Model for Simultaneous Actions in Situated Multi-agent Systems

Danny Weyns and Tom Holvoet

AgentWise, DistriNet, Department of Computer Science,
K.U.Leuven, B-3001 Heverlee, Belgium
{danny.weyns,tom.holvoet}@cs.kuleuven.ac.be

Abstract. The main focus of multi-agent research so far has been on concepts and techniques to analyze and specify multi-agent systems. Much less attention has been devoted to the implementation of the concepts and techniques. This paper intends to bridge the gap between the mere concept of simultaneous actions and its implementation. Simultaneous actions are actions that are executed by different agents at the same time. We study simultaneous actions in the context of situated multi-agent systems where agents and objects have an explicit position in the environment. To clarify the concept of simultaneous actions, first we propose a classification for simultaneous actions and illustrate each type with examples. Then we present a generic model for simultaneous actions that is independent of the applied agent architecture. Support for simultaneous actions involves two aspects: first it must enable agents to act together and second, it must ensure that the outcome of a combination of simultaneously performed actions is in accordance with the domain that is modeled. In the model, acting together is established through synchronization, while the domain requirements are ensured through reification of actions and subsequently combining the simultaneously performed actions in accordance with the valid laws. We used the model to implement the Packet–World with centralized as well as with regional synchronization. In the paper we illustrate the model for both approaches and discuss the implications for the complexity of implementation, the autonomy of agents and the scalability of the multi-agent system.

1 Introduction

Interaction is a central issue of multi-agent systems (MAS). An interaction occurs whenever two or more agents come into contact with each other. The focus of this paper is on interactions in *situated MASs*. In particular we focus on infrastructure to support the implementation of *simultaneous actions*, i.e. actions performed by different agents at the same time.

Situated Multi-Agent Systems. In situated MASs, agents as well as objects have an explicit position in the environment. Situatedness reflects the local relationships between agents and objects. Through its situatedness, a situated agent is placed in a local context that he is able to perceive and in which he can

M. Schillo et al. (Eds.): MATES 2003, LNAI 2831, pp. 105–118, 2003.
© Springer-Verlag Berlin Heidelberg 2003

act. The model for simultaneous actions we discuss in this paper is independent of the applied architecture of the agents in the MAS.

Model for Action. For actions, we use a model that is based on the theory of influences and reactions to influences, proposed by J. Ferber [4]. Roughly spoken, this theory separates what an agent wants to perform from what actually happens. Agents produce influences into the environment and subsequently the environment reacts by combining the influences to deduce a new state of the world from them. The reification of actions as influences enables the environment to combine simultaneously performed activity in the MAS. Based on this theory, J. Ferber and J.P. Müller developed a model for action with centralized synchronization that is described in [5]. In [13], we extended this model for regional synchronization.

Support for Simultaneous Actions. Simultaneous actions are actions that conceptually happens at the same time, but physically are executed separated in time, e.g. on a single or sequential processor system. Support for simultaneous actions involves two aspects: first such support must enable agents to act together and second, support must ensure that the outcome of a combination of simultaneously performed actions is in accordance with the domain that is modeled.

Outline of the Paper. To clarify what simultaneous actions are, we first propose a classification for simultaneous actions and illustrate each type with examples. Then, in section 3, we present a generic model for simultaneous actions in situated MASs. This model functionally describes how simultaneous actions can be treated towards implementation. Section 4 illustrates the model for centralized and regional synchronization and discuss the implications for both approaches with respect to the complexity of implementation, the autonomy of agents and the scalability of the MAS. Finally, in section 5 we conclude and look at future work.

2 Simultaneous Actions

In this section we elaborate on simultaneous actions. First we present a classification for simultaneous actions and then we illustrate each type of simultaneous actions with examples in the Packet-World.

2.1 Classification of Simultaneous Actions

In the literature, several researchers distinguish between different kinds of simultaneously performed actions. Some examples: Allen and Ferguson [1] differentiate between 'actions that interfere with each other' and 'actions with additional synergistic effects'. Boutilier and Brafman [2] distinguish 'concurrent actions with a positive or negative interacting effect'. Griffiths, Luck and d'Iverno [6] introduce the notions of a 'joint action that a group of agents perform together' and 'concurrent actions, i.e. a set of actions performed at the same time'. These latter

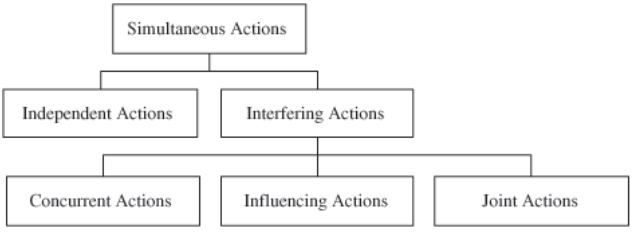


Fig. 1. Classification of Simultaneous Actions.

notions are build upon the concepts of 'strong and weak parallelism' described by Kinny [8].

We propose a classification for actions that happen at the same time as depicted in Fig. 1. We use the common name of *simultaneous actions* as general designation for actions that happen together. Further, we make a distinction between two kinds of simultaneous actions: *independent actions* and *interfering actions*. Independent actions are actions that do not interfere with one another. Interfering actions on the other hand, bring two or more agents directly in contact with each other. Depending on the nature of these interactions, we distinguish between *concurrent actions*, *influencing actions* and *joint actions*. Concurrent actions are of a conflicting nature. The result is typically non-deterministic, e.g. one arbitrary agent of the set of involved agents succeeds in his action while the other agents fail. Influencing actions are actions that positively or negatively affect each other. For this kind of interaction the outcome of the simultaneous actions is the resultant of the individual actions. Joint actions are actions that must be executed together in order to produce a successful joint result. In joint actions agents typically play complementary roles in a compound interaction.

This classification for simultaneous actions takes the viewpoint of the observer of the actions. An observer interprets the interactions as a whole and distinguish types on the basis of the possible outcomes of the interactions. Whether or not the individual agents intend to, or are aware of their participation in the interaction is independent of the classification.

2.2 Examples of Simultaneous Actions

Before we illustrate the different types of simultaneous actions, we first introduce the the example case: the Packet–World.

The Packet–World. The Packet–World[1] consists of a number of different colored packets that are scattered over a rectangular grid. Agents that live in this virtual world have to collect these packets and bring them to their corresponding colored destination. The left part of Fig. 2 shows an example of a Packet–World

[1] The Packet–World is based on an exercise proposed by Huhns and Stephens in [7] as a research topic to investigate the principles of sociality in MASs.

with size 10 in which 10 agents live. Small squares symbolize packets that can be manipulated by one agent, larger rectangles symbolize packets that must be manipulated by two agents and circles symbolize delivery points.

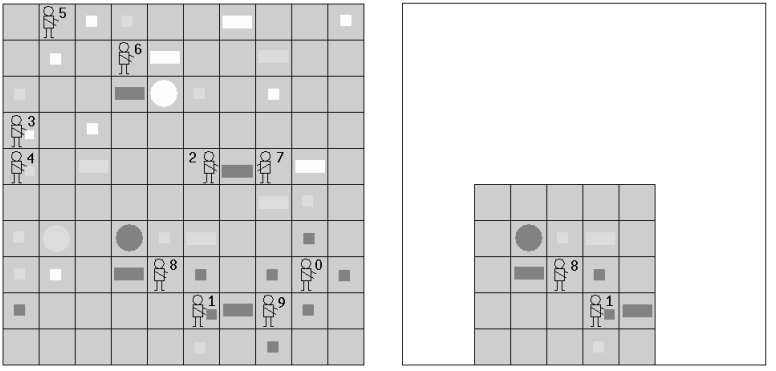


Fig. 2. The Packet–World.

In the Packet–World agents can interact with the environment in a number of ways. An agent can make a step to one of the free neighbor fields around him. If an agent is not carrying any packet, he can pick up a small packet from one of his neighbor fields. An agent can put down a small packet he carries at one of the free neighbor fields around him, or of course at the delivering point of that particular packet. In addition, we allow agents to transfer small packets directly to one another. During such a transfer, the agent that carries the packet passes it to the receiver, while the receiver simultaneously accepts the packet. Agents can also push small packets to a neighboring square. A push only succeeds when there is no obstacle on the destination square of the pushed packet. In case two agents simultaneously push the same packet, the packet moves according to the resultant of both actions. Contrary to small packets, to pick up a large packet two agents have to lift up the packet together, each of them on one short side of the packet. Agents that carry a large packet can only move together in the same direction. Large packets too can be put down at any free field or at the delivering point of the packet. However, to put down a large packet, both agents have to release the packet simultaneously. Finally, when there is no sensible action for an agent to perform, he may wait for a while and do nothing.

Besides acting into the environment, agents can also send messages to each other. Conversations between agents follow a specific protocol. Examples are: a request for information followed by an answer or a refusal; a request to set up a form of cooperation followed by an acceptance to cooperate and later on a notification of the end of the cooperation, or a refusal to cooperate.

It is important to notice that each agent of the Packet–World has only a limited view on the world. The view–size of the world expresses how far, i.e. how

many squares, an agent can 'see' around him. The right part of Fig. 2 illustrates the limited view of agent 8, in this example the view–size is 2.

We monitor the Packet–World via a number of counters that measure the efficiency of the agents in performing their job. There are counters to measure the energy invested by the agents, the message transfer between the agents and the number of conflicts that happens between two agents. The overall performance can be calculated as a weighted sum of this counters. For more details about the Packet-World we refer to [11].

2.3 Examples

Now we illustrate the different types of simultaneous actions in the Packet–World.

An example of *independent actions* are two neighboring agents that make a step to a different location. When in the depicted situation agent 3 decides to step in the direction NE[2] while agent 4 simultaneously decides to step SE, both these actions can happen independent of one another.

An example of *concurrent actions* are two agents that simultaneously try to pick the same small object. Which of the involved agents gets the packet is not determined. When for example in the situation of Fig. 2 agent 5 picks up the packet positioned E to him while agent 6 simultaneously picks up the same packet, one randomly selected agent of the two gets the packet while the other misses it.

When two agents in the Packet–World push the same object at the same time then these are *influencing actions*. If for example in Fig. 2, agent 9 pushes the packet N to him while agent 0 pushes the same packet at the same time, the packet will move on to the square NW to its depicted position. Whether or not this resulting movement is profitable for the individual agents depends on their possible intentions. But in case two agents push the same packet at the same time in opposite directions both will be frustrated since the result of such interaction is that the packet will not move at all.

Finally, in the Packet–World different kinds of *joint actions* are possible. A first example in Fig. 2 is agent 1 who passes the small packet he carries to agent 8. As stated above, such transfer only succeeds when the involved agents act together, i.e. agent 1 has to pass the packet while agent 8 simultaneously has to accept the packet. Another example of joint actions are agents 2 and 7 who make a step with the large packet they carry. Such a step only succeeds when both agents step in the same direction, in the situation of Fig. 2 for example, in the direction SW toward the destination of the packet they carry.

[2] We denote each neighboring field of a field with the first capital letter(s) of the direction from the field to that neighboring field.

3 A Generic Model for Simultaneous Actions

In this section first we give a high level description of the model for simultaneous actions. Subsequently we discuss each layer and the flow between layers in detail. At the end, we reflect on issues with respect to the implementation of the model.

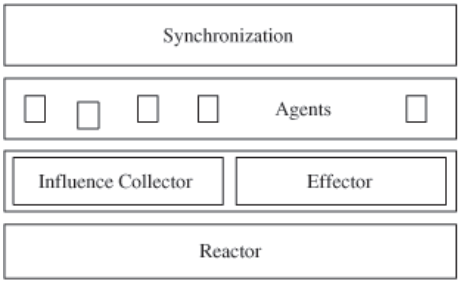


Fig. 3. High level model for simultaneous actions.

3.1 High Level Description of the Model

Fig. 3 gives a graphical description of the model. As stated in section 1, support for simultaneous actions must: (1) enable agents to act together and (2) ensure that the outcome of a combination of simultaneously performed actions is in accordance with the domain that is modeled. Different layers in the model take care of these requirements. At the top we have the *synchronization layer* that accounts for the first requirement: enabling agents to act together. This layer is responsible for composing sets of synchronized agents, i.e. sets of agents that act together. The second layer contains the *agents* of the MAS. The two lowest layers are responsible for the second requirement to support simultaneous actions: combining simultaneously performed actions in accordance with the domain laws. The third layer has a double functionality spread over two components. First this layer contains the *collector* which is responsible for collecting influences and composing sets of influences for simultaneously acting agents. The second component is the *effector* that must ensure that the consequences of the activity of the agents are realized, keeping the state of the environment up to date and bringing the effects of their actions to the agents. The fourth, bottom layer contains the *reactor*. It is the reactor's responsibility to calculate the effects of the influences of simultaneously acting agents according to the actual state of the environment and the laws of the modeled MAS.

3.2 Layers and the Flow between Layers

A detailed overview of the model for simultaneous actions is depicted in Fig. 4. To explain the different layers of the model and the flow amongst the layers,

we follow one action cycle for a particular agent, say A_i. Note that since the model of Fig. 4 is a generic model, several aspects remain abstract. Concrete interpretations of these aspects are discussed in the next section.

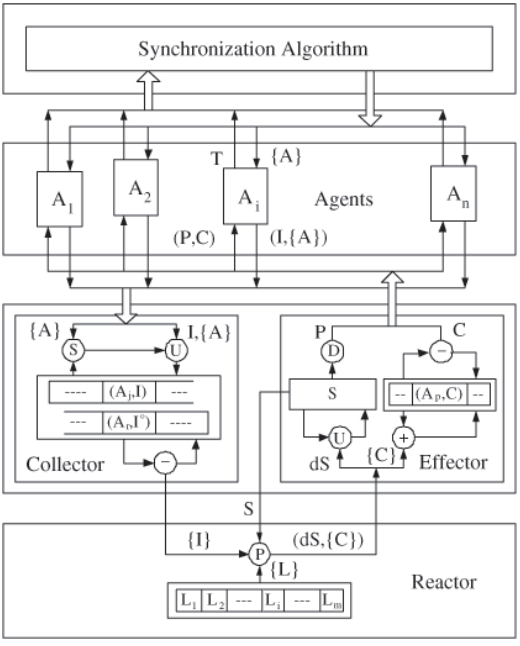


Fig. 4. Detailed model for simultaneous actions.

The cycle starts when agent A_i of the agent layer triggers the synchronization layer to compose a new set of synchronized agents. This request is denoted as T. Depending on the synchronization algorithm implemented by the synchronization layer, T may just be a signal, possibly containing the identity of agent A_i or it may contain more required information, e.g. the set of visible agents of A_i at that moment. As soon as the synchronization algorithm completes, the synchronization layer sends the set of synchronized agents, denoted as $\{A\}$, to agent A_i. Now the agent starts decision making, resulting in the selection of an action. This action is sent as an influence, together with the set of synchronized agents to the environment, denoted as $(I, \{A\})$.

It is the influence collector who collects such tuples. The collector maintains sets of *pending influences*, each set representing the influences produced by one group of simultaneously acting agents. Each element of a set is a tuple (A, I), where A represents an agent and I is the influence produced by A. However, as long as A has not yet produced its influence, I is registered as I^0 denoting that the influence is expected to be produced soon. Based on the set of synchronized agents passed by agent A_i, the collector first searches for a set of pending influences that corresponds to the agents in set $\{A\}$. This functionality is represented

as the encircled S in the model. A set matches if at least one of the agents of $\{A\}$ belongs to the set. If the collector found such a set he updates it, based on the $(I, \{A\})$ tuple. If no set is found, a new set is composed with $(I, \{A\})$ and added to the repository of sets of pending influences. The update of a set of pending influences with the $(I, \{A\})$ tuple is denoted as the encircled U. For each agent of $\{A\}$ that does not belong to the selected set of pending influences, U adds a new entry in the set, initialized as (A, I^0). Other members are left untouched. Finally, U updates the entry of the invoking agent with its actual influence, thus for invoking agent A_i the entry (A_i, I^0) is update to (A_i, I_i), with I_i the influence of the tuple $(I, \{A\})$ invoked by A_i.

As soon as the collector detects that a set of pending influences is completed, he passes the set of corresponding influences, denoted by $\{I\}$, to the reactor. A set of pending influences is completed if all agents of the set have produced their influences, i.e. no tuple in the set contains an initial I^0. Together with passing the set of influences, the collector removes the corresponding set of pending influences from its repository.

In the reactor the set of influences $\{I\}$ is composed with a set of applicable laws, denoted as $\{L\}$, given the current state of the environment denoted as S. This composition is represented by the encircled P. P results in a tuple $(dS, \{C\})$, whereof dS denotes the state changes in the environment, while $\{C\}$ denotes the set of consumptions. A consumption is an element from the environment reserved for a particular agent. When an agent 'consumes' a consumption, the consumed element can be absorbed by the agent (e.g. food that is turned into energy) or the agent may simply hold the element (e.g. a packet he has picked up in the Packet–World).

The reactor passes the tuple $(dS, \{C\})$ to the effector. With dS, the effector updates the environmental state. This update is represented by the encircled U. Furthermore, the effector adds the set of consumptions $\{C\}$ it has received from the reactor to the repository of *pending consumptions* it maintains. Pending consumptions are consumptions that have not yet been picked up by the agents. A pending consumption is a tuple (A, C) whereof C is a consumption intended for agent A.

Subsequently agent A_i can perceive the updated environment and consume the results of its previous action, denoted as (P, C). Since agents have only a limited view on the world, P is only a segment of S. The demarcation of S is represented as the encircled D. Finally, the agent triggers the synchronization layer to produce the next set of synchronized agents for him, starting a new action cycle.

It is important to notice that from the point of view of the agents, the model for simultaneous actions offers *implicit* support for simultaneous actions. Whether the agents are aware of the possible simultaneity of their actions is unimportant for the model. To put it another way, this model *enables* simultaneous actions in situated MASs, however the model does not offer support for the agents to decide about *what* actions they should perform simultaneously. Some agents may follow complex negotiation protocols to agree about the kind

of simultaneous actions they perform, other may simply act based on local perception.

3.3 Issues with Respect to the Implementation of the Model

The layered model presented in the previous section is a generic, conceptual model for simultaneous actions that abstracts from concerns such as scheduling, distribution or fault-tolerance. This model is suitable to guide an implementation of simultaneous actions, e.g. with a framework [9], or even with language technology [10]. As such the reader should be aware that the model only gives a conceptual view on support for the implementation of simultaneous actions. While for example, conceptually the collector layer is accessible for all agents in the MAS, physically the collector layer may be distributed over different hosts and contains one collector for each location where the MAS is deployed. When distribution is required, it should be implemented as a separate concern using available middleware support.

4 Centralized versus Regional Synchronization

In this section we illustrate the generic model for simultaneous actions for two concrete synchronization approaches: centralized and regional synchronization. Since our focus is on infrastructure for simultaneous actions, we do not go into details of the synchronization algorithms. The interested reader is referred to the references in the text.

4.1 Centralized Synchronization

With centralized synchronization, all agents act at one global pace. Synchronization is regulated by one central synchronizer. An example of centralized synchronization is discussed in [3]. Fig. 5 depicts the model for simultaneous actions applied for centralized synchronization. The major advantage of centralized synchronization is simplicity. The synchronization layer contains a collection to store synchronization requests T. Each entry is reserved for a particular agent of the MAS. Initially all entries are set to the initial state, i.e. (A, T^0) for each agent A in the MAS. When an agent A_i sends a request T to synchronize, the synchronizer replaces the corresponding entry to (A_i, T). When *all* agents have sent their request, the synchronizer triggers the agents to act by means of sending them the complete set of synchronized agents. Simultaneously, the collection for synchronization requests is re-initialized. This functionality is represented in the model as the encircled R.

Subsequently, the agents send their influences to the collector. For centralized synchronization the repository of influences is a simple data structure, containing one entry for each agent of the MAS. A start each entry is initialized to (A, I^0). When an agent sends his influence to the collector, the corresponding entry is updated with the passed influence. Only when all agents have sent their

influences, the collector passes the complete set of influences $\{I\}$ to the reactor and re-initializes the influence repository. This functionally of the collector is represented as R in Fig. 5.

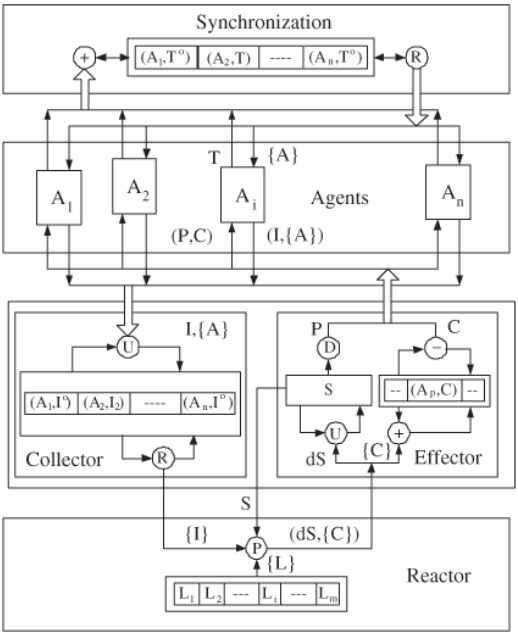


Fig. 5. Model for simultaneous actions with centralized synchronization.

Then the reactor handles the influences and passes the state changes and consumptions to the effector. This latter updates the state of the environment, composes a new set of consumptions and makes them available for the agents. When the agents start a new cycle they perceive the new local state of the environment, and consume the available consumptions from the effector.

Evaluation. The implementation of simultaneous actions with centralized synchronization is rather straightforward. The synchronization layer and influence collector have a simple structure. However, with respect to the acting pace, the autonomy of the agents is a serious problem. Centralized synchronization implies centralized control. All agents act at one global pace, and that ignores the opportunity costs of agents waiting while other agents spend time for decision making. Especially for MASs populated with heterogeneous agents this can be a serious disadvantage. For scalability we have to weigh the cost for collecting influences against the cost of reacting to influences and this in relation to the number of agents in the MAS. With centralized synchronization no search is needed to find the right set of pending influences. However, since the influences for all agents are passed together to the reactor, and since each influence can possibly interfere with any other influence in the set, the complexity to calculate

the reaction of the influences is $O(n^2)$ for a MAS with n agents. Therefore we have to conclude that centralized synchronization scores poorly for scalability of the MAS.

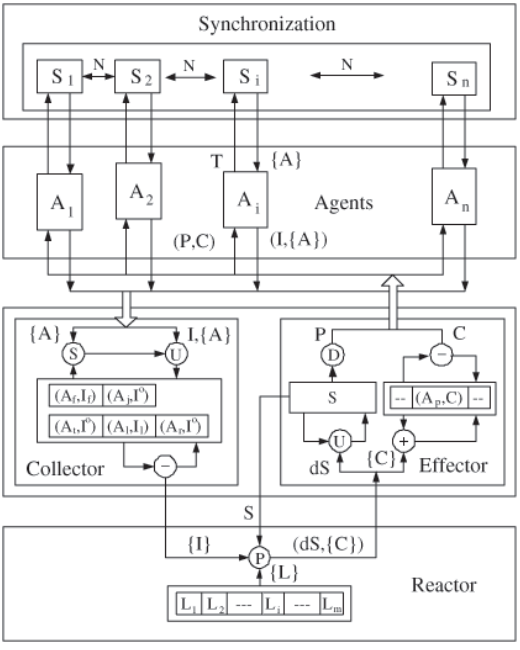


Fig. 6. Model for simultaneous actions with regional synchronization.

4.2 Regional Synchronization

With regional synchronization agents themselves take care of their synchronization. Each agent of the MAS is equipped with a personal synchronizer. Before acting, each agent triggers his synchronizer to synchronize with the agents within his perceptual range. The result of the synchronization process is the formation of independent groups of synchronized agents. The composition of these groups depends on the actual locality of the agents. When agents enter or leave each others perceptual range, the composition of synchronized groups dynamically changes at the same time. For a detailed description of regional synchronization, we refer to [12]. Fig. 6 depicts the model for simultaneous actions applied for regional synchronization. The synchronization layer with regional synchronization is populated with synchronizers S, each synchronizer connected to an agent. Before an agent A_i acts, he sends a request T to his synchronizer S_i to establishing synchronization with his neighboring colleagues. To do so, T must contain the set of agents visible to A_i. A_i deduces this set from his last perception P. Then S_i starts synchronization by requesting the set of visible agents to

synchronize. Synchronization messages are represented by N. Only when negotiation is concluded and a mutual agreement is reached with the synchronized agents, S_i informs his agent A_i to proceed, sending him the set of synchronized agents $\{A\}$.

Then the agent decides about his next action and sends an influence, together with the set of synchronized agents to the collector. Based on this latter set, the collector searches for a matching set of pending influences. Now there are three possible scenario's: (1) the collector can not find a matching set, (2) he finds one such a set or (3) he finds more sets. In case he did not found any set he adds a new set to the repository of sets of pending influences according to $(I, \{A\})$. For the case he found just one set he updates this set. We explain the third scenario (with more then one matching set) by means of an example. Such scenario occurs when a set of regional synchronized agents is composed of different subsets whereof at least two subsets have no agents in common[3]. Suppose that the collector receives the tuple $(I_1, \{A_1, A_{11}, A_6, A_9\})$ sent by A_1. Further, we suppose there are two sets of pending influences, $S_1 = \{(A_7, I_7), (A_{11}, I^0)\}$ and $S_2 = \{(A_9, I_9), (A_{14}, I^0)\}$. Now there is a match of the set of synchronized agents sent by A_1 with both S_1 and S_2. In this case the collector combines all the sets to one resulting set. For the example, the resulting set is $\{(A_7, I_7), (A_{11}, I^0), (A_9, I_9), (A_{14}, I^0), (A_6, I^0), (A_1, I_1)\}$. As soon as the collector detects that a set of pending influences is completed, he removes this set from the repository and passes the set of corresponding influences $\{I\}$ to the reactor.

The reactor composes the influences with the current state of the environment S and the applicable laws $\{L\}$. The resulting state changes dS and the set of consumptions $\{C\}$ are passed to the effector who updates with them the state of the environment and the repository of pending consumptions. Subsequently the agent can perceive the updated state and consume its consumption to start a new action cycle.

Evaluation. The implementation of simultaneous actions with regional synchronization is more complicated than with centralized synchronization. The synchronization layer must implement a non–trivial synchronization algorithm to set up regional synchronization, for details see [12]. Further, the influence collector must provide a dynamic data structure for sets of pending influences. This structure must be maintained in a consistent manner, including possible merges of sets as the one discussed in the example above. Contrary to centralized synchronization, the approach with regional synchronization guarantees much better autonomy for the agents. Since agents only synchronize with their direct neighbors (these are exactly the candidates for simultaneous actions), the pace at which they are able to act only depends on this set of synchronized agents. The price that is paid for this gain is the communication overhead to establish regional synchronization. [12] reports simulation results that compares

[3] For regional synchronization each agent establishes synchronization with only the agents visible to him, so when not all agents of a set of regional synchronized agents see each other, each agent only passes a subset of synchronized agents to the collector.

the pros and cons. Collecting influences is clearly more expensive for the regional synchronization approach. Selecting matching sets of pending influences requires a search through the repository of the collector. In addition, if more then one set is found these sets have to be merged into one compound set. On the other hand, the cost to calculate reaction is much lower than for centralized synchronization. Since only sets of influences on a per region basis are passed to the reactor, the cost for calculating reaction only depends on the size of such sets. Therefore, regional synchronization scales much better then centralized synchronization. Agents that have no colleagues to synchronize with can act asynchronously, while the calculation of the reaction for a group of synchronized agents only depends on the size of the group. This makes the complexity to react to the influences $O(n * r)$ for a MAS populated with n agents that synchronize in clusters with an average size r. As explained in section 4.1, the complexity of the reaction to the influences for centralized synchronization is $O(n^2)$.

Contrary to centralized synchronization where all agents simply act together, with regional synchronization, the possibility for acting together is established as a *natural consequence* of the situatedness of the agents. In other words, agents that are in each others neighborhood are able to perform simultaneous actions. Such support for simultaneous actions can be implemented in a meta–layer, where synchronizers are meta–agents that act on behalf of their associated agents to establish synchronization at the beginning of each action cycle.

5 Conclusion and Future Work

The objective of this paper was to present a model that can bridge the gap between the concept of simultaneous actions and its implementation. To bring order in the range of simultaneous actions we first proposed a classification. This classification takes the viewpoint of the observer, i.e. it distinguishes between different kinds of simultaneous actions based on the way how such actions interfere with one another. To underpin the different types of simultaneous actions, we illustrated each of them with examples in the Packet–World.

Then we presented a generic model for simultaneous actions in situated MASs. This model is independent of the applied agent architecture. The model functionally describes how simultaneous actions in situated MASs can be treated towards implementation. The model is composed of four layers. The top layer is responsible for composing sets of synchronized agents, i.e. agents that act together. The second layer contains the agents in the MAS. The third layer contains the collector who is responsible for composing sets of influences for simultaneous acting agents and the effector that keeps the state of the environment up to date and brings the effects of their actions to the agents. The fourth and final layer contains the reactor that is responsible for calculating the effects of the influences of simultaneous acting agents according to the actual state of the environment and the laws of the modeled MAS.

In the final section we illustrated the model for simultaneous actions for centralized and regional synchronization. The approach with centralized syn-

chronization is the easiest to implement, however the price is poor autonomy for the agents and bad scalability of the MAS. With regional synchronization the implementation is more complex, but this approach results in better autonomy of the agents and better scalability with respect to the number of agents in the MAS.

As a proof of concept, we have implemented the model for simultaneous actions in the Packet–World. We applied the model for centralized and regional synchronization. The interested reader is referred to [11]. The next step is to integrate the experiences with simultaneous actions from the Packet–World into a generic framework for situated MASs. Our goal is then to develop a language to program situated MASs that can be executed on top of this framework. Such a language can extend an existing programming language with different constructs that capture key concepts for situated MASs such as simultaneous actions.

References

1. J. F. Allen and G. Ferguson, *Actions and Events in Interval Temporal Logic*, Journal of Logic and Computation, Special Issue on Actions and Processes, 1994.
2. C. Boutilier and R. I. Brafman, *Partial–Order Planning with Concurrent Interacting Actions*, Journal of Artificial Research 14, 4-2001.
3. J. Ferber, *Multi-Agent Systems, An Introduction to Distributed Artificial Intelligence*, Addison-Wesley, ISBN 0-201-36048-9, Great Britain, 1999.
4. J. Ferber, *Un modele de l'action pour les systemes multi-agents*, Journees sur les systemes multi-agents et l'intelligence artificielle distribue, Voiron, 1994.
5. J. Ferber, and J.P. Müller, *Influences and Reaction: a Model of Situated Multiagent Systems*, Proceedings of ICMAS'96, AAAI Press, Nara, Japan, 1996.
6. N. Griffiths, M. Luck and M. d'Iverno, *Cooperative Plan Annotation through Trust*, Workshop Notes of UKMAS'02, Eds. P. McBurney, M. Wooldridge, UK Workshop on Multi-agent Systems, Liverpool, 2002.
7. M. N. Huhns and L. M. Stephens, *Multi-Agent Systems and Societies of Agents*, G. Weiss ed., Multi-agent Systems, MIT press, 1999.
8. D. Kinny, M .Ljundberg, A. Rao et al., *Planning with Team activity*, MAAMAW'92, LNCS 830, S. Martino al Cimino, Italy, 1992.
9. M. E. Markiewicz, C. J. P. Lucena, *Object Oriented Framework Development*, ACM Press, 2001. See: http://www.acm.org/crossroads/xrds7-4/frameworks.html
10. B. Robben, *Language Technology and Metalevel Architectures for Distributed Objects*, Ph.D, K.U.Leuven, Belgium, ISBN 90-5682-194-6, 1999.
11. D. Weyns and T. Holvoet, *The Packet-World as a Case to Investigate Sociality in Multi-agent Systems*, Demo presented at AAMAS 2002, Bologna, Italy, 2002. For information see: www.cs.kuleuven.ac.be/˜ danny/aamas02demo.html.
12. D. Weyns and T. Holvoet, *Regional Synchronization for Simultaneous Actions in Situated Multi-Agent Systems*, CEEMAS 2003, LNAI 2691 pp. 497–511, Prague, Czech Republic, 2003.
13. D. Weyns and T. Holvoet, *A Model for Situated Multi-Agent Systems with Regional Synchronization*, CE/AMAS, Balkema Publishers, ISBN 90-5809-622-X vol. I, 2th chapter, Madeira, Portugal, 2003.