Managing Contradictions in Multi-Agent Systems

Rubén Fuentes-Fernández, Jorge J. Gómez-Sanz, and Juan Pavón

Summary
The specification of a Multi-Agent System (MAS) involves the identification of a large number of entities and their relationships. This is a non-trivial task that requires managing different views of the system. Many problems concerning this issue originate in the presence of contradictory goals and tasks, inconsistencies, and unexpected behaviours. Such troublesome configurations should be detected and prevented during the development process in order to study alternative ways to cope with them. In this paper, we present methods and tools that support the management of contradictions during the analysis and design of MAS. Contradiction management in MAS has to consider both individual (i.e. agent) and social (i.e. organization) aspects, and their dynamics. Such issues have already been considered in social sciences, and more concretely in the Activity Theory, a social framework for the study of interactions in activity systems. Our approach applies knowledge from Activity Theory in MAS, especially its base of contradiction patterns. That requires a formalization of this social theory in order to be applicable in a software engineering context and its adaptation to agent-oriented methodologies. Then, it will be possible to check the occurrence of contradiction patterns in a MAS specification and provide solutions to those situations. This technique has been validated by implementing an assistant for the INGENIAS Development Kit and has been tested with several case studies. This paper shows part of one of these experiments for a web application.

Key words:
Agent-Oriented Software Engineering, Multi-Agent System Analysis and Design, Activity Theory, Conflict solution, Contradiction Patterns.

1. Introduction

A contradiction in a software process stands for an inconsistency, obscure point, or misunderstanding in the models that specify the system to be. Contradictions are inherent to the development; they appear no matter how disciplined a developer is, what the underlying software paradigm is, or which methodology the project uses. Moreover, given the dynamic nature and interrelationships of agents and their human environment, contradictions in agent-systems arise in more (and different) forms than in classical software engineering, requiring newer treatments.

There are at least three different types of contradictions in Multi-Agent Systems (MAS) [12]. First, there are contradictions inherent to the system under study, for instance due to competing goals of the agents in the MAS. Second, there are tensions that have their origin in the dynamic nature of the MAS and its evolution. For example, this is the case of trade-offs between learning capabilities of agents and their response time in a changing environment. Third, there are analysis mistakes, caused by misunderstandings among customers and developers, or by the number and variety of system entities and viewpoints that developers must maintain consistent and updated.

Most agent-oriented methodologies propose some kind of guidelines for the management of contradictions (e.g. i* [15]), syntactic rules to check the validity of MAS (e.g. INGENIAS [8] and ODAC [5]), or they try to overcome these problems by relying on formal models (e.g. Tropos [8] and the work of Van Eijk et al. [13]). Although these methods are focused on the intentional and organizational issues of MAS, these mainly appear at their foundation (e.g. principles, modelling primitives, or theorems). Thus, these methods need to adopt a given perspective about how the system is (or should be), and they are not applicable otherwise. This implies limitations in the management of the partial and contradictory information that appears in real-human environments of MAS. We think that solving problems about contradictions is not just a question of giving a notation and processes to gather information or to prove properties over specifications in that language. It is also a problem of what the relevant properties are, since the intentional and social issues of MAS and their environments are based on complex psychological and sociological patterns, which do not belong to the expertise of software engineers.

As a solution to the above problems, this paper proposes an alternative that facilitates the detection and solution of contradictions in MAS specifications. This approach starts from the assumption that the management of the complexity in MAS requires the consideration of agents as intentional entities that participate in higher order structures, i.e. organizations [3]. This assumption allows considering how contradictions have been studied by social sciences in human organizations. More concretely, we consider Activity Theory (AT) [9], which offers a conceptual and analytical framework to understand the motivation, the relationship between individual and society, multiple perspectives, and networks of interacting activity systems. AT studies contradictions (i.e. conflicts) between individuals and communities in productive environments. According to AT, contradictions can be the force guiding development. Our approach follows this principle. It is based on the identification of contradiction patterns from AT to detect these conflictive situations in the models of MAS; the
proposal of alternative solutions to improve and evolve these models relies on the answers to those contradictions, extracted from AT research too.

AT has been already introduced in some fields of computer science [1], such as computer supported cooperative work, requirements engineering, human-computer interaction, or web design. Recently, agent research has started to consider AT to deal with some of its open issues (e.g. coordination and engineering of societies through the use of mediation artefacts [11]). In these works, social sciences inspire researchers to identify relevant functionality of MAS, which results in the development of new frameworks or features. Of course, our research also builds a framework over sociological concepts of AT but, besides, we consider the concrete knowledge of AT studies as useful experiences that can be adapted and applied to build MAS. These adapted experiences constitute libraries for development, like the one presented here for contradiction patterns.

The rest of the paper is organized as follows. Section 2 reviews the main concepts of AT and the language UML-AT which is used to represent them in software practices. UML-AT is the basis of the format to represent contradiction patterns explained in Section 3. Section 4 gives an overview of the current state of the repository of contradictions and a detailed description of an example. Next, Section 5 considers the application of these contradictions as an additional facility during the analysis and design of MAS using a given methodology. A case study of this application for a recommender system is reported in Section 6. Finally, conclusions in Section 7 discuss the benefits of applying AT for MAS verification and the value of analyzing contradictions taking into account the results of our experimentation.

2. UML-AT: a notation for Activity Theory

Activity Theory (AT) [14] is a framework for the study of different forms of human practices and their evolution in a social and historical context. Its study focuses on the interactions and conflicts between individuals and their environment, which include their societies. Individual behaviour cannot be understood out of its context but, at the same time, people actively change their context. This complex interaction has been called activity [2] and it is the fundamental unit of analysis in AT. Essentially, the activity reflects a process, with individual and social levels interleaved. The context of the activity is called the activity system and comprehends the following elements.

The individual level of the activity is first characterized by the subject, which is the active element that carries out the activity. It can represent an individual or a group of individuals. The subject has some needs represented by the objective. The objective is satisfied by the outcome, which is produced transforming an object. In order to carry out that transformation, the subject uses tools: tools mediate the subject’s processes over objects [14]. A tool can be anything used in the process, including both material tools (e.g. a computer) and tools for thinking (e.g. a plan).

At the social level, the community represents those subjects who share the same object [2]. An activity may concern many subjects and each subject may play one or more roles and have multiple motives. Rules [1] mediate between subject and community. They specify how subjects fit into communities and cover both explicit and implicit norms, conventions, and social relationships within a community. The division of labour [1] mediates between object and community. It refers to the explicit and implicit organisation of the community as related to the transformation process of the object into the outcome.

All of these concepts are interconnected with mediation relationships. These relations reflect conceptual links according to the meaning of the concepts they associate. Examples of them are the already mentioned mediations of tools between subjects and objects or rules governing subjects in their communities.

These elements are the core notation commonly agreed in AT studies. AT uses these concepts to describe the social situations under scrutiny and their changes, contradictions, and solutions. Its descriptions are of a discursive nature and heavily rely on a deep knowledge of the exact meaning of the involved concepts and the underlying conceptual background. So, its use by non-experts in the field is quite difficult.

The intention of employing AT concepts in Agent-Oriented Software Engineering, demands that they are expressed in a language understandable and applicable by developers. With this purpose, AT concepts have been described using the UML profile extension mechanism [10]. The resulting language is called UML-AT. Besides previous elements, UML-AT includes additional ones to reason about the contradictions and solutions that AT identifies. These elements comprehend, for instance, contribution relations, which are inspired by i* [15]. Contributions show how artefacts influence each other. Examples of this kind of relationships are the satisfaction of objectives, construction of objects, or damage of tools. They are represented with several types of relationships: contribute (positively, negatively, or undefined), essential, and impede. There are also other concepts in UML-AT, like that of artefact, which is used when there is a need to represent any kind of concept as a generic abstraction. The full specification of UML-AT can be found at http://grasia.fdi.ucm.es/at/uml-at. Some examples of its use appear in the figures of this paper.
3. Description of AT contradictions for MAS

Leontiev [9] describes in one of the fundamental AT works what kind of conflictive situations appear in human activities. A contradiction is a deviation of the common societal norms about an activity, or a tension between its components, which carries the seed of new forms of that activity. Frequently, AT studies also consider how societies solve their contradictions.

![Table](image)

<table>
<thead>
<tr>
<th>Name of the contradiction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Textual description of the contradiction</td>
</tr>
</tbody>
</table>

**Pair of patterns = match + solution**

<table>
<thead>
<tr>
<th>Match pattern</th>
<th>Solution pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text + UML-AT</td>
<td>Text + UML-AT</td>
</tr>
</tbody>
</table>

Examples = Text + UML-AT

… (More pairs of patterns)

Fig. 1. Structure for the description of AT contradictions.

Using UML-AT, we propose a way to represent AT contradictions considering two main concerns: its automated processing and the multiple situations in which a contradiction can appear. Fig. 1 shows this structure.

The textual description of a contradiction tries to explain developers what the contradiction means in the context of MAS. It describes, for instance, the roots of such contradiction from the psychological and sociological point of view and the kind of undesired effects that can come up from it.

A contradiction can have several related pairs of patterns that correspond to different settings in original textual sources where the contradiction rises. Every pair consists of a match (that represents the contradiction) and a solution pattern (a solution that human societies give to the contradiction). Both patterns include a textual description of their meaning and an UML-AT diagram. These explanations help end-users and developers to understand the purpose of the contradiction.

In addition to its explanatory role, UML-AT diagrams are also used in the checking process (see Section 5). The diagram of a match pattern declares what elements must exist in the original specification. Solution patterns are described as rearrangements of the elements appearing in their match pattern with the introduction, sometimes, of new elements (which are denoted with dotted lines in the examples below). These diagrams also indicate if their elements should contain specific values (by declaring them quoted) or anyone (without quotes).

4. Repository of AT contradictions

From the study of AT we have been able to extract several contradictions that can appear in MAS. The result is a repository of AT contradictions, which developers can check in their specifications in order to work out the intentional and social aspects of their systems. Currently, the repository comprises ten contradictions (http://grasia.fdi.ucm.es/at/contradictions).

One example of contradiction is the Conflict Producer-User [7], which emerges in a network of what AT calls neighbouring activities. The neighbouring activities of a given activity are those activities that produce its artefacts (e.g. tools, object, subject, or rules) or use its outcomes. The Conflict Producer-User appears between the activities producing a tool and those that use it. For example, this is the situation when an agent (i.e. a subject) performs the task (i.e. an activity) that generates a request (i.e. a tool) for other agent. These producer and user activities pursue different objectives about the tool, and may not consider the role that this tool should play in other activities. Without this consideration, there cannot be true collaboration in the network of activities and that is the origin of this contradiction.

The network of neighbouring activities covers a Producing Activity and a Using Activity. These activities are connected by the Artefact that one of them produces and the other uses as a tool. The relationship change of role points out that the same Artefact has different uses (and therefore objectives) in the neighbouring activities. The absence of collaboration in the network appears as the independence of the objectives and activities in the diagram that are not interconnected but for the Artefact.

One possible solution proposed by AT research to a Conflict Producer-User is creating context awareness in the neighbouring activities. The UML-AT model in the lower diagram of Fig. 2 shows this solution. The new elements are inside the dashed ellipse. The solution differs from the contradiction in the two new objectives Objective for the User and Objective for the Producer. The Objective for the Producer points out that the Using Activity has to generate some outcome that helps the Producer to satisfy its objectives. In the same way, the Producing Activity follows the Objective for the User, which represents how it works to satisfy the needs of the agent User. For the sake of simplicity, the solution does not include the possibly needed new elements related with these goals.

As a common remark about contradictions, their solutions just try to provide a hint about a possible improvement. Developers need to refine those solutions in order to implement them for their particular problem.
5. Contradictions management

Fig. 3 shows the general scheme of the method for checking a model against a set of contradiction patterns. In order to apply the process, two preliminary tasks must be done. First, experts in AT and the agent paradigm must produce a library of AT contradictions that can be used with different methods and projects. An example of such repository has been discussed in Section 4. Second, experts in AT and specific agent-oriented methodologies must elaborate the mappings for translations between UML-AT and methodologies. Mappings are correspondences between structures in different languages. They allow applying this process over different methodologies. A detailed discussion of the mappings and their creation appears in [4]. The process of contradiction management itself (see Fig. 3) refers to how developers check specifications, using the products of the preliminary tasks.

The first steps of the process are to obtain the specification of a MAS (task 1 in the diagram) by means of some agent modelling tool, and translate them to UML-AT (task 2). The translation process [4] uses the mappings of the preliminary tasks.

Before the beginning of the detection stage, developers select the AT contradictions to check and customize them. They can change variables in the patterns (see the structure of contradictions in Section 3), like the name of the entities or the concrete stereotype for an artefact. These actions are covered by task 3 in Fig. 3.

Next, the translated specification is traversed looking for groups of entities and relationships that fit into the match patterns (see Section 3) of the selected contradictions. This means having the same elements, related in the same way, and compatible values in their fields. This corresponds to task 5. When one of these groups is found, the correspondence between the match pattern and the specification determines an instantiation function between variables in that match pattern and its actual instances in the models. The instantiated match
pattern of the contradiction provides further explanation on the potential problem (see task 7).

After detecting a contradiction, the user has to decide if it makes sense in the context of the specification. If this is the case, the next step is finding solutions to it (task 8 in Fig. 3). These solutions come from the solution patterns related with the detected match pattern (see Section 3). They provide hints to modify the model of the system by removing or mitigating the contradiction. Solution patterns rearrange the elements in their match pattern, perhaps with some new elements too. The instantiation function obtained in task 7 allows proposing an instantiated solution pattern for the specific situation. Again, the user has to decide whether to apply this result.

The final step is task 9. The modified specifications are translated back from UML-AT to the language of the agent-oriented methodology using the mappings. If several translations are possible for new, the user has to decide which one is the correct.

From all these tasks, the first ones demand the greater efforts. They require expert knowledge about AT, its contradictions, and the target agent-oriented methodology. Nevertheless, their results can be reused over different projects. The process in Fig. 3 is supported by a plug-in of an agent-oriented development tool, the INGENIAS Development Kit (IDK) (http://ingenias.sourceforge.net). This plug-in, called the Activity Theory Assistant (ATA), assists developers to carry out tasks 3, 8, and 9 and fully automates tasks 1, 2, 5, and 7. The ATA uses knowledge about contradictions, UML-AT, and the agent-oriented languages as XML files.

6. An example: a recommender system

The application of AT contradictions in MAS analysis and design is illustrated here with a case study of a recommender system that relies on collaborative filtering techniques. It is described in [6] and its full specification with the INGENIAS methodology [8] can be found at http://grasia.fdi.ucm.es/ingenias. As it is shown later, the existing design was improved with the use of the techniques described in this paper.

Collaborative filtering assumes that if a user finds interesting a piece of information, then other users with similar opinions and preferences may also find interesting the same piece of information. Therefore, this system implements workflows to evaluate and distribute documents (i.e. information) among users, who are grouped in communities. Both users and communities have their corresponding agents in the system.

A Personal Agent plays several roles in the system. Two of these roles are intrinsically related with the collaborative filtering itself. They are the role of Suggester, who makes a suggestion to the community, and...
the role of Advisor, who evaluates the suggestion of a member of the community. On one side, the Suggester pursues the goal Provide Interesting Documents, which is necessary in order to supply information to the community of users. On the other side, the agent’s interest is that her community deals with relevant information. This is represented with the role of Advisor that pursues the goal Preserve Document Quality.

The workflows that relate these goals are shown as a producer-consumer chain of tasks and agent mental entities (what is produced and consumed) in Fig. 4. The workflow begins when the Personal Agent, who is playing the role of Suggester, makes a Suggestion to the community. This Suggestion represents a contribution to the information of the community. Members of the community have to evaluate that Suggestion in order to know if it is relevant according to the topics of the community. The Community Agent executes the workflow Share Documents to select the evaluators for the Suggestion and gives them the information about it. Those members of the community playing the role of Advisors execute the task Begin Document Evaluation that starts the workflow Evaluate Documents. These Personal Agents satisfy their goal of Preserve Document Quality through the goal Detect Bad Information. The result of Evaluate Documents is the fact Action, which the Community Agent uses to elaborate the final evaluation.

This evaluation process was the result of guessing that users could adjust their recommendations to the topics of the community, just studying the information usually exchanged by its members. The results of the prototype that implemented this policy were not as expected. Many new users were not able to continue as members of the community despite of their interest, because their recommendations were always rejected and, however, they did not really know the reasons.
To detect what the problem was exactly, we analysed contradictions management with the ATA tool (see Section 5). The ATA was asked to detect all possible contradictions (between the 10 available in the repository) that involved the roles of Suggester and Advisor and the Suggestion. The ATA obtained the current models from the IDK, made the translation to UML-AT using the mappings for INGENIAS, and looked for instances of the match patterns. The aforesaid lack of feedback for suggesters was a kind of Conflict producer-user contradiction (see Section 4). A Personal Agent that play the role of Suggester generates a Suggestion to satisfy its goal of Provide Interesting Documents. Other members of the community playing the role of Advisors analyze this Suggestion, as it contributes positively to their goal of Preserve Document Quality. The results of this analysis, i.e. the Actions, are processed by the Community Agent. If these results are not communicated to the Suggesters, they cannot know why their Suggestions were rejected.

The solution pattern related with the Conflict producer-user (see Section 4) suggests that involved activities should also satisfy goals for their neighbouring activities, that is, the producer’s activity should accomplish some goal for the consumer and the consumer’s activity some for the producer. The ATA uses the matching to instantiate the marked up solution. Fig. 5 shows the result of this instantiation, which is the solution that the ATA proposes to change the models and solve the contradiction. This solution allows the information exchange required by Suggesters to adjust their behaviour to what the community requires.

Some remarks about the semantics of the entities in the matching of Fig. 5 (i.e. the elements outside the dashed ellipse). Note that in the original match pattern for the Conflict Producer-User (see Fig. 2), only subjects appear while in Fig. 5 there are subjects and roles. This replacement is semantically correct because the goals and tasks of a subject (that is an agent) are those of her own and those that belong to the roles that she plays. Activity 1 was created by the ATA to comprise several tasks and workflows of the original models that are not relevant to characterise this contradiction. Other comment is that the objectives of the Activity 1, i.e. Detect Bad Information, and that of its subject, i.e. Preserve Document Quality, are not the same. The pattern is applicable thanks to the relationship OR-decompose between both goals. This relation implies that a way of satisfying the goal Preserve Document Quality is satisfying Detect Bad Information.

Fig. 6 shows the translation of modifications in Fig. 5 to INGENIAS, with dashed ellipses enclosing new elements. On the right, there is part of the modified workflows about suggestions. According to the solution pattern, the goal Do Not Send Undesired Information is related to Propagate Suggestions, which is a task without further decomposition. Nevertheless, Detect Annoying Users was linked to Activity 1, which comprehends several tasks and workflows. So, it is not possible to create a new pursue relation between the goal and some concrete task of the role Advisor. The assistant can only propose the task Begin Document Evaluation and the workflow Evaluate Documents of the Advisor, as possible points to connect the new goal and the developer has to decide.

As stated in Section 3, the solution pattern in Fig. 5 does not include other elements that could be needed to implement the solution. For example, the original tasks included in the Activity 1 may need to create some kind of report about the reasons of a given evaluation, which does not appear in the figure. However, this is not a limitation of the process. Solution patterns try to give hints about possible changes to enhance the models. They do not pretend to be exhaustive about how to make the changes. If developers want this particularity, they could add to the library of contradictions a new solution.

7. Conclusions

The development of MAS can benefit the methods and tools used for managing contradictions. This paper contributes to this issue by presenting a framework that adapts the extensive knowledge of AT about contradictions to the agent domain.

The framework includes a structure to represent AT contradictions, a repository of such contradictions, and mappings for specific agent-oriented methodologies. This infrastructure aims to facilitate the use of this knowledge by non-experts in AT (such as software engineers) and in an automated way. The repository gives a set of contradictions ready-to-use in MAS developments that comprises the knowledge about contradictions extracted from AT. Mappings give portability over agent-oriented methodologies and allow developers to work transparently with AT techniques in their preferred methodologies.

The result of applying this process is better designed MAS specifications, that is, with less conflictive situations about social and intentional features. This improvement is provided through support tools, relieving developers of obtaining new skills about social sciences to deal with those features. Even the simpler contradictions can be neglected or be difficult to find in real medium-size specifications, what produces problems in the final product. So, developers need all the available knowledge and support to deal with such troublesome configurations.

The experimentation has exposed two main concerns when applying this approach. The first one is how to detect whether a contradiction is meaningful and the better way to solve it. The described process is interactive in the sense that the developer has to decide about the relevance of the contradiction and its solution. Currently, our group
is considering different approaches to improve the support for these decisions as case-based reasoning for intelligent adaptation of ATA or libraries of components to allow its customization. The second one is concerned with the scalability of the pattern matching process, as finding isomorphic graphs is known to be NP-complete.

Acknowledgments

This work has been performed in the context of the INGENIAS 2 project, which is funded by the Spanish Council for Science and Technology under grant TIN2005-08501-C03-01.

References