

Operational Semantics for Agents by Iterated Refinement

Federico Bergenti¹, Giovanni Rimassa¹, and Mirko Viroli²

¹ AOT Lab - Dipartimento di Ingegneria dell'Informazione
Parco Area delle Scienze 181/A, 43100 Parma, Italy
{bergenti,rimassa}@ce.unipr.it

² DEIS, Università degli Studi di Bologna,
via Rasi e Spinelli 176, 47023 Cesena, Italy
mviroli@deis.unibo.it

Abstract. In this paper we evaluate transition systems as a tool for providing a rule-based specification of the operational aspects of autonomous agents. By our technique, different aspects of an agent can be analyzed and designed in a loosely coupled way, enabling the possibility of studying their properties in isolation.

We take as a use case the ParADE framework for building intelligent agents, which leverages a FIPA-like ACL semantics to support semantic interoperability. Our grey-boxing technique is exploited to provide a specification where aspects related to the ACL, the adopted ontology, the agent social role, and the other agent internal details are described separately, in an incremental way.

1 Formalisms, Software Engineering, and Agents

This paper addresses the issue of sound multi-agent design trying to keep a system engineering perspective; that is, both when analyzing a problem and when synthesizing a solution the focus is kept on the whole, large-scale structure of the software artifact that is to be realized.

Striving for a sound design process naturally suggests to rely on some kind of mathematical tools: the precision and coherency afforded by formal and analytical reasoning holds great promises in terms of soundness. But, design is above all a creative process, and care has to be taken so as not to hamper it with too rigid a framework. In particular, formal methods during design activity should act as a tool to nurture an evolving idea, suggesting to the designer viable future choices while checking desirable properties of the current solution.

The formal tool we propose to provide effective assistance to the sound design of multi-agent systems (MAS) applies *labeled transition systems* to the description of interactive behaviors (of software abstractions) [6], as elaborated and promoted in the field of concurrency theory. When specifying a transition system semantics for an agent, a number of rules are given that describe in a quite declarative way the dynamics of its inner machinery, also providing insights on its internal architecture – expressed at a given level of abstraction. We believe that this formalism is particularly well suited because of its calculus-like nature,

which combines the description of a system with the prescription of its possible evolutions; this property buys designers some generativity, while remaining in a well grounded mathematical landscape.

In particular, in this paper we develop a formal framework that exploits basic features of labeled transition systems to address both composability and extensibility. Composability is obtained by dividing an agent model into several transition system specifications, that we call *tiers*; each tier captures the possible evolutions of the agent conceptual subpart dealing with a specific aspect. Beyond providing formal decoupling, tiers also mirror conceptually significant views over an agent behavior, thus easing the adjustable abstraction process. Moreover, starting from a partial specification of an agent behavior, namely a *grey-box model* [18, 15, 17], tiers are used to extend that specification by taking into account a new behavioral aspect, and leading to a new model that is a refinement of the former – according to the standard notion of refinement as “more deterministic implementation” introduced in the context of concurrency [6].

The tiers we use to describe an agent are more than a useful formal technique to us. They are closely related to the different levels of abstractions of an agent description.

The accepted scientific formalization of the idea of level of abstraction is the definition of *system level* [12]. A system level is a set of concepts that provides a means for modeling implementable systems. System levels abstract away from implementation details and are arranged in stack fashion so that higher levels provide concepts that are closer to human intuition and far away from implementation. System levels are structured in terms of the following elements: *(i)* Components, atomic building blocks for building systems at that level; *(ii)* Laws of compositions, laws that rule how components can be assembled into a system; *(iii)* A medium, a set of atomic concepts that the system level processes; and *(iv)* Laws of behavior, laws that determine how the behavior of the system depends on the behavior of each component and on the structure of the system.

In the specific case of agent systems, a relevant application of this idea is the well-known Newell’s *knowledge level* [12]. This level describes a single agent as structured in goals, actions, and a body, capable of processing knowledge and pursuing its goals according to the rationality principle. On the other hand, Jennings’ proposal of the *social level* [8] moves from agents to MAS trying to raise the level of abstraction of the knowledge level. There, the system is an agent organization, that is, a group of agents with organizational relationships and dependencies on one another, which can interact through channels. Other alternative systems level have been described such as e.g. in [1], taking the benefits of both Newell’s and Jennings’ trying to provide a concrete and useful compromise.

The various system levels cited above (and all the other, lower abstraction levels that may actually concur to fully describe a real system) provide a conceptual framework that suggests how to meaningfully partition an agent formal model, so that each part expresses properties pertaining to a specific system level. In this paper we adhere to this very methodology, dividing the specifica-

tion of an agent collaborative behavior into different tiers. In particular, due to the increased accuracy of a formal description with respect to a natural language one, these tiers are clearly finer than the usual conceptual system levels, but they can still be associated to a well defined system level.

As a use case for our approach we consider ParADE [2], a complete environment for the development of agents, taking into account social, agent and knowledge level issues, beyond of course interfacing itself with the lower system levels by means of its runtime support.

2 Outline

The remainder of the paper is organized as follows. Section 3 is devoted to describing the basic framework of labeled transition systems [6], which is exploited in this paper to formalize the behavior of agents and of their conceptual subparts.

Section 4 describes our novel approach to agent modeling. This is based on the idea of considering a partial specification of an agent behavior – namely, a grey-box model abstracting away from the agent inner details below a certain abstraction threshold [18] –, which can be refined by the specification of a *completion*. Such a refinement takes into account new aspects of the agent behavior, and leads to a new grey-box model that can be later refined again.

In order to show how our approach can be used to formalize an agent behavior in an incremental way, by separating different aspects of an agent design in different specification tiers, we take as a reference the ParADE framework for building agents [2], whose main characteristics are described in Section 5. ParADE exploits the formal semantics of a FIPA-like Agent Communication Language (ACL) to gain semantic interoperability, and provides *interaction laws* as a design mechanism to specify the agent social role. We chose to use this framework instead of others such as those based on the FIPA standard, the 3APL language [7], or Jack [13], since ParADE incorporates and keeps conceptually separated the many distinctive features that we believe are crucial when modeling MAS.

Section 6 provides a formalization of an agent behavior adhering to the main design choices of ParADE framework and considering five different tiers each dealing with a relevant aspect, namely, interaction, language, ontology, social role, and internal reasoning. Section 7 reports on related and future works, drawing concluding remarks.

3 Transition Systems

The semantic approach we describe in this paper is based on the framework of *(labeled) transition systems* (LTS) [6, 14], which originated in the field of concurrency theory to provide an operational semantics for process algebras [3]. Here, LTS are used in a slightly different, yet common fashion: instead of focusing on the semantics of a language or algebra, we apply them to the description of the interactive behavior of a system, which in our case is an agent (or a subpart of it). This application of LTS is still referred to as an operational

semantics, in that it describes the single-step capabilities of an agent to interact with the environment and to perform internal computations, resembling the idea of operational semantics in the context of programming languages.

A LTS is a triple $\langle X, \longrightarrow, Act \rangle$, where X is the set of states of the system of interest, Act is called the set of *actions*, and $\longrightarrow \subseteq X \times Act \times X$ is the *transition relation*. Let $x, x' \in X$ and $act \in Act$, then $\langle x, act, x' \rangle \in \longrightarrow$ is written $x \xrightarrow{act} x'$ for short, and means that the system may move from state x to state x' by way of action act .

Actions in a LTS can be given various interpretations. In the most abstract setting, they are meant to provide a high-level description of a system evolution, abstracting away from details of the inner system state change. In the sequence of transitions $x_0 \xrightarrow{a_0} x_1 \xrightarrow{a_1} x_2 \xrightarrow{a_2} \dots \xrightarrow{a_{n-1}} x_n$, the system evolution characterized by states x_0, x_1, \dots, x_n is associated to the actions sequence a_0, \dots, a_{n-1} , which can be thought of as an abstract view of that evolution. In a sense, actions can be used to describe what an external, virtual entity is allowed to perceive of the system evolution, generally providing only a partial description. In the case where LTS are exploited to represent the behavior of interactive systems, actions typically represent the single interaction acts of the system.

Several remarkable concepts are promoted by LTS, such as the notions of system *observation* and *refinement* of specifications [11]. Since actions in a LTS can be interpreted as a view of the system transition from a state to another, it is then possible to characterize a whole system evolution in terms of the actions permitted at each step and the actions actually executed. This characterization is made according to a given *observation semantics*, which associates to each evolution a mathematical description called *observation*. For instance, according to observation semantics called *trace semantics* [6], an observation is simply made of an allowed sequence of actions. By nondeterminism, generally more observations are possible for a system – either because of the environment interacting in different ways, or because of different internal behaviors –, so that according to the given observation semantics the software component of interest can be characterized by the set of all its possible observations. The notion of refinement then naturally comes in: a system description is considered an “implementation” of another – i.e., it is more directly executable – if it is more deterministic, that is, if it allows for strictly fewer observations [6]. This notion is particularly crucial in the context of software engineering: when some safety property of interest is verified on a system specification, it continues to hold when the system is substituted with a refinement of it.

4 Grey-box and Refinement

In [17], the rationale behind the grey-box modeling approach for agents is introduced. Its key idea is to represent an agent behavior by focusing on its part dealing with the interactions with the environment, called the (*agent*) *core*, while abstracting away from the complex inner details, namely, from the (*agent*) *internal machinery*. Clearly, deciding which aspects should be specified in the core

and which should be abstracted away depends on the abstraction level of interest. In order to deal with agent proactiveness, one of the key notions introduced by this approach is that of *spontaneous move*, which is an event occurring within the internal machinery that may influence the agent interactions, thus affecting the behavior of the core. So, while complexity of agents can be harnessed by choosing the proper abstraction level, the internal machinery behavior can be anyway taken into account by supposing that spontaneous moves can nondeterministically occur.

In [17, 18] a formal framework based on LTS is introduced to show that the grey-box modeling approach is particularly suitable for describing the agent abstraction. Based on this general idea, in this paper we go further developing this formal approach. We not only represent grey-box models as interactive abstractions, but also define a mechanism by which they can be refined by adding the specification of a new behavior, thus lowering the abstraction level and focusing on new details.

Notation In the remainder of the paper, given any set X , this is automatically ranged over by the variable x and its decorations x', x'', x_0, \dots – and analogously, a set Any is ranged over by any, any' , etcetera, and similarly for all sets. The set of multisets over X is denoted by \overline{X} and ranged over by the variable \overline{x} and its decorations; union of multisets \overline{x}_1 and \overline{x}_2 is denoted by symbol $\overline{x}_1 || \overline{x}_2$; \bullet is the empty multiset. Given any set X , symbol \perp is used to denote an exception value in the set X_\perp defined as $X \cup \{\perp\}$, which is ranged over by the variable x_\perp and its decorations.

A Formal Framework for Grey-Box Modeling Formally, an agent core is defined by a mathematical structure $\mathcal{A} = \langle I, O, P, X, E, U, \rightarrow_{\mathcal{A}} \rangle$. I is the set of acts that can be listened by the agent, O is the set of acts that the agent can perform on the environment. Both I and O can be *communicative acts*, involving the exchange of messages between agents, or *physical acts*, involving physical action in the agent environment, such as e.g. listening a stimulus from a sensor or moving a mechanical device by an actuator. P is the set of *place* states (or the set of *places* for short), which are the states of the agent core that can be observed by the agent internal machinery. X is the set of states of the remaining part of the agent core, so that $P \times X$ define the set of core states. E is the set of *events* occurring within the agent core and possibly affecting the internal machinery, while U is the set of *updates* notified by the internal machinery to the agent core, modeling the notion of spontaneous move. Relation $\rightarrow_{\mathcal{A}}$ is a transition relation of the kind $\rightarrow_{\mathcal{A}} \subseteq (P \times X) \times Act_{\mathcal{A}} \times (P \times X)$, defining how the agent core state evolves as actions in the set $Act_{\mathcal{A}}$ occur. These actions can be of five kinds according to the syntax $Act_{\mathcal{A}} ::= \tau \mid ?i \mid !o \mid \triangleright e \mid \triangleleft u$. Orderly, an action can represent the silent, internal computation τ , the agent listening act i , the agent executing act o , the event e occurring, and the update u being notified by the internal machinery.

Refinement of a Grey-Box Model It is common practice of system analysis and design to start considering a system at an high abstraction level, which

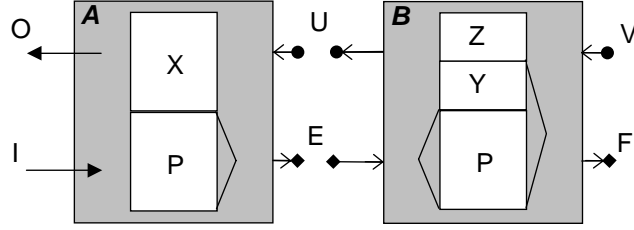


Fig. 1. Core and Completion

is later lowered in order to consider much more details. Then, a key feature of our methodological approach is to refine the specification \mathcal{A} by specifying some of aspects of the internal machinery, that is, by conceptually moving them from the internal machinery to the agent core. This is done by composing the specification \mathcal{A} by a new specification \mathcal{B} called *completion*, so that the resulting system $\mathcal{C} = \mathcal{A} \otimes \mathcal{B}$ has still the structure of an agent core specification likewise \mathcal{A} – hence it is still a grey-box model –, yet providing a number of additional details about the agent behavior.

A pictorial representation of our composition technique is shown in Figure 1. Formally, \mathcal{B} is a structure of the kind $\langle Y, Z, F, V, \rightarrow_{\mathcal{B}} \rangle$. $W = Y \times Z$ is the set of local states added by \mathcal{B} to the core state, Y is the part of it which is observable – along with P – to the new internal machinery, while Z is the local, hidden part of \mathcal{B} . $F \subseteq E$ and $V \subseteq U$ are respectively the new events and updates by which \mathcal{C} interacts with its internal machinery. Transition relation $\rightarrow_{\mathcal{B}}$, describing the behavior of the completion, is of the kind $W \times Act_{\mathcal{B}} \times W$, with $Act_{\mathcal{B}}$ being defined by the syntax $Act_{\mathcal{B}} := p : \tau \mid p : e \triangleright f_{\perp} \mid p : u_{\perp} \triangleleft v$. Action $p : \tau$ is the silent action occurring when \mathcal{A} is in place p . The action $p : e \triangleright f_{\perp}$ is executed when the place of \mathcal{A} is p : event e occurs that causes f to be propagated towards the internal machinery. The subcase $p : e \triangleright \perp$ means that no event f is generated by event e . The action $p : u_{\perp} \triangleleft v$ means that when the update v is listened from \mathcal{B} while \mathcal{A} is in place p , then u is notified to \mathcal{A} (or nothing if u is \perp). Notice that the occurrence of place p in these actions justifies the characterization of the place as the core’s subpart observable by the internal machinery.

Given models \mathcal{A} and \mathcal{B} , $\mathcal{C} = \mathcal{A} \otimes \mathcal{B}$ is defined as the agent core specification $\langle I, O, P \times Y, X \times Z, F, V, \rightarrow_{\mathcal{C}} \rangle$. The relation $\rightarrow_{\mathcal{C}}$, which is here of the kind $\rightarrow_{\mathcal{C}} \subseteq (P \times X \times W) \times Act_{\mathcal{C}} \times (P \times X \times W)$, is defined by the rules shown in Figure 2. Rules [I] and [O] state that input and output actions of the overall system \mathcal{C} are executed on the agent core \mathcal{A} . Rules [AT] and [BT] say that a τ action in either \mathcal{A} and \mathcal{B} is executed in isolation, and makes \mathcal{C} execute a τ action itself. Rules [AE] and [BE] describe the occurrence of an event in \mathcal{A} : in the first case no event f is generated by \mathcal{B} , in the second case \mathcal{B} produces an event f . Analogously, [AU] and [BU] deal with updates, with [BU] considering the case where an update is propagated from the internal machinery to \mathcal{A} and [AU] where no update is propagated to \mathcal{A} .

Following an approach similar to the one reported in [15], which is not reported here for brevity, it is possible to prove that agent core \mathcal{C} has a more

$$\begin{array}{l}
\frac{\langle p, x \rangle \xrightarrow{?i} \mathcal{A} \langle p', x' \rangle}{\langle p, x, w \rangle \xrightarrow{?i} \mathcal{C} \langle p', x', w \rangle} \quad [\text{I}] \qquad \frac{\langle p, x \rangle \xrightarrow{!o} \mathcal{A} \langle p', x' \rangle}{\langle p, x, w \rangle \xrightarrow{!o} \mathcal{C} \langle p', x', w \rangle} \quad [\text{O}] \\
\frac{\langle p, x \rangle \xrightarrow{\tau} \mathcal{A} \langle p', x' \rangle}{\langle p, x, w \rangle \xrightarrow{\tau} \mathcal{C} \langle p', x', w \rangle} \quad [\text{AT}] \qquad \frac{\langle p, x \rangle \xrightarrow{\triangleright e} \mathcal{A} \langle p', x' \rangle \quad w \xrightarrow{p':e \triangleright \perp} \mathcal{B} w'}{\langle p, x, w \rangle \xrightarrow{\tau} \mathcal{C} \langle p', x', w' \rangle} \quad [\text{AE}] \\
\frac{w \xrightarrow{p:\tau} \mathcal{B} w'}{\langle p, x, w \rangle \xrightarrow{\tau} \mathcal{C} \langle p, x, w' \rangle} \quad [\text{BT}] \qquad \frac{\langle p, x \rangle \xrightarrow{\triangleright e} \mathcal{A} \langle p', x' \rangle \quad w \xrightarrow{p':e \triangleright f} \mathcal{B} w'}{\langle p, x, w \rangle \xrightarrow{\triangleright f} \mathcal{C} \langle p', x', w' \rangle} \quad [\text{BE}] \\
\frac{w \xrightarrow{p:\perp \triangleleft v} \mathcal{B} w'}{\langle p, x, w \rangle \xrightarrow{\triangleleft v} \mathcal{C} \langle p, x, w' \rangle} \quad [\text{AU}] \qquad \frac{w \xrightarrow{p:u \triangleleft v} \mathcal{B} w' \quad \langle p, x \rangle \xrightarrow{\triangleleft u} \mathcal{A} \langle p', x' \rangle}{\langle p, x, w \rangle \xrightarrow{\triangleleft v} \mathcal{C} \langle p', x', w' \rangle} \quad [\text{BU}]
\end{array}$$

Fig. 2. Rules for composition of an agent core and its completion

refined behavior than agent core \mathcal{A} in the sense specified e.g. by trace semantics [6]. Hence, our refinement technique can indeed be considered as a way of deepening a specification towards implementation issues.

5 ParADE

ParADE, the *Parma Agent Development Environment* [2], is a development framework that provides the agent developer with high-level abstractions like beliefs and goals. It implements a hybrid agent architecture capable of supporting autonomy and intelligent behaviors by exploiting the semantics of its ACL – resembling the FIPA ACL [5] but with a much lighter semantics. Such an architecture is basically goal-oriented but it also integrates reactive behaviors.

ParADE ACL provides an operational means for agents to exchange representations of beliefs, intentions, and capabilities. The semantics is modeled as the effect that the sender wishes to achieve when sending the message. We stick to a syntax similar to that of FIPA ACL: $\phi \in \Phi$ stands for any predicative formula, a and b for actions, s for the identifier of an agent sending a message, r for the receiver, $B_j\phi$ for “entity j believes ϕ ”, $I_j\phi$ for “entity j intends ϕ ”, and $done(a)$ for “action a has just happened”. Each message is associated to a feasibility precondition (FP) that must hold in the sender and a rational effect (RE) that the send should intend: for instance $inform(s, r, \phi)$ has the precondition $B_s\phi$ – namely, the sender must believe ϕ – and $request(s, r, a)$ has the rational effect $done(a)$ – the sender should intend a to be executed. As a result, a precise semantics can be assigned to a message by its receiver, for example message $request(s, r, a)$ is associated to the semantics $B_r I_s done(a)$ (the receiver believes that the sender intends a to be executed) – see [2] for more details.

The ParADE ACL provides a means for exchanging complex logic formulae through simple messages as the receiver can assert what the sender is intending. Isolated messages are not sufficient to allow agents to communicate fruitfully. The classic example is the case of an agent requesting another agent to perform an action: in the case of asynchronous messages there is no guarantee that the

receiver would act in response to a message. Moreover, the semantics of a single message might not be sufficient to express application-specific constraints. The semantics of performative *request* does not impose the receiver to communicate to the sender that the requested action has been actually performed. The sender might hang indefinitely while waiting for the receiver to tell it that the action has been performed.

Using an ACL provides an agent with a linguistic environment that attaches semantics to its utterances. By virtue of the speech act theory, utterances become actions, with their attached pre- and post-conditions. Still, an agent is situated within an environment that is not limited to its social milieu; there are actions to perform and events to perceive that are not linguistic. Moreover, even linguistic acts often refer to entities belonging to something else than the language itself. These entities belong to the *domain* of the discourse, i.e. the external environment itself, and are typically described within a *domain model* or *ontology*. This ontology not only shapes the content of exchanged messages, but can also constrain the available actions of an agent associating them with domain-specific pre-conditions. In ParADE, the domain model works as the non-linguistic counterpart of the ACL semantics, enabling uniform processing of both the interactions with other agents and with the external environment.

In order to support socially fruitful communication, the ParADE agent model provides *interaction laws*. These are rules that an agent decides to adopt to govern its interactions with other agents. The interaction laws that an agent decides to follow are part of its capabilities and they are published. As an example, consider the interaction law $I_r done(a) \leftarrow B_r I_s done(a)$ for the agent r , which means that whenever he believes that agent s intends the action a to be executed, then this becomes an intention of r as well. This law should characterize an agent always willing to cooperate with s . In particular, as shown in the example we provide in Section 6.2, if s asks r to execute action a by means of a *request* message, then r will come to believe that the rational effect of the message is intended by the sender, that is $B_r I_s done(a)$. Then, the above interaction rule may be applied that makes $done(a)$ become an intention of r as well. In general, interaction laws are an elegant and flexible way to describe interaction protocols, they can be linked to the possible roles that an agent can play in the MAS, and they may also vary over time.

The ACL semantics, the domain model, and the interaction laws work together to expose an observable model of a ParADE agent that enables semantically meaningful communication with others. However, they don't fully specify the agent behavior; an agent programmer using ParADE is able to define several rules that drive the evolution of the agent mental states, only a subset of which will be published as interaction laws. This means that there is an internal set of rules that completes the agent specification, but is visible to nobody else than the agent designer. Though in most cases the designer wants to abstract away from the agent internal evolution rules, it is still useful to notice that even this relatively low-level part of the agent specification is still described declaratively.

6 Iterated Refinement

In this section we show an application of our technique for formalizing the behavior of an agent by successively refining an initial description, taking as a reference architecture the ParADE framework. We refer to the term *tier* when describing these different levels, each of which focuses on a different behavioral aspect and is then amenable to a separated description, thus fostering the understanding of its key features. The tiers we analyze not only correspond to the main features of ParADE, but also include most of the main aspects that an agent implementation has to take into account: orderly (i) the *interaction tier*, managing interactions with the environment, (ii) the *linguistic tier*, dealing with aspects related to the ACL semantics, (iii) the *domain tier*, concerning the specific ontology of the application, (iv) the *social tier*, where interaction laws define the role and public peculiarities of the agent, and finally (v) the *internal tier*, dealing with other aspects such as private laws, planning, and proactiveness. The order of these tiers reflects their impact on the external, observable behavior of the agent, from tackling interaction issues to considering internal reasoning. It is worth noting that the actual implementation of a ParADE agent is not actually separated into these tiers, which are instead a modeling tool useful at design-time independently from the actual agent implementation, representing conceptual levels that isolate the different behavioral aspects of agents within MAS.

In order to keep the presentation clear and reasonably compact, locally to a tier we sometime abstract away from some policy or management that is of a too lower abstraction level – either concerning details of the ParADE architecture or of the specific agent peculiarities –, referring to some generic function, relation or set encapsulating the actual behavior.

6.1 Splitting the Agent Specification into Tiers

The Interaction Tier From the point of view of the MAS, the most distinctive aspect of an agent is its ability of interacting with other agents and of performing actions on the environment. So, not surprisingly, the first tier we introduce is the one dealing with the interaction abilities of an agent, namely listening and performing acts. In the following, we denote by Ic and Oc the sets of input and output communicative acts, and by Ip and Op the sets of input and output physical acts, so that $I = Ic \cup Ip$ and $O = Oc \cup Op$.

The task of this tier is to decouple the actual agent interactions with respect to their internal processing, which is a common feature of almost all agent implementations – sometimes mentioned to contrast the notion of agent with respect to that of component (or object). Here, we suppose that the set X of states unobservable by the internal machinery is defined as a multiset of pending input acts waiting to be processed and pending output acts waiting to be executed, namely, $X = \bar{I} \cup \bar{O}$. The set P of places is defined as the set of agent mental states: a mental state $\Phi_M \in P$ at a given time is a set of formulae $\phi \in \Phi$, representing current beliefs and intentions of the agent. This is explicitly represented

in this tier because, typically, the occurrence of interactions has an immediate representation in the knowledge of an agent.

A composition operator between such formulae is defined so that if $\Phi_M \subseteq \Phi$ is the current mental state and $\Phi_n \subset \Phi$ contains some new beliefs and intentions, then $\Phi_M \circ \Phi_n$ is the new mental state obtained from Φ_M by adding formulae in Φ_n . Clearly, this composition should be defined according to the specific update policy for mental states, which we here abstract away from, analogously e.g. to [16]. Similarly, given a mental state Φ_M , we suppose that if this is of the kind $\Phi' \circ \Phi_n$ then the agent mental state includes all the beliefs and intentions of Φ_n .

An element of the set $P \times X$ is hence a couple $\langle \Phi_M, \bar{i} || \bar{o} \rangle$, here denoted as $\Phi_M || \bar{i} || \bar{o}$ for uniformity of notation. The set of events E coincides to I , representing the input acts listened from outside. The set of updates U is $O \cup \mathcal{P}(\Phi)$, representing either output acts to perform or requests to change the mental state by adding a subset of formulae in Φ (with $\mathcal{P}(\Phi)$ being the powerset of Φ). Transition relation $\rightarrow_{\mathcal{A}}$, describing the behavior of the interaction tier, is defined by the rules:

$$\begin{aligned} \Phi_M \xrightarrow{?i}_{\mathcal{A}} i || \Phi_M \circ \{B_r done(i)\} & \quad \Phi_M || i \xrightarrow{\triangleright i}_{\mathcal{A}} \Phi_M \\ \Phi_M || o \xrightarrow{!o}_{\mathcal{A}} \Phi_M \circ \{B_r done(o)\} & \quad \Phi_M \xrightarrow{\triangleleft o}_{\mathcal{A}} \Phi_M || o \\ \Phi_M \xrightarrow{\triangleleft \Phi_n}_{\mathcal{A}} \Phi_M \circ \Phi_n & \end{aligned}$$

The first rule simply states that when input act i is received ($?i$), the mental state is updated so as to reflect the reception ($B_r done(i)$), and then the act i is stored in the state X waiting for being notified as an event by means of the second rule ($\triangleright i$). The third and fourth rule, conversely handle output acts, which are inserted in X by updates ($\triangleleft o$), and are later sent outside by output actions ($!o$), affecting the mental state. Finally, fifth rule handles a mental state update requested by the internal machinery ($\triangleleft \Phi_n$), according to the semantics of composition operator \circ .

Different implementations of interactions dispatching could be specified in this tier, including the case where acts are stored into queues representing the agent mailbox instead of being immediately served as in the model above – where a precondition for receiving and sending acts is that no act is still pending.

The Linguistic Tier A fundamental aspect of the ParADE framework is that it exploits an ACL semantics to enjoy true semantic interoperability between agents [1], as currently promoted by the FIPA ACL and KQML approaches. So, as next tier we take into account the constraints imposed on communicative acts by the specific ACL. Then, we define a completion \mathcal{B} specifying the main concepts of the ACL, namely, the syntax of messages and their impact on the agent mental state.

Syntax is defined by introducing sets $I_{ACL} \subseteq Ic$ and $O_{ACL} \subseteq Oc$ of communicative acts allowed by the ACL: when a message ic is received that does not conform to that syntax this is simply ignored. The set of visible states Y is here void, since no new information is to be added to the mental state Φ_M that should

be visible to the internal machinery. Instead, set Z of local, invisible states is used to store scheduled updates \bar{u} to be propagated to the interaction tier, thus $Z = \bar{U}$: in particular, such updates are requests for updating the mental state with formulae Φ_n . On the other hand, events F produced by this tier are of the kind $ic \in I_{ACL}$, while updates V coincides with U .

In order to deal with the effect of a communicative act on the agent mental state, we define $eff_M^I : I_{ACL} \mapsto \mathcal{P}(\Phi)$ as a function associating to a communicative act its intended effect on the mental state of the receiver – formed by beliefs and intentions to be added –, and $end_M^O : O_{ACL} \mapsto \mathcal{P}(\Phi)$ as the function associating to a communicative act the condition on the local mental state enabling its sending, expressed as facts the agent has to believe and intend. In the case of ParADE, and similarly to FIPA ACL, we have e.g. $eff_M^I(ic) = \{B_r FP(ic), B_r I_s RE(ic)\}$, that is, when the message is processed the receiver comes to believe that the feasibility preconditions of the act are satisfied, and that the sender intends its rational effects. Analogously, $end_M^O(oc) = \{B_s FP(oc), I_s RE(oc)\}$, that is, a message can be sent out only if the sender believes the feasibility preconditions and intends the rational effects of the act. The rules for transition relation \rightarrow_B that deal with communicative acts are as follows:

- $\frac{\Phi_M : ic \triangleright ic}{\rightarrow_B} eff_M^I(ic)$ if $ic \in I_{ACL}$
- $\frac{\Phi_M : ic \triangleright \perp}{\rightarrow_B}$ • if $ic \notin I_{ACL}$
- $\frac{\Phi_M : oc \triangleleft oc}{\rightarrow_B}$ • if $oc \in O_{ACL}$ and $\Phi_M = \Phi'_M \circ end_M^O(oc)$
- $\frac{\Phi_M : \perp \triangleleft oc}{\rightarrow_B}$ • if $oc \notin O_{ACL}$ or $\Phi_M \neq \Phi'_M \circ end_M^O(oc)$

The first rule says that when the communicative act ic arrives that is correct with respect to the ACL ($ic \in I_{ACL}$), then this is redirected as event f (by an action of the kind $p : e \triangleright f$), and an update for changing the mental state is correspondingly scheduled; the second rule says that acts that do not belong to the ACL are ignored. Dually, the third rule handles the case where an output act that is correct with respect to the ACL syntax is to be sent out, which is actually performed only if the proper conditions hold on the mental state (ϕ_M includes conditions $end_M^O(oc)$). On the other hand, if the act is not correct or these conditions are not satisfied, the fourth rule makes the update be simply ignored and discarded. The remaining rules are as follows:

- $\frac{\Phi_M : ip \triangleright ip}{\rightarrow_B}$ • • $\frac{\Phi_M : op \triangleleft op}{\rightarrow_B}$ • • $\frac{\Phi_M : \perp \triangleleft \Phi_n}{\rightarrow_B} \Phi_n$ u $\frac{\Phi_M : u \triangleleft \perp}{\rightarrow_B}$ •

The first and second rule let input physical acts and output physical act to flow across the tier. The third rule reifies requests for changing the mental states within the tier, which by the fourth rule are redirected to the previous tier. Since this reification technique is exploited in the next tiers as well, the latter two rules are assumed here to be included in the specification of both the ontology, social, and internal tier, even if they are not actually reported for brevity.

The Domain Tier The domain tier is the part of the agent specification that deals with aspects related to the ontology defining the application domain where the agent lives. First of all, the syntax of communicative acts is further constrained with respect to the ACL, since e.g. the actual content of a message is limited to those terms to which the ontology gives an interpretation. We therefore denote by $I_{CONT} \subseteq I_{ACL}$ and $O_{CONT} \subseteq O_{ACL}$ the communicative acts allowed by the specific ontology. Then, the ontology also defines what are the physical acts that may be listened and executed by the agent, which are denoted by the sets $Ip_{ONT} \subseteq Ip$ and $Op_{ONT} \subseteq Op$. The sets of acts allowed by the ontology are then naturally defined as $I_{ONT} = Ip_{ONT} \cup I_{CONT}$ and $O_{ONT} = Op_{ONT} \cup O_{CONT}$. Finally, each output physical act is associated to an expected rational effect, by a function $cmd_M^P : O_{ONT} \mapsto \mathcal{P}(\Phi)$, associating output acts to sets of formulae (facts to be intended and believed) that will become satisfied. This function is here supposed to associate to communicative acts a void set, while in the case of a physical act op we have e.g. $cmd_M^P(op) = \{I_sRE(op)\}$. In the case the domain ontology binds physical input acts to predicates in the mental state, it would be sensible to introduce also the management of feasibility preconditions analogously to the linguistic tier, however, this is not considered here for it is subject of current researches.

Similarly to the case of the linguistic tier, set F is a subset of E and $V = U$. Also, sets Y and Z are the same of previous case. The rules for transition relation $\rightarrow_{\mathcal{B}}$ handling input acts are as follows:

$$\bullet \xrightarrow{\Phi_M: i \triangleright i} \mathcal{B} \bullet \quad \text{if } i \in I_{ONT} \quad \bullet \xrightarrow{\Phi_M: i \triangleright \perp} \mathcal{B} \bullet \quad \text{if } i \notin I_{ONT}$$

stating that when input act i arrives that is correct with respect to the ontology, then this is redirected as event f ; otherwise, by the second rule the act is simply ignored. The case of output acts is handled dually:

$$\begin{aligned} \bullet \xrightarrow{\Phi_M: o \triangleleft o} \mathcal{B} \bullet & \quad \text{if } o \in O_{ONT} \quad \text{and} \quad \Phi_M = \Phi'_M \circ cmd_M^P(o) \\ \bullet \xrightarrow{\Phi_M: \perp \triangleleft o} \mathcal{B} \bullet & \quad \text{if } o \notin O_{ONT} \quad \text{or} \quad \Phi_M \neq \Phi'_M \circ cmd_M^P(o) \end{aligned}$$

the output act is let flow toward the agent core only if it conforms to the ontology and satisfies the preconditions.

The Social Tier This tier deals with those peculiar aspects of an agent that characterize its collaborative (or social) behavior within the MAS. In particular, interaction laws can be applied that change the mental state under some conditions, e.g. making the agent react to some input by producing some output act. Since interaction laws can be of different kinds, we here just suppose that the set of interaction laws of an agent are modeled by a relation $ilaw \subseteq \mathcal{P}(\Phi) \times \mathcal{P}(\Phi)$, associating to the current mental state the new facts (possibly none) that the agent will believe and intend after applying some enabled law.

In particular, the social tier has void set Y , while set Z – as for previous tier – may contain some pending update, namely, some change on the mental state that

has to be applied. Events F and updates V coincide with E and U , respectively. The rule governing the relation transition of this tier is the following:

$$\perp \xrightarrow{\Phi_M:i \triangleright i} \mathcal{B} \Phi_n \quad \text{if } ilaw(\Phi_M, \Phi_n)$$

saying that whenever a new input is received, interaction laws make a new mental state to be computed which will be propagated to the interaction tier as usual.

The Internal Tier The internal tier is the latter tier of our specification, which includes a number of remaining implementation features of the agent. Most notably, here a further set of laws – which are not however published but forms the hidden, unobservable behavior of the agent – can be applied that may have various purposes. On the one hand, these rules can be used to drive any agent behavior that cannot be ascribed to its social role, but rather to its implementation. To the end of the formalization presented here, we model this behavior by a *private laws* function $plaw$ similar to function $ilaw$. On the other hand, the internal tier is also the part of the agent responsible for proactively requesting some action to be performed. To this end, we consider a precondition function $cond_M^E : o \mapsto \mathcal{P}(\Phi)$ associating to an output act the conditions enabling its execution. In the basic case we have $cond_M^E(o) = \{I_s done(o), B_s FP(o)\}$, that is, the agent should intend to execute the action and should believe its feasibility preconditions.

Since no further refining is here considered, set of events F and updates U are here void. Moreover, sets Z and Y coincide with previous tier. Other than the two usual rules for dispatching updates as in the previous tier, we have the two rules:

$$\begin{aligned} \perp \xrightarrow{\Phi_M:i \triangleright \perp} \mathcal{B} \Phi_n & \quad \text{if } plaw(\Phi_M, \Phi_n) \\ \perp \xrightarrow{\Phi_M \circ cond_M^E(o) : \tau} \mathcal{B} o & \end{aligned}$$

While the former makes the mental state be updated by applying a private law, the second rule says that each time a silent action is performed by the agent core and the precondition for executing an action is satisfied, then that action is scheduled by sending the update o .

6.2 An Example

In order to grasp the flavor of this formalization, we here consider an example of simple conversation for an agent, and describe the corresponding sequence of transitions modeling its behavior. In particular, we consider the case that the agent i receives from another agent j a message a of the kind $request(j, i, b)$ where $b = inform(i, j, \phi)$, requesting i to send a message declaring that he believes ϕ . In the case where i actually believes ϕ , and by the interaction law $I_i done(a) \leftarrow B_i I_j done(a)$ – stating that i is always willing to execute the actions that j intends to be executed – we should obtain that i sends the message $inform(i, j, \phi)$. This protocol is a very simplified version of the FIPA-request protocol, and for the

sake of brevity does not take into account aspects such as agreement, refusal, and so on.

The represented portion of the agent state, namely the state of the agent core, is expressed as a tuple with the state of each tier in its elements, orderly. We start by considering the agent i with a mental state of the kind $\Phi_M \circ \{B_i\phi\}$, namely, initially believing ϕ .

- (1) $\langle \Phi \circ \{B_i\phi\}, \bullet, \bullet, \bullet, \bullet \rangle \xrightarrow{?a}_C$
- (2) $\langle a \parallel \Phi \circ \{B_i\phi, B_i\text{done}(a)\}, \bullet, \bullet, \bullet, \bullet \rangle \xrightarrow{\tau}_C$
- (3) $\langle \Phi \circ \{B_i\phi, B_i\text{done}(a)\}, \{B_i I_j \text{done}(b)\}, \bullet, \{I_i \text{done}(b)\}, \bullet \rangle \xrightarrow{\tau}_C$
- (4) $\langle \Phi \circ \{B_i\phi, B_i\text{done}(a), B_i I_j \text{done}(b)\}, \bullet, \bullet, \{I_i \text{done}(b)\}, \bullet \rangle \xrightarrow{\tau}_C$
- (5) $\langle \Phi \circ \{B_i\phi, B_i\text{done}(a), B_i I_j \text{done}(b), I_i \text{done}(b)\}, \bullet, \bullet, \bullet, \bullet \rangle \xrightarrow{\tau}_C$
- (6) $\langle \Phi \circ \{B_i\phi, B_i\text{done}(a), B_i I_j \text{done}(b), I_i \text{done}(b)\}, \bullet, \bullet, \bullet, b \rangle \xrightarrow{\tau}_C$
- (7) $\langle b \parallel \Phi \circ \{B_i\phi, B_i\text{done}(a), B_i I_j \text{done}(b), I_i \text{done}(b)\}, \bullet, \bullet, \bullet, \bullet \rangle \xrightarrow{!b}_C$
 $\langle \Phi \circ \{B_i\phi, B_i\text{done}(a), B_i I_j \text{done}(b), I_i \text{done}(b), B_i \text{done}(b)\}, \bullet, \bullet, \bullet, \bullet \rangle$

The first transition models the reception of communicative act a : the interaction tier enqueues the request and adds to its mental state $B_i\text{done}(a)$. In the second transition, a is processed by flowing across each tier: in the linguistic tier it makes the agent believing the sender's intentions $B_i I_j \text{done}(b)$ (feasibility preconditions are here avoided for simplicity), while in the social tier the interaction law $I_i \text{done}(b) \leftarrow B_i I_j \text{done}(b)$ is applied leading to the new intention $I_i \text{done}(b)$. In the third and fourth rule both these facts are propagated back to the interaction tier affecting the mental state. In the fifth rule, the internal tier recognizes the intention $I_i \text{done}(b)$, and by means of function cmd_M^E fires the action b , which the sixth rule moves to the interaction tier. Finally, since the feasibility precondition $B_i\phi$ to b is satisfied the communicative act b is sent outside. Notice that another precondition for sendind would be $I_i B_j \phi$ (the agent i must intend the rational effect of b), which is not considered here for in most agent architectures – such as FIPA – it is generally entailed by $I_i \text{done}(b)$. Then, the spurious formulae $\{B_i I_j \text{done}(b), I_i \text{done}(b)\}$ are meant to be subsequently dropped from the mental state by means of private laws in the internal tier.

7 Conclusions

In this paper we tackle the problem of formalizing the behavior of complex agents by a technique underlying the notion of grey-box modeling. An existing description, focusing on the agent behavior at a given abstraction level, can be refined by adding the specification of a new aspect. This formal approach is here described and put to test on the ParADE framework, describing a number of relevant agent aspects such as the ACL and its semantics, ontology, and social role, providing a means by which different system levels and views of a MAS can be represented in isolation. A crucial role in this methodology is played by the framework of LTS, which allows us to describe in an operational way the behavior of an agent implementation, and facilitates the task of composing specifications.

Related Works LTS have been applied to agent systems in other works as well. The grey-box modeling approach has been first studied in the context of the so-called observation framework [17, 18], where agents are modeled as sources of information. In [15], a grey-box formal approach similar to the one described in this paper has been applied to represent the semantics of ACLs. In [16], specifications based on operational semantics are evaluated for ACLs, using a process algebra to describe how interaction may affect the mental state. In [10], agent internal aspects such as planning are modeled through the ψ -calculus, a language whose operational semantics is meant to capture the allowed internal dynamics of the agent. Another significant formalism from concurrency theory that is being successfully applied to MAS interaction specification are Colored Petri Nets [9], as described e.g. in Ferber’s book [4]. Most of these previous works applying LTS and CPN to MAS interaction modeling focused on agent conversations and interaction protocols, while we claim that the approach presented in this paper can be fruitfully applied to the whole design of agent-based systems, enjoying the fact that separating specifications may allow to study their properties separately. A deeper investigation will reveal further relationship with the work in [10], which is closer to our spirit.

Future Works A main advantage of an operational specification is that it should allow properties of interest to be formally proved. In our case, where one such specification is applied at design-time to model an agent collaborative behavior, we may expect them to state the soundness – in the broad meaning of the term – of an agent design, especially as far as its interactions with other agents of the MAS are concerned. Notice that the specification we provided is parametric in a number of mathematical structures (I_{ACL} , O_{ACL} , eff_M^I , cmd_M^O , I_{ONT} , O_{ONT} , cmd_M^P , $ilaw$, $plaw$, cmd_M^E), describing both peculiar aspects of the agent model (e.g. ACL semantics) as well as peculiar aspects of the individual agent (e.g. its private laws).

Some of the properties of interest may be independent of these parameters, mostly concerning the internal structure of the specification. For instance, by-hand proofs should emphasize that any reception of an input act a is reified as a believe $B_i done(a)$, and that requests for mental state updates raised by the internal tier are eventually applied (flowing towards the interaction tier) preserving their order.

Other properties, that instead depend on the parameters of the specification, can be more concerned with the correctness of the resulting agent behavior, that is, concerning its rationality and social attitude. As an example, our specification may provide a suitable framework for proving that under a given ACL semantics (eff_M^I , cmd_M^O , I_{ACL} , O_{ACL}) and a given ontology (cmd_M^P , I_{ONT} , O_{ONT}), a specific set of interact laws ($ilaw$) and private laws ($plaw$) are sufficient for the agent correctly participating in a given conversation protocol, e.g. the simple request conversation showed in Section 6.2.

An expected feature of our framework is that properties concerning only one aspect of the MAS design could be studied considering agent specifications only up to the corresponding tier, by virtue of the refinement notion of transition

systems. For instance, stating that a conversation protocol is consistent with respect to the ACL semantics should require us to consider agents as made of the interaction and linguistic tiers only: as mentioned in Section 4 such a specification refines the actual agent behavior, but consider all the aspects of interest. On the other hand, evaluating the evolutions of conversations depending on the agents social attitude requires to consider the domain and social tiers as well.

In general, deepening all these issues, that concern the applicability of formal verification tools to the validation of MAS design, is the main future work of this research.

References

1. F. Bergenti. A discussion of two major benefits of using agents in software development. In *Engineering Societies in the Agents World (ESAW 2002)*, volume 2577 of *LNAI*, pages 1–12. Springer-Verlag, 2003.
2. F. Bergenti and A. Poggi. A development toolkit to realize autonomous and interoperable agents. In *Conference on Autonomous Agents*, pages 632–639, 2001.
3. J. A. Bergstra, A. Ponse, and S. A. Smolka, editors. *Handbook of Process Algebra*. North-Holland, 2001.
4. J. Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, London, 1999.
5. FIPA. FIPA communicative act library specification. <http://www.fipa.org>, 2000. Doc. XC00037H.
6. R. v. Glabbeek. The linear time – branching time spectrum I. The semantics of concrete, sequential processes. In Bergstra et al. [3], chapter 1, pages 3–100.
7. K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. C. Meyer. Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4), 1999.
8. N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117:277–296, 2000.
9. K. Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use*. Springer-Verlag, Berlin, 1992.
10. D. Kinny. Vip: a visual programming language for plan execution systems. In *AAMAS 2002*, pages 721–728. ACM Press, 2002.
11. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
12. A. Newell. The knowledge level. *Artificial Intelligence*, 18:87–127, 1982.
13. E. Norling and F. E. Ritter. Embodying the jack agent architecture. In *14th Australian Joint Conference on Artificial Intelligence*, volume 2256 of *LNCS*, pages 368–377. Springer, 2001.
14. G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Department of Computer Science, Aarhus University, 1991.
15. G. Rimassa and M. Viroli. An operational framework for the semantics of agent communication languages. In *Engineering Societies in the Agents World (ESAW 2002)*, volume 2577 of *LNAI*, pages 111–125. Springer-Verlag, 2003.
16. R. M. van Eijk, F. S. de Boer, W. van der Hoek, and J.-J. C. Meyer. Operational semantics for agent communication languages. In *Issues in Agent Communication*, volume 1916 of *LNAI*, pages 80–95. Springer, 2000.
17. M. Viroli and A. Omicini. Modelling agents as observable sources. *Journal of Universal Computer Science*, 8, 2002.
18. M. Viroli and A. Omicini. Specifying agent observable behaviour. In *AAMAS 2002*, Bologna, Italy, 15–19 July 2002. ACM.