An Abstract Architecture for Computational Reflection in Multi-Agent Systems

Martin Rehák, Jan Tožička, Michal Pěchouček, Filip Železný, Milan Rollo
Gerstner Laboratory, Czech Technical University in Prague, Czech Republic
{*rehakm1,tozicka,pechouc,zelezny,rollo*}*@labe.felk.cvut.cz*

Abstract

We propose a general framework for computational reflection in multi-agent systems and address some technical issues related to its implementation. Potentials of computational models of cognition and reflection in a multi-agent system are detailed, and using such models, an abstract architecture of a reflective agent is designed. We also propose important characteristics of reflective multi-agent systems build upon the presented architecture.

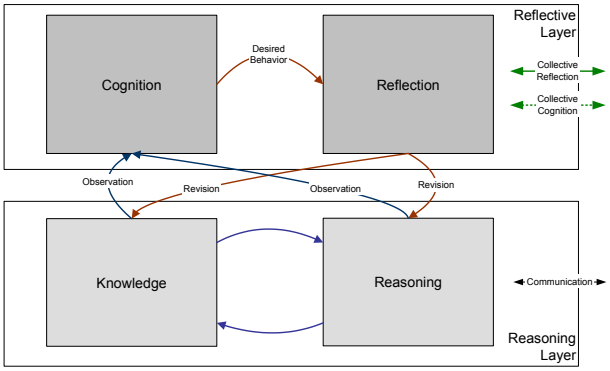

Figure 1. Role of cognition and reflection in the reflective agent.

1. Introduction

Similarly to the classical notation in computer science [7], [12], we will understand the term **reflection** as a quality of a computational system. A reflective system is capable of acquiring knowledge about its computational state, data, and execution processes. Such data can be used by the reflective system in order to build a model of its own computation on basis of which the computational process can be suitably modified.

Reflective computation consists of three non-trivial reasoning processes: (i) acquiring the relevant data about computation (ii) building the symbolic model (self-representation) of the computational process (iii) revising the computational process. Classically in the literature, (i) is referred to as introspective integrity, (iii) as introspective force, and (ii) is referred to as meta-reasoning. Similarly to [12], we understand reflection not only in the sense of agents' self-awareness and collective mutual awareness, but also as a capability to perform actions determined by the awareness.

For clarity of explanation, we will adhere to terminology simpler than that of [12]. Besides speaking of reflection as a *quality*, we will also speak of it as a *process*, with two fundamental parts shown in Fig. 1: (i) **cognition:** where agents can build models of self-awareness and mutual-awareness within the collective of agents, and (ii) **reflection:** where the operation, knowledge, inter-agent interaction and potentially the implementation structure of the agents are altered, so that reflective behaviour is achieved.

The paper is organised as follows. State of the art in techniques of computational reflection is reviewed in the next section. Section 3 puts the concept of computational reflection into the context of an autonomous agent community, outlining three major categories of reflection in such an environment and proposing the generic architecture and in Section 4 we conclude the paper.

2. State of the Art

In computer science, we distinguish two types of reflection: **structural** [1], [2] and **behavioural** [8] reflection. In the recent contribution [9], the authors define reflection types as follows: *Structural Reflection*: The system structure can be dynamically modified. *Computational (Behavioural) Reflection*: The system semantics (behaviour) can be modified. Examples provided by the authors are data structure modification and algorithm modification.

In multi-agent reflection, we use **structural reflection** in order to change agent's private data, such as its computational state or its acquaintance model. **Behavioural reflec-**

tion is used to change the agent's reasoning algorithms – the agent's component that interprets its data.

A reflective system is usually described as a system with a two-level architecture, where the object-level implements the basic functionality and behaviour, and the meta-level observes and modifies the behaviour of the object-level. Principles of the computational reflection are described e.g. in [7]. The most common approach to reflection is the meta-object approach [5]. Here each object from the object-level can be associated with a meta-level object that monitors its state and behaviour. Reflection in object oriented programming (OOP) is usually achieved by implementing the metaobject protocol as a part of the language interpreter. This approach however has some restrictions [9] – metaobject protocol restricts the amount of customizable features.

In the multi-agent research context, we use reflection for agent adaptation in mobile environment [6], to observe and adapt the whole multi-agent community [3] or even for user interaction customization [13].

3. Reflective Agent Concept

To introduce the specific types of reflection considered, we need to explain the fundamental architecture of a reflective agent [7] consisting of two layers: **agent reasoning core** containing the set of agent reasoning algorithms and agent's permanent and non-permanent knowledge, primarily regarding agent's problem solving skills, and **agent interaction wrapper** a built-in interface between the agents reasoning core and the rest of the multi-agent community. In the architectures of the socially aware agents, an insertable part of the wrapper tends to be an agent's *acquaintance model* [10] that collects the information about the other members of the community. We will distinguish three different types of reflection in multi-agent systems.

Individual reflection – Revision of the agent's isolated behaviour, that does not necessarily need to result from agents' mutual interaction. Individual reflection is an operation that works with agent's awareness of its own knowledge, resources, and computational capacity. *In this case the agent's reasoning core needs to be updated*.

Mutual reflection – Revision of agent's interaction with another agent. This kind of revision is based on agent's knowledge about the other agent, trust and reputation, knowledge about the other agent's available resources, possibly opponent's (or collaborator) longer term motivations and intentions. All these kinds of knowledge are refereed to as *social knowledge* and are stored in agents' **acquaintance models** – computational models of agent's mutual awareness. *Here the agent's acquaintance models need to be updated*.

Collective reflection – Revision of agents' collective interaction. This is the most complex kind of reflection. Here we address the revision of the collective behaviour of an agent group, as a result of their interaction. Collective reflection can be achieved either by a (i) **single reflective agent** (e.g. a meta-agent) that is busy with monitoring the community behaviour and updating agents' behaviour, or (ii) **emergently**, by a collective of agents, each carrying out its specific cognitive/reflective reasoning. In collective reflection, the agents update not only their social knowledge containers and reasoning processes but also they make attempts to *revise other agents' acquaintance models and, possibly, their reasoning* (unlike in the mutual reflection case).

In principle we can have two classes of collective reflection methods: (i) **Module sharing**, where agents share parts of their source code so that their potential functionality range is wide and their operation is lightweight and efficient at the same time, and (ii) **Collective programming**, where agents collaboratively construct the program with the intention to improve their collective behaviour. Module sharing and collective programming capability needs to be available also for agents with different types of reflective mechanisms. A feature created by one type of the agents must be described in a manner making it possible to share within the whole community.

Using the reflection processes for agent adaptation profoundly modifies the way the agent is implemented. Therefore, we present the technical issues that are related to this concept and design transformation:

Streamlined reasoning code – Currently, the efficiency, adaptation, scalability and robustness issues are handled in the reasoning module, producing code that is both more expensive to execute and difficult to maintain. Using reflection, the adaptation issues that are managed by the dedicated reflective process modifying the reasoning code, making it streamlined and goal-oriented.

Reasoning code/knowledge modification – Modification of the reasoning code and knowledge structure is often a necessary prerequisite to implement reflection. However, reflective programming is far from being easy and many new problems must be addressed in this approach.

Modular reasoning code – Parts of modular reasoning code are not always used but only available to be included by the reflection process. Modular reasoning code can be easily loaded to ubiquitous devices, as algorithm storage is inexpensive. Independent modules, for example logical clauses or executable classes can be used within reflection.

Autonomy – Autonomy of reflective devices is a crucial question, as their behaviour in various situations is difficult to predict in advance. Therefore, in most situations, we must restrict agent's behaviour using the norms or *adjustable* policies [11], to both prevent the undesired behaviour and maintain adaptability.

3.1. Abstract Architecture of a Reflective Agent

In a layered structure of the R/C agent, the classical agent architecture forms the lower half of the new structure - the *reasoning* (object) layer. On top of this layer, we add the reflective layer that includes cognition, meta-reasoning and the reflective capability (see Fig. 2).

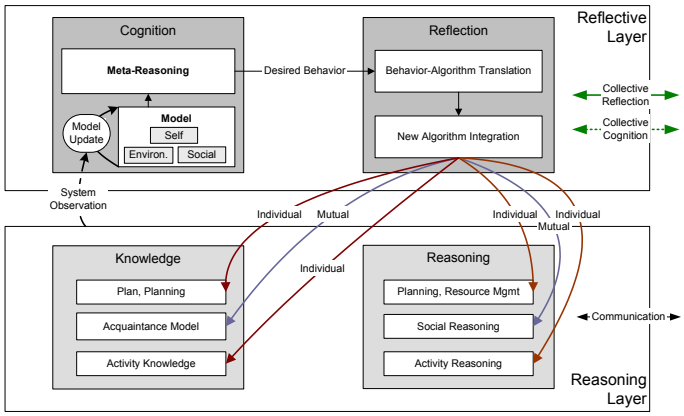

Figure 2. Reflective Agent Architecture

The *Reasoning layer* handles regular agent operation - it performs agent-specific actions (interaction with agent-specific technical resources, environment sensing), social interactions and, finally, planning and resource management.

We distinguish three functional sub-layers - the activity layer, the social layer and the planning layer. These layers will typically consist of adapted data structures containing the knowledge and corresponding algorithms processing the knowledge. Both the algorithms and knowledge in these sub-layers are subject to the reflection process.

On top of this activity, the reflective layer is responsible for system reflective behaviour. For this sake, it uses two main modules - the *cognitive module* that maintains the model of the agent in its environment and identifies the possible improvements, and the *reflective module* that performs the modifications in the reasoning layer's data, data structures and code to implement these improvements.

The *cognitive module* has two principal components - the model and the meta-reasoning working on the module data. The *model* contains knowledge about the agent's environment, social neighborhood and the agent itself. Compared with the object level, the knowledge it includes is less specific (contains classifications rather than actual values), contains historical experience rather than the mere current state, and it is represented in a form suitable for the meta-reasoning process.

One of the principal tasks of the cognitive module is to maintain the model up-to-date. This is the task of the *model update* process. This process not only handles new observations of the reasoning layer, but can also perform off-line processing on the model – such as preparing lemmas or extracting trends from the historical data.

The *meta-reasoning* process uses the data from all model components to identify the features and types of behaviour that the agent can modify to better fit the current environment. When the meta-reasoning process is finished, it may either conclude that the agent state corresponds well to the current situation, or it may identify possible improvements. The specification of such improvements is then used as an input that initializes the reflection process.

The *reflection module* is also divided into two subcomponents. The first component, *behaviour-algorithm translation*, handles the behaviour change specification resulting from the cognitive process. The exact form of this specification is not prescribed - the method used for the algorithm generation is typically the same as, or compatible with the meta-reasoning method.

Counter-intuitively, this module's output is not always an explicit algorithm. In many reflective operations, more subtle changes are possible, as many types of behaviour can be tuned by parameter modification only. On the other hand, some types of behaviour require changes even more abrupt than simple code change - we may need to completely restructure both the knowledge structure to be able to store new type of information, and the code to use the new knowledge. New parameter values, data structures and algorithms generated by this module are processed by the *algorithm integration* module. It handles issues like class loading, concurrency, agent responsiveness and syntactic and semantic compatibility.

When the new code is integrated into the baseline, agent resumes its normal operation (interrupted only during new code integration). Performance of the modified reasoning layer is still being observed by the cognition and if the need arises, new reflection iteration can be started. Concerning the amount of resources and computing time dedicated to reflection, the approaches can differ. In some situations, we will want to conserve valuable system resources and we will accept a slow pace of adaptation. In other situations, a rapid adaptation and near-optimum performance are crucial and we may dedicate more resources to achieve this goal. However, the most interesting approach is to use the reflection itself to manage reflection - reflective layer has all the knowledge about the last iterations, their success, importance of the changes in the environment and other factors.

4. Conclusion

The main ambition of this contribution has been to introduce the concept of reflection in multi-agent systems and suggest an abstract, application-neutral architecture supporting agents' individual and collective reflection.

Agent's autonomous adaptation based on the reflection processes is critical for future open ubiquitous (pervasive, ambient) systems, that will be composed of many various devices in ad-hoc manner. Once we deploy the diverse elements of these systems, they must be able to integrate themselves into the functional system and to maintain themselves operational even in a long-term perspective. Autonomous adaptation to the changing environment is critical, as it significantly increases the usability of ubiquitous systems by:

Increasing System Efficiency – A multi-agent system can increase the system performance by better adaptation to the current environment. Using reflection, we may keep the reasoning lightweight and hide the complexity of context management in the dedicated reflective layer, making the system computationally more efficient.

Robustness in Changing Environments – Using reflection, we seek to fabricate agents also able to detect changes in the environment or their own resources, and adapt to these changes.

Self-Maintenance – Self-maintenance is an important feature of modern computational systems. Using reflection, agents can autonomously maintain their reasoning code and keep themselves operational.

Functionality Extension – When we extend the system functionality, we add new goals to the system. Reflection process then modifies the overall system structure to reflect the new requirements.

Replacement and Upgrade – If we want to keep the system operational and well adapted to the environment during the transition phase, we shall not only transfer the knowledge of the existing agents, but also their reasoning and interaction patterns. Collective reflection, where old elements transfer both explicit and implicit knowledge to new ones, is a natural solution to the problem.

According to [4] reflection in programming may increase the computational requirements for the overall programme operation (running the reasoning and reflective layers). We argue that concept of collective reflection, where the agents may share the pieces of their programs and run on different levels of sophistication has got a potential to significantly improve operational efficiency of the community as a whole. This is mainly because the agents can also deploy different reflective mechanisms requiring different computational resources.

In our current work, we are implementing the above-described architecture using several AI methods (pattern matching, meta-programming, inductive logic programming) for Cognition and Reflection modules, and multi-agent interaction for collective programming and module-sharing.

Acknowledgment

We gratefully acknowledge the support of the presented research by Army Research Laboratory project N62558-03-0819.

References

[1] S. Chiba. Load-time structural reflection in Java. In *proceedings of ECOOP 2000*, *LNCS*, volume 1850, pages 313–336. Springer-Verlag, 2000.

[2] P. Cointe. A tutorial introduction to metaclass architecture as provides by class oriented languages. In *FGCS*, pages 592–608, 1988.

[3] J. Dix, V. S. Subrahmanian, and G. Pick. Meta Agent Programs. *Journal of Logic Programming*, 46(1-2):1–60, 2000.

[4] T. Elrad, R. E. Filman, and A. Bader. Aspect-oriented programming: Introduction. *Commun. ACM*, 44(10):29–32, 2001.

[5] J. Ferber. Computational reflection in class based object oriented languages. In *ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA89)*, page 317326, New York, NY, 1989.

[6] T. Ledoux and N. Bouraqadi-Saadani. Adaptability in mobile agent systems using reflection, 2000.

[7] P. Maes. Computational reflection. Technical report 87-2, Free University of Brussels, AI Lab, 1987.

[8] J. Malenfant, M. Jacques, and F. N. Demers. A tutorial on behavioral reflection and its implementation. In *Proceedings of the International Conference Reflection'96*, pages 1–20, San Francisco, CA, USA, 1996.

[9] F. Ortin and J. M. Cueva. Non-restrictive computational reflection. *Comput. Stand. Interfaces*, 25(3):241–251, 2003.

[10] M. Pěchouček, V. Mařík, and O. Štěpánková. Role of acquaintance models in agent-based production planning systems. In M. Klusch and L. Kerschberg, editors, *Cooperative Infromation Agents IV - LNAI No. 1860*, pages 179–190, Heidelberg, July 2000. Springer Verlag.

[11] M. Sierhuis, J. Bradshaw, A. Acquisiti, R. van Hoof, J. R., and A. Uszok. Human-agent teamworks and adjustable autonomy in practice. In *Proceedings of the 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space: i-SAIRAS - NARA*, Japan, May 2003.

[12] B. Smith. Reflection and semantics in lisp. In *Proc. 11th ACM Symposium on Principles of Logic Programming*. Salt Like City, Utah, Also Xerox PARC Intelligent Systems Laboratory, 1984.

[13] A. D. Stefano, G. Pappalardo, C. Santoro, and E. Tramontana. A multi-agent reflective architecture for user assistance and its application to e-commerce. In M. Klusch, S. Ossowski, and O. Shehory, editors, *Cooperative Information Agent VI – Lecture Notes in Computer Science, LNAI 2446*, Heidelberg : Springer-Verlag, 2002.