Engineering Business Processes through Accountability and Agents

Extended Abstract

Matteo Baldoni
Università di Torino, Dipartimento di Informatica, Italy

Cristina Baroglio
Università di Torino, Dipartimento di Informatica, Italy

Olivier Boissier
Laboratoire Hubert Curien UMR CNRS 5516, Institut Henri Fayol, MINES Saint-Etienne, France

Roberto Micalizio
Università di Torino, Dipartimento di Informatica, Italy

Stefano Tedeschi
Università di Torino, Dipartimento di Informatica, Italy

KEYWORDS
Accountability; Responsibility; Multiagent Organizations; JaCaMo.

ACM Reference Format:

1 RESPONSIBILITY AND ACCOUNTABILITY AS ENGINEERING CONCEPTS

A business process (BP) is “a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal.” [26]. In general, a business goal is achieved by breaking it up into sub-goals, which are distributed to a number of actors. Each actor carries out part of the process, and depends on the collaboration of others to perform its task. Multiagent Systems (MAS), in particular models for MAS organizations (MAO), are promising candidates to supply the right abstractions for describing BPs; however agent organizations still lack a systematic way to properly handle feedback of the execution of business processes in terms of good or bad functioning (e.g. exceptions). When a feedback occurs, the agent which can handle it (or which is interested to know), may be not the same agent who triggers the feedback. To make the overall system robust, the feedback should be reported to the agent with the proper means for treating it. In [2] a proposal was made to use accountability and responsibility relationships to state the rights and duties of agents in the organisation, given the specification of a normative organization. Building upon this work, we have studied how robustness can be achieved via accountability and responsibility relationships, and we used these concepts as tools to systematize and guide the design and development of the agents. It is worth noting that accountability and responsibility are not primitive concepts. Rather, they are properties that emerge in carefully designed software systems. Thus using them as engineering tools means that we actually constrain the ways in which software is designed and developed.

and certainty about the legitimacy of expectations between the community members.” [15]. In particular [18], accountability implies that some actors have the right to hold other actors to a set of standards, to judge whether they have fulfilled their responsibilities in light of these standards, and to impose sanctions if they determine that these responsibilities have not been met. It presupposes a relationship between power-wielders and those holding them accountable, where there is a general recognition of the legitimacy of (1) the operative standards for accountability and (2) the authority of the parties to the relationship (one to exercise particular powers and the other to hold them to account). BPs represent an agreed behavior, introduce expectations on the behavior of the interacting parties, and require some kind of governance in order for the process to be enacted, but the accountability results hidden into some kind of collective responsibility (“many hands problem”). As a consequence, the governance of the system is compromised as well as its functioning as a whole. This is what we aim at overcoming.

2 ENGINEERING MAO WITH ACCOUNTABILITY/RESPONSIBILITY

Since accountabilities and responsibilities imply some obligations [18], we can realize them in JaCaMo by relying on the deontic primitives that such framework provides. In other words, the fitting projection over role x (see [2]) can be mapped into a number of Jason plans of the agent playing role x by way of the following patterns, expressed in AgentSpeak(ER). Specifically, the fitting relationship represented by each pair \((R(x, q), A(x, y, r, u))\) in \(R_x \rightarrow A_x\), is mapped into an AgentSpeak(ER) g-plan as follows:

```latex
+!be_accountable(x, y, q) <: drop_fitting(x, y, q) {  
  // Well-Doing e-plan  
  +obligation(x, q) : r \land c <- body_q.  
  // Wrong-Doing e-plan  
  +oblUnfulfilled(x, q) : r \land c' <- body_f.  
}
```

Such that: (1) \(body_q\) satisfies the fitting-adherence condition (see below); (2) \(body_f\) includes the sending of an explanation for the failure from x to y. The two e-plans encode the proactive behavior of an agent assuming a responsibility. Until the responsibility is not dropped, the agent starts reacting to obligations in accordance to the accountability specified in the fitting. The agent will perceive, through the identity that is provided by the organizational role it plays, certain events as events it should tackle through some behavior of its own, but it will also be aware of its social position both (1) by knowing some other agent will have the right, under certain conditions, to ask for an account and (2) by including specific behavior for building such an account.

**Well-doing e-plan:** is triggered when the specified obligation is issued by the MAO. The context expression, \(r \land c\), is satisfied when condition \(r\) activating the agent accountability holds together with some possibly empty local condition \(c\) to choose among alternative plans, i.e., multiple ways to achieve a same result in different (local) circumstances. Due to the accountability fitting the agent has accepted, the body of the plan(s) \(body_q\) must, then, be such to satisfy the responsibility assumption represented by the pair \((R(x, q), A(x, y, r, u))\). That is, the plan body has to satisfy a fitting-adherence condition, by which there must exist an execution of the plan body that, restricted to the events that are relevant for the progression of \(u\), is an actualization of the responsibility \(q\). Intuitively, \(q\) is actually used for fulfilling the obligation. In this case, the obligation to give an account for the satisfaction of the obligation is implicitly resolved by satisfying the very same obligation. It is interesting to note that the accountability fitting is not only a functional specification of the organization, but it also specifies the “good” behavior of the agents. It is in fact this characteristic that justifies our programming patterns, that enriches the standard JaCaMo [9].

**Wrong-doing e-plan:** allows the agent to provide an account when it did not complete its task. The triggering event, \(oblUnfulfilled\), is generated by the MAO when an obligation is left unsatisfied. The context has the same structure as above; \(body_f\) produces an account of the failure. This will be an explanation that the agent produces and that some other agent will use to manage the exception and to resume the execution. The correct use of the pattern guarantees, by design, that exceptional events, when occurring, are reported to the agents who can handle them properly. Accountability fulfills this purpose because, by nature, it brings about an obligation on the a-giver to give an account of what it does.

The engineering of accountable JaCaMo MAOs involves that: (1) Each process is mapped to an organizational role; (2) A scheme representing the overall process goal is defined (its successful execution corresponds to the achievement of the process goal; (3) For each activity to be performed in sequence, a subgoal is added to the scheme by means of the corresponding operator; (4) For each structured block including a concurrent execution, the corresponding goals, grouped by the parallel operator, are added to the scheme; (5) For each choice all the schemes representing the possible courses of action should be defined. They will be instantiated dynamically by the agents, depending on their internal choices.

3 CONCLUSIONS

The systematic application of the proposed patterns makes agents aware of the process as characterization of the goal. Hence, accountabilities provide the programmer with a behavioral specification the agent has to satisfy. The proposal moves MAOs closer to other paradigms where exceptions are handled. In the actor model [19], for instance, when an actor cannot handle an exception, it reports the exception to its parent actor, which decides to either handle the exception or report it further up. In an agent-based system such a scheme is not directly applicable because agents are independent entities, and show no parent-child relationship. Approaches for modeling exceptions in a MAS setting have been proposed (see, e.g., [20, 22, 24]). However, no consensus has been reached on the use of such a concept in agent systems. The main problems rise when trying to accommodate the usual exception handling semantics with the properties of MAS; namely autonomy, openness, heterogeneity, and encapsulation. Accountabilities can fill this gap.

Commitment-based protocols, e.g. [28], as well as NorMAS [7], provide alternatives for modeling coordination. A detached commitment is an obligation on the debtor to bring about the consequent. In case of violation, the creditor has the right to complain against the debtor but cannot hold the debtor to provide an explanation. This lack of information hampers both the understanding of what has occurred, and attempts of recovery from the failure.
REFERENCES


