

An Operational Framework for the Semantics of Agent Communication Languages

Giovanni Rimassa¹ and Mirko Viroli²

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

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

Abstract. From an engineering perspective, the agent abstraction can be suitably exploited for tackling cooperation of heterogeneous systems, facilitating semantic interoperability of independently developed software components. But, in order to support the sound design of infrastructures for agent systems, adequate models have to be studied as to grasp the important aspects of cooperation at the desired level of abstraction.

In this paper we focus on the semantics of Agent Communication Languages (ACL), and study the approach based on the idea of describing an agent as a grey-box software component, representing its behaviour by means of transition systems. This framework provides an operational description of ACLs, considering the single-step evolution and interactive capability of an agent, which contrasts the classical frameworks based on intentional descriptions, which rely on the concept of agent mental state. Some examples are provided to show the flavours of the proposed model to describe various semantics aspects of communicative acts.

1 Motivation

Modern software systems become both increasingly large and more complex, moving towards more dynamic user interaction patterns. This has not just affected technical matters, but also the economical landscape of the Information Technology industry at large, creating a market where time between product conception and delivery shortens more and more. This pressure to release new features in shorter time raises very serious software engineering issues [12]. Individual productivity of software developers is quite hard to increase, so most organizations had to cope with the pressure by resorting to large-scale reuse politics.

As a result, most modern, feature-rich software systems are actually wrapping older, much less glamorous systems; while this approach is quite a cost effective one, it has one major drawback. As time passes and new parts are added, the conceptual integrity – that is, the property of being understandable through

a coherent and possibly limited set of concepts – fades away and the system becomes less maintainable.

1.1 Homogenizing heterogeneity: modern middleware and agent oriented infrastructures

Looking at these premises, it is hardly surprising that all the most successful software infrastructure technologies of the last ten years explicitly targeted the integration of heterogeneous, distributed software applications. All the popular frameworks for software infrastructure, such as CORBA and Web Services, share the common trait of being a specification for component boundaries and interaction patterns, considering the component internals as an implementation specific part. All these technologies are meant to ease the integration of heterogeneous software systems without sacrificing the maintainability of the overall system. To reach this goal, they all inject conceptual integrity into heterogeneous software systems through a *model* that coherently encompasses the whole system. A relevant example of this approach is the Model Driven Architecture (MDA) [14] promoted by the Object Management Group.

An important role in this trend is played by the approach of *multi-agent systems*, which sees the whole system as composed by autonomous interacting agents, and introduces a higher description level using social notions to capture the behaviour of interactive systems [11]. A major aim of multi-agent systems is to enable software integration on a deeper level, namely shifting the integration process from *syntactic interoperability* to *semantic interoperability* [3]. This means producing a model and an infrastructure where independently developed components can interact by making assumptions on each other, that are perceived by human users as if the components can actually understand one another.

A relevant issue in the design of complex, distributed software systems concerns the level of detail of a model. Two common approaches to system modeling are *white box* modeling and *black box* modeling. In white box modeling, the system is described in term of its inner workings; this approach is troublesome to use in our case, because there is no single model that can coherently describe all the parts of an heterogeneous system with all their implementation details. Black box modeling, instead, only deals with the system at its boundaries, providing a description based solely on the external system behavior. Unfortunately, this approach has problems also, because it provides only small information about the system behaviour, thus poorly supporting the system design.

Therefore, a hybrid *grey box* modeling approach is adopted: the system is described within a coherent framework as a collection of interconnected parts, but abstracting away from details below a certain abstraction level.

In [17], the authors propose to apply a grey box modeling approach to the specification and design of multi-agent systems, focusing on the description of agent interactive aspects. This approach is quite sensible, because multi-agent systems are particularly geared towards highly heterogeneous distributed environments.

1.2 Semantics, formalism and software development

In the field of agent systems, the problem of interoperability has in the last year faced the issue of formal semantics. The FIPA organization produced a comprehensive set of specifications for interoperable multi-agent systems, following a grey-box approach that describes software agents using a BDI framework [6] without actually requiring them to have an internal BDI architecture. In particular, semantics of communicative acts is formally described in terms of feasibility preconditions and rational effects, expressed through a modal language with operators for modeling intentional aspects such as beliefs and intentions [8]. Having a formal semantics is often considered a strength of FIPA ACL, and it is cited as making it superior to more traditional approaches like distributed objects and even to other agent communication languages like KQML [2].

A common supporting statement for a formal semantics focuses on verification: if a software infrastructure has a formal semantics, an external authority can check implementations for compliance. However, among all the successful software infrastructures very few of them are currently provided with a formal semantics and check compliance with automated tools. Still, they all achieve software integration and interoperability, at least to some extent. Moreover, even when both formal definitions and automated checkers are available, sometime programmers do not really seem to care – as the HTML case shows.

However, formal semantics can really be useful during the design phase. While implementation is highly coherent, being checked by automated tools and defined precisely enough to be executed, its very low abstraction results in a myriad of details that quickly overwhelm the capabilities of formal methods. On the other hand, requirements and specifications can be very abstract, but they are also seldom coherent. In fact, coming from interaction with customers through a typically iterative process, requirements and specifications present a twofold incoherence: they are often logically contradictory (incoherence among different parts of the specification), and they are quickly evolving (incoherence between the specification at different time points). Only design strikes the right balance between abstraction and coherency, so that it can be effectively improved by a formal framework. During design, the designer does not deal with the real system, but with a simplified system model that allows reasoning without impairing creativity with too much detail. Moreover, though the design is supposed to produce a system satisfying customer requirements, the design model tends to be somewhat decoupled from the specific customer needs, and is thus more robust against requirements creeping. As a result, during the design phase formal semantics can be fruitfully used to check the coherence of the model – commonly referred to as its *soundness* – and to verify its properties of interest, enabling interception of errors in the early stages of the design and better guiding to correct implementations.

This does not mean that a formal semantics is useful to every software designer; rather, almost all of them can create meaningful or even insightful soft-

ware without knowing any formal framework³. Though a formal semantics is directly useful only to a small subset of designers, its benefit can still be significant, because that small subset is exactly the one trying to cope with the hardest problems, often producing languages, tools and libraries that can be exploited by the development community at large. In the particular case of ACL semantics, which is the one here we are interested into, formal approaches may allow to precisely define the means by which an agent should send and receive messages, possibly supporting the design of the agent interface, the agent communicative acts, and negotiation protocols.

1.3 Overview

Given the background and motivation provided in this section, the remainder of the paper is organised as follows. Section 2 discusses various aspects concerning the design of agent-based systems, focusing on the desired features of a formal framework supporting it. Section 3 introduces the framework of transition systems on which our approach is based. Based on transition systems, in Section 4 we provide a framework for modeling agents, focusing on the operational description of their interactive behaviour. In Section 5 we go into details of defining ACL semantics by our framework, discussing aspects such as feasibility preconditions, rational effects, and conversational-based semantics. Finally, in Section 6 we provide concluding remarks.

2 Enabling sound design of multi-agent systems

The main goal of this paper is trying to support multi-agent systems design with a formal framework. In order to pursue this goal, two broad classes of issues need to be addressed. First, the general nature of the design process has to be taken into account, and then the peculiarities of multi-agent systems can enter the picture.

Design is invention and creation, it is all about dynamism and tension towards a goal, so, a design rule should both describe a property of the design and suggest a roadmap to achieve it. This property is referred to as generativity in Christopher Alexander's work [1]; this building architect inspired not only his fellows, but also the software patterns community. From the mid-nineties the software pattern community treasured Alexander's ideas and gathered together a body of patterns of software development, making design experience from the past readily available to younger designers. However, when the application domain or the programming paradigm is new and relatively untested, the designer cannot count on patterns for generativity (at least, not completely) because there will not be much past design experience to leverage. Moreover, without a lot of past history to feed his or her intuition, the designer will more likely appreciate a

³ It should be noted that, as a matter of fact, most designers hardly understand formal semantics and are not confident with their usages.

precise approach. Then, a formal framework aiming to support design will need the following features:

- 1 Being generative, that is combining description and dynamism into a calculus.
- 2 Focusing on fundamental issues, that can be solved once by a few skilled designers and leveraged many times from then on.

In the particular case of multi-agent systems design, the two properties above are not enough; others need to be added due to the specific intent of multi-agent systems, that is to obtain semantic interoperability. First of all, in the context of a specific application a shared ontology should be defined, describing entities from the domain of discourse so as to support a general understanding of the content of communicative acts. Then, an ACL semantics is to be defined in order to give an agreed upon representation of the motivations behind the observed speech act – e.g. the official semantics of FIPA ACL models agent communication using a BDI framework. The special emphasis put on semantic interoperability by the multi-agent systems approach suggests two more desirable properties for our formalism.

- 3 Supporting the description of autonomous software components, by allowing nondeterminism in the system evolution and opaqueness with respect to the actual internal model of each agent.
- 4 Being integrated with the basic speech act communication model, where an agent can affect its environment both by performing direct actions on it and by uttering communicative acts.

These four properties guide us to the definition of a new framework for the description of an agent behaviour, which in the end we will use as a tool for formally defining an ACL semantics.

First of all, we intend to rely on an operational specification of an agent behaviour – describing its capability of performing interactions and changing its state – instead of taking the viewpoint of an agent as an entity with a mental state. While this approach generally lowers the abstraction level, it better supports the need for being generative of property 1. We achieve this result by relying on the framework of labelled transition systems, as shown in next sections.

Then, according to property 2 we will adopt a grey-box modeling approach for agents, focusing on the fundamental issue of the agent interactions, describing the agent part meant to manage the interactions with the environment while abstracting away from its internal details. Other papers [17, 18] discusses why this grey-box approach is suitable for tackling agent specific aspects as requested by properties 3 and 4.

3 The framework of transition systems

The semantic approach we describe in this paper is based on the framework of (*labelled*) *transition systems* [9], which is widely used in the field of concurrency

theory to provide an operational semantics to process algebras [4], so as to specify the interactive behaviour of systems. A transition system over set X 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 a relation. Let $x, x' \in X$, and $act \in Act$, the occurrence of $\langle x, act, x' \rangle$ in \longrightarrow – written $x \xrightarrow{act} x'$ – means that the software component of interest may move from state x to state x' by way of action act . Once the set of actions is specified, and the structure of set X is set up the content of a transition relation \longrightarrow can be intensionally given in terms of a set of rules of the kind:

$$\frac{condition}{x \xrightarrow{act} x'}$$

In each rule, the upside part *condition* is a predicate on the free variables of the downside part: given a set of rules, $x \xrightarrow{act} x'$ is true if it holds for at least one rule and one substitution of the free variables ⁴.

Actions can be given different interpretations. In the most abstract setting, actions 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 characterised by states x_0, x_1, \dots, x_n is associated to the action 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.

Transition systems feature interesting properties that make them particularly appealing as mathematical structures for describing the behaviour of interactive systems, in which case actions are used to represent the system's interactions with the environment. Most importantly, transition systems intrinsically take into account non-determinism, which is a peculiar aspect of interactive systems, especially as far as asynchrony is concerned. For instance, in a given transition relation both the triples $\langle x, a, x' \rangle$, $\langle x, a', x'' \rangle$, and $\langle x, a', x''' \rangle$ can occur – meaning that in state x , the system may either perform action a or a' , and moreover, by performing a' it can either move to x'' or x''' . Non-determinism is generally used as a powerful abstraction tool, in that the specification of a system actually provides a number of allowed behaviours. Several remarkable concepts related to non-determinism are promoted by transition systems, such as the notions of system *observation* and *refinement* of specifications [13], which are here briefly discussed.

Since actions in a transition system can be interpreted as a view of the system transition from a state to another, it is then possible to characterise a whole system evolution in terms of aspects such as the actions permitted at each step and the actions actually executed. This characterisation is made according

⁴ When there are no conditions to be specified, i.e. *condition* = *true*, the upside part of the rule is completely avoided, simply writing $x \xrightarrow{act} x'$.

to a given *observation semantics*, associating to each system's evolution a given *observation*. For instance, according to the so-called *trace semantics* [9, 5], a system observation is simply made by one of the (possibly infinite) sequences of actions executed, also called *trace history*. By non-determinism, generally more observations are allowed for a system, so that according to the given semantics the system itself can be characterised 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, allowing for strictly fewer observations – which is the idea endorsed by traditional works such as [13, 16]. In this framework, transition systems can be generally exploited as a tool for either describing system abstract specifications and system implementations. Besides proving conformance via standard techniques such as algebraic verification [10], this framework allows us to describe system specifications and to derive system implementations from them.

4 The operational specification of an ACL semantics

The semantics of agent communication languages such as FIPA ACL is expressed by defining preconditions and effects of each performative, that is, by describing the conditions for sending and the effect of receiving communicative acts. This amounts to describe the behaviour of the agent part in charge of managing communications with the environment – which we call the *agent interface*. That agent part specifies how message reception and sending is related to the agent internal part, which FIPA ACL specification understands in terms of the notion of mental state. This technique relies on a grey-box approach, where agent internal aspects and dynamics are abstracted away, while only the part responsible for managing communication with the environment is represented.

Our goal here is to generalize this approach, by providing an abstract framework where representing the relationship between agent communications and changes to the agent internal part, the latter represented in an abstract way instead of relying on mental properties. Since this model concerns the interactive aspect of software components we naturally rely on the framework of transition systems, in contrast to the usual framework based on modal logics for the description of mental states and their evolution.

In the remainder of the paper we adopt the following syntactical notations. Given any set X , this is automatically ranged over by variable x, x', x'', x_0 , and so on – and analogously Y is ranged over by y , Z by z , etcetera. The set of multisets over X is denoted by \bar{X} , and is ranged over by variable \bar{x} , the set of sequences over X by X^* , ranged over by x^* . Union of multisets \bar{x} and \bar{x}' is denoted by $\bar{x}|\bar{x}'$, concatenation of sequences x^* and $x^{*'}$ by $x^*;x^{*'}$, void multiset and empty sequence by \bullet . Given any set X , symbol \perp_X is used to denote an exception value in set X_\perp defined as $X \cup \{\perp_X\}$. Variable x_\perp is automatically defined that ranges over set X_\perp .

4.1 A transition systems semantics for grey-boxes

Our intention here is to represent an agent in terms of a software abstraction interacting with its environment in one of two ways: (i) by a *reactive action* the agent receives a communicative act, changes its internal state, and possibly reactively sends a communicative act; (ii) by a *proactive action* it proactively changes state and then possibly sends a communicative act.

Following the grey-box approach, then, we mean to explicitly represent only the part of the agent devoted to managing these interactions with the environment, which we call the (*agent*) *core*, asbtracting away from the part dealing with agent internal issues, called the (*agent*) *internal machinery*. The agent core is modeled here through an *observable status* ranging over the set of states O . In order to model the effects of the internal machinery on the agent core, which is responsible for the execution of proactive actions, we introduce the set of (*internal*) *events* E . An internal event is to be understood as the result of a (possibly complex) computation within the internal machinery, which is meant to influence the agent interactive behaviour. On the other hand, the agent interactions with the environment may sometime influence the agent internal behaviour, so we suppose that each time an action is processed – either reactive or proactive – the agent core performs an update to the internal machinery. This is represented by a set of updates ranging over set U_{\perp} (supposed to be disjoint from E), where update \perp_U means that no change is currently significant for the internal machinery.

So, the behaviour of the agent core can be easily understood in terms of a transition system $\mathcal{O} = \langle O, \longrightarrow_{\mathcal{O}}, Act_{\mathcal{O}} \rangle$, where actions in $Act_{\mathcal{O}}$ are divided into reactive actions $Act_{\mathcal{O}}^r$ and proactive actions $Act_{\mathcal{O}}^p$, defined by $Act_{\mathcal{O}}^r ::= ic \triangleright u_{\perp} \triangleright oc_{\perp}$ and $Act_{\mathcal{O}}^p ::= e \diamond u_{\perp} \diamond oc_{\perp}$. Transitions in \mathcal{O} are then of the kind:

$$o \xrightarrow{ic \triangleright u_{\perp} \triangleright oc_{\perp}}_{\mathcal{O}} o' \quad o \xrightarrow{e \diamond u_{\perp} \diamond oc_{\perp}}_{\mathcal{O}} o'$$

The former represents the reactive action where observable state o moves to o' by receiving act ic , performing update u_{\perp} , and then sending act oc_{\perp} – the case where the latter is \perp_{Oc} means that no output act will be sent. Analogously, the latter represents the proactive action where o moves to o' by intercepting internal event e , performing update u_{\perp} , and sending act oc_{\perp} (again possibly void). So, given an ACL whose input communicative acts range in Ic and output communicative acts range in Oc (which are typically the same set), its formal semantics can be simply given by transition system \mathcal{O} , describing in an abstract way the behaviour of the agent interface.

Indeed, by non-determinism, it is possible to characterise the behaviour of an agent complying to this semantics by simply assuming the most abstract characterisation for the internal machinery, that is, supposing that at any time any internal event can be raised and any update is accepted. This leads us to the transition system $\mathcal{S} = \langle O, \longrightarrow_{\mathcal{S}}, Act_{\mathcal{S}} \rangle$, with $Act_{\mathcal{S}} = Ic_{\perp} \times Oc_{\perp}$ and $\longrightarrow_{\mathcal{S}}$

defined by rules:

$$\frac{o \xrightarrow{ic \triangleright u_{\perp} \triangleright oc_{\perp}}_{\mathcal{O}} o'}{o \xrightarrow{\langle ic, oc_{\perp} \rangle}_{\mathcal{S}} o'} \qquad \frac{o \xrightarrow{e \diamond u_{\perp} \diamond oc_{\perp}}_{\mathcal{O}} o'}{o \xrightarrow{\langle \perp_{Ic}, oc_{\perp} \rangle}_{\mathcal{S}} o'}$$

where transition $o \xrightarrow{\langle ic_{\perp}, oc_{\perp} \rangle}_{\mathcal{S}} o'$ means that the agent move from core state o to o' by receiving input ic_{\perp} (or proactively, if this is \perp_{Ic}) and by sending oc_{\perp} (or nothing, if this is \perp_{Oc}). The case where $ic_{\perp} = \perp_{Ic}$ or not distinguishes proactive actions from reactive actions, respectively.

On the other hand, an actual implementation of this specification – that is, the implementation of a compliant agent – can be defined by adding the description of the actual agent internal machinery. The internal machinery part can be specified as a component raising internal events and consuming updates, that is, as a transition system $\mathcal{I} = \langle I, \rightarrow_{\mathcal{I}}, E \cup U_{\perp} \rangle$. A transition $i \xrightarrow{e}_{\mathcal{I}} i'$ represents the internal machinery raising event e and correspondingly moving from state i to i' , whereas transition $i \xrightarrow{u_{\perp}}_{\mathcal{I}} i'$ represents the internal machinery moving from i to i' due to update u_{\perp} (here we suppose that $i \xrightarrow{\perp_U}_{\mathcal{I}} i$ always holds). So, by coupling the specification of the core \mathcal{O} – representing the ACL semantics – to the specification of the internal machinery \mathcal{I} – representing the agent peculiar aspects – one obtains the specification of a realisation \mathcal{R} of the agent. In particular this is a transition system $\mathcal{R} = \langle O \times I, \rightarrow_{\mathcal{R}}, Act_{\mathcal{S}} \rangle$, with same actions as \mathcal{S} , defined by rules:

$$\frac{o \xrightarrow{ic \triangleright u_{\perp} \triangleright oc_{\perp}}_{\mathcal{O}} o' \quad i \xrightarrow{u_{\perp}}_{\mathcal{I}} i'}{\langle o, i \rangle \xrightarrow{\langle ic, oc_{\perp} \rangle}_{\mathcal{R}} \langle o', i' \rangle} \quad \frac{i \xrightarrow{e}_{\mathcal{I}} i' \quad o \xrightarrow{e \diamond u_{\perp} \diamond oc_{\perp}}_{\mathcal{O}} o' \quad i' \xrightarrow{u_{\perp}}_{\mathcal{I}} i''}{\langle o, i \rangle \xrightarrow{\langle \perp_{Ic}, oc_{\perp} \rangle}_{\mathcal{R}} \langle o', i'' \rangle}$$

Taking as a reference observation criterion e.g. trace semantics [5], the system described by transition system \mathcal{R} has a more refined behaviour than \mathcal{S} , that is, the agent described by \mathcal{R} can be considered an actual “implementation” of the specification provided by \mathcal{S} . The following theorem proves the intended preorder relation between systems \mathcal{R} and \mathcal{S} according to trace semantics.

Theorem 1. *For any $o_0 \in O$ and $i_0 \in I$, $traces_{\mathcal{R}}(\langle o_0, i_0 \rangle) \subseteq traces_{\mathcal{S}}(o_0)$, i.e.:*

$$\{(a_0; a_1; \dots; a_n) \in Act_{\mathcal{S}}^* : \langle o_j, i_j \rangle \xrightarrow{a_j}_{\mathcal{R}} \langle o_{j+1}, i_{j+1} \rangle, 0 \leq j < n\} \subseteq \{(a_0; a_1; \dots; a_n) \in Act_{\mathcal{S}}^* : o_j \xrightarrow{a_j}_{\mathcal{S}} o_{j+1}, 0 \leq j < n\}$$

Proof sketch. Suffice it to prove that for any $o \in O$, $i \in I$, and $a \in Act_{\mathcal{S}}$:

$$\langle o, i \rangle \xrightarrow{a}_{\mathcal{R}} \langle o', i' \rangle \implies o \xrightarrow{a}_{\mathcal{S}} o'$$

which is true by definition of $\rightarrow_{\mathcal{R}}$ and $\rightarrow_{\mathcal{S}}$.

4.2 Intentional vs. operational specification

We refer to the specification of an ACL semantics in terms of transition system \mathcal{S} as an *operational specification*. This name comes from the typical usage of

transition systems as operational semantics for process algebras, and generally amounts to specifications describing the single-step execution and interaction capabilities of systems. In the context of ACLs, our specification describes the allowed communicative acts sent and received by a compliant agent, and their relations with the agent core O and with internal events and updates E, U – the two latter describing the mutual effects of interactions and internal behaviour.

In contrast to this approach, existing ACL semantics such as FIPA ACL adopt what we may call an *intentional specification*. In fact, they endorse the notion of *intentional stance* [7], describing the agent behaviour in terms of ascriptions to the system of beliefs, desires, and intentions, that is, in terms of the notion of mental state.

These two approaches are not completely different, e.g. they both rely on a grey-box modeling approach, even though they tackle complementary aspects. On the one hand, the operational specification abstracts away from the “meaning” of the represented agent status, whereas the intentional specification understands it in terms of mental properties. On the other hand, the operational specification sticks to the interactive aspects of the agent behaviour, thus describes the whole set of allowed interactive behaviours for the agent – enjoying the features of non-determinism. No hypothesis can be made on which actual behaviour will be chosen, whereas intentional specifications provide a means for trying to explain and predict agent choices, namely, by hypothesising that the agent mental state evolves in a *rational* way.

5 Communicative acts semantics

In this section we describe with some further detail how our approach can be exploited for describing the semantics of communicative acts.

5.1 Unspecified semantics

As first introductory example, we consider the trivial case where no actual specification is associated to communicative acts, so that no relevant information can be actually inferred by the agent interaction history. We suppose that reception of communicative acts and their processing by the internal machinery are decoupled. We define $O = Ic^*$: at any time, the observable status only contains the sequence of input acts received but not yet processed (namely, the queue of incoming requests). Internal events are of two kinds: (i) *get* represents a request for the internal machinery to consume the next input act, and (ii) *send(oc)*, represents the internal machinery producing the output act *oc* to be sent. Updates are terms *prc(ic)*, communicating to the internal machinery the new act to be processed.

So, the operational semantics of our ACL is simply defined by the three rules:

$$ic^* \xrightarrow{ic \triangleright \perp U \triangleright \perp oc} ic; ic^* \quad ic^*; ic \xrightarrow{get \diamond prc(ic) \diamond \perp oc} ic^* \quad ic^* \xrightarrow{send(oc) \diamond \perp U \diamond oc} ic^*$$

They respectively represents how (i) an incoming act is put in the queue, (ii) event *get* causes the first message of the queue to be processed, and (iii) event *send(oc)* causes *oc* to be sent out.

5.2 Modeling feasibility preconditions

In FIPA ACL semantics, some FPs and REs are associated to any communicative act. However, REs are not mandatory for the receiver: they can be just used by the sender so as to hypothesise the effect of sending a message, which is used to generally figure out plans to achieve goals. On the other hand, an agent can emit an act only if its feasibility preconditions are satisfied. We model this behaviour by considering $O = Ic^* \times \overline{Oc} \times M$ as a triple respectively containing the queue of input acts to be processed, the (multi-)set of output acts whose FPs are currently satisfied, and the state M based on which FPs are calculated (e.g. the mental state in FIPA ACL semantics). We suppose function $\mathbf{fp} \in M \mapsto \overline{Oc}$ takes elements in M and yields the set of allowed output acts. Internal events are now of three kinds: besides events *get* and *send(oc)* as in previous case, event *move(m')* represents the state M of the agent changing to new value m' – modeling e.g. a belief update. Then, we also add to $\mathit{prc}(ic)$ the new kind of update $\mathit{fea}(\overline{oc})$, used to communicate to the internal machinery the output acts whose feasibility preconditions are satisfied.

Operational semantics is built so that function \mathbf{fp} is computed each time component M changes, that is:

$$\begin{aligned} \langle ic^*, \overline{oc}, m \rangle &\xrightarrow{ic \triangleright \perp_U \triangleright \perp_{oc}}_O \langle ic; ic^*, \overline{oc}, m \rangle \\ \langle ic^*; ic, \overline{oc}, m \rangle &\xrightarrow{get \diamond \mathit{prc}(ic) \diamond \perp_{oc}}_O \langle ic^*, \overline{oc}, m \rangle \\ \langle ic^*, \overline{oc}, m \rangle &\xrightarrow{move(m') \diamond \mathit{fea}(\mathbf{fp}(m')) \diamond \perp_{oc}}_O \langle ic^*, \mathbf{fp}(m'), m' \rangle \\ \langle ic^*, oc | \overline{oc}, m \rangle &\xrightarrow{send(oc) \diamond \perp_U \diamond oc}_O \langle ic^*, oc | \overline{oc}, m \rangle \end{aligned}$$

The former two rules are analogous to the previous case. The third rule represents the agent proactively updating its state M , by which feasibility preconditions are recomputed affecting the set of admissible output acts. The fourth rule describes the agent proactively sending one of the output events whose feasibility preconditions are satisfied.

5.3 On rational effects

Whereas FIPA ACL semantics promotes full autonomy for agents, and so it mandates no REs to their communicative acts, here it is interesting to consider also the case where an agent processing a message cause some REs to be applied. First of all, a function $\mathbf{re} \in M \times Ic \mapsto M$ has to be defined that takes the current state M and the input act to be processed, and yields the new state M after applying the act's REs. The proactive action corresponding to the processing of a new input act is then of the kind:

$$\langle ic^*; ic, \overline{oc}, m \rangle \xrightarrow{get \diamond \mathit{prc}(ic) \diamond \perp_{oc}}_O \langle ic^*, \overline{oc}, \mathbf{re}(m, ic) \rangle$$

both consuming act *ic* and applying its rational effect to state *M*.

5.4 Conversation-based Semantics

As discussed in [15], one of the main drawbacks of current semantics of FIPA ACL is its lack of the concept of conversation protocol. Indeed, agents can co-operate by exploiting well-defined interaction protocols, where a statically fixed sequence of communicative acts has to be realised in order to achieve the intended aim – for instance, negotiating an auction. Whereas FIPA ACL includes protocol-oriented performatives, the burden of setting up the proper interactions protocols is left unspecified: the corresponding management is meant to be conceptually located within the internal machinery.

In our framework, we can add the notion of a conversation to the semantics of communicative acts, by taking it into account in the specification of an agent core. Formal details are omitted here for brevity, whereas only the basic intuition about the idea is provided.

An interaction protocol can be considered as a sequence of communicative acts; at a given time an agent can participate to a number of such protocols. A conversation is an instance of a protocol, keeping track of the communicative acts already occurred: the agent core can be designed so as to store all the information about the conversations the agent is participating on. As a result, the requirements an agent has to satisfy in order to participate to a protocol – in terms of the allowed sequence of communicative acts – can be directly represented in the agent core. By moving this specification from the agent internal machinery to agent core, the task of ensuring compliance of a conversation with respect to a protocol is generally facilitated.

5.5 From specification to implementation

The specification of an ACL semantics provided by our framework enjoy a very appealing property: it allows the automatic generation of a wrapper agent complying with the semantics.⁵ This wrapper has eventually to be completed with an implementation of the actual agent internal behaviour, which can be built using any architecture and implementation technique. For instance, the wrapper may actually hides a an internal BDI machinery, constraining its deliberation process.

Our framework can be fruitfully applied also when the internal behaviour is not implemented by the BDI framework, but for instance through a simple active object implementing a daemon agent. In spite of its simplicity, our wrapper would make this agent cooperate with existing, possibly smart agents in a compliant way with respect to the ACL semantics, thus enjoying the services these agents may be able to offer. In the case of legacy systems, on the other hand, the compliant wrapper may be used as a proper interface for the cooperation with

⁵ Notice that transition system semantics are well recognised as suitable tools for directly leading to implementations, in that their specification is operational.

other agents of the MAS, while only an intermediate part remains to be designed to fill the gap between the wrapper and the legacy system.

6 Conclusions

In this paper we study an approach to the semantics of ACLs based on transitions systems, which we refer to as the operational specification of ACLs. By our framework, we mean to strengthen the cooperation of agents built over different actual architectures: in particular, our aim here is to allow simple agents of a MAS to benefit from the cooperation with smart agents – e.g. implementing intelligent behaviour through a BDI framework. In particular, our approach focuses on the interactive behaviour of agents and abstracts away from the concept of mental state on which most existing ACLs are based

The framework provided here is a generalisation of the approach presented in [18], which is based on the *observation framework* for agents [17]. There, instead of describing the issue of ACL semantics in general as developed here, a particular abstract architecture is provided based on an ontology for the *observation* issue. This abstract architecture is claimed to be more widely applicable to different agent actual architectures, because it captures the very notion of agent interaction with the environment, and the idea of an agent allowing its status to be observed and altered from other entities. On the other hand, the generalised framework presented here simply sticks to idea of grey-box specifications for agents, so it better allows to foster comparison with existing ACL semantics.

One of the main future works of this research is trying to adopt the typical proof techniques for process algebras – which exploit transition systems semantics – to the domain of ACLs, providing relevant tools by which designers can enforce semantics interoperability in multi-agent systems.

References

1. Christopher Alexander. *The Timeless Way of Building*. Oxford University Press, 1979.
2. ARPA Knowledge Sharing Initiative. Specification of the KQML agent-communication language. ARPA Knowledge Sharing Initiative, External Interfaces Working Group, July 1993, July 1993.
3. Federico Bergenti. On agentware: Ruminations on why we should use agents. Technical report, AOT Lab - DII - Parma University, 2002.
4. Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors. *Handbook of Process Algebra*. North-Holland, 2001.
5. Manfred Broy and Ernst-Rüdiger Olderog. Trace-oriented models of concurrency. In Bergstra et al. [4], chapter 2, pages 101–195.
6. Philip R. Cohen and H.J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(2-3):213–361, 1990.
7. Daniel Dennett. *The Intentional Stance*. Bradford Books/MIT Press, Cambridge, MA, 1987.

8. FIPA. FIPA communicative act library specification. <http://www.fipa.org>, 2000. Doc. XC00037H.
9. R.J. van Glabbeek. The linear time – branching time spectrum I. The semantics of concrete, sequential processes. In Bergstra et al. [4], chapter 1, pages 3–100.
10. J.F. Groote and M.A. Reniers. Algebraic process verification. In Bergstra et al. [4], chapter 17, pages 1151–1208.
11. N. R. Jennings and J. R. Campos. Towards a social level characterisation of socially responsible agents. *IEEE Proceedings on Software Engineering*, 144(1), 1997.
12. Steve McConnell, Donald Reifer, John Favaro, and Martin Fowler. Engineering internet software. *IEEE Software*, 19(2), 2002.
13. Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.
14. OMG. Model Driven Architecture. <http://www.omg.org/mda>, 2002.
15. Jeremy Pitt and Ebrahim Mamdani. A protocol-based semantics for an agent communication language. In Thomas Dean, editor, *16th Intl. Joint Conf. on Artificial Intelligence (IJCAI '99)*, pages 486–491, Stockholm, Sweden, 1999. Morgan Kaufmann.
16. A. W. Roscoe Stephen D. Brookes, C. A. R. Hoare. A theory of communicating sequential processes. *Communications of the ACM*, 31(3):560–599, June 1984.
17. Mirko Viroli and Andrea Omicini. Modelling agents as observable sources. *Journal of Universal Computer Science*, 8, 2002.
18. Mirko Viroli and Andrea Omicini. Towards an alternative semantics for FIPA ACL. In Robert Trappl, editor, *Cybernetics and Systems 2002*, volume 2, pages 689–694, Vienna, Austria, 2002. Austrian Society for Cybernetic Studies. 16th European Meeting on Cybernetics and System Research (EMCSR 2002), 2–5 April 2002, Vienna, Austria, Proceedings.