

Modelling and Analysing Systems of Agents by Agent-aware Transition Systems¹

Marek BEDNARCZYK^{a,2}, Luca BERNARDINELLO^b, Wiesław PAWŁOWSKI^a and
Lucia POMELLO^b

^a*Institute of Computer Science, Polish Academy of Sciences, Gdańsk, Poland, and
Institute of Informatics, the University of Gdańsk, Poland*

^b*Dipartimento di informatica, sistemistica e comunicazione,
Università degli studi di Milano–Bicocca, Italia*

Abstract. We propose a method to specify, in a modular way, complex systems formed by interacting agents. The method is based on the notion of *view*, that is a partial representation of the system, reflecting one of its specific aspects. By composing the different views, we get the overall system, described as a special kind of transition system. By means of a suitable logical language, we can express interesting properties of the system; model-checking techniques can then be used to assess their validity. Views can be specified using different languages or notations, provided they can be translated in so-called agent aware transition systems. The method is explained with the help of a simple, but non trivial example.

Keywords. Transition systems, agents, views, model checking, modularity

Introduction

Large scale data mining is usually performed on data that are distributed in space and heterogeneous in formats. It is therefore natural to think of setting it in a framework allowing for parallel and coordinated action of several agents (see [1]).

The research in this field is active, as witnessed, for instance, by [2], where the reader can find contributions dealing with the specific issues raised by the use of agents in data mining.

In this contribution, we focus on a more general matter, and devise a method for modelling potentially complex systems, formed by “agents” who can interact to perform their tasks. The term *agent* is used here in a very generic way, and simply denotes a component of a system which bears an identity and exists throughout the time span of the system.

Modelling can be used for two reasons: either for designing a new system, or for describing an existing system that we want to analyze. In both cases, we need rigorous

¹Partially supported by PAN and CNR, project CATNET, and by MUR.

²Corresponding Author: Marek Bednarczyk, IPI-PAN Gdańsk; E-mail: m.bednarczyk@ipipan.gda.pl.

techniques allowing us to examine the model, in order to derive information on the behaviour and on the structural and dynamical properties of the real (or to be built) system.

That need requires the use of formal techniques, so that the model be amenable to automatic analysis by means of efficient algorithms. In the present context, our most general reference model is given by *transition systems*, that is structures defined by a set of states and by a set of possible state transitions. Each transition is labelled by the name of an “action”, or event, whose occurrence triggers the change of state. The same action can obviously occur in different states, but generally is constrained by specific preconditions that must be satisfied in a state.

Such transition systems lend themselves to a kind of formal analysis, namely *model-checking*, on which we will return later. Suffice it to say, by now, that such analysis requires the specification of properties to be checked in a logical language, usually based on temporal logics. Specific algorithms examine the state space of the system and check whether the given property is satisfied or not. These algorithms are now very efficient, and allow the analysis of very large state spaces (for an introduction to the subject, see [3]).

Although highly efficient algorithms for model-checking are available, using transition systems as a way to specify or describe real systems runs across a fundamental obstacle, particularly for systems composed of a large number of interacting components, which run in parallel; the problem consists in the so-called *state explosion*: the overall, or global, state of such a system is given, intuitively, by the combination of the local states of all components. The number of possible combinations of local states grows exponentially with the number of components, and can soon become unmanageable.

We are thus led to look for modelling techniques which allow to develop a specification in a modular way. A typical solution would consist in defining a language for specifying single components, and then rules of interaction. This solution has been used, for instance, in association with *process algebras*, languages in which sequential processes (components) can be specified by using different control structures (like sequence, choice, repetition), and then composed by a parallel composition operator (see, for instance, [4], [5], and [6]). A different basic strategy underlies *Petri nets*, where a system is directly defined in terms of a set of local states and a set of local state transitions. Here, components can be recovered as higher level structures within a net (for a wide review on the theory and applications of Petri nets, see [7]).

In this paper, we will follow a different path, based on the idea of *views*. By *view* we mean, intuitively, a sort of projection of the system to be described, as if seen along one of its many dimensions. Such a dimension may coincide with the observation of the system by one of its agents, or components, but is a more general concept. One view might, for instance, express some constraints, like legal or physical constraints, which limit the behaviour of agents; another one might represent spatial relations among agents, which govern their possibility to interact, and the way in which agents can move in space. Still other views can be used to express organizational features of the system.

With this approach, the designers can apply, while developing their models, principles of separation of concerns which help them in managing complexity.

In general, a view is a partial representation of a system. Each view is concerned with only some of the agents and with only some of the actions. Of course, the same agent and the same action can be observed in several views.

A view can be studied in isolation; however, the potential behaviour of a view is in general constrained when the view is composed with other views. Hence, only safety properties (namely, properties expressed by statements of the kind “no *bad* state will be reached”) are preserved after composition. For liveness properties (expressed by statements of the kind “something good will eventually happen”) we need to design views respecting stronger conditions.

Combining all the views, we get the overall system, whose observable behaviour results from superimposing the constraints coming from each view.

This kind of modular representation derives from ideas developed first within the theory of basic Petri nets [8], and later recast in a new formal setting, where agents and their changing hierarchical relations are explicitly represented [9]. The resulting model, called *hypernet*, can be seen as a sort of compact notation for Petri nets. A rather drastic generalization, or rather abstraction, led us to the idea of *agent aware transition system* [10], which is the formal tool described in this paper.

The main aspects of the framework we propose will be introduced by means of an example. The example is informally described in the next section, together with a short discussion on the kind of formal analysis that we would like to perform on the corresponding model.

Section 2 takes the step from the verbal description to a formal representation. The notions of agent aware transition system, view, and so on, will be gradually introduced with some comments on their applicability.

In Section 2.6 we discuss the kind of structural and dynamical properties that we can express in a suitable logical language, and suggest how to check whether those properties are satisfied.

Finally, in Section 3, we summarize our approach, and try to briefly assess its merits, drawbacks, and limitations.

1. An Example: Dynamic Coalitions

The example chosen to illustrate our ideas is, for obvious reasons, quite artificial, and much simplified with respect to a realistic setting. However, we hope that even such a simple model can convey the main ideas that underlie our approach. The example shows the main features of the formal framework: the components of the system (abstractly called agents in the following) are classified in several types. Mutual relationships among the agents can change in time as a result of the execution of actions. Some actions involve a number of agents; we refer to such a situation by talking of a *synchronization* of the involved agents. However, agents can also interact in more indirect ways, like exchanging messages.

The system we want to model is formed by a set of *players* who communicate within dynamically forming and changing *coalitions*. Here, we do not attach a specific meaning to the word *coalition*; the reader might think, for instance, of groups of interest, but other interpretations are possible. What matters is the constraint that an actual exchange of information can happen only between two players which currently belong to the same coalition.

A player can belong to several coalitions, can promote new coalitions, and can leave a coalition. Joining a coalition is only possible after receiving an *invitation* from a player which is already a member of that coalition.

Within such a general description, we can imagine several properties that a designer might want to check, or to enforce. For instance, we might be interested in proving that two given players will never belong to the same coalition, or to show an admissible sequence of events that will violate such a property; or, we might want to prove that, given three distinct players a, b, c , whenever a and b belong to the same coalition γ , c does not belong to γ . Later we will suggest how to express such properties, and how to check them.

2. Dynamic Coalitions: Formal Setting

In this section, we translate the plain English description of the system into a formal definition, and introduce, along the way, the needed notions.

2.1. Agent aware transition systems

As a first step along the way, we define the general concept of agent aware transition system. In the end, we want to be able to specify our system as such an object, possibly through a more expressive specification language.

Definition 1 *An agent aware transition system (AAS), is a tuple $\mathcal{S} = \langle A, T, S, \delta, s_0 \rangle$, where A is a finite set of agents, S is a finite set of states, T is a finite set of actions, δ is the transition function, and $s_0 \in S$ is the initial state. A single transition is specified by an action, the set of involved agents, the starting state, and the final state. More formally, the transition function is a partial map*

$$\delta: T \times 2^A \times S \longrightarrow_* S$$

where 2^A denotes the set of subsets of A .

The map δ is defined as partial to reflect the fact that in certain states, a given action cannot be performed by a given set of agents.

With respect to classical transition systems, an AAS introduces agents, and the specification of the set of agents which take part in a transition. The same action can thus be performed by different groups of agents.

In the formal model that we are going to develop for the system of players and coalitions, each view will be specified as an AAS. In the next section we will define views, but first we must answer a basic question. How can we build up the entire system from the collection of views? As suggested in the introduction, the different views must be superimposed. Formally, this superimposition is defined in a way which resembles the usual synchronous product for transition systems, where an action can occur in the composite system only if all components which have that action in their alphabet are ready to execute it. This mechanism gives an abstract form of synchronization, or interaction, among components.

The corresponding operations can be defined as a method to compose n AASs, irrespective of their interpretation as views of the same system.

Here, we will content ourselves with an intuitive description of the operation. The reader interested in formal definitions is invited to look at [11].

Assume then to have several AASs, \mathcal{S}_i , with i varying over a set I . We want to define an AAS, call it \mathcal{S} , as the composition of the \mathcal{S}_i . The states of \mathcal{S} are all combinations of states of the components. On the other hand, actions and transitions are defined as the union of actions and transitions, respectively, of the components.

Two, or more, transitions, taken from different components, are superimposed, or synchronized, when they are consistent. Consistency means that they can be seen as a sort of projections of a more general transition. By *projection* of a transition t on a component \mathcal{S}_i , we mean a transition with the same label, taken from T , and such that the set of agents involved is exactly the intersection of the set of agents involved in t and the set of agents belonging to \mathcal{S}_i .

The underlying idea can be intuitively explained as follows. A state transition is, in general, a complex entity; several agents can participate in a transition. We can observe the transition from different standpoints, corresponding to the different views in which the system is articulated. When looking at the transition from a standpoint corresponding to view i , we can only see agents belonging to that view; so in view i we must find the corresponding transition, labelled by the appropriate set of agents.

2.2. Agents, Actions, and Views

From the description of the system, we can derive the entities that must be explicitly represented in the model. They can be classified in three categories: *players*, representing people who gather in groups of interest and exchange information, *coalitions*, representing the groups themselves, which have players as members, and *messages*, which are used by players in order to invite other people to join a coalition; notice that these messages do not represent information exchanged within a coalition: in our simple example, we do not deal with the actual exchange of information, which will be represented by a generic action called *talk*.

Players, coalitions, and messages are the agents of the AAS that we are going to define. Each view, defined later will deal with a subset of the agents.

We can now identify the views through which we look at the behaviour of agents. We define three of them.

The first view is concerned with the knowledge that players have of coalitions. Remember that a player can join a coalition only after receiving an invitation from another player; we will assume that each player has knowledge of a subset of coalitions, and can enlarge its knowledge by receiving an invitation (receiving an invitation is a prerequisite for joining the coalition, but a player can ignore an invitation if not interested in joining that coalition). We also provide for an action by which a player can forget a coalition. After forgetting a coalition, a player can not join it, unless she receives a new invitation.

The second view defines the rules by which two players can talk to each other, namely that they belong together to some coalition. This view should keep track of membership in coalitions.

Finally the third view governs the “mechanics” of invitations. It describes the correct sequences of actions involving messages (for instance, that a given message can be received only after being sent) and associates to a pending message its content, that is the coalition it refers to.

Views will now be defined more precisely, in a sort of operational style. For each of them, we will define the set of states, the set of agents, and the set of actions; then we

will give the conditions that allow a transition to occur at a given state, and the effect of its occurrence.

In the following, P denotes the set of all players, C the set of all coalitions, M the set of all messages. These sets are fixed from the beginning: in AASs, we cannot create or delete agents. We will comment on this restrictions in the conclusion of the paper.

2.3. View 1: Knowledge of Coalitions

This view represents one kind of relation between players and coalitions, expressed by the statement *player p knows about coalition c*.

With respect to the general form of an AAS, we have

$$S_1 = \langle A_1, T_1, S_1, \delta_1, s_{01} \rangle$$

where $A_1 = C \cup P$. States of this view associate to each player the set of coalitions she knows (this is not the same as the set of coalitions that she is a member of); formally, states are functions from the set of players to subsets of coalitions:

$$S_1 = \{K: P \longrightarrow 2^C\}$$

The actions relevant to this view are those actions which depend on the knowledge of a player, or which affect that knowledge:

$$T_1 = \{join, send, receive, start, forget\}$$

We have now to define the transition function. To this aim, we list a set of generic clauses that implicitly define δ_i . To keep the notation compact, we adopt the following general convention: let $f: X \rightarrow Y$ be a map; then, by $f [z/x]$ we denote a new map from X to Y which coincides with f for all elements in the domain except for x , where it takes the value z .

There are five rules for this view. The first rule states that a player can start a new coalition only if she has knowledge of it (we can say: if she knows its name).

Let K be a state of this view.

- $K \xrightarrow{start\{c,p\}} K$ if and only if $c \in K(p)$

The second rule allows a player to join a coalition provided she knows about it already.

- $K \xrightarrow{join\{c,p\}} K$ if and only if $c \in K(p)$

The third rule allows a player who knows about a coalition to send an invitation for it.

- $K \xrightarrow{send\{c,p\}} K$ if and only if $c \in K(p)$

Notice that the first three rules do not change the state of this view (the same actions will show up in other views also).

The fourth rule states that a player receiving an invitation can enlarge her knowledge. This does not exclude that a player receives an invitation for a coalition she already knows.

- $K \xrightarrow{\text{receive}\{c,p\}} K [K(p) \cup \{c\}/p]$

The state of the view changes after performing an action *receive*, reflecting the fact that the receiver now knows about coalition *c*.

The final rule for this view governs the occurrences of action *forget*. By this action, an agent voluntarily forgets about a coalition.

- $K \xrightarrow{\text{forget}\{c,p\}} K [K(p) \setminus \{c\}/p]$

2.4. View 2: Coalition Membership

This view is concerned with the relation of membership between players and coalitions. The agents involved in this view are, like in the first view, players and coalitions:

$$A_2 = P \cup C$$

but the states are different. A state of this view must keep track of actual membership. It is then natural to define states as maps from *C* to subsets of *P*:

$$S_2 = \{s: C \rightarrow 2^P\}$$

with the intended meaning that $p \in s(c)$ if player *p* is a member of coalition *c* in state *s*.

Relevant actions for this view include actions which change membership, and also the generic action, here called *talk*, representing an exchange of information between two players.

$$T_2 = \{\text{start}, \text{join}, \text{drop}, \text{talk}\}$$

Rules:

A player can start a coalition only if that coalition has no members. After performing the action, the promoter is the only member of the coalition (she will be able to invite other players later).

- $s \xrightarrow{\text{start}\{c,p\}} s'$ iff $s(c) = \emptyset$ and $s' = s [\{p\}/c]$
- $s \xrightarrow{\text{join}\{c,p\}} s'$ if and only if $p \notin s(c)$, $s(c) \neq \emptyset$ and $s' = s [s(c) \cup \{p\}/c]$
- $s \xrightarrow{\text{drop}\{c,p\}} s'$ if and only if $p \in s(c)$ and $s' = s [s(c) \setminus \{p\}/c]$
- $s \xrightarrow{\text{talk}\{p,p',c\}} s$ if and only if $p, p' \in s(c)$

2.5. View 3: Mechanics of Invitations

This view deals with invitations. A player can invite someone to join a coalition by sending a message bearing the name of the coalition. In defining this view, we must face a constraint given by the formal framework in which we operate. In order to apply the analysis techniques briefly described in a later section, the model of the system must be finite. Since messages are agents in our model, we cannot create and destroy them freely. Hence, we choose to assume that there are enough messages in the initial states, and reuse them as needed. A “quiescent” message is like an empty box. On sending a message, the

sender associates it with a coalition. On receiving the message, the addressee removes its content, so that the message can be reused.

A typical state for this view is described by a map from messages to coalitions, associating to each message its content. Since a message can be empty, the map is only partial. By checking if the map is defined for a given message, we can also decide whether a message is traveling from a player to another.

To keep the model simple, we do not explicitly represent the addressee of a message. This means that any player can catch a pending message. Several properties of a system of this kind are actually independent of that information.

The agents involved in this view are messages and their contents, namely coalitions:

$$A_3 = M \cup C$$

States are partial maps from messages to coalitions:

$$S_3 = \{s: M \longrightarrow_* C\}$$

Relevant actions include sending and receiving a message. Notice that in this view there is no way to know which player sends or receives a message, since this is irrelevant in this context. Such information is available in the first view. After composing views, a send action will be associated to a message, a coalition, and a player.

$$T_3 = \{send, receive\}$$

The rule concerning *send* requires that a message is empty before it can be sent. In performing *send*, it is filled with the name of a coalition.

- $s \xrightarrow{send\{c,m\}} s'$ if and only if $s(m)$ is undefined and $s' = s [c/m]$

The rule for *receive* is symmetrical to the former: a message can be received only if it has been sent, that is if it has a content. The act of receiving deletes its content. The effect of receiving a message on the knowledge of the receiver is represented in view 1.

- $s \xrightarrow{receive\{c,m\}} s'$ if and only if $s(m) = c$ and $s' = s [\perp /m]$

One last ingredient is needed to fully specify the views composing our system: the initial state. For each view, we must specify the initial situation. In particular, the initial state should “initialize” the knowledge of each player, so that each coalition is known by at least one player.

Building the complete description of the system as a single AAS can now be done automatically, by applying the synchronization operation informally explained above.

2.6. Analysis

In this section we hint at the kind of properties of a system that one would like to check on the corresponding model. Properties are expressed in a logical language endowed with temporal operators.

The logical language we propose is based on a fixed set of symbols, denoting predicates on agents, and relations among agents. One obvious relation for our example could, for instance, express the fact that a given player is a member of a given coalition. A sim-

ple predicate can assert that a given coalition is empty. Formulae built on the fixed set of symbols, the set of names of agents, and on variables, can be evaluated at arbitrary states, after fixing a valuation function which associates each variable with an agent. The validity of a formula can obviously change from state to state.

The temporal operators allow us to specify a rich set of formulae of the kind “for each admissible sequence of actions from the initial state, eventually the formula ϕ will be valid”, or “there exists an admissible sequence of actions from the initial state such that formula ϕ is valid until ψ becomes valid, and so on.

For instance, assume that in the example there are three players, p_1 , p_2 , and p_3 , and we want to check, maybe for security reasons, that in no state they are members of the same coalition c . Then we can write the formulae

$$\alpha = \text{in}(p_1, c) \wedge \text{in}(p_2, c) \Rightarrow \neg \text{in}(p_3, c)$$

$$\beta = \text{in}(p_2, c) \wedge \text{in}(p_3, c) \Rightarrow \neg \text{in}(p_1, c)$$

$$\gamma = \text{in}(p_1, c) \wedge \text{in}(p_3, c) \Rightarrow \neg \text{in}(p_2, c)$$

where in is a symbol denoting the membership relation between a player and a coalition.

The property we want to check can then be expressed as follows, where s_0 is the initial state of the system, and \square denotes the temporal operator “always”.

$$s_0 \models \square(\alpha \wedge \beta \wedge \gamma)$$

3. Conclusion

With the help of a simple example, we have outlined the main ideas of a method for designing formal models of complex systems. The method is based on the notion of *agent aware transition system*, which can be seen as a generalization of the standard notion of transition system. A distinguishing feature of AASs is its explicit treatment of agents as entities in the model; the actions that make such a system evolve are always referred to the set of agents involved.

In order to apply usual model-checking techniques, the models we build are bound to finite sets of states. Consequently, we do not allow creation of agents. This can be a strong limitation in the expressive power of the model in some fields of application. The extension to infinite state spaces will be explored, in association to model-checking algorithms for such classes of systems.

A central feature of our approach is the idea of view. While this is certainly not a new idea in general, the presence of agents gives it some special features, that we think may be useful while developing the specification of a distributed system.

As a generalization of labelled transition systems, AASs are a very general notion. When designing a real systems, a designer can work on more expressive or compact notations, like hypernets, Petri nets, or process calculi, provided there is an automatic way to translate them into AASs.

References

- [1] H. Kargupta, I. Hamzaoglu, and B. Stafford. Scalable, distributed data mining using an agent based architecture. In *Int. Conf. on Knowledge Discovery and Data Mining*, pp. 211-214, August 1997, 1997.
- [2] Vladimir Gorodetsky, Chengqi Zhang, Victor A. Skormin, and Longbing Cao, editors. *Autonomous Intelligent Systems: Multi-Agents and Data Mining, Second International Workshop, AIS-ADM 2007, St. Petersburg, Russia, June 3-5, 2007, Proceedings*, volume 4476 of *Lecture Notes in Computer Science*. Springer, 2007.
- [3] Edmund Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT Press, 1999.
- [4] C.A.R. Hoare. *Communicating sequential processes*. Prentice-Hall, 1985.
- [5] Robin Milner. *Communicating systems and the π -calculus*. Cambridge University Press, 1999.
- [6] Wan Fokkink. *Introduction to process algebra*. Texts in theoretical computer science. Springer, 1999.
- [7] Wolfgang Reisig and Grzegorz Rozenberg, editors. *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, volume 1491 of *LNCS*. Springer, 1998.
- [8] Marek A. Bednarczyk, Luca Bernardinello, Benoit Caillaud, Wiesław Pawłowski, and Lucia Pomello. Modular system development with pullbacks. In Will van der Aalst and Eike Best, editors, *24th International Conference on Applications and Theory of Petri Nets, Eindhoven, The Netherlands, June 2003*, volume 2679 of *LNCS*, pages 140–160. Springer-Verlag, 2003.
- [9] Marek A. Bednarczyk, Luca Bernardinello, Wiesław Pawłowski, and Lucia Pomello. Modelling mobility with Petri hypernets. In José Luiz Fiadeiro, Peter D. Mosses, and Fernando Orejas, editors, *Recent Trends in Algebraic Development Techniques. 17th International Workshop WADT 2004, Barcelona, Spain, March 27-30, 2004. Revised Selected Papers*, volume 3423 of *LNCS*, pages 28–44. Springer-Verlag, 2005.
- [10] Marek A. Bednarczyk, Wojciech Jamroga, and Wiesław Pawłowski. Expressing and verifying temporal and structural properties of mobile agents. In Ludwik Czaja, editor, *Proceedings of the Concurrency, Specification and Programming Workshop CS&P'05*, pages 57–68, 2005.
- [11] Marek A. Bednarczyk, Luca Bernardinello, Tomasz Borzyszkowski, Wiesław Pawłowski, and Lucia Pomello. A multi-facet approach to dynamic agent systems. In L. Czaja, editor, *Proceedings of the International Workshop on Concurrency, Specification and Programming (CSP07), Vol. 1, Łagów, Poland, 26-30 September 2007*, pages 33–47, 2007.