechanism design. Nevertheless, some researches were done in this way by [8], using a multi-expert system based on agent principle, or [9] which is based on an abstract representation of the kinematic structures.

With the aim of illustrating the use of the ADELFE methodology to a problem of mechanical synthesis, the article is structured as follows. Section 2 expounds the ADELFE methodology and its foundations. Sections 3 and 4 explain the application of the analysis and design phases, i.e. how to fill in the different modules of the agents. Finally, after a brief discussion in section 6, the paper is concluded with some perspectives.

2 ADELFE Overview

ADELFE enables the development of software with emergent functionality and consists in a notation based on UML/AUML, a design method, a platform made up of a graphical design tool [10] and a library of components that can be used to make the application development easier.

2.1 Process, Notations and Tools

The objective of ADELFE is to cover all the phases of a classical software design from the requirements to the deployment. ADELFE process is an extension of the RUP (Rational Unified Process) [4] and adds some specific steps to design adaptive systems. Only the requirements, analysis and design require modifications in order to be tailored to AMAS and are presented in the next paragraphs. ADELFE focuses on some aspects not already considered by existing methodologies such as complex environment, dynamics or software adaptation led by self-organization during which models are specified using UML/AUML nota-

tions [3]. OpenTool software, released by TNI-Valiosys, has been modified to fit with these notations and Adaptive Multi-Agent Systems (AMAS) requirements. Likewise, the process is supported by an interactive tool with the aim of easing following the process by monitoring the advancement, the produced models and documents, and by providing a guideline example. These tools are freely available at `www.irit.fr/ADELFE`.

2.2 Theoretical Foundations

The Adaptive Multi-Agent System (AMAS) theory has been used in several projects and applications. It aims at defining adaptive multi-agent systems in which self-organization is led by local cooperation. In such systems, adaptation is viewed as a three-level concept. At the higher level, the system has to compute an adequate coherent function as a reaction to the dynamisms of its environment. To obtain a new function that fits better with the environment, the system has to change its internal organization at the medium level. The organization of the MAS is represented by links between the agents. These links can be for instance workflow links, belief links or more simply communication links. Organizational changes are consequences of local link changes between agents. Therefore, at the lower level, agents must know how to change their links with their acquaintances to produce a new organization, and then a new coherent function, which induces the need of a local –i.e. not explicitly informed of the global function– criterion to decide when changes are necessary. The AMAS theory proposes to use the cooperation criterion as the engine of self-organization. Cooperation is defined in three points corresponding to the "perceive - decide - act" agents' life-cycle [11]:

(c_1) All perceived signals must be understood without ambiguity
(c_2) The received information is useful for the agent's reasoning
(c_3) Reasoning leads to useful actions toward other agents

Consequently, an agent must change its links when it detects it does not fit with this criterion. We then use a proscriptive definition of cooperation ($\neg c_1 \vee \neg c_2 \vee \neg c_3$) called non cooperative situations (or NCS). If an agent is in a NCS, it acts to try to act to come back to a cooperative situation (to be as cooperative as possible). This approach can be viewed as an exception oriented specification, where the program functionality is principally defined by "what it must not do" instead of "what it must do" in classical programming. This framework enables to prove the functional adequacy theorem [11]: *"For any functionally adequate system in a given environment, there is a system having a cooperative internal medium which realizes an equivalent function"*. It has an important methodological impact : to design a system, designers only have to ensure agents' behaviors are cooperative to ensure the system provides an adequate function. It also raises some problems : "How can we design agents to obtain coherent societies?" The ADELFE methodology aims at providing an answer to these design difficulties.

3 Analysis

The analysis work aims at abstracting the application domain into classes starting from the requirements. In an agent-oriented process, it should also lead to the identification of the agent classes by analyzing the interaction models.

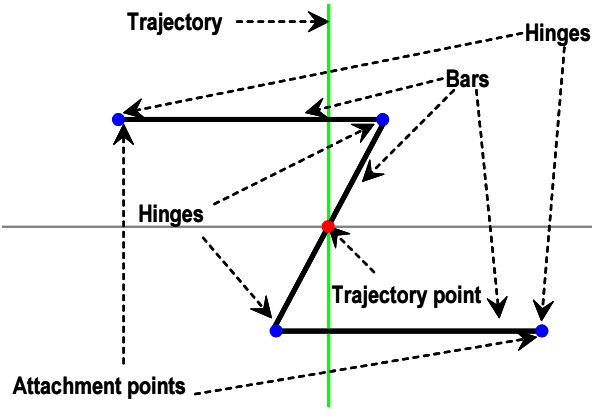

Fig. 1. An example of a 4-bars mechanism

3.1 Domain Analysis

As we are working collaboratively with partners coming from aeronautical industry (European project SYNAMEC[2]), our software – named Mechanical Synthesis Solver or MSS – focuses on solving design problems for "X-bars" mechanisms. This kind of mechanisms contains only three types of mechanical components: rigid bodies, joints (rotoidal or prismatic) and attachment points. Usually, the goal of the mechanism consists in following a given trajectory, in avoiding obstacles and in staying contained in a given envelope. Mechanisms devoted to control plane wings ailerons or outer flaps of nozzle fit with this "X-bars" mechanisms pattern. The figure 1 shows a "4-bars" mechanism made up by four rotoidal joints (hinges), three rigid bodies (bars) and two attachments points to the envelope – which is not drawn here[3]. The actuation is a rotation of the upper left hinge which cyclically performs a top-down wipe while the "trajectory point" roughly fits with the vertical trajectory all along. In section 2.2, we claimed that the system adaptation is triggered by interactions occurring between the system and its environment. Obviously, the motion computation of mechanisms is not a trivial task and so we will use an external tool MECANO to compute this

[2] SYNAMEC is a Growth European project involving Samtech (Belgium), ALA (Italy), Cranfield University (England), INTEC (Argentina), Paul Sabatier University (France), SABCA (Belgium), SNECMA (France).

[3] The envelope on which the attachment points are fixed is topologically viewed as the fourth bar

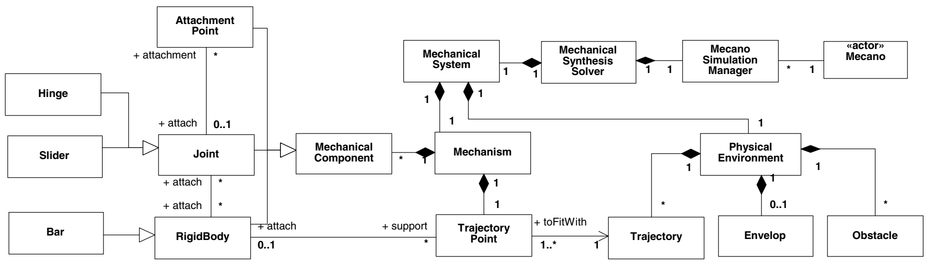

Fig. 2. Preliminary class diagram of the MSS domain

simulation of the system activity. Thus, the MSS learns a relevant mechanism by using a loop composed of the following phases:

1. The simulation engine computes the motion of the current mechanism.
2. Data related to the new mechanism state are sent to the mechanical components in order to update their state.
3. In compliance with the AMAS theory, MSS performs optimization by solving NCS detected by the agents, which leads to a new mechanism.

The interface with the simulator is performed through files which describe, in input, characteristics of the mechanism, the environment and the commands related to the desired motion (the "actuation function"). Moreover this interface allows to recover the new mechanism characteristics after MECANO has computed the motion. The domain description is represented in a preliminary class diagram shown in figure 2.

3.2 AMAS Adequacy

The next activity of ADELFE analysis is related to the AMAS theory. With through eleven questions, an ADELFE tool provides a measure of the AMAS adequacy degree. That tool answers two main questions : Is the use of an adaptive multi-agent system useful to tackle the problem ? Are there any parties of the system which need to be broken up into adaptive multi-agent systems ?

The main criterion that leads to use an AMAS for the mechanism design problem, is the complexity: there is a huge number of mechanisms to check. Moreover, this search space is discontinuous: minor topological modifications can lead to drastic modification of the global behavior. The fact that the system is open (some components are added to the mechanism during the process) and dynamical (the user can interact with the system to drive the process) increases the complexity. Finally, mechanisms are made up by several, and potentially very numerous, more simple entities, which match with multi-agent system paradigm.

3.3 Agent Identification

Agent identification relevance depends on the nature of the problem to tackle. When considering systems with which humans strongly interact, designers nat-

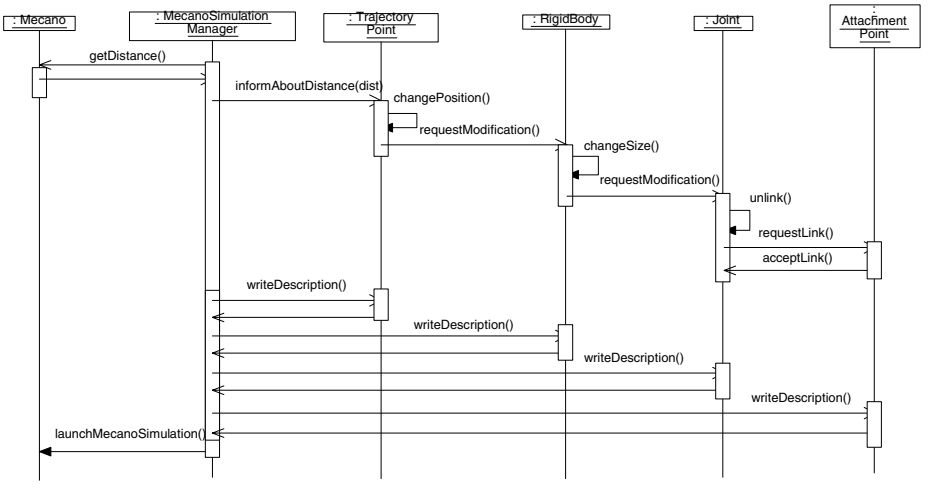

Fig. 3. A sample MSS sequence diagram

urally identify agents as representative of human actors. Therefore, methodologies which are specific to this kind of problems do not address this issue. In the analysis workflow of GAIA [2], the agents are already identified and the methodology does not provide anything to realize this identification. But, when considering simulation-based solving systems, such as MSS, agent identification is not so clear.

As a function of the theoretical paradigms used, methodologies propose several ways to identify agents. In TROPOS the agents are found inside the set of actors, but it results from the analysis of the predefined goals and soft-goals [12]. In AAII, the elaboration and refinement of the agent model and the interaction model help the designer to define agents [13]. The agent definition, which is given in MESSAGE [14] and in ADELFE, defines the features that will be ascribed to the entities that the developer will choose to consider as agents. Moreover, ADELFE proposes to analyze preliminary interaction models to identify high-risk entities in terms of cooperation or cognition.

Figure 3 shows a typical information flow from the MECANO application to each part of a mechanism (and the other way round). Considering ADELFE, agent identification focuses on the analysis of active objects. From the MECANO result file, the MecanoSimulationManager gets the distance between the Trajectory and the TrajectoryPoint. So, if this distance is different from zero, the position of the TrajectoryPoint has to be changed to get closer to the Trajectory. Either, the TrajectoryPoint changes its position by itself, or this modification is performed by the RigidBoby which it is attached to, as in this example. In the same way, a RigidBody modification could be obtained by changing its shape, its position or by delegating the problem to its neighbors –Joint in the example. Therefore, in a general way, this kind of problem can be solved either by changing properties of the given component or by propagating the problem. But, sometimes the problem

could not be solved thanks to these modifications, and so the topology of the mechanism has to be changed. In our example, the link between the RigidBody and the Joint is broken and a new link with a compatible component (AttachmentPoint for instance) has to be created. Finally, the mechanical components inform the manager of their positions as input for the next MECANO computation.

By using AMAS terminology, TrajectoryPoint can be in a NCS –called incompetence– when it does not fit with the trajectory. In the same manner, a RigidBody or AttachmentPoint are useless if not attached to joints, and vice versa. In addition, these MechanicalComponent classes and the TrajectoryPoint have to manage their acquaintances not to flood the system by information exchange, and have to be able to add or remove agents from their acquaintance data base. These criteria are sufficient to identify these classes as being cooperative agent classes. To sum up, the identified agent classes are: MechanicalComponent and all its sub-classes, and TrajectoryPoint (see figures 2 and 4). To encapsulate common properties, a top-level generic MechanicalAgent class from which the two previous classes inherit is defined.

4 Agent Design

Generally, in object-oriented methodologies, the design works lead to a precise definition of classes in terms of fields, methods, and also results in the model of the dynamic behaviors of these classes by using sequence diagrams and state-machines. In ADELFE, these two aspects – static and dynamic – are met by filling the different modules of the agents (including their cooperative attitude) and by modeling their interaction languages with AUML protocols.

4.1 Agents' Modules

ADELFE proposes a generic cooperative agent model [15]. This model decomposes agent's cognitive functions into six modules. The main mechanical agents' modules are shown in figure 4, in *bold face* : characteristics, perceptions, actions, skills, aptitudes, representations and cooperation.

The Characteristic module which manages the intrinsic properties of agents. These mainly are the points (named "noeuds" or nodes in MECANO) which describe the mechanical components (the two extremities of a bar, the two attach points of a hinge, etc.). Each characteristic can be updated either by the agent itself or consequently to a specific message reception from MECANO (after a simulation step).

The Interaction module is the interface between the agent and its environment. This module is made up of two submodules: the Action module and the Perception module. The Action module contains the actions available for the agent and manages the messages sending (sendMessage), i.e. the outputs. Figure 4 shows agents always have actions related to their characteristics (linkAttach1, linkAttach2, unlinkAttach1 or unlinkAttach2) and other specific actions (changePosition in Bar and TrajectoryPoint).The Perception module mainly manages the re-

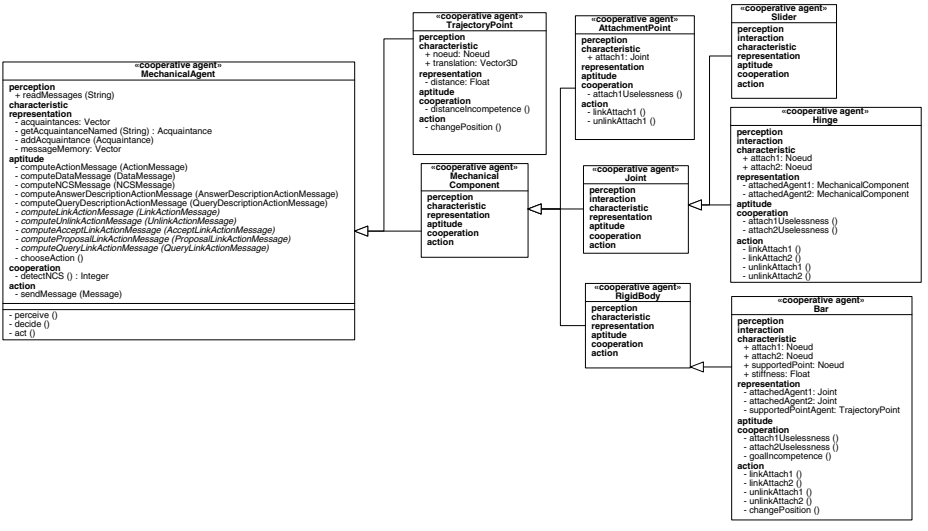

Fig. 4. The agent class diagram for MSS

ceived messages (a mechanical agent only perceives its environment through messages by using readMessage method), i.e. the inputs.

The Representation module manages information about the agent's environment (social or physical) or itself. It contains the acquaintances of the agent, i.e. information about the other agents of the system. For each known agent, the representation module contains its type, its name, etc. This knowledge base is dynamically updated during the agent lifetime (addAcquaintance). MechanicalAgent can also store previously received messages into a messageMemory. Moreover agents can have information on themselves in terms of their type – for example a Bar and Hinge owns information about agents they are connected to – or specific data such as the distance to the trajectory for the TrajectoryPoint.

The Cooperation module contains the list of the NCS (see section 2.2). Each NCS is described by a conditional trigger and by the list of actions that the agent can use to solve the problem. This module is described in the section 4.2.

The Aptitude module manages messages processing according to the message type and action selection. For instance every MechanicalAgent can ask its description to another agent and answer to such a query. Therefore, it must have aptitudes to process these kind of messages (computeQueryDescriptionActionMessage and computeAnswerDescriptionActionMessage). Moreover, since MechanicalAgents has to self-assembly, they must be able to process messages about link or unlink actions (computeLinkActionMessage, computeUnlinkActionMessage, computeQueryLinkActionMessage, etc.). These abstract methods must be implemented in every MechanicalAgent's subclass. The chooseAction method selects the actions to do at a given time considering the current state of knowledge (perception, messages, representation and triggered NCS).

The Skill module is not used for MechanicalAgents because they don't perform their "function" themselves (it is simulated by MECANO) and then they can't have any knowledge about their real activity – MECANO computation is a black-box from the agents' point of view.

4.2 Agent's Cooperative Attitude

The Cooperation module contains the local cooperation rules that lead agents' self-organization. Ideally, every agent knows when it is no more cooperative and then knows what to do to come back to a cooperative state. This is the main design challenge of AMAS: find all the NCS. In the MSS application, two main NCS have been identified:

uselessness: when an agent does not have all its partners. For example, a Bar or a Hinge or a AttachmentPoint is useless if one of its attachment points is not connected to another MechanicalComponent. To solve this NCS, agents must find a partner by negotiating for example as seen in section 4.3.

incompetence: when an agent does not reach its goal. For example, a TrajectoryPoint agent is incompetent if its distance to the trajectory is different from zero. This situation has already been detected during the agent identification as being a criterion to "agentify" the TrajectoryPoint class. The relevant action that agents must perform in such situations is to change their position to get closer to their goal for example.

Every agent owns a method called detectNCS which applies cooperation rules and put the solving actions in an "action-to-do" list that will be used by the Aptitude module when it performs chooseAction.

4.3 Agents' Interactions

ADELFE proposes to model agent interactions by specifying AUML protocols [3]. These protocols enable to describe agents' interaction languages (the meth-

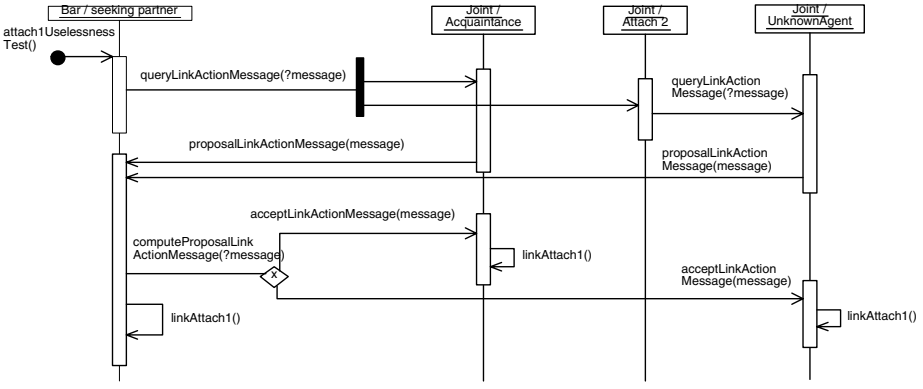

Fig. 5. A sample protocol diagram for MSS

ods they need to communicate) and resolution negotiation algorithm. For example, the figure 5 shows the resolution protocol for a Bar agent to solve a uselessness NCS. Here, the Bar sends a query to two Joints it knows: the one which is already linked to (Attach2 role) and another it may have encountered before (Acquaintance role). This last one directly answers positively. On the other hand, Attach2 cannot answer positively because it is already linked. Therefore, it delegates and informs another agent it knows about the situation (UnknownAgent role). This last one answers positively too. Now, the Bar must analyze the proposals and decide which agent will be connected to the free attachment. This decision making process is labeled by the computeProposalLinkActionMessage method on the XOR node. In terms of the results of this decision, the Bar may send acceptance to its preferred partner and then perform a link action.

5 Some Results for Dimensional Adjustment

A prototype of the modeled system has been implemented for "X-bars" mechanism problems, and especially to study the dimensional adjustment. These experiments focus on the resolution actions related to distanceIncompetence and goalIncompetence NCS.

5.1 Reactive Algorithm

Firstly, a very reactive algorithm has been tested. The Bar's NCS resolution action is directly induced by the stiffness value it perceives – the Bar applies a

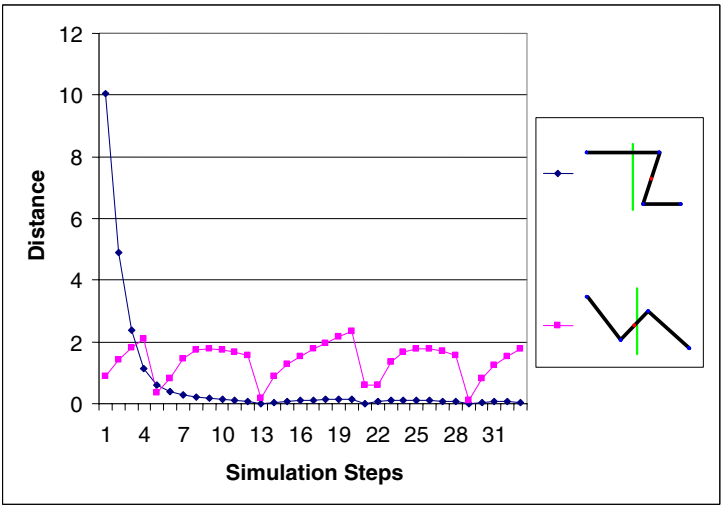

Fig. 6. Results for two different 4-bars mechanisms with a reactive resolution algorithm

translation vector to each of its ends – that leads to a modification of its position and/or length. In the same manner, the TrajectoryPoint computes a translation for its position.

Figure 6 shows the results for two different 4-bar simulations with initial mechanisms whose dimensions are faulty. Distance between the trajectory point and the trajectory quickly decreases for the first example. Simulation results in a well-sized mechanism as shown in figure 1. Nevertheless, the distance is not exactly null because the only solution is not a straight line (as specified by the user) but a very thin 8-shaped cycle. On the other hand, the reactive algorithm fails to find a solution for the second example. In fact, this algorithm cannot find solutions that requires the mechanism to reach and cross a singular position in the search space (a straight-line-shaped mechanism for example).

5.2 Learning-Based Algorithms

To address this problem, a second kind of algorithms has been tested by adding learning capabilities to agents. Each agent learns the modification it has to perform by reasoning on the intensity of the NCS it perceives (based on reinforcement learning principles). A new problem occurs with the Bar when it changes its ends, because MECANO has to re-assembly the mechanism, which disturbs the agents' learning (agents cannot rightly judge the impact of their actions). To address this second problem, a solution is to remove physical resolution actions to the Bars, i.e. they can only send messages and propagate their NCS to other agents (Hinges). Nevertheless, this solution is equivalent to algorithms based on cost functions, and therefore implies the same problems of crossing local maxima or singular points.

5.3 Further Works

Previous algorithms could be used by designers to optimize dimensions of a given roughly well-designed mechanism. But, our system aims at autonomously designing the best possible mechanism by self-assembly of elementary components (i.e. self-organization of MechanicalAgents). Therefore, the system must be able to find the best dimensioning for a given topology, independently of the initial state, which is not the case of previously proposed algorithms. The idea is now to explore other local criteria to lead dimensional adjustments, such as length balance of bars connected to the same joint, minimization of dimensions and motion lock avoidance.

6 Discussion

AMAS theory proposes an interesting approach to develop adaptive system by designing cooperative agents. The main outcome of this kind of theory is to support the design of artificial systems with a theoretical background whereas to "mimic" natural systems. In a near future, the key of artificial system design will

be to understand mechanisms which implement complex systems and relations between micro and macro behaviors.

Nevertheless, the notion of cooperation raises some methodological problems. Firstly, AMAS theory ensures the multi-agent system will produce an adequate function if all the agents composing it are cooperative. It means that designers must exhaustively list all the possible local non cooperative situation. Currently, ADELFE only proposes an analysis by NCS type, state by state, for each agent. But during this analysis there is no mean to ensure the exhaustiveness of NCS enumeration - only hints and guidance. Moreover, cooperation notion remains vague and strongly depends on the granularity of agents. Secondly, if we suppose NCS as being exhaustively found, what are the actions to perform to solve them? Once again this problem is strongly linked to the notion of cooperation and many algorithms might be used. For example, several algorithms have been tested for actions related to incompetence NCS: reactive algorithms, learning-based algorithms, cooperative memory-based algorithms. In this last solution, each agent stores errors (size and direction) it perceives into a cooperative memory module (which is recursively defined as an AMAS). Then, each error is represented by a cooperative agent which acts according to NCS rules. For example, if two errors are similar, the smallest one disappears, since it is a concurrent situation. At the agent's decision phase, this module computes the most cooperative action to decrease the size of errors –the main idea is to decrease the worse error avoiding creating a new worse one.

To tackle these problems, two paths could be explored: theoretical or/and experimental. By developing theories on adaptive multi-agent systems, we could find new properties on cooperation. Yet, the AMAS theory employs cooperation notion as a generic behavior of an entity which has to be instantiated for a given application. Therefore, discovery of new NCS types cannot result from demonstrations, but only from observation and generalization. Considering a more experimental angle of attack seems more relevant. Future works should focus on more interactive methodologies, as the *Living Design* concept does [16]. This approach proposes to shift agent design to more living phases (test, deployment and maintenance) compared to classic object-oriented design processes. Designing agents becomes a more interactive activity during which designers equip running agents with their modules, in real time. It requires defining minimal behavior and norms on agent concepts. FIPA does a lot of work in this direction. In addition, ADELFE project will now focus on such problematics thanks to the MDA paradigm, by defining more detailed agent models.

7 Conclusion

In this article, the ADELFE methodology has been applied to a mechanical synthesis problem. The two main phases, analysis and design, led to a self-organizing multi-agent model where each mechanical component is "agentified". Self-organization of the collective is a way to reach an adequate structure using agents' cooperative attitude.

Nevertheless, the notion of cooperative attitude raises some problems such as exhaustiveness of the cooperation rules for each agent and the choice of the most cooperative action at a given time to reach a more cooperative state. This implies to study different heuristics and algorithms to tackle as close as possible the notion of cooperation as defined in the AMAS theory. Thus, two main future work directions are opened. First, by formalizing agent and multi-agent models or refining their meta-models, some new properties may be discovered, such as the relevance of a local cooperation measure. Second, experience is the main way to succeed in finding cooperation rules. The concept of Living Design may be an real solution to interactively define agents' models and cooperative attitude.

Finally, a prototype called Mechanical Synthesis Solver (MSS) has been developed and shows promising results on dimensional adjustments [17]. The next step of these experiments is to use self-organization to modify the topology of mechanisms by adding, removing and spatially re-organizing agents.

References

1. Brazier, F., Jonker, C., Treur, J.: Compositional Design and Reuse of a Generic Agent Model. International Journal of Cooperative Information Systems **9** (2000)
2. Wooldridge, M., Jennings, N., Kinny, D.: A Methodology for Agent-Oriented Analysis and Design. In Oren Etzioni and Jörg P. Müller and Jeffrey M. Bradshaw, ed.: Proceedings of the 3rd International Conference on Autonomous Agents (Agents 99), ACM Press (1999)
3. Odell, J., Parunak, H., Bauer, B.: Extending UML for Agents. In: Proceedings of the Agent Oriented Information Systems (AOIS) Workshop at the 17th National Conference on Artificial Intelligence (AAAI). (2000)
4. Picard, G., Gleizes, M.P.: The ADELFE Methodology – Designing Adaptive Cooperative Multi-Agent Systems (Chapter 8). In Bergenti, F. and Gleizes, M-P. and Zambonelli, F., ed.: Methodologies and Software Engineering for Agent Systems, Kluwer Publishing (2004)
5. Gomez Sanz, J., Fuentes, R.: Agent Oriented System Engineering with INGENIAS. In: Fourth Iberoamerican Workshop on Multi-Agent Systems, Iberagents'02. (2002)
6. DeLoach, S., Wood, M.: Developing Multiagent Systems with agentTool. In Castelfranchi, C. and Lesperance, Y., ed.: Intelligent Agents VII. AgentTheories Architectures and Languages, 7th International Workshop (ATAL 2000), Springer-Verlag (LNCS 1986) (2001)
7. Bernon, C., Gleizes, M.P., Peyruqueou, S., Picard, G.: ADELFE: a Methodology for Adaptive Multi-Agent Systems Engineering. In Petta, P. and Tolksdorf, R. and Zambonelli, F., ed.: Third International Workshop on Engineering Societies in the Agents World (ESAW-2002), Springer-Verlag (LNAI 2577) (2002)
8. Campbell, M., Cagan, J., Kotovsky, K.: Agent-based Synthesis of electro-mechanical design configurations. In: Proceedings of DETC98 1998 ASME Design Engineering Technical Conferences. (1998)
9. Tsai, L.W.: Mechanism Design: Enumeration of kinematic structures according to function. CRC Press (2001)
10. Bernon, C., Camps, V., Gleizes, M.P., Picard, G.: Tools for Self-Organizing Applications Engineering. In: First International Workshop on Engineering Self-Organizing Applications (ESOA) at AAMAS'03, Melbourne, Australia (2003)

11. Capera, D., Georgé, J., Gleizes, M.P., Glize, P.: The AMAS theory for complex problem solving based on self-organizing cooperative agents. In: 1st International workshop on Theory and Practice of Open Computational Systems (TAPOCS) at IEEE 12th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2003), IEEE Computer Society (2003) 383–388
12. Castro, J., Kolp, M., Mylopoulos, J.: A Requirements-driven Development Methodology. In Dittrich, K., Geppert, A., Norrie, M., eds.: Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01), Springer-Verlag (LNCS 2068) (2001) 108–123
13. Kinny, D., Georgeff, M., Rao, A.: A methodology and modelling technique for systems of BDI agents. In de Velde, W.V., Perram, J.W., eds.: Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a MultiAgent World, Springer-Verlag (LNAI 1038) (1996) 51–71
14. Caire, G., Coulier, W., Garijo, F., Gomez, J., Pavon, J., Leal, F., Chainho, P., Kearney, P., Stark, J., Evans, R., Massonet, P.: Agent Oriented Analysis Using Message/UML. In Wooldridge, M., Wei, G., Ciancarini, P., eds.: Agent-Oriented Software Engineering II, Second International Workshop, AOSE 2001, Springer-Verlag (LNCS 2222) (2001) 119–135
15. Bernon, C., Camps, V., Gleizes, M.P., Picard, G.: Designing Agents' Behaviors within the Framework of ADELFE Methodology. In: Fourth International Workshop on Engineering Societies in the Agents World (ESAW-2003), Imperial College London, UK (2003)
16. Georgé, J.P., Picard, G., Gleizes, M.P., Glize, P.: Living Design for Open Computational Systems. In: International Workshop Theory And Practice of Open Computational Systems (TAPOCS) at 12th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03), Linz, Austria, IEEE Computer Society (2003)
17. Capera, D., Gleizes, M.P., Glize, P.: Mechanism Type Synthesis based on Self-Assembling Agents. Journal on Applied Artificial Intelligence **18** (To appear in 2004)