Journal of Universal Computer Science, vol. 8, no. 4 (2002), 423-452
submitted: 17/10/01, accepted: 6/3/02, appeared: 28/4/02 © *J.UCS*

Modelling Agents as Observable Sources

Mirko Viroli
DEIS, Università di Bologna
Via Rasi e Spinelli 176, 47023 Cesena (FC), Italy
mailto:mviroli@deis.unibo.it

Andrea Omicini
DEIS, Università di Bologna
Via Rasi e Spinelli 176, 47023 Cesena (FC), Italy
mailto:aomicini@deis.unibo.it

Abstract: Observation is a fundamental interaction pattern in today's computer-based systems. Adopting observation as the main modelling criterion, computer-based systems can be represented as composed by three class of entities: *observers*, *observables* (or *sources*), and *coordinators*, that is, the entities managing the observer/source interaction.

Also, *agents* and agent *societies* are fundamental abstractions in modelling today's complex systems. When exploiting observation in the context of agent-based systems, the most natural interpretation for agents is to see them as either observers or coordinators. However, their situatedness and autonomy, their peculiar perception and representation of the environment, and their typical ability to infer new knowledge – in short, their *individual viewpoint* over the world –, make agents suitable for an interpretation as observable sources.

Accordingly, this paper discusses the implications of using observation to model agent systems, and focuses on the interpretation of agents as observables. A formal framework is developed where multiagent systems are modelled as the composition of agents interacting by observing each other and by mutually affecting their observable behaviour.

Key Words: multi-agent systems, coordination patterns, formal models
Category: D.2.1, F.1.2, F.3.1, F.4.3, I.2.11

1 Observation, Computer Systems, and Agents

The impetuous development of information technologies is rapidly changing the computer science scenario. On the one hand, the increasing complexity of computers and computer applications is making impractical or even impossible to completely model their behaviour. On the other hand, computer systems are typically built as aggregations of components, often knowable only in terms of the services they ask for and offer – in other words, through their observable properties. Notions as *interface* and *encapsulation*, paradigms like the object-oriented and the component-based ones, have promoted *observation* to a first-class issue, by implicitly stating that complexity of computer systems could be handled only by abstracting away from the essence (in a broad sense) of the elements (like objects, or components), and focusing instead on their observable behaviour.

Agent-based technologies – one of the most promising approaches for the engineering of complex systems – are contributing to this trade-off as well. In multiagent systems (MAS henceforth), a multiplicity of autonomous, possibly intelligent agents have to set up well-constructed interaction patterns to deal with heterogeneous, distributed, and unpredictable knowledge sources – making observation at the same time a relevant and complex issue in agent-based systems. Even more, it is often the case that agents cannot be known but in terms of their *interaction history* [Wegner, 1997] – for instance, because they are used to wrap practically unknowable legacy systems, or because they are inherently unknowable, as in the case of open systems. Thus, the precise characterisation of the observable behaviour of an agent, and the study of how this can be affected by the external environment, seem necessary efforts for harnessing the intrinsic complexity of current and forthcoming agent-based systems.

In this paper, observation is applied as the fundamental criterion to model the behaviour of agents and MAS. From this viewpoint, it is natural to view agents as *observers* of information sources, – e.g. information agents –, or as *coordinators* managing the patterns of interaction among observers and sources – e.g. broker agents. Instead, this paper elaborates on the idea of representing an agent as an observable *source*, that is, an entity manifesting its knowledge or competence through its observable behaviour. In order to cope with agents' partial knowability, and with the typical need of abstracting away from their inner details, our approach is based on the ideas of (i) modelling only the *observation core* of a source – that is, the part of an agent that directly affects its observable behaviour –, and (ii) explicitly representing how the agent's observable core determines its observable manifestations, and how, conversely, the core is affected by the agent's interactions.

Based on this simple ontology, this paper develops the formal framework for observation in computer systems introduced in [Viroli *et al.*, 2001], and applies it to agent-based systems. In particular, a calculus is presented where MAS are seen as the composition of agents interpreted as observable sources, and multiagent systems' evolution is represented in terms of the source agents exchanging their individual knowledge and competence over time. This model being grown on top of a precisely-defined ontology, an alternative and original viewpoint is given over well-known features of agent-based systems, based on concepts related to observation.

1.1 Modelling Styles for Software Components

Several modelling styles can be adopted for interacting software components, which differ for their different degrees of completeness in the description of how components' interactions affect and are affected by their inner status.

In the *white-box* approach, the component's behaviour is completely modelled. Typically this is done by providing an operational semantics for the component which completely defines its dynamics and its interactions with the environment. However, in a wide set of domains – agent-based systems being a relevant case – this approach is unlikely to be effective, both because the software component is intrinsically too complex to be fully characterised, and also because – independently from its complexity – its actual behaviour is not completely known, as in the case of legacy systems.

Typically, this inadequacy is addressed by exploiting a *black-box* approach, where the component's behaviour is characterised only by means of its admissible interaction histories – that is, by the sequences of input and output communication acts the component exhibits while interacting with its environment. The most notable example of this approach, is the one underlying the observational equivalence issue [Milner, 1989]. This is the standard framework exploited in the field of process algebras [Bergstra *et al.*, 2001] – by languages such as CCS [Milner, 1989] and π-calculus [Milner, 1999] –, for stating whether two interactive behaviours are equivalent, or also, when one is a subcase of the other. This approach promotes the idea of characterising a software component only by its interactive behaviour, while abstracting away from any detail concerning its actual inner machinery. While this framework provides a basic and valuable foundation for reasoning about interactive systems in general, and for stating properties of interest about communication protocols, it is not completely satisfactory for modelling software components of any kind. As for the white-box approach, agents are a case where the black-box approach, too, seems not to succeed.

Agents are intrinsically stateful software entities. The ways they interact with the environment are not fixed once and for all, but tightly depends on their evolution over time, as well as on the decisions they autonomously take, reflecting internal deliberation, planning, and inference activities – namely, their proactive behaviour. Even more, very often the agent's communication acts cannot be simply expressed in terms of events occurring in it. For instance, an agent receiving a request for executing a given action never guarantees that action to be actually executed, for the agent can simply ignore the request.

So, in between the black-box and the white-box approaches, a *grey-box* approach is taken as an underlying model for the framework developed in this paper [Viroli and Omicini, 2002c], which seems a suitable approach for representing agents' behaviour. This is based on the idea of partitioning an agent's internal aspects in two parts: one which is completely modelled (the *white* part), as it is responsible for the interactions with the environment, and the other (the *black* part) – representing internal deliberation process – that is abstracted away, but is anyway taken into account as far as it affects the former part. Correspondingly, in this paper we refer to the *agent's observable behaviour* as the dynamics of its interactions along with the dynamics of its modelled part's status, which are conceptually driven by both the external environment and by the agent's hidden status dynamics.

In order to apply the grey-box approach, however, it is necessary to provide an ontology for defining: *(i)* what agent's aspects are better to be included either in the modelled or in the hidden part, *(ii)* what is the model for the dynamics of the modelled part, *(iii)* how the modelled part's dynamics are influenced by and can affect the interactions with the environment, and *(iv)* how the modelled part's dynamics are influenced by and can affect the hidden part's dynamics. As argued in [Viroli and Omicini, 2002b], choosing such an ontology – or equivalently, defining an agent abstract machine on which the actual model is based – is a definitely crucial issue, in particular as far as wide applicability to various agents' implementation is concerned. To this end, in this paper a grey-box model for agents is defined which is grown on an a general ontology for the *observation issue* in computer systems developed in [Viroli *et al.*, 2001]. Such an ontology is focussed on representing how software components provide facilities for allowing their status and its dynamic to be perceived and possibly affected by external entities. As a result, the corresponding model turns out to be fairly abstract so as to provide a reasonable architecture-independent specification tool for agent-based systems [Viroli and Omicini, 2002b].

1.2 Observation in Computer Systems

The ontology for observation introduced in [Viroli *et al.*, 2001] interprets computer systems as made of three kinds of entities: observable *sources*, *observers* and *coordinators*. A source is a component storing some knowledge and/or competence, and intrinsically able to *manifest* a part of its state – or of its dynamics – by delivering chunks of information in the form of asynchronous messages, called *manifestations*. Observers are those entities receiving these messages, namely, *observing* the source manifestation. Finally, coordinators are entities managing the pattern of interaction between observers and sources. This is done by *conditioning* the source's observable behaviour, that is, by interacting with a source so as to affect the dynamics of its manifestations.

Roughly speaking, according to our ontology, the basic observation pattern can be sketched as follows: a coordinator conditions a source so as to affect its observable behaviour, the source elaborates the conditioning and manifest part of its knowledge and competence, then an observer receives the source's manifestation(s). Since coordinators and observers are more or less the ends of this observation pattern, they can be easily modelled as black boxes, represented in terms of their outputs – coordinators producing conditionings – and inputs – observers receiving source manifestations.

Instead, a source evidently constitutes the core of the observation pattern, and its inner state and dynamics are worth to be represented explicitly – at least partially, that is, as far as they affect the way in which a source participates in different kinds of observation patterns. As a result, the ontology focuses on characterising the dynamics of the source's manifestations, how manifestations are related to the state of the source, and how they can be manipulated through conditionings. To this end, the source is modelled in terms of its *(observation) core*, that is, the part of the source that determines its observable behaviour over time. At any time during its evolution, the core of a source is said to be in a given *(observable) position*. In turn, a position consists of the *(observable) place*, representing the part of the source's current state that can in any way affect the source's observable behaviour, and the *(observation) configuration*, determining its dynamics and manifestations over time.

At any time during its evolution, a source is either in *equilibrium* or in *motion*. When in equilibrium, a source is characterised by its current position. The equilibrium of a source can be perturbed in two ways: by a *conditioning* from a coordinator, or by a spontaneous *move* of the source. On the one hand, a source can allow a coordinator to change the source's configuration, so as to determine new interaction patterns between the source and the observers. On the other hand, a source can spontaneously change its place: this is meant to model either an interaction with the external environment that is not desirable (or useful) to interpret in terms of observation, or an internally-triggered event affecting the source's core, such as the tick of an internal clock, or the inference of new knowledge resulting from an internal process.

When in motion, the source's configuration determines the observation *trajectory* of the source, that is, how the source produces manifestations to observers, and changes its position. At the end of the trajectory – that is, when the source's configuration requires no more manifestations or changes to the source's core –, a source goes back to equilibrium.

1.3 Observation in Agent Systems

The most natural approach to modelling agents' interactions through the observation pattern is to view agents as observers of information sources. In fact, knowledge sources often populate the environment of a MAS, where agents play the role of information agents in charge of obtaining information from sources, and possibly elaborating it. On the other hand, agents are likely to be modelled as coordinators as well. Typically, agents (and in particular, intelligent ones) have some awareness of their context, and often feature some ability to set up the agent-environment interaction patterns that better favour the accomplishment of their goals.

While it is a quite natural viewpoint to interpret an agent as either an observer or a coordinator, this paper takes a seemingly less obvious stance, and focuses on interpreting agents as sources. First of all, this interpretation conceptually matches the idea that each single agent, with its own knowledge, competence, and unrepeatable "experience" – in a broad acceptance of the term –, provides a unique individual viewpoint over the world. Then, within a MAS, agents not only aim at individually pursuing their goals, but often manifest some kind of cooperative attitude, too, being ready to help other agents pursuing their own goals, and, more generally, wishing to contribute to the accomplishment of the system's global goal. Accordingly, the social ability of an agent can be represented in terms of the knowledge, competence, resources, and abilities made available to other agents. While preserving agent autonomy, this feature, too, makes agents good candidates for being modelled as observable sources.

The way our ontology characterises the source's internal details and dynamics turns out to fit the typical agent's properties. So, the core represents the part of an agent that can determine its observable behaviour as perceived by the environment – in general, this is only a limited and well-defined portion of an agent, or rather some abstract view of it, thus satisfying the typical need of abstraction that agent-based approaches require.

The spontaneous change of a source's core can be used to take into account the agent's intrinsic proactiveness, with no need to model the details related to most of its internal aspects. Agents are typically seen as independent *loci* of control, driven by the aim of pursuing their goals. However, to our end it is not relevant to explicitly model the detail of control – which is often too complex to be fully characterised –, but rather to account for how this can affect the agent's observable behaviour. As a result, our ontology represents the agent's inner modifications that cause some manifestations as spontaneous moves, that is, as unpredictable changes of its core.

The configuration of a source models how the dynamics of an agent is manifested. For instance, when a spontaneous move occurs, the current state of the configuration determines what manifestations should be produced and what is the next equilibrium state of the agent. Also, a cooperative agent could be designed so as to deliberately allow the environment to partially modify its configuration – namely, to condition the agent – so as to affect its observable behaviour over time, and set up the desired patterns of interaction. For instance, this is how our ontology accounts for requests coming from other agents, (i) asking for some service or information, (ii) being answered by some manifestation, and also (iii) possibly causing a change in the agent status keeping track of the interaction that occurred.

As a whole result, modelling an agent as an observable source seems quite appropriate for taking into account its relevant aspects, features, and behaviour.

So, this ontology is taken as a reference for applying a grey-box modelling approach to agents. This choice makes it possible to concentrate on representing the agent's part directly affecting they way it interacts with the environment – as well as taking into account the definitely relevant aspect of an agent's proactive behaviour –, while abstracting away from details concerning agent's deliberation process – that are very difficult to model whereas they only marginally affect the dynamics of MAS.

1.4 Overview

The main goal of this paper is to develop a formal framework for modelling the behaviour of a MAS, based on the idea of representing each agent as an observable source according to the ontology for observation developed in [Viroli *et al.*, 2001]. In order to obtain this result, we conceptually divide the framework in three parts, each focussing on a different level of abstraction in specifying the overall MAS dynamics.

In Section 2, the agent's inner dynamics is studied. Based on the formal framework introduced in [Viroli *et al.*, 2001] – which is used to specify how software components let their status and its dynamics to be observed – here an operational semantics for the agents inner status is developed. The agent is interpreted in terms of an observable source which can either *(i)* receive a conditioning, *(ii)* perform a spontaneous move, *(iii)* evaluate its configuration, and *(iv)* produce manifestations.

Section 3 is devoted to specify the agent's interactive behaviour. In particular, the four events characterising the agent's inner behaviour are related to the actual interactions of the agent with its environment. In this model, the agent is seen as an interactive component that either *(i)* proactively sends output messages to the environment (which are manifestations fired by spontaneous moves), or *(ii)* receives a message from the environment and reactively replies with one or more output messages (which are manifestations fired by a conditioning).

Finally, Section 4 addresses the agents cooperation level. A MAS is seen as a composition of agents interacting with each other. In particular, the MAS dynamics is described in terms of an agent's manifestations eventually becoming conditionings for another agent. On the one hand, *(i)* an agent can spontaneously produce output messages, on the other hand, *(ii)* an agent can accept one message as input and correspondingly produce new output messages for other agents.

A simple application example is provided and developed according to the corresponding abstraction level throughout Sections 2,3, and 4. Such an example is intentionally simple yet complete enough to help understanding the formalism and the most relevant related concepts and applications.

Section 5 is devoted at giving an interpretation of the typical features of agents and MAS, such as autonomy and social ability, in terms to aspects related to the observation ontology, and according to the formal framework presented. In turn, this is meant to provide further insights on our model, as well as alternative and innovative viewpoints over well-established concepts.

Section 6 concludes by discussing related works and directions for future studies in this research direction.

2 The Formal Framework for Observation

The ontology for observation sketched in the previous section provides the conceptual basis for the uniform description, classification, and comparison of the seemingly different approaches supporting observation as they emerge independently from computer science and artificial intelligence [Viroli *et al.*, 2001]. In the following, a formal framework for observation is defined, grounded on the observation ontology. In particular, this is meant to provide a tool to denote and specify the observable behaviour of a source, focusing on the way in which interaction may influence the source's manifestation. Also, this formal framework provides the basis for the agents' calculus presented in Section 3 and Section 4.

2.1 Formal Background

Throughout this paper, variables written in uppercase letters denote sets, and variables written in lowercase letters denote the corresponding elements. So, let X be any set, variable x and its variations $(x', x'', \ldots, x_1, x_2, \ldots)$ range over X. Sometimes the content of a set is specified in terms of a grammar, that is, using a BNF-like production. In particular, in its right-hand side the occurrences of a variable x are meant to denote that any element of the set X can be used in their place. The left-hand side specifies instead the name of the set whose elements are the strings generated by the grammar production. For instance, the semantics of the expression:

$$X ::= \texttt{elem}(y) \mid \texttt{zero}$$

where $Y = \{1, 2, 3\}$, is to define the set X as the set:

$$\{\texttt{zero}, \texttt{elem}(1), \texttt{elem}(2), \texttt{elem}(3)\}$$

Also, the right-hand side is said to be the language of X, or the language defining X.

Given any set X, the special symbol $\perp_X$ is used to denote an exception value in the set $X_\perp$ defined as $X \cup \{\perp_X\}$. The set of multisets over X is denoted by $\overline{X}$, and its elements (which are multisets over X) by the variable $\overline{x}$ and its variations ($\overline{x}', \overline{x}'', ..$). The content of a multiset can be specified by enumerating its elements through the symbol $\{\}_M$ – e.g. writing $\overline{x} = \{x', x', x'', x''', ...\}_M$ – or by the union of two multisets through the binary operator $|$, as in $\overline{x} = \overline{x}'|\overline{x}''$. The void multiset is denoted by ϵ. For any $\overline{x}$, $\overline{x}'$ and $\overline{x}''$, the following equivalences are supposed to hold:

$$\overline{x}|\overline{x}' = \overline{x}'|\overline{x} \qquad \overline{x}|\epsilon = \overline{x} \qquad (\overline{x}|\overline{x}')|\overline{x}'' = \overline{x}|(\overline{x}'|\overline{x}'')$$

With an abuse of notation, sometimes x denotes a singleton element either in a multiset $\{x\}_M$ or in a set $\{x\}$. So, the expression $x|\overline{x}$ denotes a multiset containing the element x. Furthermore, we also suppose $\perp_X |\overline{x} = \overline{x}$ to hold. The symbol $\setminus$ is used for the difference between either sets and multisets: so, for instance, $(x|\overline{x}) \setminus \overline{x} = x$.

A finite sequence of elements $x, y, z, \ldots$ is denoted by the symbol $\langle x, y, z, \ldots\rangle$ and considered as an element of the cartesian product $X \times Y \times Z \times \ldots$. The set of functions from subsets of X to subsets of Y is denoted by $X \mapsto Y$. A function $\mathit{fun} \in X \mapsto Y$ can be specified using the notation $\{x' \mapsto y', x'' \mapsto y'', ..\}$. Often, when applying a function to a finite sequence, we avoid the bracket notation, writing $\mathit{fun}(x, x')$ instead of $\mathit{fun}(\langle x, x'\rangle)$. The function obtained by updating f so as to map the element x into y is denoted by $f[x \mapsto y]$.

In this paper, the formal framework of *labelled transition systems* is adopted as a means for describing the behaviour of systems – for a more exhaustive introduction see [Glabbeek, 2001]. Formally, a (labelled) transition system over a set of *processes* X, is a pair $\langle X, \longrightarrow\rangle$ where $\longrightarrow \subseteq X \times \mathit{Act} \times X$ is a ternary relation, associating processes to other processes through *actions*. In particular, the presence of a certain triplet $\langle x, \mathit{act}, x'\rangle$ in $\longrightarrow$, also denoted by the syntax $x \xrightarrow{\mathit{act}} x'$, is meant to represent the process x moving to x' due to the transition characterised by the action act.

Typically, processes are used to model the state of a certain computational activity, and transitions are used to model changes on that state. Actions characterise the kind of transition, modelling either an interaction of the computational activity with the environment, or simply a spontaneous change of the activity's state – typically modelled by the *silent* action τ, which in this paper is taken as default when none is specified.

The content of $\longrightarrow$, that is, the semantics of the transition system, is then given by rules of the kind

$$\frac{\mathit{cond}}{x \xrightarrow{\mathit{act}} x'}$$

and is defined as the least relation satisfying all the rules – in particular, when the formula *cond* is true, the triplet $\langle x, act, x' \rangle$ is considered as an element of $\longrightarrow$.[1] In order to simplify the understanding of the rules, actions are generally divided in sorts, each identified by a different symbol or character tagging the transition, such as S in $\xrightarrow{act}_S$, which has same semantics of $\xrightarrow{\langle S, act \rangle}$. As a result, often labelled transition systems are defined by means of a set of labelled transition systems, each identified by a different tag – as in the case of the transition system for sources defined in the following subsection.

2.2 The Core Calculus for Sources

A class of *source* systems is defined as a quartet $\langle P, C, M, J \rangle$. P is the set of the admissible *(observable) places* for a source system of the class. A place models at any time the part of the current state of a source that can influence its observation behaviour. C is the set of the admissible *configuration atoms*, or *c-atoms* for short, determining the dynamics of a source and its manifestation. In fact, the *(observation) configuration* of a source is defined as a multiset of c-atoms. The pair $\langle \text{place}, \text{configuration} \rangle$ determines the current *position* of the source's core. More precisely, the set of the admissible positions of a source is $P \times \overline{C}$, and the position of a source is represented by a pair $\langle p, \overline{c} \rangle \in P \times \overline{C}$, which is denoted through the syntax $p\,[\,\overline{c}\,]$. So, the state of a source in equilibrium is fully characterised by its current position $p\,[\,\overline{c}\,]$.

A *move* of the source's core is defined as a change in the source's place, denoted by elements $\langle p, p' \rangle$ of the set $P \times P$. Furthermore, $P \times P \times \overline{C}$ is defined as the set of the admissible *motions* of the source, keeping track of the move and of the current configuration. A generic motion $\langle p, p', \overline{c} \rangle$ is denoted by the syntax $\langle p, p' \rangle [\,\overline{c}\,]$, where p is the source's previous place, and $p'\,[\,\overline{c}\,]$ the source's new position – or, in other terms, where $\langle p, p' \rangle$ is the source's move, and $\overline{c}$ its new configuration. So, the state of a source in motion is fully characterised by its current motion $\langle p, p' \rangle [\,\overline{c}\,]$.

The third element of a source class' specification, M, is the set of messages that a source of the class can produce. The syntax for messages is not defined in general: instead, each class of sources specifies its own.

Finally, $J = \langle \mathit{eval}, \mathit{select} \rangle$ is a pair of functions giving semantics to c-atoms. The *evaluation function* (*eval*) models the effect of a c-atom on both the source's core and manifestation. The *selection function* (*select*) determines which c-atom in the current source's configuration should be evaluated at a given time. In particular,

$$\begin{aligned} \mathit{eval} &\in C \mapsto (P \times P \mapsto C_\perp \times P_\perp \times \overline{M}) \\ \mathit{select} &\in \overline{C} \mapsto (P \times P \mapsto C_\perp) \end{aligned}$$

[1] More precisely, in case $\langle x, act, x' \rangle$ contains free variables, if the bound triplet $\langle \hat{x}, \hat{act}, \hat{x}' \rangle$ can be obtained from $\langle x, act, x' \rangle$ with variable substitution σ – i.e., $\langle x, act, x' \rangle\,\sigma = \langle \hat{x}, \hat{act}, \hat{x}' \rangle$ –, then $\langle \hat{x}, \hat{act}, \hat{x}' \rangle$ is an element of $\longrightarrow$ if $\mathit{cond}\,\sigma$ holds.

Given a source's motion $\langle p, p' \rangle[\overline{c}]$, the evaluation $eval(c)(p, p')$ of the c-atom $c \in \overline{c}$ produces a triplet $\langle c', p'', \overline{m} \rangle = eval(c)(p, p')$, to be interpreted as follows:

- c' is the c-atom to be written in the configuration $\overline{c}$ replacing c. $c' = \perp_C$ means that no c-atom replaces c, i.e., c is simply dropped from the configuration $\overline{c}$ after its evaluation.
- p'' is the new place of the source after the evaluation of the current c-atom. In the case that $p'' = \perp_P$, the source's place does not change further, that is, $p' = p''$. This is supported by the operator $\langle p, p' \rangle :> p''$, accepting the move $\langle p, p' \rangle$ and the new place p'', and yielding the move defined as:

$$\langle p, p' \rangle :> p'' = \begin{cases} \langle p', p'' \rangle & \text{if} \quad p'' \neq \perp_P \\ \langle p, p' \rangle & \quad\text{otherwise} \end{cases}$$

- $\overline{m}$ is the multiset of messages to be sent out by the source due to the evaluation of the current c-atom.

Given the configuration $\overline{c}$ and the move $\langle p, p' \rangle$, $select(\overline{c})(p, p')$ returns the c-atom to be evaluated – or $\perp_C$, meaning that no c-atoms should be evaluated and the source can return to equilibrium.

The selection and evaluation of the c-atoms in a source's configuration determine the *trajectory* of the source – in terms of core modifications and observable manifestations – after it has been stimulated by either the registration of a c-atom or a spontaneous move. This dynamics is described by means of a labelled transition system with four kinds of transitions: *move* ($\longrightarrow_M$), *conditioning* ($\longrightarrow_C$), *evaluation* ($\longrightarrow_E$), and *stop* ($\longrightarrow_S$). More precisely,

$$\begin{aligned} \longrightarrow_M &\in (P \times \overline{C}) \times (P \times P \times \overline{C}) \\ \longrightarrow_C &\in (P \times \overline{C}) \times C \times (P \times P \times \overline{C}) \\ \longrightarrow_E &\in (P \times P \times \overline{C}) \times \overline{M} \times (P \times P \times \overline{C}) \\ \longrightarrow_S &\in (P \times P \times \overline{C}) \times (P \times \overline{C}) \end{aligned}$$

Move and conditioning transitions make a source move from equilibrium ($P \times \overline{C}$) to motion ($P \times P \times \overline{C}$), the latter transition being labelled by a c-atom (C) – so as to model an input from the environment. Evaluation transition makes a source move from motion to motion and is labelled by a multiset of messages ($\overline{M}$) – modelling an output to the environment –, while stop transition makes the source return to equilibrium from motion without any label.[2] The semantics

[2] This labelled transition system is somehow different with respect to the general definition given in Subsection 2.1, being of the kind $X \times Act \times X'$ with $X \neq X'$. On the one hand, it would be easy to show how the transition system defined here could be seen as a special case of the general definition – e.g. considering the set of processes X as the union of sources in equilibrium state and motion state, by choosing $X = (P \times \overline{C}) \cup (P \times P \times \overline{C})$ in each transition. On the other hand, we believe that our definition more naturally leads to the intuition of equilibrium state, motion state, and their dynamics.

for these transitions is given by the following rules:

$$p\,[\,\overline{c}\,] \rightarrow_M \langle p, p'\rangle[\,\overline{c}\,] \tag{T-MOVE}$$

$$p\,[\,\overline{c}\,] \xrightarrow{c}_C \langle p, p\rangle[\,c|\overline{c}\,] \tag{T-COND}$$

$$\frac{select(c|\overline{c})(p, p') = c \qquad eval(c)(p, p') = \langle c', p_0, \overline{m}\rangle \qquad \langle p, p'\rangle :> p_0 = \langle p'', p'''\rangle}{\langle p, p'\rangle[\,c|\overline{c}\,] \xrightarrow{\overline{m}}_E \langle p'', p'''\rangle[\,c'|\overline{c}\,]} \tag{T-EVAL}$$

$$\frac{select(\overline{c})(p, p') = \perp_C}{\langle p, p'\rangle[\,\overline{c}\,] \rightarrow_S p'\,[\,\overline{c}\,]} \tag{T-STOP}$$

A source in equilibrium is fully characterised by its current position $p\,[\,\overline{c}\,]$, and the only applicable transitions are (T-MOVE) and (T-COND). This corresponds to the ontological notion that the equilibrium of a source can be perturbed only by either a spontaneous move (T-MOVE)[3] or a conditioning by a coordinator (T-COND). Here, in particular, the act of conditioning a source is modelled as the addition of a single c-atom to the multiset of c-atoms currently constituting the source's configuration – in other words, a coordinator can condition a source by *registering* a single c-atom in the source's configuration. [4] [5]

After its equilibrium has been perturbed, a source starts a new trajectory towards a new equilibrium state. During its evolution, the source is fully characterised by its current motion $\langle p, p'\rangle[\,\overline{c}\,]$, and the only applicable transitions are (T-EVAL) and (T-STOP). In particular, (T-EVAL) selects (through *select*)

[3] For the sake of simplicity, in the transition (T-MOVE) the source's place p is allowed to move to any other place p'. In general, however, it could be desirable to constrain the dynamics of spontaneous moves, so as to prevent moves that the system component modelled as source does not actually perform. This can be done e.g. by adding to the tuple specifying the source semantics a relation $S \subseteq P \times \overline{C} \times P$ stating that from current position $\langle p, \overline{c}\rangle$, a source can spontaneously perform move $\langle p, p'\rangle$ only if $\langle p, \overline{c}, p'\rangle \in S$. Although this could be supported by simply adding condition $\langle p, \overline{c}, p'\rangle \in S$ to rule (T-MOVE), this is avoided here in order to keep the source specification and semantics as simpler as possible.

[4] Defining conditioning as the registration of a single c-atom is not a severe limitation: configuration modifications other than adding a c-atom (such as deleting a c-atom) can be the whole result of a trajectory caused by the registration of a single c-atom.

[5] Typically, every source defines, either implicitly or explicitly, *who* is allowed to modify its configuration (in other words, who can act as a coordinator for that source), and *how* this can be done. Then, in general, only a subset of the admissible c-atoms are available to given coordinators for conditioning a source. For every class of coordinators, a source defines the set $D \subseteq C$ of the admissible *conditioning atoms*, or *d-atoms*, for short. However, the focus on the source's behaviour, rather than on the coordinator's, makes this issue not relevant in the context of this paper. As a result, the distinction between c-atoms and d-atoms is not considered in the following, implicitly assuming $D = C$ as an acceptable approximation.

one c-atom c from the source's configuration $\overline{c}$ and evaluates it (through *eval*), leading to a new source's motion and to new manifestations towards the observers (the possibly empty multiset of messages $\overline{m}$). After a number of c-atom evaluations, the source's motion stops (T-STOP) when no more c-atoms in the configuration can be evaluated (*select* returns $\perp_C$), so that the source concludes its trajectory and comes back to equilibrium.

In general, this model is able to represent the dynamic of sources with non-finite trajectories that never restore the source's equilibrium. This happens when the selection function keeps finding c-atoms to be evaluated, leading to infinitely many evaluation transitions. However, this situation is apparently not interesting in the context of this paper, so it is ignored in the following.

2.3 An Example

As an example of how to model a system component as a source, and how its behaviour can be represented through the transitions of the above labelled transition system, a simple Internet application is introduced in the following. A monitor agent a_M is in charge of getting information from m sites of the network, each having a local agent a_L^j $(1 \leq j \leq m)$ responding to the queries. The monitor agent can delegate its work to n *gatherer* agents a_G^k $(1 \leq k \leq n)$, roaming over sites and gathering information. Thus, the monitor agent autonomously determines the list of sites assigned to each gatherer, and sends a request message to it. Each gatherer autonomously roams the assigned sites, and finally reports results to the monitor.

As argued in Subsection 1.3 the formal framework described in this paper is built on the idea of modelling agents as sources. As a result, this example application is modelled here as composed by three classes of sources, *locals*, *gatherers* and a *monitor*, respectively with names id_L^j, id_G^k, and id_M. Their semantics is specified by providing the tuple $\langle P, C, M, J\rangle$ for each class, as follows.

The simplest kind of agent is the local agent, which is modelled as a purely reactive source that can be conditioned so as to manifest the result of a certain query. The semantics of a generic local agent id_L^j is given in Figure 1. The place of a local agent is represented as a function assigning to each query $q \in Q$ a datum $d \in D$ representing the corresponding result to be replied. The c-atom $ask(q, id)$ represents the conditioning made by a certain observer id – which in our application is a gatherer id_G – asking for the local knowledge q. The local agent with name id_L^j can emit messages of the kind $id \Uparrow info(id_L^j, d)$, which contains the information d to be provided to id. The selection function is trivial, since it simply makes each c-atom in the source's configuration be immediately evaluated. When the c-atom $ask(q, id)$ is evaluated, (i) it is dropped from the configuration, (ii) it does not change the observable place, and (iii) it produces the manifestation of the message containing the reply.

$$
\begin{aligned}
P_L &= Q \mapsto D \\
C_L &::= ask(q, id) \\
M_L &::= info(s^j_L, d)
\end{aligned}
$$

$$
\begin{aligned}
select(c|\overline{c}) &= \{\langle p_L, p'_L\rangle \mapsto c\} \\
eval(ask(q, id)) &= \{\langle p_L, p'_L\rangle \mapsto \langle \bot_C, p'_L, id \Uparrow info(s^j_L, p'_L(q))\rangle\}
\end{aligned}
$$

Figure 1: The semantics of a local agent id^j_L as a source

$$
\begin{aligned}
P_G &= \mathcal{P}(ID) \times Q_\bot \times (ID \mapsto D_\bot) \times ID_\bot \\
C_G &::= info(id, d) | gath(ID', q) | query | end \\
M_G &::= id \Uparrow ask(q, id^i_G) | id \Uparrow res(f, id^i_G)
\end{aligned}
$$

$$
\begin{aligned}
select(query|\overline{c}) &= \{\langle\langle ID', q, f, id\rangle, \langle ID', q, f, id'\rangle\rangle \mapsto query : \quad \text{if} \quad id \neq id'\} \\
select(end|\overline{c}) &= \{\langle\langle ID', q, f, id\rangle, \langle \{\}, \bot_I, f, \bot_{ID}\rangle\rangle \mapsto end : \quad \text{if} \quad ID' \neq \{\}\} \\
select(c|\overline{c}) &= \{\langle p_G, p_{G'}\rangle \mapsto c : \quad \text{if} \quad c \in \{info(id, d), gath(ID', q)\}\}
\end{aligned}
$$

$$
\begin{aligned}
eval(info(id, d)) &= \left\{ \begin{array}{l} \langle p_G, \langle ID', q, f, id\rangle\rangle \mapsto \\ \qquad \langle \bot_C, \langle ID' \setminus \{id\}, q, f[id \mapsto d], id\rangle, \epsilon\rangle \end{array} \right\} \\
eval(gath(ID', q)) &= \{\langle p_G, \langle \{\}, \bot_I, \{\}, id\rangle\rangle \mapsto \langle \bot_C, \langle ID', q, \{\}, \bot_{ID}\rangle, \epsilon\rangle\} \\
eval(query) &= \left\{ \begin{array}{l} \langle p_G, \langle ID', q, f, id\rangle\rangle \mapsto \\ \qquad \langle query, \langle ID', q, f, id\rangle, id \Uparrow ask(q, id^i_G)\rangle \end{array} \right\} \\
eval(end) &= \left\{ \begin{array}{l} \langle p_G, \langle ID', q, f, \bot_{ID}\rangle\rangle \mapsto \\ \qquad \langle end, \langle \{\}, \bot_I, \{\}, \bot_{ID}\rangle, id_M \Uparrow res(f, id^i_G)\rangle \end{array} \right\}
\end{aligned}
$$

Figure 2: The semantics of a gatherer agent id^i_G as a source

Figure 2 reports the semantics of the gatherer id^i_G. The place $\langle ID', q, f, id\rangle$ of a gatherer contains four kinds of information: (i) the subset ID' of sites to visit[6], (ii) the information q to ask, (iii) the information already gathered f (as a function from ID to D) and (iv) the next site to visit id. There are four kinds of c-atom, two of which represent conditionings possibly coming from outside, the others representing a part of the configuration that always persists and is used to manifest internal changes to the outside. The conditioning of the

[6] To this end, for the sake of simplicity, the site to be visited is identified by the name of the corresponding local agent. This technique, on the other hand, also provides for location transparency, making it possible to abstract from a site' actual position, and from the path an agent needs to go through in order to reach the site.

$$
\begin{aligned}
P_M &= \mathcal{P}(ID) \times Q_\perp \times ID \mapsto D_\perp \times ID \\
C_M &::= res(f, id) | mon \\
M_M &::= id \Uparrow gath(ID', q)
\end{aligned}
$$

$$
\begin{aligned}
select(mon|\overline{c}) &= \{\langle\langle\{\}, \perp_I, f, \perp_{ID}\rangle, \langle ID', q, f, id\rangle\rangle \mapsto mon\} \\
select(c|\overline{c}) &= \{\langle p_M, p'_M\rangle \mapsto c: \quad \text{if} \quad c \in \{res(f, id)\}\}
\end{aligned}
$$

$$
\begin{aligned}
eval(mon) &= \left\{ \begin{array}{l} \langle p_M, \langle ID', q, f, id\rangle\rangle \mapsto \\ \langle mon, \langle\{\}, \perp_Q, f, \perp_{ID}\rangle, id \Uparrow gath(ID', q)\rangle \end{array} \right\} \\
eval(res(f, id)) &= \{\langle p_M, \langle ID', q, f', id\rangle\rangle \mapsto \langle mon, \langle ID', q, f \otimes f', id\rangle, \epsilon\rangle\}
\end{aligned}
$$

Figure 3: The semantics of a monitor agent as a source

c-atom $info(id, d)$ represents the information d returned by id in response to a query. The conditioning of the c-atom $gath(ID', q)$ is sent by the monitor agent asking to search for information q in all the sites of the set $ID' \subseteq ID$. The c-atom *query* is selected when the agent decides to visit a new site, and is meant to fire the corresponding query request. The c-atom *end* is selected when no more sites are to be visited, and is meant to fire the results message to the monitor. As shown in Figure 2, while the latter two c-atoms are selected when a particular condition holds on the move $\langle p_G, p'_G\rangle$ – so as to model the manifestation of a proactive behaviour –, the former two c-atoms are selected as they occur in the configuration – as a means for modelling the manifestation of a reactive behaviour. The evaluation of $info(id, d)$ causes an update on the knowledge gathered and erases the information on the next site to visit. The evaluation of $gath(ID', d)$ causes the update of the lists of sites to be visited and of the query to ask, and also erases the next site to visit. When *query* is evaluated, the query message is sent to the local agent in the next site to visit, while *end* causes the gathered knowledge to be sent to the monitor, and its place to be initialised.

Finally, the semantics of the monitor is shown in Figure 3. A generic place $\langle ID', q, f, id\rangle$ specifies the knowledge it has already received f, and the next knowledge it needs, that is: (i) the set ID' of sites to visit, (ii) the agent id that should be charged with the request and (iii) the kind of information needed q. The only kind of conditioning c-atom is $res(f, id)$, representing an agent id communicating the knowledge it has retrieved by means of the function f from sites ID to data D. The c-atom mon is selected when the monitor decides it is time to ask for a new gathering, and causes the c-atom $gath(ID', q)$ to be sent to a gatherer agent. When the c-atom $res(f', id)$ is evaluated, it causes the current knowledge of the monitor f to be updated, by means of the binary operator $\otimes$

defined as:

$$f \otimes f' = \begin{cases} id \mapsto f(id) & \text{if} \quad f'(id) = \perp_D \\ id \mapsto f'(id) & \text{otherwise} \end{cases}$$

As a brief example to help understanding the notation, and to clarify the behaviour of the labelled transition system for sources, consider the case in which a gatherer autonomously decides to query one of the sites it was asked to visit. Initially, the gatherer is in equilibrium, with state

$$\langle \{id_L^1, id_L^2\}, q, \{\}, \perp_{ID} \rangle \, [\, query | end \,]$$

representing the fact that it is expected to visit id_L^1 and id_L^2, asking them the query q, having no information retrieved yet, nor future site to roam to already decided. Its deliberation to visit id_L^2 first is modelled by a move transition making its place changing to $\langle \{id_L^1, id_L^2\}, q, \{\}, id_L^2 \rangle$:

$$\begin{array}{l} \langle \{id_L^1, id_L^2\}, q, \{\}, \perp_{ID} \rangle \, [\, query | end \,] \longrightarrow_M \\ \qquad \langle \langle \{id_L^1, id_L^2\}, q, \{\}, \perp_{ID} \rangle, \langle \{id_L^1, id_L^2\}, q, \{\}, id_L^2 \rangle \rangle [\, query | end \,] \end{array}$$

As a result, an evaluation transition is triggered, where the selection function selects c-atom *query*, and its evaluation causes (i) *query* being reinserted in the configuration, (ii) the place remaining unmodified, and (iii) a message being sent to id_L^2 asking q.

$$\begin{array}{l} \langle \langle \{id_L^1, id_L^2\}, q, \{\}, \perp_{ID} \rangle, \langle \{id_L^1, id_L^2\}, q, \{\}, id_L^2 \rangle \rangle [\, query | end \,] \xrightarrow{id_L^2 \Uparrow ask(i, s_G)}_E \\ \qquad \langle \langle \{id_L^1, id_L^2\}, q, \{\}, \perp_{ID} \rangle, \langle \{id_L^1, id_L^2\}, q, \{\}, id_L^2 \rangle \rangle [\, query | end \,] \end{array}$$

At the next step, the selection function does not find further c-atoms to be evaluated, so after a stop transition the gatherer returns to a new equilibrium state:

$$\begin{array}{l} \langle \langle \{id_L^1, id_L^2\}, q, \{\}, \perp_{ID} \rangle, \langle \{id_L^1, id_L^2\}, q, \{\}, id_L^2 \rangle \rangle [\, query | end \,] \longrightarrow_S \\ \qquad \langle \{id_L^1, id_L^2\}, q, \{\}, id_L^2 \rangle \, [\, query | end \,] \end{array}$$

3 Agents as Sources

Based on the formal framework for modelling sources defined in the previous section, here a calculus for the behaviour of agents is developed. Since the starting point is to interpret an agent as a source, it may seem natural to straightforwardly exploit the model for sources, and represent the agents' behaviour in terms of move, conditioning, evaluation, and stop transitions. However, the need for higher levels of abstraction calls for models where the details of the internal machinery handling interactions are hidden, and where the focus is put on modelling the effects of the agent's interactions on its observable core.

3.1 The Calculus

According to the above considerations, the calculus for agents as sources is built based on the idea of abstracting away from the trajectory details making a source move from an equilibrium state to another. In this way, the observable behaviour of an agent can be characterised only in terms of how conditionings and spontaneous moves affect the dynamics of its equilibrium state and its manifestations, hiding the resulting sequences of motion states. To this end, a new labelled transition system over agents (modelled as sources) is defined, wrapping the one presented in Subsection 2.2. This is made by two kinds of transition:

$$\begin{array}{l}\longrightarrow_P \in (P \times \overline{C}) \times \overline{M} \times (P \times \overline{C}) \\ \longrightarrow_R \in (P \times \overline{C}) \times C \times \overline{M} \times (P \times \overline{C})\end{array}$$

whose operational semantics is defined by the rules:

$$\frac{\begin{array}{c} p_0\,[\,\overline{c}_0\,] \rightarrow_M \langle p_0, p'_0\rangle[\,\overline{c}_0\,] \\ \langle p_i, p'_i\rangle[\,\overline{c}_i\,] \xrightarrow{\overline{m}_i}_E \langle p_{i+1}, p'_{i+1}\rangle[\,\overline{c}_{i+1}\,] \quad 0 \le i < n \\ \langle p_n, p'_n\rangle[\,\overline{c}_n\,] \rightarrow_S p\,[\,\overline{c}\,] \end{array}}{p_0\,[\,\overline{c}_0\,] \xrightarrow{\overline{m}_0|\ldots|\overline{m}_{n-1}}_P p\,[\,\overline{c}\,]} \qquad \text{(AGENT-P)}$$

$$\frac{\begin{array}{c} p_0\,[\,\overline{c}_0\,] \xrightarrow{c}_C \langle p_0, p'_0\rangle[\,\overline{c}'_0\,] \\ \langle p_i, p'_i\rangle[\,\overline{c}'_i\,] \xrightarrow{\overline{m}_i}_E \langle p_{i+1}, p'_{i+1}\rangle[\,\overline{c}'_{i+1}\,] \quad 0 \le i < n \\ \langle p_n, p'_n\rangle[\,\overline{c}'_n\,] \rightarrow_S p\,[\,\overline{c}\,] \end{array}}{p_0\,[\,\overline{c}_0\,] \xrightarrow{c \triangleright \overline{m}_0|\ldots|\overline{m}_{n-1}}_R p\,[\,\overline{c}\,]} \qquad \text{(AGENT-R)}$$

The former transition ($\longrightarrow_P$) is called *proactive (agent) transition*, and models the situation where the agent proactively produces a manifestation and changes its core. In particular, a proactive transition

$$p_0\,[\,\overline{c}_0\,] \xrightarrow{\overline{m}_0|\ldots|\overline{m}_{n-1}}_P p\,[\,\overline{c}\,]$$

moves an agent from the equilibrium state $p_0\,[\,\overline{c}_0\,]$ to the the equilibrium state $p\,[\,\overline{c}\,]$, and produces the multiset of manifestations $\overline{m}_0|\ldots|\overline{m}_{n-1}$. According to the rule [AGENT-P], starting from an equilibrium state where a spontaneous move on the agent's place occurs, this transition moves the agent to the first equilibrium state reached after n motion states $\langle p_i, p'_i\rangle[\,\overline{c}_i\,]$ $(0 \le i < n)$. This transition, then, is labelled by the multiset of all the messages sent out by the agent during the corresponding trajectory.

The latter transition ($\longrightarrow_R$) is called *reactive (agent) transition*, and models the agent receiving a conditioning from its environment and reactively producing manifestations. A reactive transition

$$p_0\,[\,\overline{c}_0\,] \xrightarrow{c \triangleright \overline{m}_0|\ldots|\overline{m}_{n-1}}_R p\,[\,\overline{c}\,]$$

is triggered by a configuration conditioning c, and makes an agent in the equilibrium state $p_0\,[\,\overline{c}_0\,]$ move towards a new equilibrium state $p\,[\,\overline{c}\,]$, while producing the multiset of manifestations $\overline{m}_0|...|\overline{m}_{n-1}$. According to the rule [AGENT-R], starting from an equilibrium state the agent receives a c-atom which is treated as a conditioning. Correspondingly, the reactive transition makes the agent move to the first equilibrium state reached after a number of motion states, and is labelled by both the conditioning c-atom that triggers the agent's motion, and the multiset of all the messages sent out by the agent during the corresponding trajectory.

In all, this transition system models an agent as a *situated* source, interacting with its environment in two ways: by proactively sending a multiset of messages to it – through the transition $p_0\,[\,\overline{c}_0\,] \xrightarrow{\overline{m}}_P p\,[\,\overline{c}\,]$ –, or by receiving a c-atom from it and reactively replying with a multiset of messages – through the transition $p_0\,[\,\overline{c}_0\,] \xrightarrow{c \triangleright \overline{m}}_R p\,[\,\overline{c}\,]$.

3.2 An Example

Based on the example application introduced in Subsection 2.3, here some examples of transitions are described that emphasise the flavour of our calculus for agents as sources.

First of all, consider the case of a monitor agent. A monitor is modelled as a situated source performing a number of proactive transitions – each producing a request for knowledge to be gathered from a given list of sites – as well as a number of reactive transitions – each receiving knowledge gathered from one site list. For instance, the following transitions represent the dynamics of a monitor agent asking agents id_G^1 and id_G^2 to gather information q from site lists ID_1 and ID_2, respectively, then receiving a reply from id_G^2:

$$\begin{array}{l} \langle\{\}, \bot_Q, f, \bot_{ID}\rangle\,[\,mon\,] \xrightarrow{id_G^1 \Uparrow gath(ID_1,q)}_P \\ \langle\{\}, \bot_Q, f, id_G^1\rangle\,[\,mon\,] \xrightarrow{id_G^2 \Uparrow gath(ID_2,q)}_P \\ \langle\{\}, \bot_Q, f, id_G^2\rangle\,[\,mon\,] \xrightarrow{res(f_2,id_G^2) \triangleright \epsilon}_R \\ \langle\{\}, \bot_Q, f \otimes f_2, id_G^2\rangle\,[\,mon\,] \end{array}$$

On the other hand, the behaviour of a gatherer agent can be characterised as follows: (i) the agent receives a gathering request specifying a list of sites (reactive transition) where to look for information q, then, while there is still at least one site to query, (ii) autonomously chooses the next site to explore, ask it about q (proactive transition), and (iii) get the required information back (reactive transition). When all the sites in the initial list have been explored, the last reactive transition also packs all the information gathered about q and emits it as a single knowledge chunk. For instance, the transitions below represent the case in which gatherer id_G gets a request for the sites id_L^1 and id_L^2, and deliberates to roam first id_L^2 and then id_L^1.

$$\langle\{\}, \bot_Q, \{\}, \bot_{ID}\rangle\,[\,query|end\,] \xrightarrow{gath(\{id_L^1, id_L^2\}, q) \triangleright \epsilon}_R$$
$$\langle\{id_L^1, id_L^2\}, q, \{\}, \bot_{ID}\rangle\,[\,query|end\,] \xrightarrow{id_L^2 \Uparrow ask(q, id_G)}_P$$
$$\langle\{id_L^1, id_L^2\}, q, \{\}, id_L^2\rangle\,[\,query|end\,] \xrightarrow{info(id_L^2, d_2) \triangleright \epsilon}_R$$
$$\langle\{id_L^1\}, q, \{id_L^2 \mapsto d_2\}, id_L^2\rangle\,[\,query|end\,] \xrightarrow{id_L^1 \Uparrow ask(q, id_G)}_P$$
$$\langle\{id_L^1\}, q, \{id_L^2 \mapsto d_2\}, id_L^1\rangle\,[\,query|end\,] \xrightarrow{info(id_L^1, d_1) \triangleright id_M \Uparrow res(\{id_L^2 \mapsto d_2, id_L^1 \mapsto d_1\}, id_G)}_R$$
$$\langle\{\}, \bot_Q, \{\}, \bot_{ID}\rangle\,[\,query|end\,]$$

Finally, the local agent is modelled as a simple reactive source. The following transitions represent a local agent id_L^1 replying to a couple of queries, then updating its own knowledge according to some event not to be interpreted in terms of observation – modelled here accordingly as a spontaneous move $\langle f, f'\rangle$.

$$f\,[\,\epsilon\,] \xrightarrow{ask(q_1, id_G^1) \triangleright id_G^1 \Uparrow info(id_L^1, f(q_1))}_R$$
$$f\,[\,\epsilon\,] \xrightarrow{ask(q_2, id_G^2) \triangleright id_G^2 \Uparrow info(id_L^1, f(q_2))}_R$$
$$f\,[\,\epsilon\,] \xrightarrow{\epsilon}_P f'\,[\,\epsilon\,]$$

In all, this example should help making clear how this calculus makes it possible to concentrate on agent's interactions with the environment where it is situated, as well as on the corresponding effects on the agent's observable core.

4 MAS as Source's Composition

Agents are situated entities, living and (inter)acting within a MAS as individuals within a society. They manifest a social attitude by making their knowledge, capabilities, and services available to other agents, which may require them in order to achieve their own goals.

In the following, a calculus for MAS is introduced based on the one described in previous section. Here, a MAS is seen as a composition of individual agents interpreted as sources, and interactions among agents and between the agents and the environment are modelled as source's manifestations and coordinator's conditionings.

4.1 The Calculus

In our framework, a MAS is denoted by the following syntax:

$$
\begin{array}{rll}
\text{Agents:} & A \ ::= id : p\,[\,\overline{c}\,] & \\
\text{Messages:} & M \ ::= id \Uparrow c & \\
\text{MAS:} & \Gamma \ ::= \epsilon_M & \text{an empty MAS} \\
 & \quad \mid \ a & \text{an agent of the MAS} \\
 & \quad \mid \ \overline{m} & \text{a multiset of messages pending in a MAS} \\
 & \quad \mid \ \gamma \ \oplus \ \gamma & \text{a composition of MAS}
\end{array}
$$

An agent $id : p\,[\,\overline{c}\,]$ is represented as a source in the equilibrium state $p\,[\,\overline{c}\,]$, and is characterised by a unique name id ranging over the set of agent names ID. As shown in the previous section, agents' evolution is amenable to a representation as sources moving through sequences of equilibrium states. So, growing the calculus for MAS on top of this model basically means viewing a MAS as a composition of sources in equilibrium state. This makes it possible to concentrate on agents' interactions, abstracting away from details about their trajectories.

Agents interact with each other and with the environment by exchanging messages of the kind $id \Uparrow c$, where id is the name of the destination agent, and the content of the message is c-atom c. On the one hand, the messages emitted by an agent (represented as a source) are to be interpreted as manifestations of its observable behaviour. On the other hand, they are sent to other (source) agents and contain a c-atom, so they are also suitable for an interpretation as conditionings. As a result, although each agent is formally specified as a source, it might also be interpreted as a coordinator or an observer as well, depending on whether it is sending or receiving a message. In particular, when a message is emitted, the sender agent id_S is viewed as a source producing a manifestation message, while the destination agent id_R can be seen as an observer for the source id_S. At the time the source agent id_R receives the message, instead, the sender id_S can be seen as a coordinator for id_R.

A MAS γ can be either an empty MAS, a multiset of messages, or a composition of two MAS. The intent of this calculus is to represent a MAS as an unordered composition of agents interacting by the exchange of asynchronous messages, and evolving through the *interleaved* evolution of each agent. This kind of representation is obtained by borrowing typical techniques introduced in the field of process algebras [Bergstra *et al.*, 2001] as follows.

The following properties are supposed to hold for MAS[7]:

$$\gamma_1 \ \oplus \ \gamma_2 \equiv \gamma_2 \ \oplus \ \gamma_1 \qquad \epsilon_M \ \oplus \ \gamma \equiv \gamma \qquad (\gamma_1 \ \oplus \ \gamma_2) \ \oplus \ \gamma_3 \equiv \gamma_1 \ \oplus \ (\gamma_2 \ \oplus \ \gamma_3)$$

[7] These rules, which assume $(\gamma, \ \oplus \ , \epsilon)$ to be an Abelian monoid, are commonly used in the definition of process algebras, such as in the case of π-calculus [Parrow, 2001a].

In particular, two MAS are considered the same when they are similar up to the congruence relation $\equiv$ defined as the least relation satisfying the three above rules[8]. As a result, $\oplus$ now can be seen as a n-ary commutative operator composing agents and (multiset of) messages in an unordered way. The reason for having messages and agents at the same level – namely, *floating* in the same *soup*[9] –, comes from the typical techniques used to model asynchronous messages [Boudol, 1992]. When a process P intends to send an asynchronous message M and then behave like P', it is seen as the composition of P' and the process emitting the message and having no continuation, the latter process being simply denoted by the message M itself. The actual consumption of M by a process Q can actually take place later than the processing of P', thus modelling asynchrony. So, according to this kind of modelling, our calculus interprets a MAS as composed by multisets of asynchronous messages and agents[10].

The interleaved semantics of our model – where the evolution of a system is modelled in terms of the interleaved evolution of their sub-parts[11] – can be naturally defined by means of a labelled transition system over MAS, composed by the following transitions:

$$\begin{array}{l} \longrightarrow_{\pi} \in \Gamma \times \Gamma \\ \longrightarrow_{\rho} \in \Gamma \times \Gamma \\ \longrightarrow_{\omega} \in \Gamma \times \overline{M} \times \Gamma \\ \longrightarrow_{\iota} \in \Gamma \times \overline{M} \times \Gamma \end{array}$$

whose semantics is specified by the operational rules:

$$\frac{p_0\,[\,\overline{c}_0\,] \xrightarrow{\overline{m}}_P p\,[\,\overline{c}\,]}{\gamma \,\oplus\, id : p_0\,[\,\overline{c}_0\,] \longrightarrow_{\pi} \gamma \,\oplus\, id : p\,[\,\overline{c}\,] \,\oplus\, \overline{m}} \qquad \text{(MAS-}\pi\text{)}$$

$$\frac{p_0\,[\,\overline{c}_0\,] \xrightarrow{c \rhd \overline{m}}_R p\,[\,\overline{c}\,]}{\gamma \,\oplus\, id : p_0\,[\,\overline{c}_0\,] \,\oplus\, (id \Uparrow c)|\overline{m}_0 \longrightarrow_{\rho} \gamma \,\oplus\, id : p\,[\,\overline{c}\,] \,\oplus\, \overline{m} \,\oplus\, \overline{m}_0} \qquad \text{(MAS-}\rho\text{)}$$

8 Including void multiset ϵ_M in the syntax for MAS, along with the composition operator $\oplus$, makes our model for MAS a process algebra [Bergstra *et al.*, 2001]. Although in this paper this fundamental aspect is not further considered, in general it allows us to exploit a rich framework of existing tools for verifying properties on system's specifications [J.F.Groote, 2001], such as the ones concerning process equivalence [Glabbeek, 2001].

9 The figurative idea of a concurrent system as a soup of floating processes derives from the chemical abstract machine model [Berry and Boudol, 1992].

10 Our syntax deals with multiset of messages – and not just messages – simply because agents generally emit multiset of messages in an atomic way. As a result, it is also convenient to suppose a void multiset of messages be equal to an empty MAS, that is $\epsilon \equiv \epsilon_M$, or analogously adding the rule $\epsilon \,\oplus\, \gamma \equiv \gamma$. In fact, this hypothesis makes it possible to completely loose track of a multiset of messages from the specification of a MAS after all of them have been consumed.

11 For a discussion on the semantics and formal differences between interleaving process semantics and non-interleaving ones, see e.g. [Parrow, 2001b].

$$\gamma \oplus \overline{m} \xrightarrow{\overline{m}}_{\omega} \gamma \qquad \text{(MAS-}\omega\text{)}$$

$$\gamma \xrightarrow{\overline{m}}_{\iota} \gamma \oplus \overline{m} \qquad \text{(MAS-}\iota\text{)}$$

It is worth noting that the former two transitions propagate reactivity and proactiveness from the individual agent level to the MAS level, modelling a MAS where one agent performs a $\longrightarrow_P$ and $\longrightarrow_R$ transition, respectively. In particular, transition $\longrightarrow_\pi$ is called *MAS proactive transition*, and models an agent in the MAS proactively sending a multiset of messages. For instance, transition

$$\gamma \oplus a_0 \longrightarrow_\pi \gamma \oplus a \oplus \overline{m}$$

models agent a_0 in MAS γ proactively sending the messages $\overline{m}$ and becoming a, according to the semantics of rule [MAS-π]. Transition $\longrightarrow_\rho$ is called *MAS reactive transition*, and models an agent in the MAS consuming one message and reactively sending a multiset of messages. For instance, transition

$$\gamma \oplus a_0 \oplus (id \Uparrow c)|\overline{m}_0 \longrightarrow_\rho \gamma \oplus a \oplus \overline{m} \oplus \overline{m}_0$$

models agent a_0 – with name id within MAS γ – consuming the message $id \Uparrow c$ (by interpreting the content of the message c as a conditioning), producing the messages $\overline{m}$ and becoming a, according to the semantics of the rule [MAS-ρ].

The transitions $\longrightarrow_\omega$ and $\longrightarrow_\iota$, called *MAS output transition* and *MAS input transition*, allow messages to be respectively dropped and put in the MAS. Since transitions with non-void labels are typical meant to represent interactions with the external environment from which the model abstracts away, $\longrightarrow_\omega$ and $\longrightarrow_\iota$ can be exploited in order to represent the MAS' environment respectively consuming messages in the MAS and sending messages to the MAS. As obvious, which agents should be directly modelled as part of the MAS, and which instead are to be considered as part of the MAS' environment is an arbitrary choice.

4.2 An Example

As an application example for this calculus, consider an agent-based system composed by one monitor agent id_M, one gatherer agent id_G, and n local agents id_L^i $(1 \leq i \leq n)$. As the initial state, consider the following MAS:

$$\begin{array}{l} id_L^1 : f^1\,[\,\epsilon\,] \oplus id_L^2 : f^2\,[\,\epsilon\,] \oplus \\ id_M : \langle \{\}, \bot_Q, \{\}, \bot_{ID} \rangle\,[\,mon\,] \oplus \\ id_G : \langle \{\}, \bot_Q, \{\}, \bot_{ID} \rangle\,[\,query|end\,] \end{array}$$

which is composed of the monitor, the gatherer and the local agents id_L^1 and id_L^2. Local agents have initially a void configuration, and knowledge f^1 and f^2. The

$$
\begin{array}{lr}
id_L^1 : f^1\,[\,\epsilon\,] \;\oplus\; id_L^2 : f^2\,[\,\epsilon\,] \;\oplus\; id_M : \langle\{\}, \bot_Q, \{\}, \bot_{ID}\rangle\,[\,mon\,] \;\oplus & \\
id_G : \langle\{\}, \bot_Q, \{\}, \bot_{ID}\rangle\,[\,query|end\,] & \\
\longrightarrow_\pi & (1) \\
id_L^1 : f^1\,[\,\epsilon\,] \;\oplus\; id_L^2 : f^2\,[\,\epsilon\,] \;\oplus\; id_M : \langle\{\}, \bot_Q, \{\}, id_G\rangle\,[\,mon\,] \;\oplus & \\
id_G : \langle\{\}, \bot_Q, \{\}, \bot_{ID}\rangle\,[\,query|end\,] \;\oplus\; id_G \Uparrow gath(\{id_L^3, id_L^2\}, q) & \\
\longrightarrow_\rho & (2) \\
id_L^1 : f^1\,[\,\epsilon\,] \;\oplus\; id_L^2 : f^2\,[\,\epsilon\,] \;\oplus\; id_M : \langle\{\}, \bot_Q, \{\}, id_G\rangle\,[\,mon\,] \;\oplus & \\
id_G : \langle\{id_L^3, id_L^2\}, q, \{\}, \bot_{ID}\rangle\,[\,query|end\,] & \\
\longrightarrow_\pi & (3) \\
id_L^1 : f^1\,[\,\epsilon\,] \;\oplus\; id_L^2 : f^2\,[\,\epsilon\,] \;\oplus\; id_M : \langle\{\}, \bot_Q, \{\}, id_G\rangle\,[\,mon\,] \;\oplus & \\
id_G : \langle\{id_L^3, id_L^2\}, q, \{\}, id_L^2\rangle\,[\,query|end\,] \;\oplus\; id_L^2 \Uparrow ask(q, id_G) & \\
\longrightarrow_\rho & (4) \\
id_L^1 : f^1\,[\,\epsilon\,] \;\oplus\; id_L^2 : f^2\,[\,\epsilon\,] \;\oplus\; id_M : \langle\{\}, \bot_Q, \{\}, id_G\rangle\,[\,mon\,] \;\oplus & \\
id_G : \langle\{id_L^3, id_L^2\}, q, \{\}, id_L^2\rangle\,[\,query|end\,] \;\oplus\; id_G \Uparrow info(id_L^2, f^2(q)) & \\
\longrightarrow_\rho & (5) \\
id_L^1 : f^1\,[\,\epsilon\,] \;\oplus\; id_L^2 : f^2\,[\,\epsilon\,] \;\oplus\; id_M : \langle\{\}, \bot_Q, \{\}, id_G\rangle\,[\,mon\,] \;\oplus & \\
id_G : \langle\{id_L^3, id_L^2\}, q, \{id_L^2 \mapsto f^2(q)\}, id_L^2\rangle\,[\,query|end\,] & \\
\longrightarrow_\pi & (6) \\
id_L^1 : f^1\,[\,\epsilon\,] \;\oplus\; id_L^2 : f^2\,[\,\epsilon\,] \;\oplus\; id_M : \langle\{\}, \bot_Q, \{\}, id_G\rangle\,[\,mon\,] \;\oplus & \\
id_G : \langle\{id_L^3, id_L^2\}, q, \{id_L^2 \mapsto f^2(q)\}, id_L^3\rangle\,[\,query|end\,] \;\oplus & \\
id_L^3 \Uparrow ask(q, id_G) & \\
\xrightarrow{id_L^3 \Uparrow ask(q, id_G)}_\omega & (7) \\
id_L^1 : f^1\,[\,\epsilon\,] \;\oplus\; id_L^2 : f^2\,[\,\epsilon\,] \;\oplus\; id_M : \langle\{\}, \bot_Q, \{\}, id_G\rangle\,[\,mon\,] \;\oplus & \\
id_G : \langle\{id_L^3, id_L^2\}, q, \{id_L^2 \mapsto f^2(q)\}, id_L^3\rangle\,[\,query|end\,] & \\
\xrightarrow{id_G \Uparrow info(id_L^3, f^3(q))}_\iota & (8) \\
id_L^1 : f^1\,[\,\epsilon\,] \;\oplus\; id_L^2 : f^2\,[\,\epsilon\,] \;\oplus\; id_M : \langle\{\}, \bot_Q, \{\}, id_G\rangle\,[\,mon\,] \;\oplus & \\
id_G : \langle\{id_L^3, id_L^2\}, q, \{id_L^2 \mapsto f^2(q)\}, id_L^3\rangle\,[\,query|end\,] \;\oplus & \\
id_G \Uparrow info(id_L^3, f^3(q)) & \\
\longrightarrow_\rho & (9) \\
id_L^1 : f^1\,[\,\epsilon\,] \;\oplus\; id_L^2 : f^2\,[\,\epsilon\,] \;\oplus\; id_M : \langle\{\}, \bot_Q, \{\}, id_G\rangle\,[\,mon\,] \;\oplus & \\
id_G : \langle\{id_L^3, id_L^2\}, q, \{id_L^2 \mapsto f^2(q), id_L^3 \mapsto f^3(q)\}, id_L^3\rangle\,[\,query|end\,] \;\oplus & \\
id_M \Uparrow res(\{id_L^2 \mapsto f^2(q), id_L^3 \mapsto f^3(q)\}, id_G) & \\
\longrightarrow_\rho & (10) \\
id_L^1 : f^1\,[\,\epsilon\,] \;\oplus\; id_L^2 : f^2\,[\,\epsilon\,] \;\oplus\; id_G : \langle\{\}, \bot_Q, \{\}, \bot_{ID}\rangle\,[\,query|end\,] \;\oplus & \\
id_M : \langle\{\}, \bot_Q, \{id_L^2 \mapsto f^2(q), id_L^3 \mapsto f^3(q)\}, id_G\rangle\,[\,mon\,] &
\end{array}
$$

Figure 4: A case of MAS evolution

monitor agent has neither knowledge nor gathering requests pending, whereas the gatherer is waiting for a gathering request: the initial place for both agents is then modelled by the tuple $\langle\{\}, \perp_Q, \{\}, \perp_{ID}\rangle$.

Figure 4 shows an example of evolution of this MAS, where the monitor asks and receives from the gatherer information on the local agents id_L^2 and id_L^3, the latter being an agent outside the MAS, i.e., living in the MAS' environment. As a first step (1), the monitor autonomously decides to get information q from sources id_L^2 and id_L^3 through the gatherer agent, so it proactively produces the request message $id_G \Uparrow \mathit{gath}(\{id_L^3, id_L^2\}, q)$. Consequently (2), the gatherer consumes the request, updates its place to $\langle\{id_L^3, id_L^2\}, q, \{\}, \perp_{ID}\rangle$, and starts deliberating which site it should visit first. At a given time (3), it chooses id_L^2 and produces a request message: this is consumed by the local agent (4), which reacts with a message $id_G \Uparrow \mathit{info}(id_L^2, f^2(q))$, providing the result $f^2(q)$. Then (5), the gatherer receives the reply and correspondingly updates its current knowledge to $\{id_L^2 \mapsto f^2(q)\}$. Later (6), the gatherer decides it is time to explore id_L^3 and to produce a request message. Since the source id_L^3 is not modelled as an agent of the MAS, the consumption of that message is modelled as the environment getting the message through an input transition (7), and the reply is modelled as the environment producing the response message through an output transition (8). The gatherer consumes that message (9) and since all the sites have been queried, it returns to the monitor the whole result of its gathering through the message $id_M \Uparrow \mathit{res}(\{id_L^2 \mapsto f^2(q), id_L^3 \mapsto f^3(q)\}, id_G)$. Finally (10), the monitor receives and processes the message, updating its knowledge.

5 Interpretation

Growing a system's model on top of a precisely-defined ontology enjoys an interesting property: through formalisation, the system's sub-parts, evolution, behaviour, and properties are straightforwardly mapped onto the ontology, thus providing for an original viewpoint over the system. In this paper, agent-based systems are modelled using an ontology for observation, so that aspects such as agent's behaviour, interactions, reactivity, proactiveness, autonomy, social ability, as well as MAS' evolution, MAS' environment, and the like, are interpreted in terms of the ontology's concepts, such as source's manifestations, conditionings, spontaneous moves, trajectory, and so on.

As a first step, the issues related to individual agents are taken into account. Among the several definitions of the agent abstraction, here the one given in [Wooldridge, 2000] is considered, where the (rational) agent is seen as an entity situated in an *environment* – which the agent *senses* and *acts* upon –, and featuring *reactivity*, *proactiveness*, *autonomy* and *social ability*. Since agents are modelled as sources, the agent's environment is correspondingly modelled as the collection of all the entities that can condition it (coordinators) or can observe its manifestations (observers). As a result, acting is modelled as the agent manifesting to the observing environment, while sensing is modelled as the environment conditioning the agent's observable behaviour.

An agent's reactivity is modelled as its ability of producing manifestations in response to conditionings. A conditioning models the intention of a coordinator to interact with an agent in order to affect its observable behaviour. In fact, when an agent receives a conditioning from its environment that changes its configuration, it reacts by moving to a new equilibrium state and by producing observable manifestations according to its (new) configuration. An agent's proactiveness is modelled as its ability of producing manifestations due to spontaneous moves. A spontaneous move models some change to the agent's observation core – by definition, affecting the agent's observable behaviour – depending on some internal events, independently of any interaction with the agent's environment. Proactiveness then results from the observable manifestations produced following a spontaneous move according to the agent's configuration.

Our model accounts for agent's autonomy as well – until autonomy, roughly speaking, is the agent's capability to drive its own control, and not to be driven by others. On the one hand, details related to the inner (autonomous) agent's deliberation process, driving its control, are typically abstracted away – an agent remains in equilibrium until its autonomous computation does affect its observable behaviour, captured by the concept of agent's core. When the effects of the deliberation process become perceivable by the environment, these are modelled as spontaneous moves of the place, producing an agent's motion and possibly causing some manifestations. On the other hand, the ability of mediating the environment's requests is a crucial aspect related to the agent's autonomy: configuration works as a control layer that prevents such requests to directly influence the agent's inner status and dynamics. In fact, the environment's requests are expressed as conditionings inserted in the agent's configuration, affecting the agent's internal behaviour only as far as their evaluation produces a change on the place. In turn, changes to the places actually affect the agent's dynamics and observable behaviour depending again on the agent's configuration.

Then, in order to capture the idea of agent's social ability, the framework is extended from individual agents to MAS, so that a MAS is viewed as the composition of agents modelled as sources. A MAS' evolution is based on the agents interacting with each other through the exchange of asynchronous messages – each specifying a c-atom and a target agent – and on the interactions with the MAS' environment, modelled as a black-box entity able to put and drop messages from the MAS. An agent interacts with another by conditioning it, and possibly receiving as a reply a manifestation of its observable behaviour. As a result, it seems natural to interpret the social ability of an agent as its disposition to let other agents to perceive its observable behaviour, meant to represent its beliefs, capabilities, competence – roughly, in one word, the *knowledge* an agent encapsulates as a source, representing its own viewpoint on the world. Correspondingly, the inner evolution of a MAS – taking place through the exchange of messages between its agents – is naturally interpreted as the continuous exchange of information amongst agents. This information exchange can be seen as each agent's knowledge flowing through the MAS agents.

Nevertheless, our model does not just represent a MAS as a closed system evolving in isolation. Instead, a key aspect concerns the MAS' ability to interact with its environment. On the one hand, the environment can contribute to the flowing of knowledge within the MAS by inserting new messages, so as to contribute to the global MAS' knowledge – and, vice versa, information is also allowed to flow outside the MAS into the environment. On the other hand, the observation pattern for the interaction between the MAS and its environment also enables an interpretation where the MAS is either seen as a whole source, that the environment can condition so as to affect its global observable behaviour, or as a single observer for the environment's manifestations.

6 Related Works and Conclusions

The goal of this paper is to provide a framework – both conceptual and formal – for reasoning about the *observation* issue in agent-based systems, for representing agent's and MAS' behaviour, and for interpreting them in terms of aspects related to observation. This work carries on the research started in [Viroli *et al.*, 2001], where the observation issue is introduced in the context of computer systems in general. There, an ontology and a formal framework are developed so as to compare the expressiveness of software components coming from different areas – such as object orientation, DBMS and coordination models [Omicini *et al.*, 2001] – from the viewpoint of observation. In this paper – which is an extension to [Viroli and Omicini, 2001] – the ontology is applied to agent-based systems by interpreting agents as sources, and the formal framework is developed so as to better fit the typical needs of abstraction of agent-based systems. A simple application example of our framework is provided, which is meant to ease the understanding of formulae's syntax and semantics, and which was designed to be as small and simple as possible, while being complete enough to show and emphasise all the relevant aspects of the framework. On the other hand, in order to verify the applicability of the framework, more complex examples are required. Even though this issue goes beyond the scope of this paper, it is in fact addressed by many of our current research efforts.

For instance, in [Viroli and Omicini, 2002a] the applicability of the framework to the specification of an agent's observable behaviour is studied, considering a number of interaction patterns that are very common in agent-based systems. Each of them is mapped in the framework, showing the power of the model as a tool for specifying complex interactive behaviours in agent-based systems. As a further application, in [Viroli and Omicini, 2002b] the framework is exploited to provide a semantics to agent communication languages. In particular, the framework is shown to be suitable for describing agents' interactive behaviour in a rather architecture-independent way, which is a property quite relevant for organisations aiming at the definition of standardised platforms for agents, such as FIPA [FIPA, 2000]. Finally, in [Viroli and Omicini, 2002c] the framework is also put to test as a formal framework for coordination media [Omicini *et al.*, 2001] – that is, for those software abstractions meant to rule and govern interaction in distributed systems. The flavours of the model are described focussing on both how it captures the media's details of interest with the desired level of abstraction, and how it can be used to explicitly model complex behaviours of coordination media [Omicini and Denti, 2001].

The mathematical framework supporting the formalisation developed here is based on techniques introduced in the field of process algebras [Bergstra *et al.*, 2001], where they are exploited as a tool to describe the behaviour of interactive systems and to provide a foundation for issues such as communication and concurrency. Since the observation model interprets an agent as an interactive system, and a MAS as a concurrent system evolving through the interactions of its sub-parts, process algebras provide for a suitable conceptual and formal basis. Nevertheless, modelling an interactive system through the classical process algebra approach has rather different goals from the observation approach presented here.

From an engineering perspective, one of the main aims of the research field on process algebras is to define equivalences between *processes* – that is, to which extent interactive software components are to be considered equivalent with respect to a given semantics [Glabbeek, 2001]. On the one hand, the study of these properties is fundamental in order to ensure that a system keeps working in a correct way after the substitution of a sub-components of its. On the other hand, process equivalence can also be exploited to verify whether the implementation of a given component satisfies its formal specification. As discussed in Subsection 1.1, this general framework follows the so-called black-box modelling approach, where a software component's behaviour is modelled by completely abstracting away from aspects related to its internal machinery. Instead, the observation framework presented here follows a grey-box modelling approach, characterising the so-called *observable behaviour* of a software component in terms of how its interactions with the environment affect its status and, conversely, how such a status is made perceivable to and can be affected from the outside.

Nevertheless, our model of the interactive behaviour of an agent within a

MAS is suitable for a comparison with existing process algebra approaches. Considering the framework of name-passing and value-passing calculi – such as CCS [Milner, 1989], π-calculus [Milner, 1999], and asynchronous π-calculus [Boudol, 1992] – our agent's model can be thought of as a process evolving by performing actions. On the one hand, an agent can asynchronously produce messages for other processes, similarly to name sending in asynchronous π-calculus. Correspondingly, the formalisation of asynchrony provided here borrows from asynchronous π-calculus – that is, it relies on the concept of pending messages *floating in the same soup* of interacting processes [Berry and Boudol, 1992]. On the other hand, an agent can perform a more complex action by atomically receiving an asynchronous message and sending zero, one, or more asynchronous messages to other agents – which can be commonly referred to as a reactive behaviour. Finally, despite in our model interactions resemble asynchronous π-calculus, their content is a generic value analogously to value-passing calculi such as CCS.

Our approach is also related to others defining architectures for software component's behaviour, whose most notable example is the Actors model [Agha, 1986, Hewitt, 1977]. An actor is defined as a purely reactive software entity, which receives message and correspondingly changes its behaviour and send other messages. Conversely, as in most agent models the observation framework deals with agent proactiveness [Wooldridge, 2000], that is, with agents featuring an inner control responsible for changing its status and sending messages to the environment.

Another approach to modelling an agent communications in terms of labelled transition systems has been developed in [van Eijk *et al.*, 2000b, van Eijk *et al.*, 2000a]. In [van Eijk *et al.*, 2000b], for instance, a programming language is defined where agents are characterised by their mental state – including belief states and a goal state – and where agent communications follow the rendezvous schema, a version of the classic remote procedure call (RPC) where the target agent processes requests using an interleaved pattern. Then, the language is defined through an operational semantics, describing how agents evolve by internal computations (such as believes update) and through communications. There, however, the focus is put on modelling an agent's mental state and its dynamics, while the goal of the observation framework is more to emphasise the agent's collaborative aspects – namely, its interactive behaviour. Another relevant difference is related to the technical treatment: instead of describing agents' aspects through a programming language, here a generic architecture is developed – the model of an agent as an observable source – which can be specialised to different behaviours by changing the source's specification.

Further work along this research direction will be devoted to the study of the impact of this approach on the engineering of MAS – on all the places of the engineering process, from specification and design, to implementation, verification, and monitoring.

Acknowledgements

The authors would like to thanks MIUR (the Italian "Ministero dell'Istruzione, dell'Università e della Ricerca") and Nokia Research Center, Burlington, MA, USA, for partially supporting this work. Also, we are grateful to the anonymous referees, whose remarks and criticisms were of invaluable help in improving this article.

References

[Agha, 1986] Gul Agha. *Actors: A Model for Concurrent Computation in Distributed Systems*. MIT Press, 1986.

[Bergstra *et al.*, 2001] Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors. *Handbook of Process Algebra*. North-Holland, 2001.

[Berry and Boudol, 1992] Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

[Boudol, 1992] Gérard Boudol. Asynchrony and the π-calculus. Rapport de Researche, INRIA, Sophia Antipolis, Number 1702, May 1992.

[FIPA, 2000] FIPA. FIPA2000 specification. Foundation for Intelligent Physical Agents, `http://www.fipa.org`, 2000.

[Glabbeek, 2001] R.J. van Glabbeek. The linear time – branching time spectrum I. The semantics of concrete, sequential processes. In Bergstra et al. [Bergstra *et al.*, 2001], chapter 1, pages 3–100.

[Hewitt, 1977] Carl Hewitt. Viewing control structures as patterns of passing messages. *Artificial Intelligence*, 8(3):323–364, 1977.

[J.F.Groote, 2001] M.A. Reniers J.F.Groote. Algebraic process verification. In Bergstra et al. [Bergstra *et al.*, 2001], chapter 17, pages 1151–1208.

[Milner, 1989] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Milner, 1999] Robin Milner. *Communicating and Mobile Systems: The π-calculus*. Cambridge University Pres, 1999.

[Omicini and Denti, 2001] Andrea Omicini and Enrico Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 41(3):277–294, November 2001.

[Omicini *et al.*, 2001] Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf, editors. *Coordination of Internet Agents: Models, Technologies, and Applications*. Springer-Verlag, March 2001.

[Parrow, 2001a] J. Parrow. An introduction to the π-calculus. In Bergstra et al. [Bergstra *et al.*, 2001], chapter 8, pages 479–544.

[Parrow, 2001b] J. Parrow. Partial order process algebra (and its relation to Petri nets). In Bergstra et al. [Bergstra *et al.*, 2001], chapter 13, pages 769–873.

[van Eijk *et al.*, 2000a] Rogier M. van Eijk, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Open multi-agent systems: Agent communication and integration. In Nicholas R. Jennings and Yves Lespérance, editors, *Intelligent Agents VI*, volume 1757 of *LNAI*, pages 218–232. Springer-Verlag, 2000.

[van Eijk *et al.*, 2000b] Rogier M. van Eijk, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Operational semantics for agent communication languages. In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*, volume 1916 of *LNAI*, pages 80–95. Springer-Verlag, 2000.

[Viroli and Omicini, 2001] Mirko Viroli and Andrea Omicini. Multi-agent systems as composition of observable systems. In *WOA 2001 – Dagli oggetti agli agenti: tendenze evolutive dei sistemi software*, Modena, Italy, 4–5 September 2001. Pitagora Editrice Bologna.

[Viroli and Omicini, 2002a] Mirko Viroli and Andrea Omicini. Specifying agents' observable behaviour. In *1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002)*, Bologna, Italy, 15–19 July 2002. ACM.

[Viroli and Omicini, 2002b] Mirko Viroli and Andrea Omicini. Towards an alternative semantics for FIPA ACL. In *16th European Meeting on Cybernetics and Systems Research (EMCRS 2002), 3rd International Symposium "From Agent Theory to Agent Implementation" (AT2AI-3)*, Vienna, Austria, 3–5 April 2002.

[Viroli and Omicini, 2002c] Mirko Viroli and Andrea Omicini. Tuple-based models in the observation framework. In Farhad Arbab and Carolyn Talcott, editors, *Coordination Languages and Models*, volume 2315 of *LNCS*, pages 364–379. Springer-Verlag, 2002. 5th International Conference (COORDINATION 2002), 8–11 April 2002, York, UK, Proceedings.

[Viroli *et al.*, 2001] Mirko Viroli, Gianluca Moro, and Andrea Omicini. On observation as a coordination pattern: An ontology and a formal framework. In *16th ACM Symposium on Applied Computing (SAC 2001)*, pages 166–175, Las Vegas (NV), 11–14 March 2001. ACM. Track on Coordination Models, Languages and Applications.

[Wegner, 1997] Peter Wegner. Why interaction is more powerful than computing. *Communications of the ACM*, 40(5):80–91, May 1997.

[Wooldridge, 2000] Michael Wooldridge. *Reasoning about Rational Agents*. The MIT Press, 2000.