Expert Systems With Applications 95 (2018) 190–200

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Remediating critical cause-effect situations with an extended BDI architecture

J. Faccin [a,*], I. Nunes [a,b]

[a] Universidade Federal do Rio Grande do Sul, Instituto de Informática, Av. Bento Gonçalves, 9500, Porto Alegre, RS 91501–970, Brazil
[b] TU Dortmund, Fakultät für Informatik, Otto-Hahn-Straße 12, Dortmund 44227, Germany

ARTICLE INFO

Article history:
Received 7 June 2017
Revised 14 November 2017
Accepted 15 November 2017
Available online 15 November 2017

Keywords:
Software agents
BDI Architecture
Remediation action
Cause-effect
Goal generation
Plan selection

ABSTRACT

Remediation actions are performed in scenarios in which consequences of a problem should be promptly mitigated when its cause takes too long to be addressed or is unknown. Such scenarios are recurrent in the real world, including in the context of computer science. Existing approaches that address these scenarios are application-specific. Nevertheless, the reasoning about remediation actions as well as cause identification and resolution, in order to address problems permanently, can be abstracted in such a way that they can be incorporated to autonomous software components, often referred to as agents. They can thus autonomously deal with these scenarios, which we refer to as *critical cause-effect situations*. In this paper, we propose a domain-independent extension to the belief-desire-intention (BDI) architecture that provides such agents with this automated reasoning. Our work provides an extensible solution to this recurrent problem-solving strategy and allows agents to flexibly deal with resource-constrained scenarios. This solution removes the need for manually implementing the coordination of actions performed by agents, using causal models to capture the knowledge required to carry out this task. Therefore, it not only allows the development of systems with remediative behaviour, but also enables the reduction of development effort by means of a reusable infrastructure that can be used in several different domains. Our approach was evaluated based on an existing solution in the network resilience domain, which showed that our extended agent can autonomously address a network challenge, with a reduction in the development effort and no impact in agent performance.

1. Introduction

Human problem-solving is complex and involves the adoption of different strategies when facing problems. A not unusual scenario involves the realisation that what is being addressed is actually a consequence, or *effect*, that has a *cause*. In order to permanently solve the problem, both cause and effect must be dealt with. In such scenarios, there are issues to be taken into account in the practical reasoning (Bratman, 1987), i.e. the process of deciding how to act. First, *remediation actions* may be performed in order to mitigate effects, before addressing the cause, which must also be dealt with; otherwise, effects will likely reappear. Second, the cause may be *unknown* and, in this case, this should be investigated so that the real problem can be identified and resolved. If causes remain unaddressed, effects may return even with greater impact.

There are many real-world situations that match the scenario described above. In computer science, different examples can be observed. An example is in the context of network resilience (Sterbenz et al., 2010), which is the ability to provide and maintain an acceptable level of service in the face of faults and challenges to normal operation (e.g. a malicious attack). When dealing with a distributed denial-of-service (DDoS) attack, network operators must minimise the number of users affected by the unavailability of the service provided by the infrastructure being attacked. At the same time, they must act towards the identification and isolation of the attack origin to have it permanently blocked (Schaeffer-Filho et al., 2012). Another example can be seen in self-healing systems (Breitgand, Goldstein, Henis, Shehory, & Weinsberg, 2007), which are software systems able to monitor, diagnose, analyse and heal their problems, and prevent them from reappearing. To automate software memory management, objects that are no longer used, but still referred to, in Java programs (loitering objects, a form of memory leak in Java) may be “paged”, in order to keep the system operational while the software is debugged to identify such objects (Goldstein, Shehory, & Weinsberg, 2007).

* Corresponding author.
E-mail addresses: jgfaccin@inf.ufrgs.br (J. Faccin), ingridnunes@inf.ufrgs.br (I. Nunes).

https://doi.org/10.1016/j.eswa.2017.11.036
0957-4174/

Although promising, all these approaches were individually analysed and implemented, thus being application-specific solutions. Furthermore, the actions of mitigating effects and searching for causes are all explicitly modelled and *hard coded*. Existing solutions can potentially support a more flexible implementation of these approaches by exploiting the use of autonomous software components, called *agents*. However, the reasoning towards the selection of remediation actions, and cause identification and resolution is still manual. The belief-desire-intention (BDI) architecture (Rao & Georgeff, 1995) is such a solution. This architecture provides such agents with a reasoning cycle that includes an *option generation* function, in which goals to identify and resolve causes can be generated, and a *plan selection* function, in which remediation plans (sequences of actions) can be selected because other plans may not prioritise the mitigation of effects. However, such functions are abstract in the BDI architecture, and must be customised in specific applications to provide the desired behaviour. In fact, there are many existing approaches that extend, adapt and customise the gaps of the BDI architecture (Faccin & Nunes, 2015; Nunes & Schaeffer-Filho, 2014; Singh, Sardina, Padgham, & Airiau, 2010), but none are able to cope with the aforementioned scenario.

In this paper, we abstract from this described target scenario and propose a domain-independent extension of the BDI architecture that allows agents to autonomously select appropriate, possibly remediation, plans to solve a problem (i.e. achieve a goal) and deal with possible problem causes. The goal of our extension is not to serve as replacement of the traditional BDI architecture in agent development, but to automate the coordination of an agent's actions in our target scenario. This approach thus promotes *reuse* across domain-dependent solutions, designed and implemented in an *ad hoc* way, and allows agents to flexibly decide the best action according to the current context, agent goals and preferences. Our BDI architecture extension includes a set of components to capture the required domain knowledge to allow agents to make such decisions. This knowledge is then used in a customised reasoning mechanism, which selects remediation plans, when needed, and generates goals to search and deal with problem causes. Our approach is evaluated by taking an existing network resilience scenario (Nunes & Schaeffer-Filho, 2014; Schaeffer-Filho et al., 2012), which is implemented in an application-specific way, and developing it with our approach. Results show that our approach is able to reproduce the behaviour in a domain-independent way. Moreover, an empirical evaluation shows that the performance is not compromised by postponing the practical reasoning process regarding remediation plans to runtime and the developers' implementation effort is reduced. Our approach thus promotes software reuse in an important recurrent scenario, thus freeing developers from the complex reasoning to deal with critical cause-effect situations. Moreover, it is a flexible solution that chooses between remediation and definitive plans, depending on the current preferences and constraints.

In Section 2, we describe our problem by presenting an illustrative scenario that is used as a running example. We detail the elements comprising the extended BDI architecture in Section 3 and describe our customised reasoning mechanism in Section 4. Our approach evaluation is presented in Section 5. Finally, related work is discussed in Section 6, and conclusions are presented in Section 7.

2. Problem and running example

Before detailing the elements that comprise our meta-model, we introduce an illustrative example to provide a better understanding of the scenario addressed. It is used throughout this paper as a running example. Despite its simplicity and perhaps not adequacy of an agent-based solution to this problem, it allows us to clearly illustrate the class of problems we are targeting and concepts of our approach without having to detail additional domain background. Consider the problem of dealing with a ceiling leak. *Alice* is a person who notices a ceiling leak in the room of her house. To deal with it, she has three options (or **plans**): (i) cover the leak with *duct tape*; (ii) use a *towel* to absorb the liquid coming from the ceiling; or (iii) put a *bucket* underneath the drip.

The three available plans can achieve the **goal** of *dealing with the ceiling leak*. However, each of them has particular characteristics concerning how this goal is achieved. Each plan requires different amounts of *time* to be performed. Covering the leak with duct tape, for example, requires much more time than putting a towel under the drip. Plans are also related to different execution *costs*. If Alice chooses to use the towel to achieve the goal, she will have to afford the cost of such towel (assuming that it would be thrown away after being used). If she chooses to use the bucket instead, the cost she has to afford will be lower, given that a bucket can be reused. Therefore, plan executions are directly associated with the consumption of **resources**. Further, considering that the floor can become wet if the leak is not addressed as soon as possible, Alice has **constraints** over the execution time of her plans. In this context, using duct tape to cover the leak may not be a feasible solution. Moreover, every time Alice wants to deal with a ceiling leak, she may have different **preferences** on how to spend resources. For instance, if she wants to address it quickly, the plan execution time becomes more valuable than its cost.

Although performing one of the available plans achieves the given goal, Alice's problem is not completely solved. The leak is an **effect** of several possible **causes**, e.g. a broken pipe or a cracked roof tile, which must be addressed in order to stop the leak permanently. Moreover, there may be plans that deal with both cause and effect. However, they possibly cannot be executed in such a way that constraints are met. Therefore, assuming that an agent *A* is in charge of resolving this whole problem, two key issues must be addressed: (i) *how can agent A select the most adequate plan, possibly a remediation one, to achieve its goal based on its constraints and current preferences?*; and (ii) *how can agent A identify and address the causes of the effect associated with this goal to permanently solve the problem?*

In the traditional BDI architecture, an agent is structured in terms of beliefs, desires and intentions, which are mental attitudes that represent the information the agent has about the world, the state of the world it wants to bring about (i.e. its goals), and its commitment to achieve its desires, respectively. Agents based on this architecture have their goals explicitly specified, and plans to reach these goals are provided at design time. However, there is no specified strategy to choose among available plans. Moreover, depending on how the problem is modelled, plans that deal with the effect of the problem may not achieve the causes of this problem. Therefore, once a plan achieves the goal associated with the effect, its causes may remain unaddressed. Finally, the BDI architecture does not include means of searching for causes of problems being tackled.

3. Software agent architecture

The example presented above introduces many key concepts, such as resources and preferences. Some of these, e.g. goals and plans, can be associated with existing components of the BDI architecture. Others, however, are domain-independent concepts that can be incorporated to this architecture to provide agents with the ability of dealing with what we refer to as *critical cause-effect situations*. These situations are those in which goals must be achieved considering a set of constraints and correspond to effect mitigation before searching and dealing with its cause. Examples are the goals of dealing with a ceiling leak, potential network attack or problem