Robust JaCaMo Applications via Exceptions and Accountability

Demonstration Track

Matteo Baldoni
Università degli Studi di Torino, Dip. di Informatica
Torino, Italy
matteo.baldoni@unito.it

Cristina Baroglio
Università degli Studi di Torino, Dip. di Informatica
Torino, Italy
cristina.baroglio@unito.it

Roberto Micalizio
Università degli Studi di Torino, Dip. di Informatica
Torino, Italy
roberto.micalizio@unito.it

Stefano Tedeschi
Università degli Studi di Torino, Dip. di Informatica
Torino, Italy
stefano.tedeschi@unito.it

ABSTRACT

Robustness is the degree to which a system can function correctly in the presence of perturbations. We present two extensions to the JaCaMo agent platform to realize robust MAS applications. The first extension delivers an exception handling mechanism suited for MAS; the second one is grounded on the notion of accountability to create feedback chains among the agents. Both extensions provide high-level abstractions that facilitate the design and development of a MAS that meets robustness requirements.

KEYWORDS

JaCaMo; Engineering MAS; Exception Handling; Accountability

ACM Reference Format:
Matteo Baldoni, Cristina Baroglio, Roberto Micalizio, and Stefano Tedeschi. 2023. Robust JaCaMo Applications via Exceptions and Accountability: Demonstration Track. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 3 pages.

1 INTRODUCTION

Robustness, *"the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions"* – generally called *perturbations* [13], is a crucial requirement of distributed software systems [9, 10, 14, 15]. Multi-Agent Systems (MAS) [18] are an effective approach to realize distributed systems by means of heterogeneous, and autonomous agents. Agent *organizations* (MAO), in particular, provide abstractions for modularizing code spread over many components, and orchestrate their execution by way of norms. JaCaMo [8] is one of the best-known platforms for implementing MAOs, but it focuses on providing the means for capturing the normal, correct behavior of the system and lacks of structural mechanisms allowing agents to exchange and propagate information (feedback) when they face perturbations. As in [1], the availability of *feedback* about perturbations is crucial to build robust distributed systems. Also MAS robustness should ground on the ability to convey feedback about perturbation to the agents that can handle it. But since agents generally are peers, and are not related by relationships like caller-callee or parent-child, the realization of robustness should occur through the definition of distributions of responsibilities among the agents, that become part of the MAO. This demonstration presents two extensions to the JaCaMo platform which allow building robust agent organizations. The first borrows from software engineering the concepts of *exception* and *exception handling*, while the second relies on the notion of *accountability*. Exception handling is suitable for treating perturbations anticipated at design time (i.e., exceptions) by activating handlers, that are also specified at design time. Accountability, instead, defines feedback "channels" that agents can use at runtime to gain situational awareness about what occured and then take actions. Raising and handling exceptions as well as asking and returning for an account will be tasks, under the responsibility of specific agents. The two extensions provide the means for representing such tasks as goals, and for distributing the reponsibilities of such goals to the capable agents. Note that each such goal can be assigned to many agents, specifying the minimum and maximum cardinality of how many agents need to achieve the goal (as standard in a JaCaMo organization specifications). Moreover, we allow specifying many raising and handling goals for each exception, and many requesting, accounting and treatment goals (possibly involving many agents) for each accountability.

2 JACAMO + EXCEPTIONS

We consider a production cell for metal plates inspired to [16]. The system involves five robots (agents) that coordinate their activities for producing plates. The process can be realized as a JaCaMo organization where the organizational goal of producing a plate is decomposed into sub-goals the robots should achieve. We introduce the management of the possible malfunction of one of the motors of the elevating rotary table (*ERT*). Such a condition should be detected as soon as possible to stop the production and schedule repair. A first solution would be to add the treatment of the perturbation as a part of the original goal decomposition. This solution, however, is strongly discouraged by practice. In fact, mixing business logic and exception handling logic complicates the verification of processes as well as later modifications [11]. Our proposal is to keep the original goal decomposition distinct from any malfunction treatments, and introduce new abstractions for modeling treatments as exceptions to be raised and handled[1]. For instance, the following *Notification Policy* complements the goal decomposition.

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom. © 2023 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

[1]Source code available at http://di.unito.it/moiseexceptions.

```
<notification-policy id="npTable"  target="turnTableMoveUp"
                     condition="scheme_id(S) & failed (S,turnTableMoveUp)">
    <exception-specification id="exMotor">
        <exception-argument id="motorNumber" arity="1" />
        <raising-goal id="notifyStoppedMotorNumber" />
        <handling-goal id="scheduleTableMotorFix" />
    </exception-specification>
</notification-policy>
```

The policy specifies how the ERT motor malfunction exception is handled by scheduling repair. In fact, the policy targets goal turnTableMoveUp, assigned to ERT in the original decomposition, and is activated whenever such a goal fails (see the condition expressed in NOPL syntax [12]). The policy then specifies the type of exception, exMotor, and its argument: the number identifying the motor affected by the problem (fundamental in order to act on the right motor). The exception, thus, amounts to a piece of information to be exchanged between some agent that detects the problem, and some that can handle it. To model this relationship, we extend the notion of goal, native in JaCaMo, in two ways. A **Raising Goal** is used to make an agent produce an exception (i.e., the structured piece of information), and raise it by making this exception available, through the organization, to the agents that can handle it. A **Handling Goal** is used to model the treatment of an exception when it becomes available. Both types of goals can be included in agent missions, as any other goal in JaCaMo. This has an important consequence: whenever an agent enacts an organizational role, it is asked to commit to all the goals included in the missions associated with that role, including raising and handling goals. So, from the perspective of agent programming, the treatment of exceptions is completely transparent: agents just need to bring about their goals when asked to, independently of whether these goals are in the original decomposition or part of some notification policy.

In the example, the goals notifyStoppedMotorNumber and scheduleTableMotorFix must be included in the missions of agents having the right capabilities for completing them. So, either ERT itself could the exception raiser, or the exception could be raised by external, observing agent. An important feature of our proposal, in fact, is a clear separation of concerns among the agents where a perturbation occurs, where it is detected, and, then, where it is treated. That is, the agent whose goal fails may be different from the agent that actually detects the failure and raises the exception. And the agent handling the exception is usually different from the one that raised it. The details of the implementation can be found in [2, 3, 6, 17].

3 JACAMO + ACCOUNTABILITY

Accountability allows agents to get runtime information to be used in their decision making. Specifically, a party, named account-taker (*a-taker*) is entitled to ask for an account about a goal of interest to another party, named account-giver (*a-giver*), that is obliged to provide such an account upon request. Through accountability, thus, agents have access to information otherwise inaccessible, and, hence, have greater awareness of what is going on in the overall system. This allows the agents to take advantage of opportunities, and to adapt to changing system conditions.

For instance, suppose the production cell is part of a production plant that should possibly never be stopped. Assume that an agent is in charge of supervising the production process. This agent can, under certain conditions, ask the *feed belt* robot the amount of plates still to be processed. Depending on such a number, the supervisor can decide to slowdown the production, in order to avoid a full stop. Also in this case, this behavior could be included within the original goal decomposition of the production process, but the result would be a mix-up between business and control logic. Our solution keeps separate business and control logic, and exploits accountability to allow the supervisor to obtain the needed information from the *feed belt* robot. The following *Accountability Agreement*, included in the definition of the organization, serves this purpose[2].

```
<accountability-agreement id="aa1">
    <target  id="conveyPlateToTable"  />
    <requesting-condition value="true" />
    <account-template>
        <account-argument id="availablePlates" arity="1"  />
        <goal id="requestRemainingStock" atype="requesting"  />
        <goal id="notifyRemainingStock" atype="accounting"  />
        <goal id="slowDownProduction" atype="treatment"
           when="account(_, availablePlates (N)) & N <= 10 & N > 0"  />
        <goal id="stopProduction"  atype="treatment"
              when="account(_, availablePlates (0)) "  />
    </account-template>
</accountability-agreement>
```

The accountability agreement is the abstraction we offer to allow accounts to flow from a-givers to a-takers. Specifically, an accountability agreement targets a goal, e.g., conveyPlateToTable, which represents the object of the account. An agreement is activated by a requesting condition, that can even be true, meaning that it can be asked at any time throughout the execution. An important part of the agreement concerns the structure of the account, and how it can be asked and provided. The structure of the account is given as a list of arguments with their corresponding arity. In our example, a single argument availablePlates with arity one, is sufficient to convey the number of plates in the queue. Concretely, we leveraged the model presented in [4] and the formalization from [5, 7].

To model the request and the notification of an account we extend JaCaMo goals, as we did for exceptions, by introducing the notions of **Requesting Goal** and **Accounting Goal**. Intuitively, when an agent wants to get some specific information outside its context, and has the permission to ask for them, the agent needs just to accomplish a requesting goal. This activates, by way of the normative system of the organization, the associated accounting goal specified in the agreement. For instance, when goal requestRemainingStock is marked as achieved, goal notifyRemainingStock is activated, and the agent responsible for it (i.e., *feed belt*) receives the obligation to carry it out. The agreement reports also an optional **Treatment Goal**. When specified, the goal specifies how the account should be addressed by the a-taker. In our scenario, there are two alternative treatment goals: one to be activated when the number of available plates is between 0 and 10, then the production is slowed down; and one to be activated when such a number is 0, and hence all the production cell is stopped.

Also in this case, the requesting, the accounting and the treatment goals are part of role missions, as for standard JaCaMo goals. By committing to such missions, agents take on the responsibility to perform these goals whenever a corresponding obligation is issued by the normative system of the organization.

[2]Source code available at http://di.unito.it/moiseaccountability.

ACKNOLEDGEMENTS

This publication is part of the project NODES which has received funding from the MUR –M4C2 1.5 of PNRR with grant agreement no. ECS00000036.

REFERENCES

[1] David L. Alderson and John C. Doyle. 2010. Contrasting Views of Complexity and Their Implications for Network-Centric Infrastructures. *IEEE Tr. on Sys., Man, and Cyber.* 40, 4 (2010).

[2] Matteo Baldoni, Cristina Baroglio, Olivier Boissier, Roberto Micalizio, and Stefano Tedeschi. 2021. Demonstrating Exception Handling in JaCaMo. In *Advances in Practical Applications of Agents, Multi-Agent Systems, and Social Good. The PAAMS Collection - 19th International Conference, PAAMS 2021, Salamanca, Spain, October 6-8, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12946)*, Frank Dignum, Juan M. Corchado, and Fernando De La Prieta (Eds.). Springer, 341–345. https://doi.org/10.1007/978-3-030-85739-4_28

[3] Matteo Baldoni, Cristina Baroglio, Olivier Boissier, Roberto Micalizio, and Stefano Tedeschi. 2021. Distributing Responsibilities for Exception Handling in JaCaMo. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems* (Virtual Event, United Kingdom) *(AAMAS '21)*, Ulle Endriss, Ann Nowé, Frank Dignum, and Alessio Lomuscio (Eds.). International Foundation for Autonomous Agents and Multiagent Systems, 1752–1754. http://www.ifaamas.org/Proceedings/aamas2021/pdfs/p1752.pdf

[4] Matteo Baldoni, Cristina Baroglio, Roberto Micalizio, and Stefano Tedeschi. 2021. Reimagining Robust Distributed Systems through Accountable MAS. *IEEE Internet Computing* 25, 6 (2021). https://doi.org/10.1109/MIC.2021.3115450

[5] Matteo Baldoni, Cristina Baroglio, Roberto Micalizio, and Stefano Tedeschi. 2021. Robustness Based on Accountability in Multiagent Organizations. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS '21)*, Ulle Endriss, Ann Nowé, Frank Dignum, and Alessio Lomuscio (Eds.). International Foundation for Autonomous Agents and Multiagent Systems, 142–150. http://www.ifaamas.org/Proceedings/aamas2021/pdfs/p142.pdf

[6] Matteo Baldoni, Cristina Baroglio, Roberto Micalizio, and Stefano Tedeschi. 2022. Exception Handling as a Social Concern. *IEEE Internet Computing* 26, 6 (2022), 33–40. https://doi.org/10.1109/MIC.2022.3216272

[7] Matteo Baldoni, Cristina Baroglio, Roberto Micalizio, and Stefano Tedeschi. 2023. Accountability in multi-agent organizations: from conceptual design to agent programming. *Autonomous Agents and Multi-Agent Systems* 37, 1 (2023), 1–37.

[8] Olivier Boissier, Rafael H. Bordini, Jomi Hübner, and Alessandro Ricci. 2020. *Multi-agent oriented programming: programming multi-agent systems using JaCaMo.* MIT Press.

[9] Samuel H. Christie, Amit K. Chopra, and Munindar P. Singh. 2021. Bungie: Improving Fault Tolerance via Extensible Application-Level Protocols. *Computer* 54, 5 (2021), 44–53. https://doi.org/10.1109/MC.2021.3052147

[10] Samuel H. Christie, Amit K. Chopra, and Munindar P. Singh. 2022. Mandrake: multiagent systems as a basis for programming fault-tolerant decentralized applications. *Autonomous Agents and Multi-Agent Systems* 36, 1 (2022). https://doi.org/10.1007/s10458-021-09540-8

[11] Claus Hagen and Gustavo Alonso. 2000. Exception Handling in Workflow Management Systems. *IEEE Trans. Software Eng.* 26, 10 (2000), 943–958. https://doi.org/10.1109/32.879818

[12] Jomi F. Hübner, Olivier Boissier, and Rafael H. Bordini. 2009. A Normative Organisation Programming Language for Organisation Management Infrastructures. In *Coordination, Organizations, Institutions and Norms in Agent Systems V (Lecture Notes in Computer Science, Vol. 6069).* Springer, 114–129.

[13] ISO/IEC/IEEE. 2010. Systems and software engineering - Vocabulary. *24765:2010(E) - ISO/IEC/IEEE International Standard* (2010).

[14] Anuj K Jain, Manuel Aparico IV, and Munindar P Singh. 1999. Agents for process coherence in virtual enterprises. *Commun. ACM* 42, 3 (1999), 62–69.

[15] Anup K. Kalia and Munindar P. Singh. 2015. Muon: designing multiagent communication protocols from interaction scenarios. *Autonomous Agents and Multi-Agent Systems* 29, 4 (2015), 621–657.

[16] Claus Lewerentz and Thomas Lindner. 1995. *Case study "production cell": A comparative study in formal specification and verification.* Springer, 388–416.

[17] Stefano Tedeschi. 2021. *Exception Handling for Robust Multi-Agent Systems.* Ph.D. Dissertation. Università degli Studi di Torino, Dipartimento di Informatica, Torino, Italy.

[18] Michael Wooldridge. 2009. *An introduction to multiagent systems.* John Wiley & Sons.