Tag Interactions in Multi-Agent Systems: Environment Support

Eric Platon[1], Nicolas Sabouret[2], and Shinichi Honiden[1]

[1] National Institute of Informatics, Sokendai, 2-1-2 Hitotsubashi, Chiyoda, 101-8430 Tokyo
[2] Laboratoire d'Informatique de Paris 6, 8, Rue du Capitaine Scott, 75015 Paris
{platon,honiden}@nii.ac.jp, nicolas.sabouret@lip6.fr

Abstract. Tag interactions refer to interactions in which software agents are intentionally or opportunistically involved. Intentional tag interactions allow agents to observe each other, like when one observes the physical condition of others. Opportunistic tag interactions occur when one agent receives information about others without requesting for it, like when one realizes that a friend is ill. Such cases seem natural for human-beings, but software agents need specific support. In this paper we model tag interactions based on the agent *environment* and *computational bodies* to enact, maintain, and regulate their execution. We discuss our model and we identify further issues in the current state of the research. An example application is described in detail to show the potential of introducing tag interactions. Future work aims at addressing the present research issues and to implement the example application to experimentally evaluate the computational and communication cost of tag interactions.

1 Introduction

In Software Multi-Agent Systems (SMAS), interactions are usually message-passing between two agents, so that the main research activities deal with communication languages, interaction protocols, and their exploitation (e.g. negotiation, argumentation). However, systems capabilities appear limited in comparison to interaction opportunities in human agencies [11]. The potential for interaction diversity of software agents seems underexploited, although some situations can leverage interaction schemes with different semantics, such as indirect (e.g. stigmergy, see [14] for a survey), implicit [26], or opportunistic interactions [1]. Typically, one can conclude a friend is ill just due to her appearance, even though she *does not communicate* her state intentionally. How to model such an interaction? To this end, we exploit the idea that human-beings do not only communicate through their languages, but also through their bodies. We name *tag interaction* this type of interactions and propose to endow software agents with an explicit computational body that exposes observable information labeled as tags.

The management of tag interaction based on a software body differs from message-passing techniques. The source agent is not always aware of the information it expresses and it cannot always be considered as the 'operational sender' (e.g. my ill friend). Similarly, agents that receive information via tag interaction do not always know they play the role of receivers (e.g. realizing my friend is ill). Consequently, tag interaction requires an active entity that allows and executes such situations by dealing with the software body. The scope of this entity must encompass all agents that can participate in tag interaction and we think a natural candidate for this function is a *computational environment.* This paper describes how the environment supports tag interaction with intuitive examples from social and natural sources, and a detailed example application.

Section 2 first details our terminology and it demonstrates that an environment is required by tag interaction. Section 3 then exposes our formalization of the notion of tag interaction and environment. Section 4 exploits this formalization to describe environmental mechanisms. Section 5 exposes an example application devoted to an agent-based load balancing problem. Section 6 discusses the current state of our work, and section 7 summarizes our current plans for future improvements.

2 Tag interaction and Environment

2.1 Software Agent for Tag interaction

In the literature, a software agent is thought of as an autonomous computational entity with interactive capabilities [22, 5]. Our definition in the frame hereafter is related but separates explicitly the interactive nature of agents from their internals as shown on Fig. 1.

A software agent is an autonomous problem-solving entity endowed with an explicit boundary named *softbody* that exposes:

Sensors to receive information from the environment
Actuators to send information to the environment
A *public state* of the agent, observable in the environment

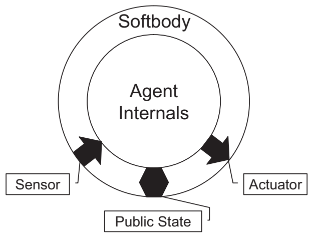


Fig. 1. A Software Agent.

The problem-solving capabilities of the agent internals refer to the various existing agent architectures, like BDI [18] and KGP [8]. Internals are usually hidden from other agents in SMAS. The boundary delimits the agent and it captures the interactive capabilities with other entities in the system. We refer to this boundary as the *softbody* of software agents. This computational body features sensors and actuators for interactions with other agents and entities in the SMAS [22].

In addition, the softbody exposes a *public state* of the agent. A public state exposes information about the agent that can be sensed by other agents in the environment. Concretely, the public state is a list of variables named *tags* whose values reflect agent internals. The contents and types of information in the public state can be configured by the designer or dynamically by the system to define what can be observed about each agent. For example, we can infer in a discussion that someone may be lying when we observe her to blush, i.e. her public state exposes a change to a 'red skin color'. On the other hand, system designers might choose to prevent such an observation in a software auction system to avoid collusion means by body signals [21]. Public state content information can indeed influence agent reasoning processes and consequently their interactions.

In the remainder of the paper, we will now refer to *software* agents simply as *agent* for readability concerns, although our model does not pretend to be generalized to any kind of agent.

2.2 Tag interaction

Definition We define tag interaction as follows;

Tag interaction is an ensemble of mechanisms that model and enact in a SMAS: (a) the expression of agent public state, (b) the sensing of agent public state, and (c) the intentional and opportunistic cases. All tag interaction mechanisms are performed and controlled by a SMAS environment.

Tag interaction mechanisms expose and publish changes of public states in (a) and (b), so that agents are notified about such events. We distinguish two types of tag interaction to describe different situations in (c). *Intentional tag interaction* is related to observation, when an agent's mental state is to collect information about others. This active inquiry mode is initiated by an agent and contrasts with the 'passive' reception mode of *opportunistic tag interaction.* In that case, an agent receives information about the public states of other agents without explicitly requesting for it. For instance, one can feel a presence in a dark room through senses, even though we do not request this information in the first place. Fig. 2 depicts the two types of tag interaction for the public state.

The left part of Fig. 2 shows the observing agent acting (e.g. looking at) so that to sense the public state of the observed agent. The right part shows that the public state of an observed agent is spread out to the sensor of any 'passing agent' (e.g. mobile). We argued in section 1 that these two situations need an active third-party entity to perform the information flows represented by the arrows of the two above figures, and the natural candidate is the environment.

Environmental Requirements Observing an agent means the public state is 'readable', such as on the left of Fig. 2. But in usual agent approaches, reading

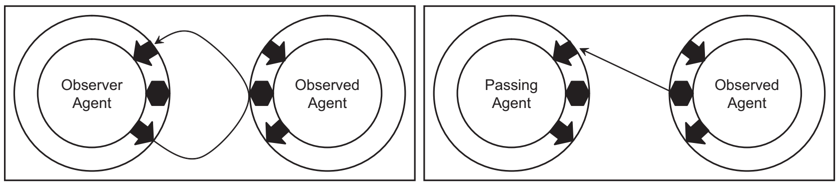


Fig. 2. Left: Intentional tag interaction of the public state. Right: Opportunistic tag interaction of the public state.

actions are realized by contacting the observed agent that consequently becomes aware of this observation. In a discussion, we might however talk and simultaneously observe interlocutors to detect clues such as attitudes. The observation requires for the observing agent to *receive* information, instead of to request for it, and the environment can perform this function.

Second, opportunistic tag interaction emphasizes the previous argument, since no agent triggers any information transfer, as on the right of Fig. 2 where the environment delivers public state information. Typically, a working place tends to foster members to focus on their tasks as a global effect: the environment carries 'implicit messages' about the ambiance to any agent.

Finally, tag interaction requires an environment as a *regulating entity* [27]. The public state is a feature of the softbody that lets malicious agents fake attitudes if no regulation is enforced. As a general example, the public state 'position' should be reliable if agents are to have a fair ordering in a queue. The environment can enforce correctness of public states against system rules (under the hypothesis of a perfectly functioning environment).

To summarize, tag interaction requires from the environment the following support, and our model aims at providing them.

- Enact intentional interactions (observation)
- Enact opportunistic interactions
- Regulate interactions

This support needs to be provided transparently to avoid introducing into the agent any complexity related to the environment responsibilities. The model abstracts this issue at this point by only specifying what are the environment responsibilities. Implementation issues lie at another level of analysis. The environment might be centralized or distributed depending on application requirements. This matter is further discussed in section 6.

2.3 Environment Model

Our definition of environment specializes a generic version [28], notably based on the work of Russell and Norvig [22], and Ferber [5]. An environment for SMAS is a first-class entity to encapsulate part of the application logic that cannot be handled by agents, such as system-wide regulation, resource access, and interaction mediation [27]. Our definition emphasizes the characteristics that matter for tag interaction.

The environment of a SMAS is the entity where agents exist and that:

- Maintains the system topology
- Maintains mapping information of the agent population to the topology
- Performs tag interaction mechanisms
- Defines and enforces tag interaction rules

The environment is a stateful entity that defines and maintains a topology of the system, which can be spatial as in many simulations [10], network domains, file systems, or web-sites [2]. The environment also maintains information about agent situations in the system to manage information delivery and regulate their interactions. This information is only related to softbodies. That is, the environment does not need to deal with agent internals, since these are encapsulated into agents. Furthermore, the environment mediates tag interaction, which refers to public state evolution and change notification. On the one hand, the environment applies rules in the SMAS to enforce certain public state values. Rules define ranges of possible values, so that system states remain consistent for the application. On the other hand, rules define how public state information is spread out in the system, for instance by defining a range of interaction [12, 17]. These rules allow specifying change notification strategies to control the amount of tag information that is exchanged in the system (which can cause a significant cost variation as illustrated with the example application of section 5).

The definition meets the requirements we defined for tag interaction.

- Interaction mediation enacts an observation framework whereby the environment delivers observable events.
- Opportunistic events delivery is configurable by specific rules.
- Regulation is enforced by the rules while mediating interactions.

We notice that the environment has no deliberative capability and no decision power. The environment merely accomplishes its responsibilities in strict compliance with rules defined by design.

3 Formalization

3.1 Agent

Our formalization of an agent follows from the definition of section 2.1:

$$Agent = (\psi, \varphi, INF)\text{ , where } \varphi = (\mathcal{S}, \mathcal{A}, \mathcal{P}_s) \tag{1}$$

First in this formula, ψ denotes the problem-solving abilities of the agent (its internals), and φ is its softbody. A pair (ψ, φ) then represents a state of the agent. The softbody φ is further developed into a 3-tuple where $\mathcal{S}$ is the set of sensors of the agent, $\mathcal{A}$ the set of actuators, and $\mathcal{P}_s$ the *public state*, which can be a set of variables, e.g. in predicate logic.

The last element of the formalization is INF, a set of two reaction rules INF_ψ and INF_φ to determine the evolution of agent states (ψ, φ) on change of internals or softbody respectively. For any ψ and φ:

$$\frac{(\psi, \varphi)\ and\ \psi \rightarrow \psi'}{\varphi \rightarrow \varphi'} INF_\psi \tag{2}$$

$$\frac{(\psi, \varphi)\ and\ \varphi \rightarrow \varphi'}{\psi \rightarrow \psi'} INF_\varphi \tag{3}$$

The operational semantics of INF_ψ expresses the evolution of the state (ψ, φ) after the evolution from ψ to ψ'. The result of INF_ψ is the evolution of the softbody to reach φ'. The agent final state is therefore the pair (ψ', φ'). For instance, when an agent wants to open a door, it first *intends* (ψ evolution) and then *acts* (φ evolution) to complete its intention. INF_φ expresses similarly the evolution of internals due to the evolution of the softbody, e.g. input on sensors. INF operators explicate *how* internals and softbody are linked by a cause to consequence relation. *What* is modified along the evolution is application-dependent and relies on instances of our model.

3.2 Environment

Our formalization of the environment follows from the definition of section 2.3:

$$Environment = (\Omega, \Phi, TRANS) \tag{4}$$

First, Ω is a 2-tuple, representing the environment internals:

$$\Omega = (\mathcal{T}opology, \mathcal{R}ules) \tag{5}$$

where $\mathcal{T}opology$ describes the structure (possible dynamic) of the system, e.g. the ground in simulations or the hyperlink network of a web-site [2]. Agents are situated in this topology to define their neighborhood. Management of tag interactions by the environment is performed according to this topology. $\mathcal{R}ules$ is the set of rules that define how the environment executes agent interactions.

Back to (4), Φ is the set of all softbodies in the system. The environment exploits softbodies to serve the population of agents (e.g. information delivery) and to enforce system rules by imposing environmental regulation. Consequently in our model, each softbody is *owned* and *controlled* by an agent in (1), and the control is *regulated* by the environment in (4). The regulation lets agents control their softbodies, but the effect in the system of this control is bounded by the environment rules. Only softbodies are required to enable the environment functions, since the softbody synthesizes public features of agents. A complete state of the environment is then a pair (Ω, Φ).

We finally introduce $TRANS$, a set of reaction rules $TRANS_{\Omega}$ and $TRANS_{\varphi}$ to model the regulation mechanism of the environment over agent softbodies. For any Φ and Ω:

$$\frac{(\Omega, \Phi) \textit{ and } \Omega \rightarrow \Omega'}{\exists A \subset \Phi :\ \forall \varphi \in A,\ \varphi \rightarrow \varphi'} TRANS_{\Omega} \tag{6}$$

$$\frac{(\Omega, \Phi) \textit{ and } A \subset \Phi :\ \forall \varphi \in A,\ \varphi \rightarrow \varphi'}{\Omega \rightarrow \Omega'} TRANS_{\Phi} \tag{7}$$

From an environment state (Ω, Φ) and an evolution of the internals Ω to Ω', $TRANS_{\Omega}$ causes the set of softbodies Φ to evolve, so that for softbodies in the subset $A = (\varphi_1, ..., \varphi_n) \in \Phi^n$, the transformation $TRANS_{\Omega}$ entails all φ_i to evolve to some $\varphi'_i \in \Phi'$. Similarly, $TRANS_{\Phi}$ models the converse transformation. The subset A depends on the $\mathcal{T}opology$ and typically contains a 'neighborhood' of agents defined by application-dependent needs, such as an Euclidean or social (same taste, *etc.*) distance. In particular, A can be empty (e.g. only one agent in the SMAS) or the entire set of softbodies (e.g. only two agents in a virtual shop).

4 Environment Mechanisms for Tag interaction

4.1 Agent Influence on the Environment

When an agent intends to execute an action (i.e. change own public state or observe) in the BDI sense [18], its internals evolve from ψ_{init} (intention selection) to ψ_{act} (intention attempt). The agent is initially in a state $(\psi_{init}, \varphi_{init})$, so that the change of internals causes the softbody to evolve due to the INF_{ψ} reaction rule. The softbody consequently evolves to φ_{act}:

$$\frac{(\psi_{init}, \varphi_{init}) \textit{ and } \psi_{init} \rightarrow \psi_{act}}{\varphi_{init} \rightarrow \varphi_{act}} INF_{\psi} \tag{8}$$

The modification of the softbody entails a reaction on the environment with the $TRANS_{\Phi}$ rule, from Ω_{init} to Ω_{check}:

$$\frac{(\Omega_{init}, \Phi) \textit{ and } \varphi_{init} \rightarrow \varphi_{act}}{\Omega_{init} \rightarrow \Omega_{check}} TRANS_{\Phi} \tag{9}$$

The environment then checks whether its new state is valid according to applicable rules. If so, it continues the action by applying either process in sections 4.2 and 4.3 and completes by informing the source agent. A successful action entails Ω_{ok}. Success is observed with the softbody that becomes φ_{ok} under the application of $TRANS_{\Omega}$.

$$\frac{(\Omega_{check}, \Phi) \textit{ and } \Omega_{check} \rightarrow \Omega_{ok}}{\varphi_{act} \rightarrow \varphi_{ok}} TRANS_{\Omega} \tag{10}$$

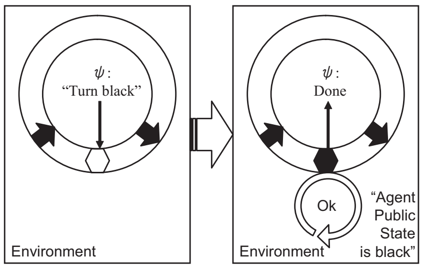


Fig. 3. Environment validates and commits the influence.

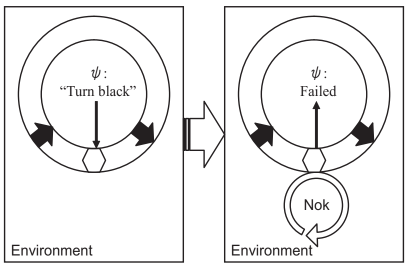


Fig. 4. Environment prevents the influence.

Fig. 3 shows the case where a public state is turned from 'white' to 'black' successfully.

In case rules oppose the agent's intention, the environment evolves from Ω_{check} to Ω_{nok} and counter-balances the agent attempt.

$$\frac{(\Omega_{check}, \Phi) \ and \ \Omega_{check} \rightarrow \Omega_{nok}}{\varphi_{act} \rightarrow \varphi_{fail}} TRANS_{\Omega} \tag{11}$$

Fig. 4 shows a case where the agent's intention to turn its public state to black has failed. Formula (11) follows formula (9) and cancels the agent's action before occurring in the system (e.g. if one wants to push a wall, the body does not move). In the end, the softbody influences back the agent internals with INF_{φ} to 'report' the opposition of the environment.

4.2 Environmental Effect on Agents

The environment acts on agents with the mechanisms described in the previous section. However, the environment cannot be overruled by agents, since the environment *is* the rule, so that the sequence of reaction rules differs. With the same notations as before:

$$\frac{(\Omega_{init}, \Phi) \ and \ \Omega_{init} \rightarrow \Omega_{act}}{\varphi_{init} \rightarrow \varphi_{update}} TRANS_{\Omega} \tag{12}$$

The environment acts on the softbody that evolves to φ_{update}, representing either a change of public state (Fig. 5) or an observation received on sensors. Then, the softbody informs its agent internals:

$$\frac{(\psi_{init}, \varphi_{init}) \ and \ \varphi_{init} \rightarrow \varphi_{update}}{\psi_{init} \rightarrow \psi_{update}} INF_{\varphi} \tag{13}$$

At this point, the agent *cannot* reverse the process, since it is an environmental effect. Such a situation is shown on Fig. 5.

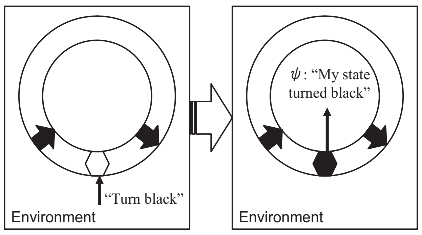


Fig. 5. Environmental effect on the agent public state.

However, some agents may *ignore* the environmental effect, but this is an epistemological issue at the agent internals level. Typically, we can ignore someone is stepping on our feet on a crowded train, either because we are too concentrated on a specific task (agent busy and unaware of the fact) or because we just think it is not worth discussing and decide to move on (agent aware of the fact).

Although the agent cannot oppose the environment's action in the first place, it can react afterward. The agent can take subsequent actions to attempt to modify the environment, and such behaviors are governed by the same regulation sequence as in 4.1. We can illustrate such situations with someone entering a river stream. The stream has an overwhelming strength at first and it carries the swimmer downstream. In reply, swimmers can try to oppose the stream and may succeed in crossing it if their swimming abilities and strength are sufficient.

4.3 Public States Spread Management

Given an agent, three types of events imply a spread of the public state in the environment, namely modification of the public state by the agent, other agents' modification attempts through the environment, and environmental dynamics (such as wind and gravity). Fig. 6 illustrates the case where an agent modifies its public state (left part). Validation of the modification by the environment (central part) is followed by the publication of the resulting state to agents in the neighborhood defined by the topology (right part). We describe hereafter how to process the public state spread management in the three aforementioned cases.

Agent modification Each agent controls its softbody and can modify the public state, under regulation by the environment. The procedure of modification is initially the same as the successful agent influence on the environment detailed in section 4.1. The sequential application of formulas (8) and (9) modifies the public state of the agent and validates it by the environment. Then, acknowledgment of the conformance of the new public state is performed with the publication of the modification in the neighborhood:

$$\frac{(\Omega_{check}, \Phi) \ and \ \Omega_{check} \rightarrow \Omega_{ok}}{\varphi_{act} \rightarrow \varphi_{ok}, \ \forall \varphi \in A, \ \varphi \rightarrow \varphi_{news}} TRANS_{\Omega} \tag{14}$$

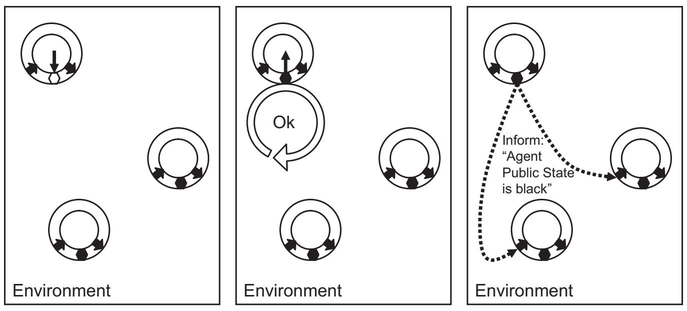


Fig. 6. Public State Management from left to right: the top-left agent modifies its public state; the environment validates the change; the change is spread to neighbors.

A is included in the set of softbodies $\Phi \setminus \{\varphi_{act}\}$. Each softbody in A receives a notification of '$\varphi_{act} \rightarrow \varphi_{ok}$' on its sensors. Formula (14) is a generalization of formula (10) that only stated the successful completion of public state change without publication (case where the agent is alone in the system).

Environmental dynamics Environmental dynamics apply to subsets of agents. Typically, Archimedes' Law applies to agents under water in a simulation, while a clock interrupt to represent time concerns all agents in the system. The application of environmental dynamics on public state follows the procedure of section 4.2. Each dynamics corresponds to an environmental rule set that targets a particular type of public state variable. If p in the public state of a softbody is the target of a rule, p is assigned a new value p' after application of the reaction operator $TRANS_{\Omega}$. Environment and softbody are consequently updated, and the new value p' is spread in the system. The corresponding formula is a generalization of formula (12) that modifies all softbodies, similarly to (14).

Other agents' attempts When an agent intends to act on another agent to modify its public state (e.g. push an agent to change its position), the interaction is mediated by the environment. The procedure begins with a source agent that intends to act on another's public state. The intention modifies with INF_{ψ} the softbody that acts in turn on the environment with $TRANS_{\Phi}$. If the action is authorized in the system, the environment reaction is three-fold by applying the action to the target agent, publishing the action to other agents, and sending an acknowledgment to the source agent:

$$\frac{(\Omega_{check},\Phi)\ and\ \Omega_{check}\rightarrow\Omega_{ok}}{\varphi^{s}_{act}\rightarrow\varphi^{s}_{ok},\ \varphi^{t}\rightarrow\varphi^{t}_{changed},\ \forall\varphi\in A,\ \varphi\rightarrow\varphi_{news}} TRANS_{\Omega} \tag{15}$$

A is the set of softbodies $\Phi \setminus \{\varphi^{s}_{act}, \varphi^{t}\}$, where φ^{s}_{act} is the softbody of the source agent and φ^{t} the one of the target agent. In the end, each softbody informs its internals with INF_{φ}, so that agents are informed about the action. In particular, the target agent can react to this action.

5 Tag interactions applied to a fault tolerance scenario

In this section, we describe an agent-based application with 'classical' and tag interaction approaches. The application is a load-balancing scenario that requires a fault tolerance mechanism to support the activity of the system. We detail the application, the different approaches, and a run of the system where fault tolerance is required in the case of the 'agent death' [9].

5.1 The load-balancing scenario

In this scenario, a client (e.g. the user) submits a set of tasks to the system. The role of the system is to perform the tasks and to report the results to the client. All tasks have same importance, they are independent, and they must all be completed. In other words, we suppose that tasks are not ordered and there is no time constraint to complete them. The base architecture of this scenario is a task repository where clients submit their tasks and wait for the results. The repository is concretely an indexed queue where tasks have three states, namely `todo`, `doing`, and `done`. The system must perform the tasks marked `todo`, signal tasks under process with `doing`, and mark completed tasks with `done` so that the client can take them. The index of the queue points to the next task marked with `todo`. The index approach is mainly introduced as a mean to simplify the task selection process and to ensure mutually exclusive access to tasks when necessary. Different elements[3] that access the repository can only choose to perform the task pointed by the index (which is then automatically updated to the next task to be done).

A classical approach in MAS is a supply-chain where a supervisor agent delegates the tasks in the repository to worker agents, it collects the results, and it updates the task state. The delegation by the supervisor is the load-balancing mean of this approach: Tasks are allocated to available workers. The usual schema of this application is depicted on figure 7. Worker agents interact with the supervisor to ask for tasks and return their results. Their internal state cycle is shown on figure 8. A typical run of the cycle is to receive a task, do it, and inform the supervisor about the result.

A typical problem in this application occurs when a worker fails. Without any specific mechanism to deal with this issue, the supervisor agent may wait indefinitely the completion of the task, while the worker agent is 'dead', i.e. the underlying process anomalously stopped. One solution to this issue is to introduce a system-wide clock and timeouts. Each delegation of a task is completed with a rendezvous timeout by which the worker agent must finish the task. Beyond the timeout, the supervisor considers the worker has encountered a problem and cannot complete the task. The task is then assigned once more to an available worker.

[3] I.e. agents in the following. This description tries to remain general in terms of Software Engineering.

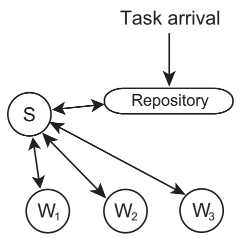


Fig. 7. Supervisor-worker approach to the load-balancing problem.

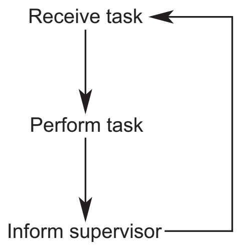


Fig. 8. Internal state cycle of worker agents. The semantics of the arrow is the transition from one label to the other, once the associated action is completed.

Some problems of this approach are the supervisor failure point and the limited flexibility. Distributed computing techniques exist to recover from supervisor failures (e.g. the 'primary-backup' architecture [24]) but they maintain a centralization point that may cause performance drops in case of failures. Workers cannot interact directly to recover locally from some issues without the supervisor. The flexibility of a MAS approach may be improved.

5.2 Tag interaction approach

The purpose of using tag interactions in the load-balancing scenario is to improve the flexibility of the application by removing the supervisor role and opting for a fully distributed approach. Tanenbaum explains that fully distributed approaches are ideally 'better' than centralized ones, but they are also often less efficient [24]. In our case, experiments to evaluate the performance of the approach is let for future work. We focus in this paper on describing how tag interactions can improve flexibility in the fault tolerance mechanisms of the system.

Figure 9 shows the system workers and their tags. Sensors and actuators on the softbody are simply message boxes to communicate in the environment and they are not represented for clarity reasons. Workers are endowed with two types of tags in their public state. `Ref` is the reference to the task the agent is executing (agents get a reference to the task in the object-oriented meaning). `ECD` (Expected Completion Time) is the second tag to indicate the time by which the worker expects to complete the task it is performing.

The internal state machine of the workers is more complex due to the removal of the supervisor role. Figure 10 shows a cycle with a disjunctive branch. Workers start by 'looking around' in the environment about tags on agents in their neighborhood. If all `ECD` tags are future dates, the worker gets a new task from the repository. If an `ECD` tag is past, it means the corresponding agent is considered as dead and the worker prehends its task by reading the `Ref` tag. The next step in the cycle is for the worker to update its own tags to let others know

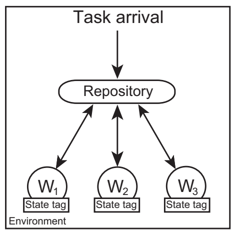


Fig. 9. Tag interaction approach to the load-balancing problem.

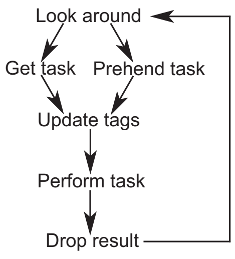


Fig. 10. Internal state cycle of worker agents with tag interactions. The semantics of the arrow is the same as figure 8.

about its current state. The worker then performs the task and drops the result in the repository.

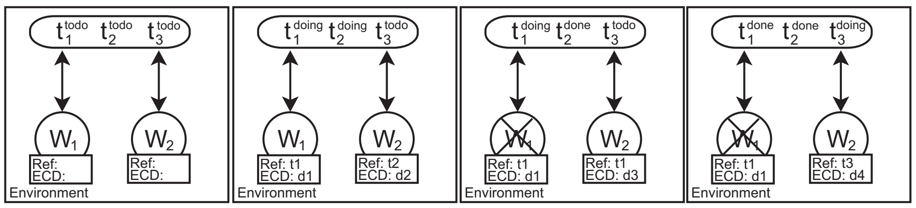
The role of the environment is to support the tag interaction processes. It publishes any change in agent public states and it enforces rules that can depend on the task type. One general rule serves to maintain the system consistency: Agents can only publish `Ref` tags that correspond to references to existing tasks in the repository, i.e. the environment checks that tag values belong to the set of task references.

Notice that such a tag interaction process is arguably very costly if the environment is to publish any change to all workers in the system. This is the reason why the topology of the environment has to constrain the spread of change publication by the definition of the worker neighborhood. We chose to place two agents in the neighborhood of each worker. In terms of spatial metaphor, workers are like equally distributed over a circle. The environment publishes the public state change of a worker to the two agents in its neighborhood, which significantly reduces the theoretical communication cost of tag interactions (from $O(n^2)$ to $O(n)$, where n is the number of workers).

5.3 System run with fault tolerance

Tag interactions allow to decentralize the fault tolerance aspect of the application as can be observed in the following run of the application. Initially, the client places three tasks (t_1, t_2, t_3) in the repository and there are two workers in the system (w_1 and w_2). Figure 11 represents a run where w_1 fails and w_2 exploits the tag information to automatically recover the failure.

The worker w_1 looks around for tags, but does not notice any problem (initial stage), so it gets t_1 and starts performing it with a public state (`Ref`, t_1) and (`ECD`, d_1), where d_1 is a date in the future. The second worker w_2 does the same and performs t_2 with the corresponding public state where $d_2 > d_1$. During the performance, w_1 fails and w_2 completes its current task. The worker w_2 can start a new cycle and looks around. As $d_2 > d_1$, w_2 can deduce that w_1 is dead, so that



t_1^{todo} t_2^{todo} t_3^{todo}
W_1
W_2
Ref:
ECD:
Environment
t_1^{doing} t_2^{doing} t_3^{todo}
Ref: t1
ECD: d1
Ref: t2
ECD: d2
t_1^{doing} t_2^{done} t_3^{todo}
Ref: t1
ECD: d3
t_1^{done} t_2^{done} t_3^{doing}
Ref: t3
ECD: d4

Fig. 11. Sequence of system states in the considered run.

it prehends the task t_1 and performs it. In the last cycle, w_2 looks around but no problem exist in the neighborhood, so that it performs the last task t_3. The system request is then completed, despite one fault that has been automatically recovered.

6 Discussion

6.1 Applications

A criteria of an application that lends itself to tag interactions is *observation*. If observation is an essential characteristic of agents in an application, then tag interactions can be relevant. Also, the load balancing scenario presented in section 5 shows how tag-based observation can contribute to the flexibility of software that rely on more classical schemes (e.g. the supervisor approach). Applications that could leverage tag interaction are first simulations that need more realistic interaction patterns encountered in nature, as we illustrated in this paper. In particular, the environment plays an important role in simulations and the introduction of tag interaction can rely on extending existing work.

Beyond simulation, tag interactions are useful in some electronic market types, where interaction opportunities are a critical factor. Auction systems (see [25] for a survey) and normative agencies such as Electronic Institutions [4] are active research areas that rely explicitly on direct interactions, and tag interaction can introduce more flexibility. We described such an application to show such a flexibility, and tag interactions contribute positively to the system performance [15]. Buyer agents expose the item types they want as public state. While they are actively searching appropriate sellers (e.g. with the directory facility of the FIPA), interested sellers can detect the public state of buyers (observation) and take the initiative to contact them, thus leading to a more dynamic market. Regulation from the environment implements and enforces functions of Electronic Institution to state what is authorized in the market, e.g. ranges of authorized market prices.

6.2 Present Issues

Various improvements can be applied to the current model of tag interaction. In previous work [16], agents could interact with others and objects. Although the present paper focuses only on agents, the introduction of the softbody allows us in future work to exploit and generalize our approach to objects, where an object is assimilated to an ‘empty softbody’, i.e. without agent internals. In such a case, the public state would not reflect agent internals but an interface to handle the object, like with coordination artifacts [13] or the interface of web services. Another extension is to deal with simultaneous and concurrent events in the system, whereas we currently treat actions in sequence. In fact, these issues are discussed for many years [5, 28], and we let them open for the moment. One consequence is that current implementations of our model feature a centralized component for the environment, whereas MAS would rather leverage distributed approaches. The model abstracts distribution issues for system design as it seems more convenient to first specify a solution by defining the responsibilities of the environment, and then to analyze appropriate implementation. In the case of a fully distributed approach, further research must be conducted. The work on AGV from Weyns et al. seems an appropriate base solution [29]. Agents are associated to ‘pieces’ of environment that are synchronized when required.

Finally, the issue of environment regulation deserves further studies on how to specify and implement rules. In particular, the issue of ‘conservative rules’ (in a physical sense) seem to be a powerful mean to control the system entropy.

6.3 Related Work

Castelfranchi described the Behavioral Implicit Communication (BIC) in [3] to explain why agents can communicate and coordinate by acting instead of using languages. For example, an escaping prey ‘communicates’ its position to the predator only by moving, when it would prefer avoiding it. However, the prey is not explicitly the sender of messages about its position in the sense of usual Agent Communication Languages, i.e. it has no mental attitude of a sender. So who is the sender in the prey example? BIC explains this issue by identifying a *power of observation* that allows the observer to ‘pull’ messages. Even though the prey does not communicate its position, the predator fetches the information by observation. BIC has been related to the environment of agents as the source whereby observers can get information about others. The further stage we aimed at in this paper is to enact the observation of agents (the public state) in a concrete interaction model.

In the present paper, we proposed to exploit embodiment in tag interaction among machines, and thus allow software agents using their body and information about others. Our proposal can be compared to the position of Kushmerick [11], with more emphasis on a model that can be engineered and the environment. To our knowledge, little work has discussed the exploitation of a body in the interactions among software agents, in spite of the potential benefit of such an approach. Wrapper agents are usually interfaces to legacy programs or they add some functionalities such as code mobility. Such wrappers are not entities that reflect the state of the wrapped entity, so that we think it differs from the softbody. The KGP model of agency is implemented with a stateful body that features sensors and actuators [23, 8]. The state of this body collects the sensed information to feed the agent knowledge base about the world. Such state is complementary but differs from our public state which exposes information about the agent itself to other agents in the system.

The public state is related to the 'tags' proposed by Holland in his theory of complex adaptive systems [7]. This theory has been exploited in the work of Hales in biologically-inspired systems [6]. The public state is another version of the tags. The essential difference with the work of Holland and Hales is that the public states is an attempt to use tags as an engineering element in engineering SMAS.

Finally, software environments have been considered in various work to mediate and rule interactions. In particular, the coordination artifacts (CA) of Ricci et al. allow to build such environments [19]. Engineering environments with CA populates SMAS not only with agents, but also with artifacts that mediate interactions among agents (coordination). Although CA can provide an engineering approach to address tag interaction, we think CA cannot cover the notion of body. The Agent Coordination Context of CA [20] shares similar concepts with the softbody, but the ACC is an interface that is lent to agents during interactions, whereas the body belongs to the agent architecture.

7 Conclusion

In this paper, the environment in SMAS is seen as a solution to support tag interaction. The environment provides an adequate abstraction to describe the mechanisms involved in tag interaction and to lead to an implementation framework. Also, tag interaction exemplifies a category of applications that require an environment as first-class entity. We think that the connection between tag interaction and environment is an indication that future development of interaction theories should rely increasingly on the environment as key element.

In future work, we intend to refine this model so as to aim at more flexible and realistic systems. We aim at relating our model to distributed implementation schemes, whereas the current methodology leads to a rather monolithic implementation of the environment. Rules of the environment are also an important research direction with the definition of 'conservative rules'. Such rules may help in defining a metrics of the system entropy to evaluate its states. The example application presented in this paper is the subject of a series of experiments to evaluate the benefit of using tag interactions, compared to its computational and communication costs. In addition, we are studying other application scenarios that leverage tag interaction, so as to identify a candidate methodology for design issues (e.g. the selection of public state). Although the present paper focuses on agents, the softbody can be exploited in future work to model objects, assimilated to an 'empty softbody', to seamlessly represent interactions among agents and objects. Finally, the current mechanisms underlying tag interaction are to be modified to handle action simultaneity and concurrent interactions in the system, which should take advantage of other improvements in the modeling of tag interaction spaces.

Acknowledgment

The authors would like to thank Danny Weyns, Paolo Petta, the teams of AgentLink III and the Jožef Stefan Institute.

References

1. F. Balbo and S. Pinson. Toward a Multi-agent Modelling Approach for Urban Public Transportation Systems. In A. Omicini, P. Petta, and R. Tolksdorf, editors, *Engineering Societies in the Agent World'01*, volume 2203 of *LNAI*, pages 160–174. Springer–Verlag, 2001.
2. S. Bandini, S. Manzoni, and G. Vizzari. Web Sites as Agents' Environments: General Framework and Applications. In D. Weyns, H. V. D. Parunak, and F. Michel, editors, *Proceedings of Environment for Multi–Agent Systems at Autonomous Agents and Multi–Agent Systems'05*, 2005.
3. C. Castelfranchi. Silent Agents: From Observation to Tacit Communication. In G. A. Kaminka, P. Gmytrasiewicz, D. Pynadath, and M. Bauer, editors, *Proceedings of Modeling Others from Observations*, 2004.
4. M. Esteva, J. A. Rodríguez-Aguilar, C. Sierra, P. Garcia, and J. L. Arcos. On the formal specifications of electronic institutions. In F. Dignum and C. Sierra, editors, *AgentLink*, volume 1991 of *Lecture Notes in Computer Science*, pages 126–147. Springer, 2001.
5. J. Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence.* Addison-Wesley, 1999.
6. D. Hales. *Tag Based Co-operation in Artificial Societies.* PhD thesis, University of Essex, England, 2001.
7. J. H. Holland. *Hidden Order: How Adaptation Builds Complexity.* Addison Wesley, 1996.
8. A. C. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. The KGP model of agency. In R. L. de Mántaras and L. Saitta, editors, *ECAI*, pages 33–37. IOS Press, 2004.
9. M. Klein, J. A. Rodríguez-Aguilar, and C. Dellarocas. Using domain-independent exception handling services to enable robust open multi-agent systems: The case of agent death. *Autonomous Agents and Multi-Agent Systems*, 7(1-2):179–189, 2003.
10. F. Klügl, M. Fehler, and R. Herrler. About the Role of the Environment in Multi-agent Simulations. In D. Weyns, H. V. D. Parunak, and F. Michel, editors, *Environment for Multi–Agent Systems*, volume 3374 of *Lecture Notes in Computer Science*, pages 127–149. Springer, 2004.
11. N. Kushmerick. Software agents and their bodies. *Minds and Machines*, 7(2):227–247, 1997.
12. M. Mamei and F. Zambonelli. Motion Coordination in the Quake 3 Arena Environment: a Field-Based Approach. In D. Weyns, H. V. D. Parunak, and F. Michel, editors, *Environment for Multi–Agent Systems'04*, volume 3374 of *LNAI*, pages 264–278. Springer, 2005.
13. A. Omicini, A. Ricci, M. Viroli, C. Castelfranchi, and L. Tummolini. Coordination Artifacts: Environment-based Coordination for Intelligent Agents. In *Autonomous Agents and Multi–Agent Systems*, pages 286–293. ACM Press, 2004.
14. H. V. D. Parunak. Expert Assessment of Human-Human Stigmergy. Analysis for the Canadian Defence Organization, Altarum Institute, Ann Arbor, Michigan, May 2005.

15. E. Platon. Artificial intelligence in the environment: Smart environment for smarter agents in open e-markets. In *Proceedings of the Florida Artificial Intelligence Research Society.* AAAI, 2006.
16. E. Platon, N. Sabouret, and S. Honiden. Overhearing and Direct Interactions: Point of View of an Active Environment, a Preliminary Study. In D. Weyns, H. V. D. Parunak, and F. Michel, editors, *Proceedings of Environment for Multi–Agent Systems at AAMAS'05*, 2005.
17. E. Platon, N. Sabouret, and S. Honiden. Oversensing with a softbody in the environment: Another dimension of observation. In G. A. Kaminka, D. V. Pynadath, and C. W. Geib, editors, *Proceedings of Modeling Others from Observation at IJCAI'05*, 2005.
18. A. S. Rao and M. P. Georgeff. BDI Agents: From Theory to Practice. Technical report, Australian Artificial Intelligence Institute, 1995.
19. A. Ricci and M. Viroli. Coordination Artifacts: A Unifying Abstraction for Engineering Environment–Mediated Coordination in MAS. *Informatica*, 2005.
20. A. Ricci, M. Viroli, and A. Omicini. Environment-based coordination through coordination artifacts. In D. Weyns, H. V. D. Parunak, and F. Michel, editors, *Environment for Multi–Agent Systems'04*, volume 3374 of *LNAI*, pages 190–214. Springer–Verlag, 2005.
21. A. Rogers and N. R. Jennings. Collusion in Agent–Based Systems. *AgentLink News*, 17:6–8, 2005.
22. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, Edition 2003.
23. K. Stathis, A. C. Kakas, W. Lu, N. Demetriou, U. Endriss, and A. Bracciali. PROSOCS: A platform for programming software agents in computational logic. In J.-P. Müller and P. Petta, editors, *Proceedings of 'From Agents Theory to Agent Implementation'*, 2004.
24. A. S. Tanenbaum. *Distributed Operating Systems.* Prentice Hall, 1994.
25. M. Tsvetovatyy, M. Gini, and Z. Mobasher, B.and Wieckowski. MAGMA: An Agent-Based Virtual Market for Electronic Commerce. *Journal of Applied Artificial Intelligence*, 11(6):501–523, 1997.
26. L. Tummolini, C. Castelfranchi, A. Ricci, M. Viroli, and A. Omicini. "Exhibitionists" and "Voyeurs" Do It Better: A Shared Environment for Flexible Coordination with Tacit Messages. In D. Weyns, H. V. D. Parunak, and F. Michel, editors, *Environment for Multi–Agent Systems'04*, volume 3374 of *LNAI*, pages 215–231. Springer–Verlag, 2005.
27. D. Weyns, A. Omicini, and J. Odell. Environment, First-Order Abstraction in Multiagent Systems. *Autonomous Agents and Multi-Agent Systems*, To appear in July 2006.
28. D. Weyns, H. V. D. Parunak, F. Michel, T. Holvoet, and J. Ferber. Environments for Multiagent Systems, State-of-the-Art and Research Challenges. In D. Weyns, H. V. D. Parunak, and F. Michel, editors, *Environment for Multi–Agent Systems'04*, volume 3374 of *LNAI*, pages 1–47. Springer, 2005.
29. D. Weyns, K. Schelfthout, T. Holvoet, and T. Lefever. Decentralized control of e'gv transportation systems. In F. Dignum, S. Kraus, and M. Singh, editors, *AAMAS*, pages 67–74. ACM Press, 2005.