Software Agents as Boundary Objects

Migle Laukyte

CIRSFID, University of Bologna, via Galliera 3, 40121 Bologna, Italy
migle.laukyte@unibo.it

Abstract. Despite the wide use of agent-based applications in different areas of human activity, there hasn't been paid much attention to understand how these applications are possible, taking into account that they are build by people coming from such conceptually distant fields of study as, for example, law, artificial intelligence, and software engineering. This paper aims to fill in this gap addressing the different approaches to software agents—understood as building blocks of agent-based applications—adopted in each of these fields of study and suggesting that the way to understand how do these fields manage to work together in building a single agent-based application resides in seeing these agents as boundary objects.

Keywords: law, artificial intelligence, software agent, multi-agent system, boundary object, software engineering.

1 Introduction

Software agents[1] have been around for a while: perhaps no mistake will be made to date their appearance among computer science's topics in the late 1970s or early 1980s. From then on, software agents have been (and still are) studied, developed and discussed by the scholars of artificial intelligence, robotics, psychology, neurosciences, software engineering, and—the last but surely not the least—law.

Obviously, the nature, interests and aims of each of these scientific fields have led each of these fields to build a specific idea of what a software agent is. I am not pretending—alas!— to contribute in any way to the descriptions of agents developed in any of the fields listed above: what I instead would to like to do is to suggest to step back and take a wider perspective through which to look at these numerous agents around. This perspective, based on the idea of boundary objects, explains how it is possible to put together so many agents and involve as much many scientific fields in building a single agent-based application, and succeed in doing so. Hence, the main thesis of this paper is that the key to understand how the existence of so many agents cannot hinder, but on the contrary, can only enhance the development of a single agent-based application, is to conceive software agents as boundary objects.

To illustrate this idea I have organized this paper as follows: in Section 2, I describe how software agents are conceived in artificial intelligence (AI), while in

[1] I must clarify from the outset that I use software agent as a generic term covering all agents that come under this heading, thereby designating electronic agents, intelligent agents, artificial agents, and so on.

Section 3 I focus on software agents in software engineering (SE).[2] Section 4 then discusses how legal scholars understand software agents, while Section 5 offers a general introduction to the notion of boundary object used in sociology, and explores how this notion applies to our case, that is, to software agents in AI, SE and law. Finally, I conclude by explaining the benefits of considering software agents as boundary objects in developing agent-based applications.

2 Software Agents in AI

Software agents can be described in AI as rational-intentional agents based on the BDI model.[3] As AI suggests in its very name (artificial intelligence), AI agents are intended to artificially model natural intelligence. Human-intelligent agents, however, have not yet been created: what researchers have managed to develop are rational agents whose rational computing capacity is enriched with the human attributes of belief, desire, and intention, which gives the so-called Belief-Desire-Intention (or BDI) software model developed by Michael Bratman in the late 1980s ([2]).

Let us consider each of the model's three mental states in turn: beliefs are what an agent knows about the world, and are a necessary basis on which an agent figures out how to act in its own best interest. Beliefs are necessary since we couldn't work out any plan and couldn't even form intentions if we didn't believe anything to be the case. Also, an agent's beliefs need to reflect its changing environment, because that is how the agent responsively adapts to it. And if the environment is populated with other BDI agents, belief also makes it possible to infer what their beliefs and intentions might be, and hence what their behavior will be like.

Desires and intentions are close kin, in the sense that they both describe states of affairs the agent would like to come true. Desires and intentions thus enable an agent not so much to respond to its environment (because that is what beliefs are for) as to act on and modify it. Intentions then are a subset of an agent's desires: the latter are whatever an agent would wish to see in an ideal construction of the world, if only the world would correspond to its imaginings, while the former, by contrast, are those specific desires an agent has committed to.[4] They can thus be understood as

[2] My assumption here is that AI and software engineering are two of many disciplines in computer science. I am aware that there is no common position in this regard (see, for example, [1]), but the question as to how these disciplines are best classified falls outside the scope of this discussion, and for the purposes of this paper AI and software engineering will be sometimes be referred to by the general term of computer science.

[3] This is not to say, however, that BDI is the only agent model available in AI.

[4] This interestingly echoes the *legal* distinction between motive an intent: "intent is a state of mind [or *mens rea*] preceding or accompanying the act. Motive is the overall goal [good or bad] that prompts a person's actions" [5]. Which is to say that when we have a motive (e.g., revenge) we have not thereby also formed the intent to commit a specific act by which to satisfy that motive (a specific way to carry out the revenge, e.g., destroying so-and-so's property)—exactly as in BDI, which can thus be said to closely model the way the *law* views our decision-making. This is an example where different communities are using different names for the same concept (here the same distinction between the states of mind that move one to action). And if we only could see that the distinction between desire and intention in BDI is operationally the same as that between motive and intent in law, then we will at least have a platform on which to work in enabling the two communities to work together on future projects.

designating currently chosen courses of action [4], and that makes them deliberative in two respects, in that (i) deliberation is required to form an intention (or pick out a desire to be achieved), and (ii) once an intention has been formed, it will constrain an entire range of subsequent decisions. Intentions can thus be understood as a deliberative tool that simplifies decision-making by constraining all further reasoning and possibilities for action.

We have just seen one of the ways in which software agents can be understood in AI, namely, as entities whose rationality is enriched with beliefs, desires, and intentions. This combination gives them some autonomy, enabling them to decide on their own and act independently (at least to some extent). Clearly enough the scholars of AI study, address and develop in agents the qualities and characteristics that describe their area of research: this is why AI focuses on agent's mental states, such as intentionality or beliefs, and presents us with an idea of agent which vaguely manages to mimic some of our abilities, but is not yet ready to compete with us.

The observation to be made at this point is that even though researchers in AI work closely with software engineers, the latter do not share with the former this approach to software agents. So let us see how software engineers understand software agents.

3 Software Agents in Software Engineering

Software agents in software engineering (SE) are abstractions. I shall now have something to say about abstraction such as it relates to SE, and I will then explain how software agents fit into this concept. Then (in Section 3.1), I will illustrate these ideas by considering the use of abstractions in a specific SE context.

First of all, SE is based on abstractions: they capture content and knowledge and act as recipes on which basis to carry out different tasks, thereby simplifying them and enabling us to solve problems without requiring us to do work that has already been done. Practically this means, as engineers argue themselves, "you have to write less code and make fewer errors, and can therefore tackle more complex problems in the same time-scale" [6].

The reason for so abstracting—by simplifying and finding common elements—is to help software engineers manage the complexity involving the design and management of software systems, especially those representing a new approach in SE, namely the ones based on software agents and coined under the name of multi-agent systems (MASs) (see [7], [8], [9], and [10]). While software agents are indeed abstractions, they come in many varieties (for an overview and classification, see [11]), and this makes it necessary to investigate such agents in context: therefore we need to look at specific uses of agents to see what roles they play as abstractions in the agent-based approach to SE, or agent-oriented SE (AOSE).

3.1 Software Agents as Abstractions in AOSE

Agent-oriented SE (AOSE) is concerned with setting out approaches and methodologies on which basis to develop agent-based systems, understood as ones "in which the key abstraction used is that of an agent" [12]. An agent in such a system can be understood as autonomous and capable of interacting with other agents to satisfy the system's design

objectives: these objectives then require the agents to have additional properties, such as pro-activeness (agents can act on their own accord rather than just in response to the environment), reactivity (agents perceive their environment through sensors and can appropriately respond to changes in the environment in timely fashion), situatedness (agents inhabit an physical or virtual environment), and social ability (agents never exist in isolation but always form part of a society of agents with which they interact).

Each of these properties is itself an abstraction: for example, autonomy cannot be modeled simply by matching input to output, because in this way we would limit agent to a finite set of options, and this is certainly not what we would call autonomy (an ability to make choices on one's own). So instead we need to abstract from these items and set out a model on which basis the agent can reason and plan using the information or knowledge at its disposal.

An example illustrating the role of agents in a MAS is the SODA methodology (Societies in Open and Distributed Agent Spaces, online at http://www.apice.unibo.it/xwiki/bin/view/SODA/), which brings out two ways in which agents figure as abstractions in a MAS. First, in a MAS architecture, the entire design revolves around agents, since all of its components (its non-agent abstractions, such as actions, workspaces, operations, or resources) are meant to enable agents to work (perform actions) and to interact (form into an agent society) in a way that is functional to the system's overall purpose. Agents so considered might be called functional or architectural abstractions, or again system-component abstractions.

And, second, SODA also shows a sense in which agents are abstractions as role-playing agents: since a role is itself an abstraction, it can be structured in any number of ways, but it should not identify the individuals filling that role or the specific actions that satisfy it. And since agents as roles need not specify who is acting in their respective roles, they function as abstractions. Similarly, roles may be deemed abstract as metaphors by which to say that agents act on behalf of their human users.[5] Agents so considered might be called representative abstractions, or abstractions by proxy.

In this section we considered software engineers' approach to agents, which differs from the approach of those who work in AI: software engineers see agents not as slight reproductions of humans (or at least of some human capacities), but as abstract entities which are part of a overall interactive and abstract whole of elements, each driven by its goal and all aiming to achieve a single objective. But as much as these two approaches may differ, this difference will not be as big as the one that exists between these two approaches (AI and SE) and the legal one. Hence, the next section is dedicated to describe how law conceives software agents.

4 Software Agents in Law

As already mentioned briefly in the introductory part of this paper, software agents are studied not only in computer science, but also in social sciences: here we will be concerned with the law's approach to software agents, making the case that software agents in law can hypothetically be understood as entities whose actions may bring

[5] This is another instance where we can strike an analogy with the legal understanding of the concept we are dealing with, here that of agency, defined in law as "a legal relationship in which one person represents another and is authorized to act for him or her" [4].

about legally relevant consequences. For this reason it has become a matter of discussion how such agents ought to be classified and treated. In fact, legal scholars[6] seem to no longer doubt *whether* autonomous and intelligent agents will be among us, and have accordingly begun to prepare for the moment *when* this will happen. The issues to be solved are many, but they prominently include that of the rights and liabilities agents should have. But let us take the whole question from the beginning.

From the legal point of view, agents can be considered in either of two ways: (a) as goods, that is, software programs (standard approach), or (b) as legal persons, that is, as entities recognized as having a capacity to act in legally relevant ways. This means that the agents have rights and duties, such as the right to sue and be sued, the right to enter into contracts, and the right act on another's behalf, just as if they were real persons (nascent speculative approach).

On the current approach (a), the accent in the discussion on software agents is placed on the word *software* rather than on the word *agents*, meaning that software agents are treated as software products: they are accordingly protected by intellectual property law, and in particular by either copyright or patent law, depending on the characteristics of the application they are used in.[7]

But we will be concerned here with the second approach (b), on which the accent falls on the word *agent* (rather than on *software*), meaning that software agents would be treated not as products, but as creatures that can deliberate about what to do and can act on that decision. This, of course, describes a fully human agent (a self-conscious one), and it is because software agents cannot yet be described as an artificial equivalent of agency so conceived that I am calling this approach speculative.[8]

So the first question, as we enter into this speculative approach, is how do legal scholars conceive software agents and how they translate the technological complexity of agents into legal terminology? We can do this by running through a selection of definitions and then considering what they all have in common.

For [13], software agents are electronic forms of real human agents, and just like human agents in law, they do things for us. This analogy is premised on three similarities: (i) both types of agents can acquire and retain knowledge; (ii) both can perform a given task; and (iii) both can communicate. Specifically, software agents can "react autonomously to changes in their environment and solve their tasks without any intervention of the user" (ibid). On this basis these two authors analogize software agents more broadly to legal persons.

---

[6] When I refer to legal scholars, I don't have in mind only legal professionals (such as attorneys), but also people, who—having studied law—actively participate in building software-agent based applications for legal or business domains (for example, in knowledge acquisition or knowledge representation phases of the application's development), or are in any other way interested in the impact that such applications (might) have on the legal issues.

[7] Furthermore, they are also regulated by norms on consumer protection.

[8] We might also call this a what-if approach: what if software agents *were* like human agents in every respect except for the fact that they do not have a biological life? For a discussion based on the claim that such fully autonomous agents *are* already with us, see [21].

[14] argues that "something is a person [i.e., an agent] iff it has states whose interactions appropriately mimic our rational architecture."

[15] similarly presents the idea of an electronic person, a legal construct on which basis to specifically consider the personhood of software agents in law. An electronic person would be recognized as having limited liability: on the one hand, this would limit its owner's liability, and on the other hand, the contracting party would be able to check a registry of agents to see whether the agent is solvent before entering into a contract.[9]

[16] presents software agents as "digital entities capable of executing autonomously the mandates assigned to them," and he ascribes to them cognitive stances (beliefs, desires, and intentions) on which basis to qualify their actions in legal terms (in e-commerce, for example).

For [17] software agents are agents "capable of independent action rather than merely following instructions": they "exhibit high levels of mobility, intelligence, and autonomy according to which their actions are not always completely anticipated, intended, or known by their users," while [18] describe software agents as "intelligent and autonomous electronic agents."

What we can extract from these definitions is that software agents are regarded as agents owning two characteristics: (a) intelligence and (b) autonomy. And it is in particular this latter characteristic that makes the action of software agents legally relevant: autonomy is understood as an agent's capacity (i) to learn from experience; (ii) modify its own instructions; and (iii) work out new instructions to follow [19].

Intelligence and autonomy make software agents more than just tools or electronic devices: in [20] we can proceed on this basis "to treat [these] programs as legal agents of their principals, empowered by law to engage in all those transactions covered by the scope of their authority."

A third characteristic that legal scholars view as essential in making software agents worthy of consideration as legal persons is their intentionality (this is implicit, for example, in Sartor's account of software agents as endowed with the cognitive attributes of desire, belief, and intention). And still other characteristics are their belonging to someone else—and usually acting on that someone else's behalf—along with their ability to socialize. [21] describes these as "external characteristics of an independent individual," which takes us back to the idea of autonomy and intentionality.

4.1 Expanding the List of Legal Persons

The consequence we can extract from these considerations is that we will sooner or later reach the point where it is reasonable for us to add software agents to the list of entities recognized as legal persons. We have seen this kind of expansion before: in the early 19th century, for example, Chief Justice John Marshall found that this is how we are to consider a corporation, "that invisible, intangible, and artificial being,

[9] The idea of an agent registry is also considered in [19].

that mere legal entity, a corporation aggregate" [22]. He used the legal fiction of the corporation as a person, and no doubt the time will come when software agents must also be understood in the same way.

Of course, conditions must be ripe before anything can be viewed as a legal person: this was true of the corporation, and I am arguing that the same goes for software agents.[10] And so, if we can observe legal scholars working on how to extend legal personality to software agents, trying to work out the appropriate legal analogies to be drawn in making the extension, there must be a compelling set of reasons for so doing. One such reason, I would argue, lies in the need to protect humans who interact directly or indirectly with software agents, and it seems that the law is already equipped for the extension: [23] argue that "in principle the law can attribute conditional legal personhood to any well-defined type of entity." Hence, the right thing to do would be to make the agents to become "well-defined."

Then, too, as [20] argues, the extension importantly rests on the ground that software agents have evolved to the point where they acquire social meaning, in that we recognize them as entities which populate our social existence, our modes of social and economic interaction. It might be some time before this prediction comes true, but this will certainly be an essential criterion. And since it is unrealistic to posit a specific moment when software agents become a significant part of our social life, we can think of this as a process, whereby software agents will initially attain a kind of legal personhood and will thereafter incrementally become legal persons in one way or another. This suggests that, in establishing criteria on which basis to determine that software agents are no longer just tools but qualify as legal entities, we do not have to at once make them into full-fledged legal persons: as [24] suggests, we could work out a kind of legal personhood specifically tailored to software agents in a sense "akin to Roman law," where they would be "not legal persons in their own right, but with power to enter into binding arrangements, and receive information, on behalf of their owners, in circumstances where their owners would be bound by those arrangements or that knowledge."

I have briefly outlined the process by which the law is working out its own understanding of software agents: it proceeds by taking into account a number of criteria centered on what it is that makes something an agent, focusing on what is prominent among these criteria, namely, the capacities that make one (or something) a practical agent, one that can engage on its own in practical reasoning about what to do; we are thus looking at an autonomous intentional agent, one that can accordingly be deemed morally responsible and whose actions carry consequences for those with whom it interacts. At first sight this may seem to have little to do with the previously considered conceptions of software agents developed in AI and SE. And yet, despite these differences, those who work in these three areas—AI, SE, and law—understand that they

[10] At the same time, what may also be at play in this extension of personality is what has been called Topffer's Law (named for Rodolpfe Topffer, the father of the modern comic book), stating that we are inclined to attribute personality and character to any squiggle we recognize as a face: we are "meaning-making, pattern-seeking creatures [...] we read personalities into all kinds of interactive artifacts" [25], and I submit that if we are including software agents among such artifacts, this psychological law may well have something to do with it, too.

are all dealing with the same object, namely, software agent. This means that there must be a common ground, and so I will devote the rest of this paper to explore it and to argue that this ground explains why the three approaches are actually fruitfully complementing one another in building agent-based applications.

5 Software Agents and Boundary Objects

The differences between software agents as conceived in AI and SE, on the one hand, and in law, on the other, are obvious ones indeed, but the point here is not the amount of differences, but that despite these differences, AI, SE and law have found a way to collaborate with each other in different enterprises which lead to successful agent-based applications. And the question is how they managed to do that?

This is indeed a very important point in discussing how the very sophisticated and complex systems are being build: in fact, in building such systems the heterogeneity of the scientific fields involved and fluent communication among them is something that ensures a positive outcome. Still, the heterogeneity doesn't mean communication: on the contrary, different fields might not understand each other. So how is it possible in case of agent-based applications?

The answer to these questions is that this is possible if only we see software agents as boundary objects, understood as cross-border objects used in at least two different areas of activity. So let us briefly consider what a boundary object is (Section 5.1) and then see how this notion applies to software agents (Section 5.2).

5.1 What Is a Boundary Object?

A boundary object is a term coined in 1989 by Susan Leigh Star (computer scientist and sociologist) and James R. Griesemer (philosopher), and it designs any object—whether it be abstract or concrete—that different communities of practice (or CPs)[11] use in different ways while still recognizing the object as such. In this sense it is a plastic object—for its use and meaning change depending on who is using it and for what purposes—and yet it is solid enough at its core that the different communities using it will still know they are essentially dealing with the same object.

But what is CP? And how it relates to our discussion? The CPs are characterized by three basic features [26]: mutual engagement of community members, communally negotiated goals, and a shared repertoire. Mutual engagement means that members pursue the same interest, developing mutual relationships in such a way as to increase everyone's sense of belonging to that particular community. Community goals and guidelines are broad, with much latitude for the community's individual members, in that each member can frame specific goals within the broad outline, thereby

[11] I use the term "communities of practice," but Star and Griesemer have first applied boundary objects to social worlds, and only then expanded their applicability to communities of practice, information worlds, electronic community systems, digital libraries, etc. The difference among these fields of application is not the focus of this paper. More about boundary objects, fields of their application and other details, see [29].

contributing to the community in an individual way. A shared repertoire, finally, is everything that gets built over the course of a CP's activities: experiences, stories, constructs, categories, tools, events, images, expertise, know-how, and more.

Practically any field of human activity (professional, cultural, social, …) is CP: for example, lawyers, physicians, or astrologers could all be seen as members of their CP, because they all pursue their community's interests—legal issues, illness, or stars' influence on our characters—and all are engaged in discussions which permit them to work out the goals for their community, at the same time creating a shared repertoire of the practices, experiences and narratives that define each CP. Hence, in this paper AI, SE and law can also be seen as CPs. Obviously, this is a very simplified definition of CP which omits many important aspects and features, but I believe that it is still clear enough to give a reader an insight of what it is all about.

Where do boundary objects fit into this? In short, they are objects that different CPs can share and can use to interact, this owing to the ability of boundary objects to maintain a constant identity even as they travel across borders. Thus, on the one hand, different communities can recognize a boundary object for what it is (without mistaking it for something else), all the while tailoring that object to their different needs. Boundary objects have been described in this sense as having a structure at once weak and strong: weak because malleable—they can be fashioned into different "shapes"—but strong once they have been so fashioned, with each CP customizing the object for its own purposes. Or, if we flip this around, a boundary object makes it possible for each CP to use the object in its own way, all the while collaborating with other CPs, or at least communicating with them, in linking up the different types of knowledge they have each developed.

Hence, for example, we can say that astrologers and physicians share the concept of prognosis, which in medicine is attributed with a different meaning and based on different parameters than in astrology, even if in both cases it deals with hypothesizing about the future. Same applies to philosophy and SE which share the concept of abstraction, to art and civil engineering which share the concept of design and drawing, to political science and computer science which share the concept of institution, and so on and so forth. This shows that boundary objects populate different areas of human activity and interest, enabling the interaction among them and leading to different applications, results, uses and practices.

Having said that, we can move on to explain the idea of software agents as boundary objects used in three CPs, that is, in AI, SE, and law.

5.2 Software Agents as Boundary Objects

A software agent becomes a boundary object the moment we consider the use it is put to in the context of CPs of AI, SE, and law. As much as there is a large body of knowledge shared among the first two CPs—that is, AI and SE—this does not mean that any member of one CP will ipso facto be able to understand, communicate, and work with someone belonging to another CP. For example, a software engineer building an AI system may have to work with an AI researcher who is specialized in the psychology and physiology of the brain, two areas a typical software engineer will know nothing about, and yet the two may have to work together if they want to develop an intelligent system.

Even less common ground, than there might be between these two CPs, there is between these CPs and legal CP. But, again, this does not preclude their working together on common problems: evidence of this lies in the fact that we have an entire research area—legal informatics, or computer science and law—which (among other issues) is dedicated to bridge the law and computer science working out the best ways for legal scholars to contribute to the development of software applications dealing with legal problems.

And so we ask, How can a software agent understood as a boundary object enable the three CPs in question to benefit from one another's work? It can because a software agent so considered becomes a malleable entity, one that all CPs can recognize as such (as a software agent), but that each can use in its own way for its own purposes. This variety of uses explains why each CP will offer a different account of what a software agent is, but what appears to be a weakness (this lack of a shared definition) turns out to be a strength, for it shows that different CPs can approach software agents from different angles, or study them in different ways, all the while recognizing that they are, after all, software agents. Thus legal scholars are interested in seeing whether software agents exhibit properties akin to those that would make one an intelligent agent that can be held morally responsible for its own actions; software engineers seek to develop software agents with social abilities, such as a goal- or task-orientedness, interactivity, proactivity, and reactivity; and researchers in AI seek to model software agents having a capacity for rational behavior, endowing them with beliefs, desires, and intentions, and a capacity to act rationally on these mental states.

Now, these are not just different approaches, but different approaches that can be made complementary, because the CPs that pursue these different interests recognize them as revolving around the same object, namely, software agents. It is on this basis that we establish the common ground necessary to achieve the needed complementarity.

Thus, for example, we can recognize that all three communities understand software agents as performing precisely the role their name suggests, the role of agents or proxies, that which consists in doing something on someone's behalf: we saw this with the agency relationship in law (where an agent acts on a principal's behalf), and the same is true in computer science, where agents are entrusted with carrying out tasks for their users.

At the same time, software agents are understood in all three communities as autonomous agents: even when they carry out instructions for someone's benefit, they act with some latitude within that framework. Of course, each CP has its own view of what such autonomy means. Thus, for legal scholars, an autonomous agent is a practical agent that can be held accountable for the consequences of its actions;[12] for

[12] It must be noted, however, that the law of agency does limit or qualify this liability if (*a*) the agent is acting within the "scope of employment" (under the common law doctrine of *respondeat superior*), in which case the agent and the principal are both simultaneously liable for a tort of the agent (joint and several liability), or (*b*) the agent contracts with a third party on the principal's behalf and the principal is disclosed to such third party, in which case the agent bears no personal responsibility for the contract.

researchers in AI, an autonomous agent is one that can make rational decisions on its own using the information it receives and its data-processing logic in a virtual or "logical" environment; and for software engineers, autonomy is a software agent's interactive ability to be part of a MAS, in such a way as to handle the unpredictability the system may present on account of the way the other software agents in the system might behave. Hence, what we have here is the way in which the conceptual vagueness and malleability inherent in the idea of a software agent's autonomy can be an asset rather than a liability, because while each CP has its own idea of what it means for a software agent to be autonomous, they all understand that they are essentially dealing with the same thing, and indeed that the work each CP is doing in building autonomy into a software agent, or otherwise dealing with such autonomy, is complementary to the work of the others.

The complementarity involved can be also appreciated if we consider that while computer scientists deal with the question of what problems software agents could solve in the legal domain, legal scholars deal with the question what problems software agents could cause: in combination, these approaches can make it possible for computer scientists to develop better or improve the existing agent-based systems for domains where legally relevant consequences are possible (such as e-health or e-commerce). In e-commerce, for example, legal scholars have suggested taking into account the criteria of good faith and fair dealing in contract negotiation (see [27]). And many others have drawn developers' attention to the issue of privacy. As [24] have argued, "designers of artificial agents which access users' personal information or private communications need to be mindful of possible privacy implications: the more sophisticated their systems become, the more likely it is that corporations that deploy those agents will be attributed with knowledge of their users' personal information, possibly triggering significant legal liability." There is a twofold advantage to be gained here if computer scientists can interact with legal scholars, for on the one hand consumers can benefit from agent software that takes the legal scholars' concerns into account, and the law, for its part, would stand to benefit by forging rules stemming from a deeper understanding of what it is exactly that software agents actually do and how they behave.

6 Conclusions

We have considered three different approaches to software agents: the approach in AI, which considers software agents as rational entities guided by beliefs, desires, and intentions; the approach of software engineers, who treat agents as abstractions on which to build MAS; and the legal approach, which focuses on the practical implications the use of software agents brings about, and which works out hypothesis for their future legal personhood.

Then the question was put forward: how is it possible for all the people who work in AI, SE and law to work together on a single agent-based application when they all have different ideas on what an agent is. The answer to this question is given in the second part of this paper where I argue that the multilateral collaboration, interaction

and understanding among these professionals is based on the idea that software agents are boundary objects among AI, SE and law.

What does this mean practically? What advantages (if any) do we get from considering software agents as boundary objects? This multiplicity of agents around us is an advantage: this way we can open a space in which AI, SE, and law can share knowledge and work together in developing products that can benefit business and consumers at the same time: in fact, agents are seen by software engineers as offering "a uniform conceptual space where all the findings of the AI field can be easily framed and related and can eventually find mainstream acceptance" [28]. And, by the same account, these software engineers also argue that software-agent abstractions provide for "a new, powerful approach to the construction of intelligent systems": an understanding of software agents as boundary objects only supports this vision, and legal scholars cannot but contribute to improve it.

References

[1] Newell, A.: The Knowledge Level. Artificial Intelligence 18(1), 87–127 (1982)

[2] Bratman, M.E.: Intention, Plans, and Practical Reason. Harvard University Press, Cambridge (1987)

[3] Binmore, K., Castelfranchi, C., Doran, D., Wooldridge, M.: Rationality in Multi-Agent Systems. Knowledge Engineering Review Archive 13(3), 309–314 (1998)

[4] Rao, A.S., Georgeff, M.P.: BDI Agents: From Theory to Practice. In: Lesser, V., Gasser, L. (eds.) Proceedings of the First International Conference on Multi-Agent Systems, pp. 312–319. MIT Press, Cambridge (1995)

[5] Emerson, R.W., Hardwicke, J.W.: Business Law. Barron's Educational Series, Hauppauge, NY (1987)

[6] Henderson-Sellers, B., Gorton, I.: Agent-Based Software Development Methodologies. White paper for the OOPSLA 2002 Workshop on Agent-Oriented Methodologies (2002)

[7] Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. In: Müller, J.P. (ed.) Proceedings of the Fifth International Conference on Autonomous Agents, pp. 648–655. ACM, New York (2001)

[8] Jennings, N.R.: An Agent-Based Approach for Building Complex Software Systems. Communications of the ACM 44(4), 35–41 (2001)

[9] Zambonelli, F., Van Dyke Parunak, H.: From Design to Intention: Signs of a Revolution. In: Proceedings of AAMAS 2002, pp. 455–456. ACM, New York (2002)

[10] Weiß, G.: Agent Orientation in Software Engineering. The Knowledge Engineering Review 16(4), 349–373 (2001)

[11] Nwana, H.S.: Software Agents: An Overview. Knowledge Engineering Review 11(3), 1–40 (1996)

[12] Wooldridge, M.J., Ciancarini, P.: Agent-Oriented Software Engineering: The State of the Art. In: Ciancarini, P., Wooldridge, M.J. (eds.) AOSE 2000. LNCS, vol. 1957, pp. 1–28. Springer, Heidelberg (2001)

[13] Wettig, S., Zehendner, E.: A Legal Analysis of Human and Electronic Agents. Artificial Intelligence and Law 12, 111–135 (2004)

[14] Pollock, L.J.: How to Build a Person: A Prolegomenon. MIT Press, Cambridge (1989)

[15] Karnow, C.E.A.: Future Codes: Essays in Advanced Computer Technology and Law. Artech House, Boston (1997)
[16] Sartor, G.: Cognitive Automata and the Law: Electronic Contracting and the Intentionality of Software Agents. Artificial Intelligence and Law 17, 253–290 (2009)
[17] Dahiyat, E.A.R.: Intelligent Agents and Liability: Is It a Doctrinal Problem or Merely a Problem of Explanation? Artificial Intelligence and Law 18, 103–121 (2010)
[18] Andrade, F., Novais, P., Machado, J., Neves, J.: Contracting agents: Legal personality and representation. Artificial Intelligence and Law 15(4), 357–373 (2007)
[19] Allen, T., Widdison, R.: Can Computers Make Contracts? Harvard Journal of Law and Technology 9(1), 25–50 (1996)
[20] Chopra, S.: Rights for Autonomous Artificial Agents? Communications of the ACM 53(8), 38–40 (2010)
[21] Willmott, S.: Illegal Agents? Creating Wholly Independent Autonomous Entities in Online Worlds (2004), http://www.lsi.upc.edu/dept/techreps/llistat_detallat.php?id=695
[22] Bank of the United States v. Deveaux, 9 U.S. 61 (1809)
[23] Koops, B.J., Hildebrandt, M., Jaquet-Chiffelle, D.O.: Bridging the Accountability Gap: Rights for New Entities in the Information Society? Minnesota Journal of Law, Science & Technology 11(2), 497–561 (2010)
[24] Chopra, S., White, L.: Privacy and Artificial Agents, or, Is Google Reading My Email? In: Proceedings of the International Joint Conference on Artificial Intelligence. Morgan Kaufmann, San Francisco (2007)
[25] Mishra, P., Nicholson, M.D., Wojcikiewicz, S.K.: Seeing Ourselves in Computer: How We Relate to Technologies? Journal of Adolescence & Adult Literacy 44(7), 634–641 (2001)
[26] Wenger, E.: Communities of Practice: Learning, Meaning and Identity. Cambridge University Press, Cambridge (1999)
[27] Weitzenböck, E.: Good Faith and Fair Dealing in Contracts Formed and Performed by Electronic Agents. Artificial Intelligence and Law 12(1-2), 83–110 (2004)
[28] Omicini, A.: Challenges and Research Directions in Agent-Oriented Software Engineering: Autonomous Agents and Multi-Agent Systems. Special issue, Challenges for Agent-Based Computing 9(3), 253–283 (2004)
[29] Bowker, G.S., Star, S.L.: Sorting Things Out: Classification and Its Consequences. MIT Press, London (1999)