INTERNATIONAL DESIGN CONFERENCE - DESIGN 2006
Dubrovnik - Croatia, May 15 - 18, 2006.

DESIGN 2006

AGENT-AIDED PRELIMINARY AIRCRAFT DESIGN

J. B. Welcomme, M.P. Gleizes and R. Redon

Keywords: computer aided design, cooperative agents, complex systems, multi disciplinary components

1. Introduction

Preliminary aircraft design involves a lot of disciplines like weight, range, aerodynamic and operating cost estimations [Kroo 2004]. A simulation approach is done during this design phase in order to discover the main characteristics of a desired aircraft. So the intuitive decomposition between models according to the involved disciplines is also very useful to reduce the complexity of the global problem into more manageable subtasks.
Mostly of the disciplinary models represent a physic as a mathematical function with a set of inputs/outputs. Thus, building a multidisciplinary simulation consists first in selecting and then linking the modules (encapsulating a given model), which match better the target criteria and the context of the product to design. After that, two different approaches in design, and two different directions of simulation computation can be done:

- When geometrical and main design parameters are known, the simulation computation is used to calculate the product performances: this is the « **analysis direction** ». In this case the simulation can be computed directly.
- When the technical requirements (product performances) are known, the simulation is used to calculate the design parameters; this is the « **design direction** ». Unfortunately, a mathematical inverse problem must be solved iteratively with a lot of simulations because we only know the computational models of "analysis direction".

Generally, the main objective is to build a simulation, in order to solve the inverse problem according to a set of constraints shared between some design parameters and performances (like operating cost, mission…). So an optimization process iterates the built simulation to find feasible design parameters respecting design constraints (like wing, fuselage and tail shapes) and performances.
In this paper, we study the advantages of a cooperative multi-agent framework to support the design of preliminary aircraft. The section 2 presents the main approaches used in Multi-Disciplinary Optimization (MDO). The section 3 explains our agentification approach for the MDO problem and the basic cooperative behaviour of agents allowing convergence to optimal solutions. We give some experimental results in the section 4 and analyse the consequence of this agentification process for the design.

2. Related works in Multi Disciplinary Optimization

During the last three decades, various types of computational or computer-aided design systems have been developed in MDO domain. A lot of issues were addressed like interoperability, problem decomposition, design robustness analysis, uncertainty propagations [Kroo 2004]. Several strategies were proposed for the global optimisation and the subsystems linkage, (figure 1) exploiting the synergy of interactions through Fixed Point Iteration (FPI) algorithms [Allison 2004]. Many relationships between mathematic analyser and optimiser were studied, in which an analyser defines

an execution order for computing the different models, whereas an optimiser compares their results and adapts the design parameters to converge on target criteria [Allison 2004]:

- Multi Disciplinary Feasible (MDF) is composed of an analyser and an optimiser. In this strategy, analysis and optimisation are done independently. In consequence during the analysis step, the system is completely dependent on the FPI limits leading to useless executions.
- Individual Disciplinary Feasible (IDF) decomposes the complexity of the analyser and provides more flexibility. Because each discipline has its individual objectives, solutions are often better by putting shared parameters in dynamic concurrence than to define a static sequence of resolution through an analyser of systems. So in this kind of strategy, the optimiser has a central role in providing inputs to subsystems and controlling their executions. Nevertheless each subsystem goes on doing mathematic analysis to compute its local models.
- Finally, in All at Once (AAO) approach, optimiser has all solving abilities. Each disciplinary iterates/evaluates its local models, then optimiser controls shared parameters and subsystem convergence.

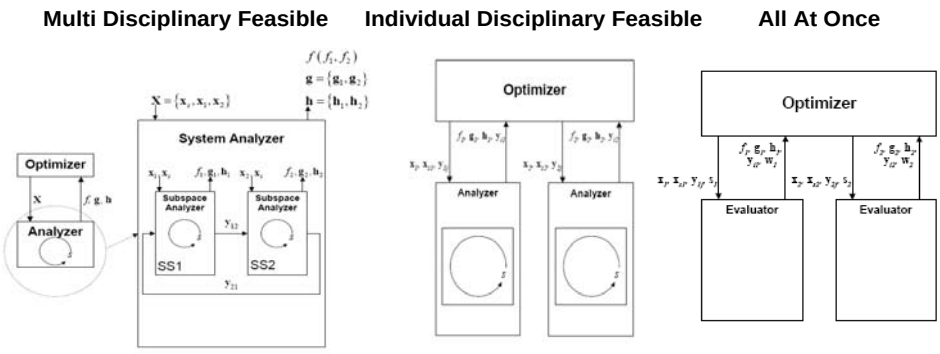

Figure 1. Main MDO strategies [Allison 2004]

These three strategies are centralised on the optimiser, and not always adapted to the reality. More complete approaches such as collaborative optimisation (CO), Concurrent Sub-Space Optimization (CSSO) offer multi-level architectures [Kroo 2004], where each disciplinary has its individual optimisation strategy. Analytical Target Cascading (ATC) [Allison 2004] is another alternative, in which each component is itself an optimiser. As a consequence, the system is hierarchical and each component tries to minimise its individual objectives and those of its neighbours.

MDO strategies have offered good analysis and solutions on the decomposition of the problem and on the interactions between shared parameters. Nevertheless, the complexity of the problem is not limited to the optimisation of the shared parameters through interactions, because the process organisation acts also on results. By organisation, we mean that a better selection of models and a better-decomposed organisation can bring better solutions. As the dynamic context of preliminary aircraft design makes the optimisation formulation evolving during the simulation, we think that the module/model fitness changes, and that the initial organisation needs to be adapted during the simulation. More precisely, [Fujita et Yoshida, 2004] introduces the following three classes of optimisation; the first class considers the parameters; the second class the module combination; and the last class both parameters and module combination. In the next parts, we show how cooperative multi-agent systems (MAS) offer very interesting perspectives to deal with this last class of problem.

3. Principles of the MASCODE[4] system

In this general context, we want to give assistance for solving complex product design thanks to an adaptive and self-organized structure of software components. Generally speaking, self-organization is the apparition of a functional structure spontaneously maintained in a dynamic equilibrium by all the participating components [Heylighen et Gershenson, 2003]. As described in [Di Marzo Serugendo et al., 2006], self-organising in Multi Agent Systems (MAS) offer opportunities to simulate real complex

[4] MASCODE : Multi-disciplinary Aircraft Simulation for COnceptual DEsign.

systems, because of agents autonomous behaviours; to adapt constantly their state relative to each other; to learn from experience; and to create dynamically group and organization. Because all these features correspond to a part of the system design complexity, they help us to understand the general structure and behaviour of complex systems.

3.1 The basic cooperative agent behaviour

Researchers have experienced several mechanisms leading to self-organisation [Di Marzo Serugendo et al., 2006], but we use in the MASCODE system a specific one based on the cooperation mechanisms provided in the AMAS theory [Georgé et al., 2004]. The cooperative strategy is a meta-heuristic that identifies a set of local Non-Cooperative Situations (NCSs) for the agents.
NCSs are used to help the agents to find collective solutions and avoid them to choose always individual optimal choices, which imply non-optimal whole. Generally, behaviour based on NCSs is relevant when a subsystem over constrains other subsystems. NCSs are defined locally, because this enables each agent to individually rearrange its interactions with others depending on its objectives and on its representation of environment. For example when an agent considers that one of its neighbour is over constrained, it can decide to change its individual objectives to help it. Nevertheless, NCSs do not lead to altruistic agent, because this kind of reasoning is also not cooperative.
In the animal behaviour society, a system is said cooperative when individual cooperative costs compensate the benefices of the society. According to these principles, we consider a first simple experimentation based on aircraft weight estimation.

3.2 The MASCODE experiment

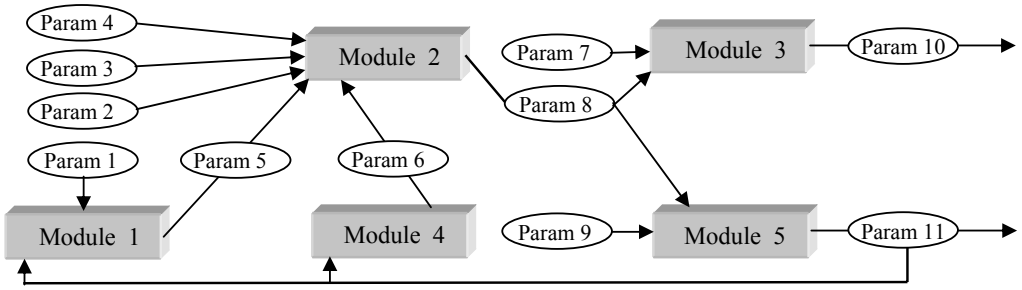

Figure 2. "Take off weight" modules simulation

The experimentation showed here (figure 2) is a simple test case representative of a real problematic[5] composed of 5 methods/models. This simple test case makes possible to test the cooperative reasoning of the agents, their ability to negotiate collectively on a set of parameter values, and to converge on a collective solution. In this first experimentation, the organisation is static but the final objective is a self-organizing system, by adding new NCSs in the future as described at the end of the paper.
The function of the test case (figure 2) is to estimate the "take off weight" (parameter 11) of an aircraft, which is obtained through an addition of the main aircraft component weights. However, the final results "parameter number 11" is also an input of the 1st and 4th modules whose results are themselves used in the "take of weight" estimation across the 2nd module. In consequence, the system is composed of two loops that introduce non-linearity in the system, because each component affects other components. Thus the cause-and-effect relation is *circular*: any change in the first component is fed back to itself via its effects on the other components [Heylighen, 2001]. So the aim of this experiment is to verify if the cooperative reasoning and the identified NCSs can control these phenomena and converge on a solution before further improvements. In the next subpart, we present the main choices of the implemented agent by using ADELFE design methodology [Picard et Gleizes, 2004].

[5] For industrial reasons, the parameters and functions name are hidden

3.3 The agent knowledge

In MASCODE, one agent controls one model and possesses knowledge and behaviours to communicate, to learn and to adapt its reasoning. Agent knowledge is static or dynamic, and divided into knowledge on its model and knowledge on its relations (connection with neighbours).

Knowledge on model defines the validity domain of each input.

- *An execution validity interval* provides an input value interval, in which the model is computable (physical limits).
- *An objective validity interval* describes a preferred interval. All the values inside this range fit the user constraints.

Then, each model agent is able to compute a critical value on its inputs by using this two ranges and a defined mathematic function. When the input value is inside the objective validity interval, the associated critical value is negative and inversely proportionally positive outside. Finally, the agent non-satisfaction degree is equal to the most critical value of its inputs.

Knowledge on relations is simple at this time. It informs which agent(s) provide an input or use an ouput. In addition to this static knowledge, agent learns experiences during the simulation execution, and builds memories. Memories are a key element in the AMAS theory, because agent adapts its behaviour, and takes decisions in function of its past experiences. In our case, the memory is decomposed as following:

- For each output, an agent keeps in memory its last computed values, and the last feedback of agents, that use this parameter.
- For each input, an agent knows its last input values; the last tried input modifications and the last received values provided by predecessors.

3.4 The agent behaviours

A model agent possesses the four following behaviours:

1. ***Management of the physical model***: each agent is able to compute its model. In direct mode for execution, it computes its function and sends the new output values to the next agent. In indirect mode it computes its Jacobian matrix to find the local dependencies between its inputs and its outputs. The Jacobian is akin to a derivative of a multivariate function, because it is a linear approximation of a differentiable function near a given point.
2. ***Communication***: A forward message is used to send the new computed output values to the successor agent, while a backward message is used to send desired input value (feedback) to the provider agents. In addition to the desired or computed value, each message contains its associated critical value and the non-satisfaction degree of the sender agent.
3. ***Adaptative computation:*** To obtain a more efficient process, each agent adapts a minimal execution/modification step in function of its number of received message. In consequence when receiving a backward or forward message, the agent compares the received value with its current value. If the two values are smaller than the minimal execution/modification step, the agent ignores the message; else it processes the message.
4. ***Adaptive input*** variations parameters. While moving to a solution, if the modification direction of an input is successively the same, the agent considers it as a positive feedback and increases the input variation step. Conversely if the modification direction is changing, agent considers it as a negative feedback, and decreases the variation step.

4. MASCODE results and analysis

4.1 The Non Cooperative Situations of agents

The Non Cooperative Situations (NCSs) are defined with a description and a set of actions. The description can be viewed as a rule containing all the conditions necessary to recognize the NCSs. The sets of actions describe how the agents can improve the cooperation of its environment. In our context, we can classify the NCSs in two main classes: NCSs that adapt the parameters, and NCSs that change the organisation. Table 1 and 2 give a general description of the main NCSs for adapting the parameters, because in the current MASCODE version we do not deal with changing organisation yet. Concerning changing organisation, the work is in progress, and the objective is to let the agents taking

initiatives to leave the system, to create/search other cooperations during the simulation, when the context is evolving.

Table 1. Main NCSs on forward message

Name	Description	Action
NCSForwardValidity1	One or several inputs don't respect their validity domain	Send a backward message to providers
NCSForwardValidity2	One or several inputs don't respect their expected value	Send a backward message to providers Send an forward message to users
NCSForwardWithModif	Forward message is not compatible with the modification memory	Resend the message Don't execute the module Update the memory
NCSUselessForwardMessage	The forward message matches the current input value	Cancel the message

Table 2. Main NCSs on backward message

Name	Description	Action
NCSBackwardAndModuleCriticalValue	The Module is more critical than the message	Adapt only the intputs, whose critical value is less important than the request.
NCSBackwardMessageAndOthersMessages	Another message is more critical.	Cancel the message
NCSModificationAndOthersModifications	A critical value implied by another modification is more important.	Cancel the message
NCSUselessBackwardMessage	The backward message matches the last message or matches the current output value	Cancel the message

To illustrate the non-cooperative reasoning, we detail a small scenario example with one NCS, the "*BackwardAndModuleCriticalValue*" NCS (see figure 3). It appears only when an agent receives a backward message on one of its output parameters. On the figure 3, the agent has a critical value equal to "50", and in this situation, a backward message is received on its "parameter D" with a critical value equal to "30". This message asks simply the agent for modifying the provided "parameter D" value. However to modify the "parameter D", the only possibility of the agent is to adapt one or several of its input.

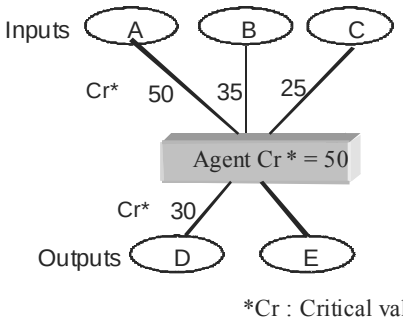

Figure 3. BackwardAndModuleCr

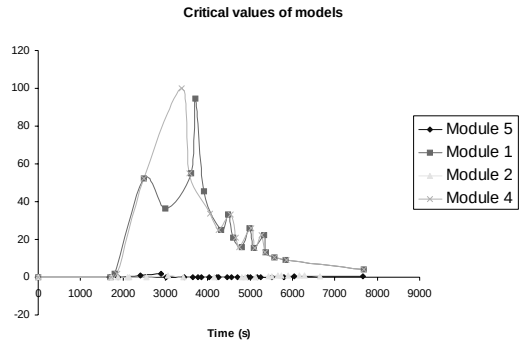

Figure 4. Critical values of models

But as the agent is already more critical than the request, the cooperative reasoning permits to adapt only inputs of the model that are less critical than the request. By favouring most critical parameters, we progressively decrease the critical values inside the network, and we achieve the main objective of cooperative reasoning. Thus in the example, the only input that can be adapted, is the "parameter C", because less critical than the request.

Now the agent has identified the input to adapt, it needs to know the modification step to apply. To do this, it computes a relation taking into account the requested variation on "parameter D" and the partial derivative value between "parameter C and D". Finally, it proceeds with the new value and when necessary informs its neighbours of this new situation (backward and forward message).

4.2 Results

The figure 4 presents the non-satisfaction degree of modules 1, 2, 4 and 5 during one simulation. But each agent also stores locally its states and its chosen values, which are indicated on the figure 5 for the same modules 1, 2, 4 and 5.

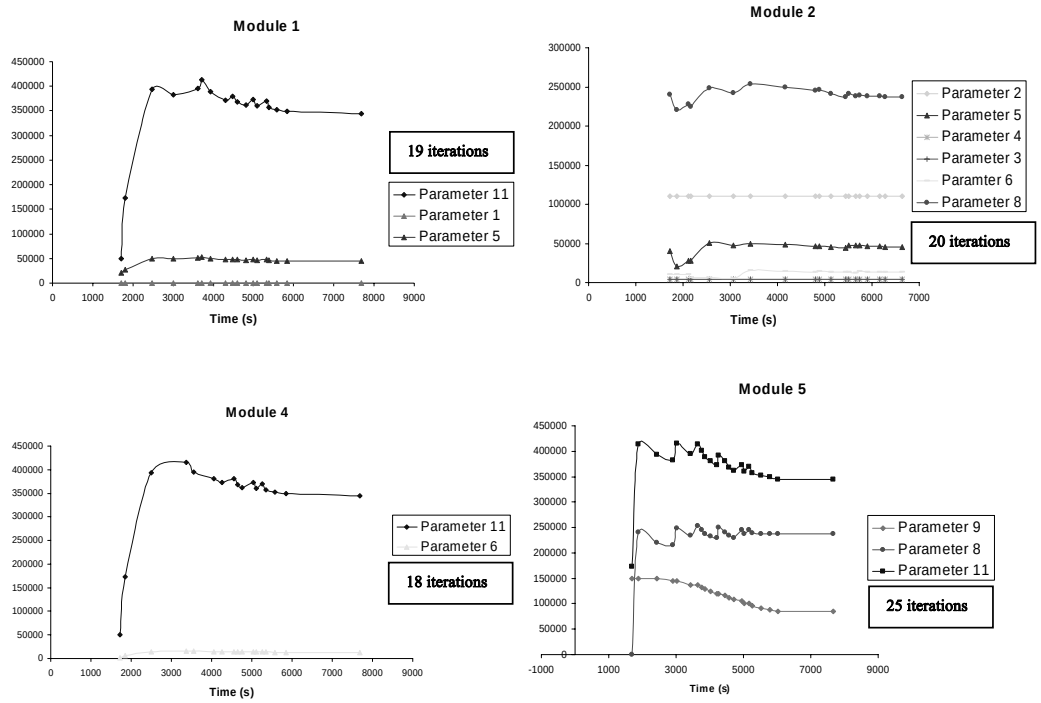

Figure 5. Example of stored values

The cooperation finds equilibrium after several iterations. At the beginning of the negotiation none of the models has a real non-satisfaction value, because input and output values are not synchronized. When the value are transmitted, the most non-satisfaction models are the models number 1 and 4, because their input value ("11th parameter") is not appropriate with their objective validity domains. So the two agents encapsulating these models inform their neighbours of the situation, and step-by-step the constraints are propagated across the network. In this way, the parameter values are adapted, and at the end of the process the non-satisfaction values of agents are comparable and constraints are shared across the network. These results show:

1. In the built MAS, each agent has an individual iteration number, because computing its model is a local behaviour. For example, we can see on the figure 5 that iteration number of "the 5th model" is the most important with 25 iterations. This result proves that agents are able to know if they need to compute their model and so that the rearrangements are propagated only on the concerned part of the network.
2. At the end of the simulation, the process stops because the critical values are similar and because the modification proposed are included in the minimal execution/modification step. So, the agents decide to ignore their mutual messages. We observe clearly that the cooperative behaviour enable to decrease the NCS degree of the two most constrained modules over the time.

4.3 MAS aided engineering design

For the computer-aided engineering design point of view, these first results allow us to conclude that agentification offers new perspectives in the design of preliminary aircraft design. Cooperative multi-agent systems provide a new conceptual architecture (distributed and asynchronous):

1. ***For designing each component independently of each other***. It enables to add engineering knowledge on modules/models involved in the simulation without global system considerations. It also makes the system opened; new modules can be added easily, because of component autonomy. Nevertheless, a trial and error process is very required to ensure that the local NCSs are well defined and managed. So this is the most difficult part in the design of cooperative systems with AMAS.
2. ***For integrating this component in a common system without using a global control.*** Thus, the system is adaptive and dynamic: the user can change constraints and the system will search a new equilibrium, moreover an interesting feature of the system is the user could follow the most constrained model and associated physics during the system execution. So we can say the cooperative reasoning offer new views on the system.
3. ***For reducing redundant execution***

All these features favour autonomy, robustness and adaptivity, and make the agent simulation based on cooperative disciplinary entities promising in engineering design activity. A generalisation of this work will offer other aided engineering design perspectives to compare and enrich with [Shen et al., 2001].

5. Conclusion and future work

In this paper, we have shown an approach of MDO supported by cooperative agents. Each agent is designed separately and during a simulation it has only a local view of the global system based on its relationships with its neighbours. Our first experimentations enable to conclude that we are able to find a solution with a distributed and cooperative decision process. As we described, the cooperative reasoning is based on a meta-heuristic that tries to find equilibrium between local optimal solutions. In consequence:

- Local agent behaviours are very important for the efficiency and the performance of the resolution, and improvements are obtained through trial and error processes on NCS definition.
- When the good NCS are found, the system seems to find an optimal global solution.

However additional features are necessary to address our final objectives; providing a self-organizing simulation on a real problem composed of more than one hundred models. To do this, we are now improving the MASCODE platform in three directions:

1. ***Cooperative reasoning on resolution:*** The trial and error processes should be continued to improve the system performances. We need to compare several communication protocols with more or less synchronisation. The size of the memories could also be increased, because for simplifying this first experimentations memories contained only the last computed values. With these short-term improvements, we expect to compare our results with others MDO approaches. However the numeric resolution performance is not a final objective, because we want address other issues by taking into account more engineering knowledge as described bellow.
2. ***Adaptive organisation***: The implemented agents represent physical models. For every one of them, we defined new knowledge on validity domains. Nevertheless if we want deal with the full self-organized problematic, we need other cooperation's criteria on precision, on concurrent models, on execution time… to provide more autonomy. Then, the self-organized process will be obtained simply by using this additional knowledge to define other NCSs for changing the organisation.
3. ***Abstract levels:*** However, we think the knowledge on models will not be sufficient to efficiently self-organise complex systems, because it could not support more abstract knowledge like:
    - Physics models dependencies, that constrains the solution;
    - Design organisation, that influences the simulation;

- Quality objectives, that provides information on the critical sections of the design.

That is why, we are thinking to add one abstract level representing domains like weight estimation, mission performances, and operating cost. It will provide an easiest way for the user to constrain/understand/monitor the problem per domain, and it will help to limit the communication quantity.

Finally, our approach is based on a cooperative multi-agent framework, that favour adding engineering knowledge on modules, and that provides autonomous and adaptive behaviours. It offers promising perspectives by optimising simultaneously both module organisation and module parameters, which are both important and linked for efficiently exploring the best solution in preliminary aircraft design.

References

Allison J. T,."Complex System Optimization: A Review of Analytical Target Cascading, Collaborative Optimization, and Other Formulations.", Master's thesis, Department of Mechanical Engineering, University of Michigan, 2004.

Di Marzo Serugendo G., Gleizes M.P., Karageorgos A.,"Self-Organisation and Emergence in MAS: An Overview", to be published in "Informatica an International Journal, Institute of Mathematics and Informatics, Lithuanian Academy of Sciences, 2006

Fujita K., and Yoshida H., "Product Variety Optimization Simultaneously Designing Module Combination and Module Attributes", Concurrent Engineering, Research and Applications, Vol. 12, No. 2, June 2004, pp. 105-118.

Georgé J.P., Edmonds B., Glize P., "Making Self-Organising Adaptive Multiagent Systems Work", in Methodologies and Software Engineering for Agent Systems, The Agent-Oriented Software Engineering handbook, Kluwer Publishing, 2004, pp. 321-340.

Heylighen F., "The Science of Self-organization and Adaptivity", in The Encyclopedia of Life Support Systems, 2001

Heylighen F., Gershenson C., "The Meaning of Self-organization in Computing", IEEE Intelligent Systems, section Trends & Controversies - Self-organization and Information Systems, June 2003.

Kroo I., "Distributed Multidisciplinary Design and Collaborative Optimization", VKI Lecture Series on Optimization Methods & Tools for Multicriteria/Multidisciplinary Design, November 2004.

Picard G., Gleizes M.P., "The ADELFE Methodology - Designing Adaptive Cooperative Multi-Agent Systems, Methodologies and Software Engineering for Agent Systems, The Agent-Oriented Software Engineering handbook. Kluwer Publishing, 2004, pp. 157-176.

Shen, W., Norrie D. H., Barthès J.P.,"Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing", Taylor and Francis, 2001.

Jean-Baptiste Welcomme, Mr
IRIT, Universié Paul Sabatier
31062 Toulouse, 118, Route de Narbonne, France
EADS CCR, 31700 Blagnac, Centreda 1, Av. Didier Daurat, France
Tel.: 33 (0) 5 61 18 48 54
Email: welcomme@gmail.com