SN Computer Science (2021) 2:317
https://doi.org/10.1007/s42979-021-00706-5

SURVEY ARTICLE

Engineering Multi-agent Systems with Statecharts

Theory and Practice

Nikolaos I. Spanoudakis[1]

Received: 18 September 2020 / Accepted: 15 May 2021 / Published online: 3 June 2021
© The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd 2021

Abstract
The purpose of this paper is to map the works in the Agent Oriented Software Engineering and Engineering Multi-agent Systems fields that use the statecharts paradigm to aid the software development process. We did not only focus on finding out the methods that employ statecharts, but also on identifying the application domains suitable for this kind of modeling. To achieve this goal we researched the available literature. We found out that statecharts are used by numerous methods that target diverse application domains such as robotics, internet agents, simulation, health and safety. Moreover, statecharts have been employed for modeling different things, such as the behaviour of agents, agent plans, agent interaction protocols. We also found reports of real-world systems and applications that were developed using these methods in the last 20 years and we identified trends and common characteristics that they have. Concluding, we have mapped the area where statecharts meet agent-oriented methodologies both in theory and practice. Moreover, we provide some exciting directions for future works.

Keywords Statecharts · Finite state machines · Agent-Oriented Software Engineering · Engineering Multi-agent Systems · Applications

Introduction

The area is known as Agent-Oriented Software Engineering (AOSE), or, more recently, as Engineering Multi-agent Systems (EMAS), is located in the place where the Artificial Intelligence (AI) and the Software Engineering (SE) domains overlap. This area emerged when the agent paradigm, a subdomain of AI, also referred to as Multi-agent Systems (MAS), or Distributed AI (DAI), was mature enough to start producing applications and it intersects the general area of statecharts research. Figure 1 attempts to depict this area. It may seem small in the figure, however, we will show that it is a live area that has produced significant results in the form of extensions to the state of the art but also in real-world applications.

The areas of Agent Technology and Statecharts have another characteristic in common, they both emerge in the eighties. Agent technology firstly appeared as Distributed Artificial Intelligence (DAI) [59], and later as Multi-agent Systems (MAS), while the statecharts as a method for engineering complex and reactive systems [34]. Statecharts combine data flow and control and exploit ideas present in the Finite Automata, or Finite State Machines, and the Data Flow Diagrams. Modern statecharts combine the above ideas with the concept of orthogonality (concurrent execution modeling) and the hierarchical states, along with execution semantics and are also referred to as "Harel statecharts", after the person that first conceived them [34].

Statecharts and Finite State Machines were quickly employed by AOSE methods and processes. They were initially used for modeling agent interactions [55, 70] and agent plans [16, 60, 65]. The agent interaction protocols have also been referred to as *inter-agent control* models [16, 87]. Later, AOSE methods adopted a modular approach for defining agents, which employed modules as capabilities,

This article is part of the topical collection "Advances in Multi-Agent Systems Research: EUMAS 2020 Extended Selected Papers" guest edited by Nick Bassiliades and Georgios Chalkiadakis.

✉ Nikolaos I. Spanoudakis
nikos@amcl.tuc.gr

[1] Applied Mathematics and Computers Laboratory, School of Production Engineering and Management, Technical University of Crete, Chania, Greece

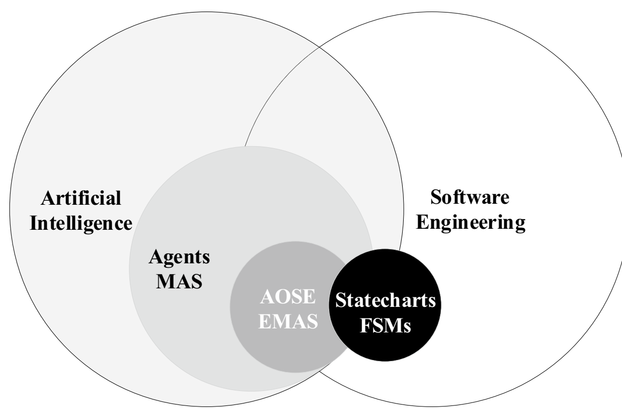


Fig. 1 The Statecharts and AOSE scientific domains

and needed a formalism for coordinating them. Statecharts were capable for modeling this kind of interaction (or agent behaviour) and such models were referred to as *intra-agent control* models [16, 91]. Moreover, newer platforms, such as the Repast Simphony agent-based modeling toolkit, catered for agent-based social simulation through statecharts [67].

Nowadays, as AI has moved from theory to practice and the emergence of 5G aims to connect AI-enabled components [97], we expect that MAS applications will start appearing in the market. Until recently, MAS were usually prototypes, systems participating in scientific contests, such as Robocup, and results of R&D projects. Thus, they were limited in scale and many researchers opted to develop them without following a methodology. When they come to the market, though, and they need teamwork, documentation and maintenance, AOSE methods will become invaluable.

This paper extends a previous work [89] and aims to collect all this experience of using statecharts for agent-related research and to show that the EMAS/AOSE area has not only used this formalism as is, but has proposed extensions and practices to further the state of the art. Moreover, and this is novel with respect to the previous work [89], this paper explores real-world systems developed using this formalism and tries to extract trends and to outline the application domains. Finally, we will also take a look into the future by summing up the challenges identified by these works.

Therefore, this paper is intended as a first comprehensive attempt to gather all the works in the EMAS/AOSE field related to the language of statecharts, to outline the uses of this formalism and identify the application areas. It is addressed to researchers in both areas hoping to stimulate interaction. Towards this end, in the next section (background), both the statecharts and multiagent systems areas are introduced.

Thus, section two, first focuses on statecharts and finite state machines, also trying to disambiguate these terms, and, secondly on agent technology. Section three provides an overview of the use of the language of statecharts in AOSE and EMAS. Section four illustrates some key features that statecharts have enabled in EMAS/AOSE. Section five provides information regarding the application domains and characteristics of real-world agent-based systems modeled as statecharts. Section six discusses these findings, section seven takes a look at related work, and, section eight identifies future research directions.

Background

Statecharts and Finite State Machines

Statecharts are often confused with automata [19], finite state automata, or finite state machines (FSMs) [7]. We will firstly define FSMs aiming to give the definition in such a way that it will later help the reader to identify the formal differences with statecharts (see Definition 1).

Definition 1 A Finite State Machine can be defined as a tuple (L, δ) where:

- $L = (S, Name)$ is a set representing the states of the statechart, and:
  - $S \subseteq \mathbb{N}^*$
  - Name is a mapping from nodes to their names
- $\delta \subseteq S \times TE \times S$ is the set of state transitions, where TE is a set of transition expressions

It is not unusual to see more information appear in such definitions, such as the FSM's initial state, or the alphabet used for transition expressions, however, for the needs of this work Definition 1 will suffice.

Harel proposed the modern language of statecharts for modeling software systems [36] by adding control information to a hierarchical data-flow diagram. Data Flow Diagrams (DFD) used to depict the flow of data within a system and the different processes that work with the data. In DFDs, processes can be further refined in a lower level of abstraction. This idea was used to allow for composite states, i.e., such that can contain other states. Lately, there are works that incorporate this feature into FSMs, in the so called *hierarchical FSMs* [10, 30, 106]. Another innovation with relation to FSMs is the orthogonal components, i.e., concurrently active states.

In this paper, when we refer to statecharts we mean Harel statecharts [34, 36]. To define them, we will expand Definition 1 with the following information. There are different state types:

- *or-type* states are composite states that contain other states (sub-states), only one of which can be active at any given time. The *or-type* state at the highest level, i.e., the one that contains all other states of a statechart is called the *root*
- *and-type* states are composite states that contain *or-type* states all of whom can be active at the same time and are also called "orthogonal components"
- *basic-type* states are simple states that may be active or not at any given time
- The *start-type* state is an *auxiliary state*, i.e. a state type that helps in defining the control flow of a statechart but it cannot be active at any given time (this is why auxiliary states are also referred to as *pseudo-states*). It is the default first state to enter when an *or-type* state is activated, thus each *or-type* state can have at most one *start-type* state
- When an *end-type* state (an *auxiliary state*) is entered this means that the *or-type* state that contains it has finished executing all its sub-states
- The *condition-type* state (an *auxiliary state*) forks a transition
- The *shallow_history-type* state (an *auxiliary state*) allows for "remembering" the last active state in an *or-type* state
- *history-type* state (an *auxiliary state*) allows for "remembering" the last active state in a whole branch of L

Note that all the auxiliary state types along with the basic state type are leaves of L. Now we can give Definition 2.

Definition 2 A Statechart can be defined as a tuple (L, δ) where:

- $L = (S, \lambda, Name, Activity)$ is an ordered rooted tree structure representing the states of the statechart, and:
  - $S \subseteq \mathbb{N}^*$
  - *Name* is a mapping from nodes to their names.
  - $\lambda : S \rightarrow \{and, or, basic, start, end, shallow_history, history, condition\}$, is a mapping from the set of nodes to labels giving the type of each node.
  - *Activity* is a mapping from nodes to their algorithms in text format implementing the processes of the respective states.
- $\delta \subseteq S \times TE \times S$ is the set of state transitions, where TE is a set of transition expressions

Each transition from one state (source) to another (target) is labeled by a transition expression (TE), whose general syntax is $e[c]/a$, where e is the event that triggers the transition; c is a condition that must be true in order for the transition to be taken when e occurs; and a is an action that takes place when the transition is taken. All elements of the transition expression are optional. Moreover, there can also be compound transitions (CT), that can have more than one source or target states. We will not refer to that level of detail in this work. The *scope* of a transition is the lowest level *or-type* state that is a common ancestor of both the source and target states in L. A transition without an event is enabled when the activity of its source state is completed.

The statechart formalism also defines execution semantics. We will give a brief overview, for the details the reader is referred to Harel and Naamad [36]. The execution of a statechart is a sequence of *steps*. After each step, we view a *snapshot* of the statechart. Execution starts at *start* states. When a step is taken, the events that have happened are sensed, including retrospection events (such as the entering of a state at the previous step). When the step finishes, the statechart is in a valid *configuration*, i.e. specific *basic-type* states are active and the respective *or-type* and *and-type* states up to the root. Other types of states cannot be included in the configuration (e.g. the *start-type* state cannot be active in a snapshot).

When a transition occurs all states in its scope are exited and the target states are entered. Multiple concurrently active statecharts are considered to be orthogonal components at the highest level of a single statechart. If one of the statecharts becomes non-active (e.g. when the activity it controls is stopped) the other charts continue to be active and that statechart enters an idle state until it is restarted.

The language of statecharts has also been adopted by the Unified Modeling Language (UML), which has been standardized by the Object Management Group (OMG), (https://www.uml.org) for object-oriented design. UML employs statecharts for defining the dynamic behavior of an object. The main difference between Harel and UML statecharts is that the latter focus only on defining the dynamic behaviour of an object (class instance), while Harel statecharts focus on defining the behaviour of a system.

Multi-agent Systems

EMAS/AOSE is concerned with the methodological approach to modeling agent-based systems. Agents are the descendants of objects and have several properties that are not mainstream for classical software engineering and this is why AOSE produces new metaphors, concepts and methods to aid the development process. Agents are [102]:

- proactive (have goals and act to achieve them),
- reactive (respond to events occurring in their environment)
- social (are acquainted with other similar software and can cooperate or compete with it),

- autonomous (do not need human intervention to act), and,
- intelligent (may perform such tasks that, when performed by humans, we consider as evidence of a certain intelligence).

The reader just needs to keep these properties in mind. The main differences of agents with objects is that they perceive their environment and can act on it and that they usually follow the pattern *sense* (update the knowledge about the environment based on the latest signals sensed), *think* (update the agent's goals based on the updated knowledge about the environment), *act* (select and apply the best action based on the updated goals) [102, 105].

Now, acting is related to selecting and activating a valid plan for reaching a selected goal. Moreover, since their conception, MAS were intended to solve more complex problems than those that a single agent can solve. Therefore, their functionality depends on communication and coordination, which are achieved through the definition of agent interaction protocols. Finally, as agents are complex software they are built of diverse components that must work together to present a coherent behaviour (consider, e.g., a robot).

Engineering Multi-agent Systems with Statecharts

Among the first to use Harel statecharts in AOSE was Moore [55], who adopted them for defining agent interaction protocols. He also defined the Formal Language for Business Communication (FLBC), an Agent Communication Language (ACL) that incorporated speech acts to distinguish between message types. A speech act is an act that a speaker performs when making an utterance [4]. Moore introduced speech acts as *Performatives* that are meant to express the intent of an agent when it sends a message to another agent. Thus, a compact representation of an ACL message uses the formalism $performative(sender,\ receiver, content)$, where the performative is the speech act, the sender is the agent that sends the message, the receiver is the intended recipient of the message and the content is the data, i.e. the message itself.

According to the work of Moore, a *conversation policy* (CP) defines (a) how one or more conversation partners respond to messages they receive, (b) what messages a partner expects in response to a message it sends, and, (c) the rules for choosing among competing courses of action.

Moore introduced the idea of modeling the activities of the participants in a conversation as orthogonal components of a statechart. The transition expressions contain the actions of sending and receiving a message. Moore's conversation policies allow for exceptions when a conversation is interrupted by assuming that an agent has stored all allowed CPs in a kind of repository where it can browse for a new policy to handle the exception, in the form of a sub-dialog to the original one. When this sub-dialog terminates the original one can resume.

The first AOSE methodology to adopt the language of statecharts was the Multiagent Systems Engineering methodology (MaSE) [15, 16]. In the MaSE design phase, the first activity is about creating agent classes and then agent classes can connect to other classes indicating the possible interactions or conversations. The latter are defined in the Communication Class Diagram (CCD), which is in the form of a statechart. MaSE defines a system goal-oriented MAS development methodology. MaSE introduced the concepts of *inter-* and *intra-agent* interactions that must be integrated in the agent design.

A Communication Class Diagram (CCD) aggregates as many statecharts as are the participating roles. Each role behaviour within the communication protocol is defined in its own statechart.

To define MaSE statecharts the reader can use Definition 2, however, the λ mapping changes to $S \rightarrow \{basic, start, end\}$. As there are no composite states (*and*, *or*), the depth of the routed tree structure is always equal to one. Thus, the plans or CCDs are individual statecharts. *Basic-type* nodes can have *entry* and/or *do* actions. The transition expressions are in the form $e[c]\hat{}a$, where e and a are respectively incoming and outgoing messages.

A message is defined by its performative and optional arguments (variables) in parenthesis. E.g. *acknowledge* or *reportingStatus(status)*. The recipients are optionally used if more than two roles participate in the interaction.

Dumas et al. [17] proposed the combination of statecharts and defeasible logic for defining the behaviour of an agent. They used statecharts for the control module of the agent and defeasible logic for the reasoning module. Moreover, in their formalism they define a communication module that sends and receives messages. When it receives a message, it generates an event for the control statechart to catch. Likewise, the statechart can trigger the sending of a message through appropriate action expressions. Although it was mostly theoretical work, the way that they motivated the use of statecharts is very interesting and is also valid until today. For selecting the appropriate modeling language they used criteria such as:

- *formal*, in the sense that its syntax and semantics are precisely defined. In that way the models behaviour is predictable, explainable, verifiable and executable
- *conceptual*, i.e. it should allow its users to focus at the task at hand, at the right level of abstraction, and not have to deal with other aspects, such as implementation language. This quality has also been related to model-

driven engineering and the platform independence of design models [47]
- *comprehensible* by humans, e.g. offering an intuitive graphical representation
- *expressive* enough to allow for modeling all possible behaviours

The above criteria can also be considered as advantages of the statechart formalism, as it was selected because it covered all of them.

Statecharts were also employed for modeling robotic applications. Arai and Stolenberg [2] proposed an extension to state machines with a new symbol that is placed between two orthogonal components and connects states that need to execute concurrently, thus allowing for synchronization of activities within statecharts.

König [48] used the state transition diagrams (STD) formalism for modeling protocols, but also for the agent's decision activities. An STD is a special case of a Finite State Machine (FSM) that allows transitions between states either when an external or an internal event occurs to the system (according to his work, transitions in FSMs can only contain external events).

König defined a protocol as a structured exchange of messages. Then, he compared three approaches to modeling conversation policies, i.e. those based on STDs, FSMs and Petri nets. He observed that all approaches modeling conversations from the viewpoint of an observer are using either STD or petri nets, in contrast to those using FSM (or statecharts) that are representing the conversation from the viewpoint of a participating agent. For modeling a conversation from the point of view of a participating agent who receives and sends messages, König argued that a model supporting input and output operations is more suitable. When a conversation should be modeled from an observer's view, it is sufficient to use a model which is able to express that a message has been transmitted from one agent to another, like a transition in a STD or in a petri net. He chose STD aiming to model both activities and protocols, allowing also for object-oriented development.

König made the assumption that only two agents are involved in a protocol, i.e. the primary (who initiates the interaction) and the secondary. Moreover, the exchange of messages is always synchronous, i.e., when one agent sends a message, the other agent is in a state of receiving a message (they cannot both be sending at the same time). Then, he defines an FSM for the observer and from it he derives the FSMs of the participants. In the next level (higher level of abstraction) he defines communication acts that can make use of the protocols in the form of STDs. Finally, in a third level he defines the activities of the agents that can invoke one or more communication acts and assume a wait state until the acts finish. The acts themselves can choose to execute one or more protocols and enter a wait state until they are finished. All these can only happen sequentially.

The Gaia methodology [101] emphasized the need for new abstractions to model agent-based systems and supported both the levels of the individual agent structure and the agent society in the multi-agent (MAS) development process. Gaia added the notion of situatedness to the agent concept [105]. According to this notion, the agents perform their actions while situated in a particular environment. The latter can be a computational environment (e.g. a website) or a physical one (a room) and the agent can sense and act in it.

MAS, according to Gaia, are viewed as being composed of a number of autonomous interactive agents that live in an organized society, in which each agent plays one or more specific roles. Gaia defined the structure of a MAS in terms of a role model. The model identifies the roles that agents have to play within the MAS and the interaction protocols between the different roles.

The objective of the Gaia analysis phase is the identification of the roles and the modelling of interactions between the roles found. Roles consist of four attributes: responsibilities, permissions, activities and protocols. Responsibilities are the key attribute related to a role since they determine the functionality. Responsibilities are of two types: liveness properties - the role has to add something good to the system, and safety properties - the role must prevent something bad from happening to the system. Liveness describes the tasks that an agent must fulfil given certain environmental conditions and safety ensures that an acceptable state of affairs is maintained during the execution cycle. To realize responsibilities, a role has a set of permissions. Permissions represent what the role is allowed to do and, in particular, which information resources it is allowed to access. The activities are tasks that an agent performs without interacting with other agents. Finally, protocols are the specific patterns of interaction, e.g. a seller role can support different auction protocols. Gaia defined operators and templates for representing roles and their attributes and schemas for the abstract representation of interactions between the various roles in a system.

The Gaia2JADE process appeared in 2003 [56, 58] and was concerned with the way to implement a multi-agent system with the Java Agent Development Framework (JADE) [5] framework, one of the most popular agent platforms [6], using the Gaia methodology for analysis and design purposes. This process used the Gaia models and provided a roadmap for transforming Gaia liveness formulas to Finite State Machine (FSM) diagrams. It provided a method for transforming the FSMs to JADE behaviours using the *FSM-Behaviour* native JADE construct [5].

Paurobally et al. [70] combined statecharts with the Agent Negotiation Meta-Language (ANML), which is based on Propositional Dynamic Logic (PDL) for use in actions and

transition expressions. PDL blends the ideas behind propositional logic and dynamic logic by adding actions while omitting data; hence the terms of PDL are actions and propositions. Then, the authors defined templates for transforming the ANML formulas to statecharts, extending the statecharts language in the process. The representation of all computation is in transitions, while states just describe a situation (where specific conditions hold). The representation can be general, or specialized for a specific agent participant. The expressions in the transitions are ANML formulas.

The proposal of Paurobally et al. [70] and later of Dunn-Davies et al. [18] did not employ the orthogonality feature of the statecharts because they considered that the agents are not subsystems and, thus, execute on their own. If they were combined as orthogonal components for execution they would have to combine parts of interactions between temporally autonomous agents into a pseudo whole.

To define propositional statecharts the reader can use Definition 2, however, the λ mapping changes to $S \rightarrow \{or, basic, start, end\}$. BASIC nodes can have actions. The transition expressions are in the form e; $c?a$, where e and a are respectively incoming and outgoing messages.

In 2004 there was also a proposal for the use of Distilled StateCharts (DSCs) for modeling mobile agents [26]. The proposal came along an object-oriented implementation based on UML modeling. DSCs define some limitations to the language of Statecharts, e.g. only the OR-state decomposition is used, states do not have properties such as activities, therefore activities are only carried out under the form of atomic actions attached to transitions. If their source is not *start* and *history* states, transitions always include an *event*. In a later work, Fortino et al. [24] proposed a JADE implementation for DSC.

Murray [60] worked in the direction of defining Robocup soccer player agents. He used statecharts to define plans for the different roles that an agent can assume, e.g. goalie, defender, attacker. For example, when an attacker is in the state of having the ball he can exit that state if he passes it (state activity completion), or if the referee signals (an external event).

Murray also recognised the need for synchronizing co-players' actions. He coped with this requirement in two different ways. On one hand he employed *wait states* where one agent can reside while waiting for another to complete an action. On the other hand, he also proposed an extension to statecharts with *synch* states for synchronizing the actions of different agents.

His work, along with the previous one of Obst [66], supported semi-automatic code generation for Robolog, a robot programming language based on Prolog. A similar layered approach was used later [45] for modeling the behavior of non-player characters in computer games. Murray proposed a methodology and tool (StatEdit) for capturing a player's behavior based on a three-layered approach:

- In the top level, the different roles (*modes*) that the player can assume when active are represented as states and the transitions indicate a change of role
- In a middle level an agent chooses among a set of plans adding detail at each mode of the previous level. The states here capture the agent general activity and show where the player synchronizes its actions with other roles (e.g. wait for the center player to pass the ball and then shoot to score).
- On a bottom level of the hierarchy each activity of the role is detailed to specific actions (e.g. acquire the ball and then kick towards the goal)

Later, ASEME [85, 91], emerged as an evolution of the Gaia2JADE process [58] influenced by the requirements analysis phase of Tropos [9] and the work of Moore [55] on conversation policies. It collected best practices from previous works and it uniquely, among AOSE methodologies, used the statecharts formalism both for inter- and intra-agent control modeling. Moreover, it extended the statechart formalism by adding state-dependent variables. Thus, each state is associated with variables that it can monitor and change/update. To propose this extension, the authors were motivated by the Gaia methodology and the role's access to data structures with the read or write/update permissions [101]. Thus, ASEME proposed the addition of the *Var* property to the statechart nodes. The different states can be connected with variables that can be used for exchanging information.

Definition 3 The tuple (L, δ) defined in Definition 2 is extended by adding *Var* to *L*:

- $L = (S, \lambda, Name, Activity, Var)$ is an ordered rooted tree structure representing the states of the statechart, where:
  - *Var* is a mapping from nodes to sets of variables. $var(l)$ stands for the subset of local variables of a particular node l.

According to ASEME, a state name that starts with the string "send" implies an inter-agent message sending behavior for the state's activity. A send state has only one exiting transition and its event describes the message(s) sent. Similarly, a state name that starts with the string "receive" implies that the activity of the state should wait for the receipt of one or more inter-agent messages. The type and quantity of the expected messages can be implied by the events monitored by the transition expressions that have this

Table 1 A summary of the AOSE methods using the statecharts formalism

Work	Use	Formalism	Year	Extension
CPs [55]	I-P	Def. 2	2000	Transition expressions (TE)
MaSE [15, 16]	P	Def. 2	2001	TE, no composite states
Dumas et al. [17]	C	Def. 2	2002	TE with communication module
Arai & Stolenberg [2]	C	Def. 2	2002	Synchronization states
STD [48]	I–C–P	Def. 1	2003	TE with internal events
Gaia2JADE [56, 58]	C–P	Def. 1	2003	–
ANML [18, 70]	I–P	Def. 2	2004	TE, no orthogonality
DSCs [24, 26]	C–P	Def. 2	2004	TE, no state actions, no orthogonality
StatEdit [45, 60, 66]	I–C–P	Def. 2	2004	Synchronization states
ASEME [85, 91]	I–C–P	Def. 3	2008	TE
Tropos [104]	C–P	Def. 2	2008	TE
TSTATES [75]	C–P	Def. 1	2012	Callable FSMs
KSE [94]	C–P	Def. 3	2013	TE
Repast simphony [67]	C–P	Def. 2	2015	TE, no orthogonality
Teleo-Reactive [76]	C–P	Def. 2	2017	TE

The following abbreviations have been employed in the second column (*Use*): C for (intra-agent) control, P for plans and I for interactions (inter-agent control). Additionally, the abbreviation TE has been used in the fifth column (*Extension*) for transition expressions

state as source. The events that can be used in the transition expressions can be:

- a sent or received (or perceived, in general) inter-agent message,
- a change in one of the executing state's variables (also referred to as an intra-agent message),
- a timeout, and,
- the ending of the executing state activity (empty event).

This formalism allows also for environment-based communication by defining state activities that monitor for a specific effect in the environment. This effect can be expected to be caused by any other agent or a particular agent. Such activities can be, for example, "wait for someone to appear" or "wait until my counterpart lifts the object" respectively.

At about the same time, researchers proposed a method for mapping goal-oriented requirements, such as those captured in the Tropos requirements analysis phase [9], to Harel statecharts [104]. These allowed for high-variability design as Tropos allows for defining alternative ways to achieve a system goal.

Later, researchers explored the translation of agent models defined using the Distilled StateCharts (DSC) [23, 26] into a Belief-Desires-Intentions (BDI) framework [25], including a BDI-like code generation feature. BDI is an example of an agent architecture including an execution paradigm besides ontological features [71]. BDI advocates the fact that an agent first senses its environment and updates its *beliefs*, then it searches possible *desires*, i.e. goals that are valid in this environment state, and, finally, selects some of these desires to actively pursue. The latter are now its *intentions*. Spanoudakis [89] provided a statechart model for modelling the dynamic behaviour of a BDI agent following the 3APL [13] agent development language.

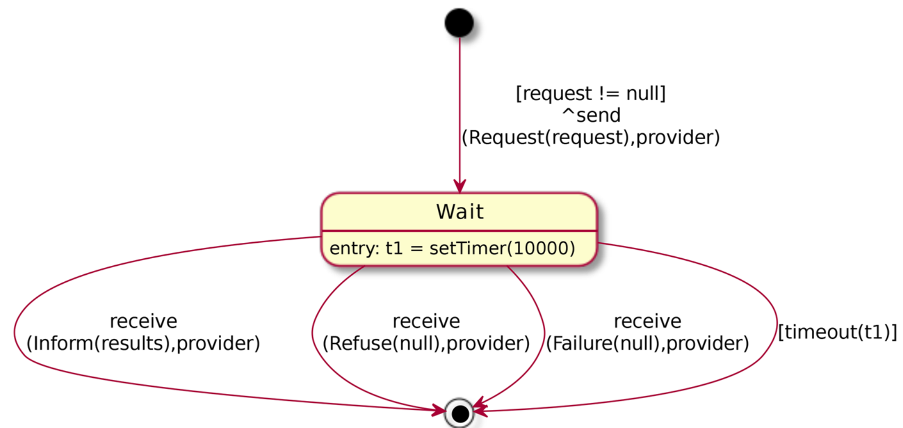
In another work, researchers provided the Kouretes Statechart Editor (KSE) CASE tool for authoring robotic behaviours [94]. Given existing bottom level functionalities [60], e.g. kick the ball, the modeler could define a robotic behaviour visually and immediately generate the code and upload it to a humanoid (Nao) robot.

Another work aims to aid the development of physical agents (robots) using statecharts to model Teleo-Reactive behaviours [76]. A Teleo-Reactive program can be seen as a set of prioritized condition/action rules that trigger actions whose continuous execution leads the system to satisfy a goal [64].

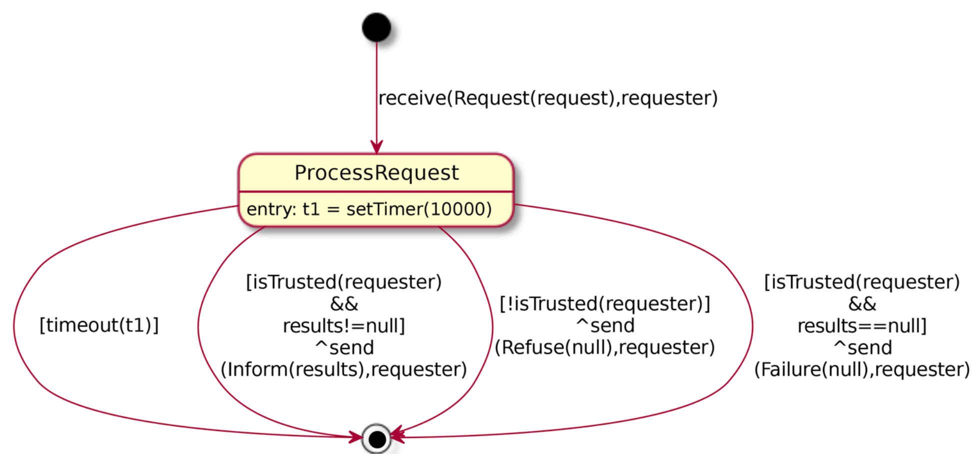
An interesting type of applications for multi-agent systems is that of simulations. Statecharts have been used for modeling the behaviour of agents in simulations. The TSTATES Domain-specific Language (DSL) for the NetLogo simulation platform [81] employs state machines for defining the behaviour of agents (also called turtles) [75]. Moreover, in his work, Sakellariou proposes the concept of *callable* state machines that can be invoked by a transition from any state, a concept similar to that of nested functions. In their work, Ozik et al. [67] also use statecharts in the Repast Simphony tool for social simulations.

In Table 1, the reader can find the summary of our findings. There, we outline the AOSE works related to statecharts in chronological order, including the formalism that they follow and the possible extensions or limitations they

Fig. 2 The plans for the requester and provider roles



(a) Statechart representation of a plan for a requester agent.



(b) Statechart representation of a plan for a provider agent.

impose. Moreover, there is an indication of whether the statecharts formalism has been used for modeling intra-agent control, agent plans, interactions (inter-agent control), or their combinations.

AOSE Features Supported with Statecharts

In this section, we provide some examples that demonstrate how to use statecharts for representing plans and agent interaction protocols; how their use can facilitate the integration of inter-agent control models to the intra-agent control models; how to support sub-dialogs and how to embed dialogs in dialogs.

We will consider the meetings scheduling example, where end users are assisted in managing their meetings by a personal assistant. The latter manages the user's schedule. The meetings organization process is managed by a meetings manager. The meetings manager contacts the users' assistants whenever she needs to negotiate a meeting date. This example (the meetings management system) has been widely used in the past for demonstrating the use of AOSE methodologies, e.g. for the Prometheus, MAS-CommonKADS [38] and ASEME [91]. To demonstrate the different approaches we will model a simple schedule request protocol, where an agent, the requester role, wants to get the meetings schedule of another agent, the provider role. The provider checks if the requester is trusted. Then it checks if it has a schedule to share. If yes, then it replies informing the requester about its schedule. If not, then it replies with failure. If the requester is not trusted, then the provider refuses to send it information. Finally, both participants agree that the answer must take place within 10 seconds or else the protocol terminates without the requester receiving a response.

Modeling Plans

MaSE used statecharts as plans for roles. For our meetings scheduling example, the schedule requester's plan is depicted in Fig. 2a. The plan starts when the *request*

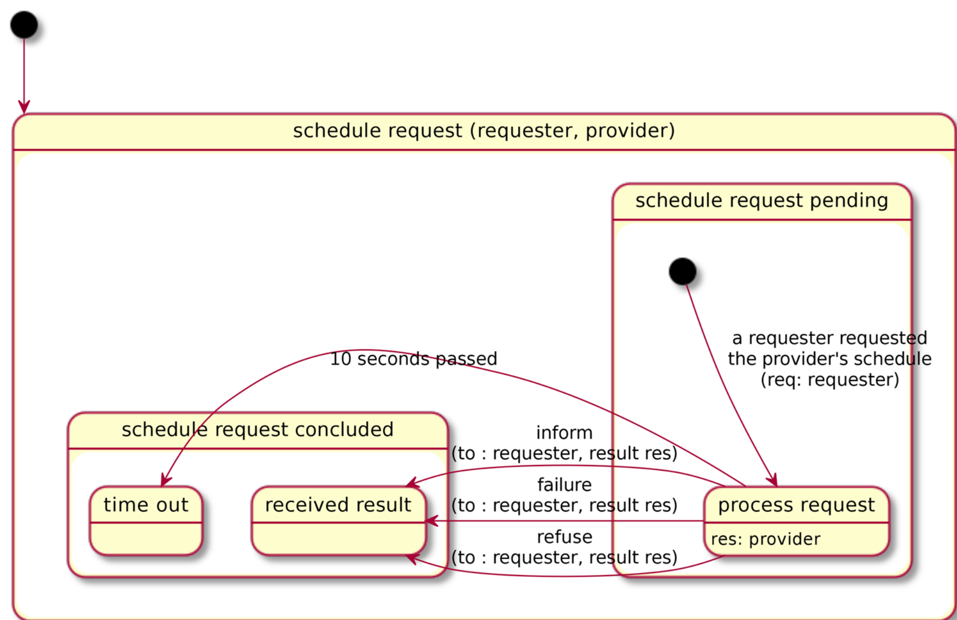


Fig. 3 A propositional statechart representation of a protocol for requesting a schedule

intra-agent message is instantiated (the guard checks that it is not null) and the role sends the *Request* message. The intra-message means that the *request* has been instantiated by another plan of this agent. The role senses this instantiation and initiates this new plan.

The plan goes to the *Wait* state and on entry it sets a timer for 10,000 milliseconds (10 seconds). The plan finishes when the role receives one of three messages or the timer time-outs. The three possible responses are :

– **Performative**: *Inform*, **Content**: *results*. The *results* will typically be an instance of a *Schedule* class (it is assumed that it has been defined in an ontology).
– **Performative**: *Refuse*, **Content**: *null*. The *null* entry shows that this item is empty, i.e. the role does not have to provide more information other than the utterance indicated by the performative.
– **Performative**: *Failure*, **Content**: *null*.

The provider's plan is depicted in Fig. 2b. The plan starts when the Request inter-agent message is received. It goes to the *ProcessRequest* state. The plan finishes when the role sends its reply or the timer time-outs.

Modeling Agent Interactions

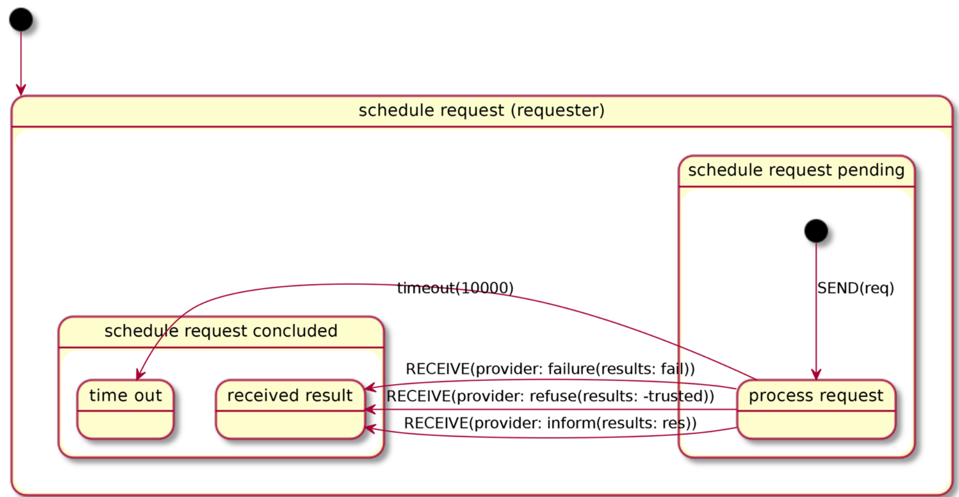
In this section, we try to see this interaction protocol from the point of view of the propositional statecharts method [18]. In propositional statecharts, there are three (or more) views, a global view where the protocol is defined abstractly, and then individual views, one per participating role.

In the case of the schedule request protocol a possible global view is depicted in Fig. 3. The protocol is represented by an *or-type* state that contains two *or-type* substates, one for encapsulating all the different states that represent the execution of the protocol (in our example the *schedule request pending* state) and one for encapsulating the possible concluding states (the *schedule request concluded* state). In the former, we have just one *basic-state* for processing the request, where the provider is expected to deliver a result - *res*. The possible concluding states are that the protocol finished with a timeout (*time out*) and that the protocol finished by the requester sending a message to the provider (*received result*).

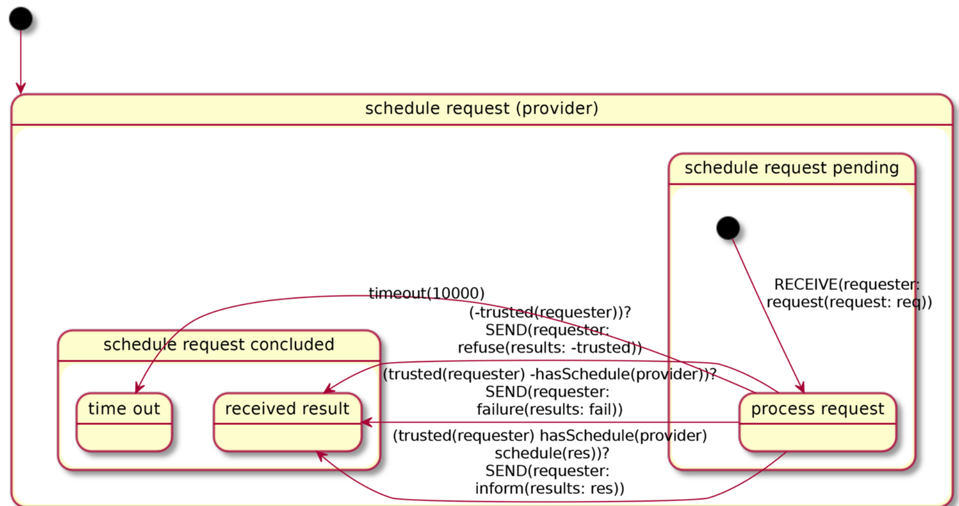
The different roles' implementation of the abstract protocol are subsequently defined. The states remain as they are, however, the transition expressions and state actions change to reflect each role's actions and monitored events. The reader can check the possible implementations of the abstract protocol in Fig. 4a for the requester role and in Fig. 4b for the provider, Note how the different inbound and outbound messages are modeled and the logical language in the expressions.

Modeling Inter- and Intra-agent Control

ASEME defines protocols as statecharts where the participating roles are defined as orthogonal components. See for example the schedule request protocol model in Fig. 5, with the schedule requester (sr) and the schedule provider (sp) as orthogonal components in the *ScheduleRequest* protocol state. Moreover, these roles are abstract roles, any user's personal assistant (PA) can use this protocol either as a



(a) Propositional statechart representation for a requester agent.



(b) Propositional statechart representation for a provider agent.

Fig. 4 The individual statecharts for the requester and provider roles

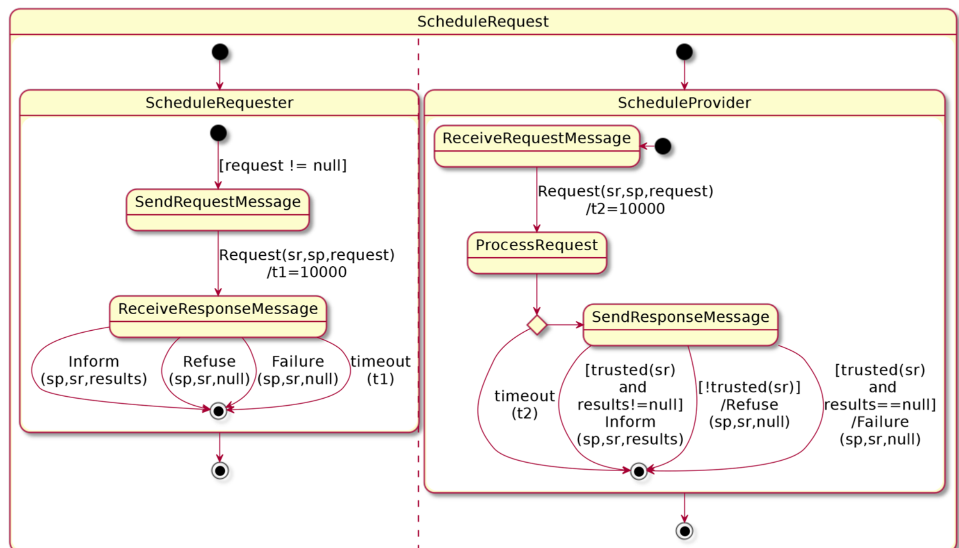


Fig. 5 Statechart representation of a protocol for requesting a schedule. The diamond shape represents a condition state

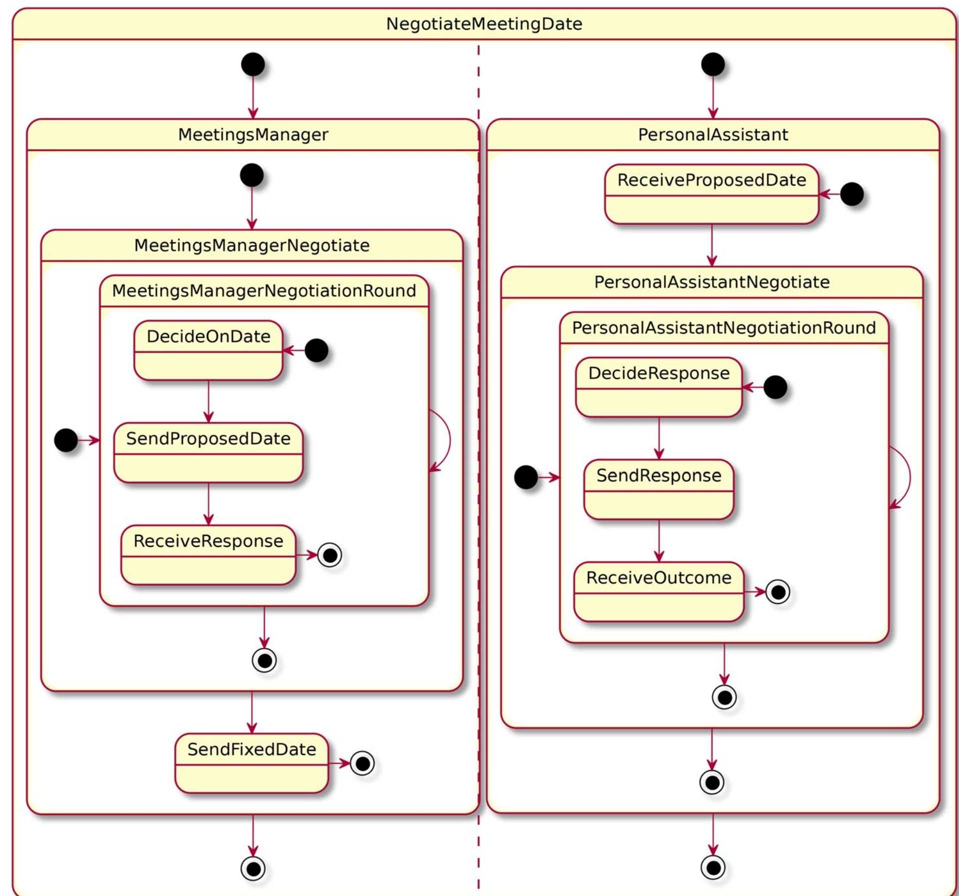


Fig. 6 Statechart representation of a protocol for negotiating a meeting date

requester or as a provider (we use the abbreviation *sr* for schedule requester and *sp* for schedule provider).

The *sr* sends a *message* using the *Request* performative whose variables are the sending and receiving agents and the *request*, which can be an object for object-oriented implementations or a query for logic-based implementations. On the other hand, the schedule provider waits to receive this message, then processes the request and either replies with a *Refuse* (the service is refused for this agent), *Failure* (failed to reply), or *Inform* (with the user's schedule data) performative. Note that the protocol terminates for both roles after a timeout of 10,000 milliseconds.

A similar model also appears in the work of Seo et al. [78] for buying products. Note the use of the message receiving states as synchronization points where an agent waits for another to finish its task, similarly to the work of Murray [60].

In Fig. 6 the reader can see yet another protocol definition, the one for negotiating the meeting date. This protocol is between the meetings manager (*mm*) and personal assistant (*pa*) roles. Now these are concrete roles, not abstract such as the *sr* and *sp*. In the figure, the reader can check out the composite states representing negotiation rounds. The transitions that have these states (i.e. *MeetingsManagerNegotiationRound* and *PersonalAssistantNegotiationRound*) both as source and target facilitate this. Note that transition expressions have been omitted so as not to clutter the diagram. This statechart is valid for one or more *pa* role participants as the *ReceiveResponse* state can be set to receive any number of requests. The number of requests can even be a variable so that the number of *pa* role participants is dynamic.

Then, when the designer defines the intra-agent control, again as a statechart, she can get the desired role (orthogonal component from a protocol definition) and use it. Moreover, although protocols define the control aspect of the role (events, conditions, actions in TE), the activity part is left open for the agent developer to realise, similarly to the approach of Moore [55]. Thus, orthogonality is very helpful for providing a complete view of the protocol including all possible actors. Then, when it comes to implementation, each agent type can realize only the orthogonal component that corresponds to its role. Also, using orthogonality, one can develop (and simulate) agents that can concurrently participate into more than one protocols.

The work of Moore [55] supported the possibility of an agent getting involved in a *sub-dialog* when in a dialog. Moore supposed that the agent has access to a repository of

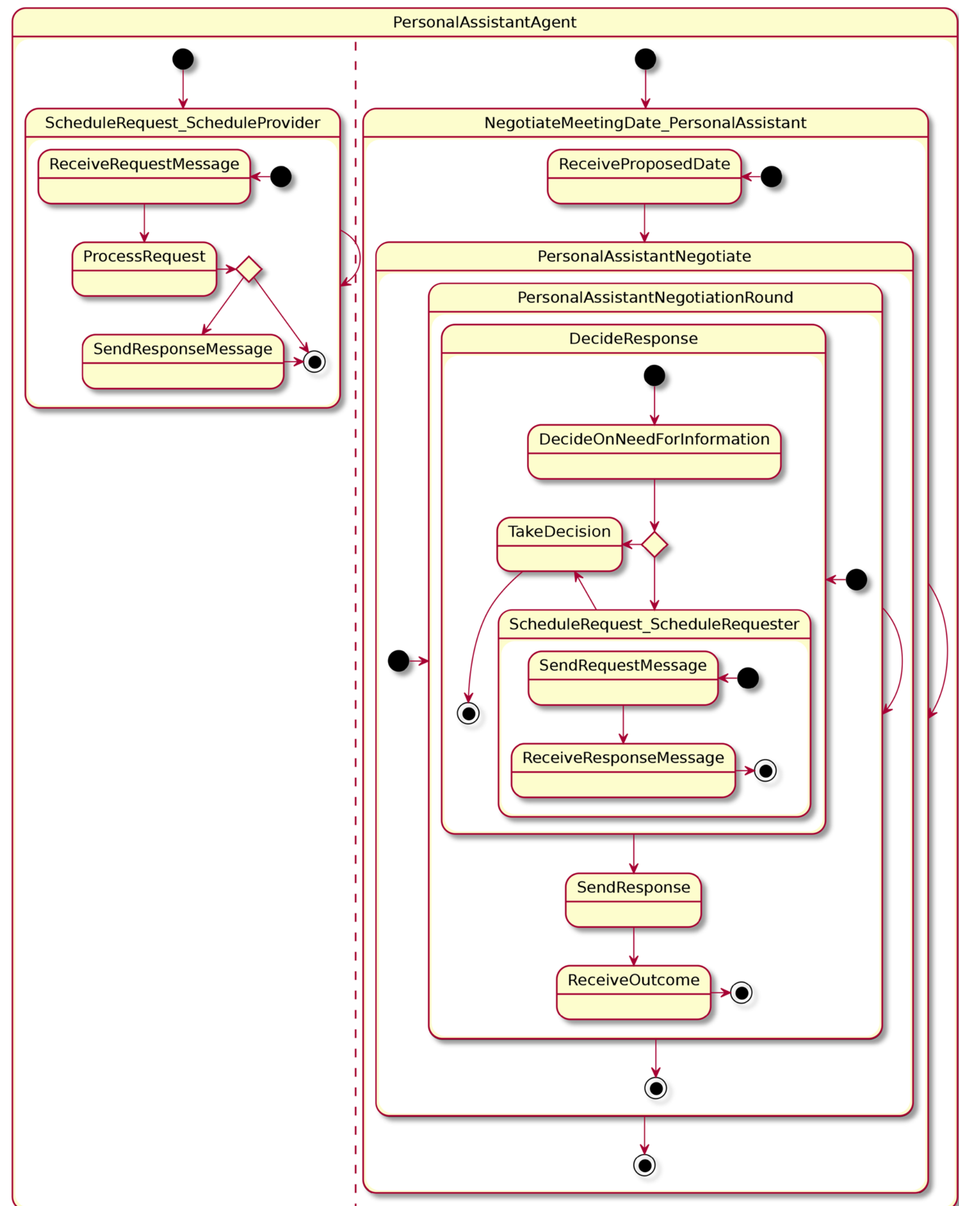


Fig. 7 Statechart representation of a personal assistant agent featuring **a** an embedded dialog (*ScheduleRequest_ScheduleRequester* embedded in *PersonalAssistantNegotiate*) and **b** a sub-dialog (*ScheduleRequest_ScheduleProvider*)

dialogs and dynamically selects a sub-dialog model whenever an incoming message is not permitted by the existing dialog but is permitted by another in the repository. In the intra-agent control model, ASEME allows for this possibility as all roles the agent can participate in can be instantiated as orthogonal components. Information between orthogonal components can be exchanged through the use of common variables and their usage in transition expressions, thus, a given protocol can remain in a given state until some information becomes available (an implicit intra-agent message).

Another feature of ASEME is the catering for *embedded dialogs* in an agent's design, i.e. in its intra-agent control model. Dialogs occur when an agent participates in an agent interaction protocol. Instances of dialogs contained entirely within other dialogs are said to be embedded [53]. ASEME defines that when a role in a protocol model is integrated into an intra-agent control model, the protocol role *or-type* state is inserted as-is in the intra-agent control. Then, the designer is free to define the activities of the basic states. The designer can even select to expand a basic state and turn it to an *or-type* state. A very interesting recent paper of Syriani et al. [92] researches exactly this issue, i.e. what refinements are possible to a part of a statechart that preserve its structure and behavior.

To illustrate these features, we provide the intra-agent control model of a personal assistant agent that has the capability to participate in both the above protocols in Fig. 7. The *pa* realizes the like-named role in the *NegotiateMeetingDate*

Table 2 Real world agent-based systems modeled using statecharts and their application domains

Work	Year	Keywords
[61]	2001	Robocup; simulation
[29]	2003	Safety critical; dynamic adaptation; mechatronic systems
[57]	2004	Infomobility services; personal travel assistance
[28]	2005	E-commerce; automated negotiations
[14]	2008	Conference management system
[79]	2008	Project management; productivity and quality improvements
[27]	2009	Educational system
[86]	2009	Product pricing; retail business
[80, 100]	2010	Clinical decision support systems
[82, 83]	2012	Wind turbine; structural health monitoring; smart structures
[75]	2012	Crowd behaviour simulation; termites simulation
[94]	2013	Robocup; soccer; physical robot;
[68]	2014	Wumpus world; blackboard
[88]	2015	Health care; ambient intelligence; ambient assisted living
[1]	2016	Automation; supervisory control; data acquisition; real-time monitoring; large-scale SCADA; adaptive industrial networks; PLC
[39]	2016	Smart grid; demand response; flexible energy market
[62]	2018	Manufacturing systems; internet of things
[20]	2018	Demand response; home energy management; flexibility; access control; device abstraction
[3]	2019	Robotic application; personal assistant

protocol (in the *NegotiateMeetingDate_PersonalAssistant* composite state). While negotiating, it is always possible for another agent or the meetings manager to ask it for its user's schedule (a sub-dialog), as it realises the *ScheduleProvider* role of the *ScheduleRequest* protocol (in the *ScheduleRequest_ScheduleProvider* composite state).

Another interesting feature that exploits the composite states of a statechart is the ability to expand a *basic* state and further elaborate it. For example, in Fig. 7 the intra-agent control modeler has chosen to expand the *DecideResponse* state in the *PersonalAssistantNegotiationRound* state to give to the agent the possibility to explore another agent's schedule before taking its decision. This is how a dialog is embedded within another. This time the PA uses the *ScheduleRequest* protocol as a *ScheduleRequester*. A similar approach has been followed by Dumas [17].

Real World Applications of Agents Modeled with Statecharts

In this section, we try to provide a view of the agent-based real-world systems that have been developed using methods that employ the language of statecharts. We searched the literature for papers reporting on the development of such systems and we found out that several such exist. With the term real-world systems, we mean systems that have been implemented and evaluated, usually in the context of a research and development project. And, of course, they must have been reported in the literature.

Table 2 shows the systems/applications that we collected sorted by the year of publication to show the time on which these applications started drawing attention. A first use of this table is to identify the application domains. Another use is the application domains trend. For example, the robotic applications domain seems to be spread throughout the covered years. Smart grid and internet of things-related systems are currently trending.

Specifically, among them we find single-agent or MAS applications for the *smart grid*, e.g. secure, automated home energy management [20], and a power distribution network management system [39], *internet agents*, e.g. information for people on the move using their smart phones [57], *automated negotiations* [28], a conference management system [14], a product pricing agent for retail market chains [86], a personalized learning platform [27], *structural health monitoring* systems [82, 83], *robotic applications*, e.g. a soccer playing robot for the robocup competition [94], a *personal assistant* robot [3], *production and management in industry 4.0*, e.g. *safety critical* mechatronic systems [29], project management and quality assurance [79], a system integrating IoT (Internet of Things) devices to a MAS-based manufacturing environment [62], *simulation*, e.g. robotic soccer [61], social simulation [75], playing the wumpus game [68], *remote monitoring* and *health care* [1, 88].

Table 3 Real world agent-based systems modeled using statecharts. This table aims to provide information about how they were developed

Work	Language	Engine	Method	Agent type	Comm.	Std
[61]	UML	Robolog	Ad-hoc	Software	None	None
[29]	UML clone	Proprietary	Fujaba	Software	ACL	FIPA
[57]	FSM	JADE	Gaia2JADE	Software	ACL	FIPA
[28]	UML	JADE	Ad-hoc	Software	ACL	FIPA
[14]	agentTool	Proprietary	MaSE	Software	Events	None
[79]	FSM	JADE	Gaia2JADE	Software	ACL	FIPA
[27]	agentTool	Proprietary	MaSE	Software	Events	None
[86]	AMOLA	Rhapsody	ASEME	Software	None	None
[80, 100]	AgentTool	JADE	MaSE	Software	ACL	FIPA
[82, 83]	AMOLA	JADE	AGEME	Software	ACL	FIPA
[75]	TSTATES	Netlogo	Ad-hoc	Software	None	None
[94]	AMOLA	Monas	KSE	physical	Blackboard	None
[68]	AMOLA	Proprietary	ASEME	software	None	None
[88]	AMOLA	JADE	ASEME	Software	ACL	FIPA
[1]	FSM	JADE	Ad-hoc	Software	ACL	FIPA
[39]	FSM	JADE	Gaia2JADE	Software	ACL	FIPA
[62]	UML	AnyLogic	AUML	Software	Events	FIPA
[20]	FSM	JADE	Gaia2JADE	Software	ACL	FIPA
[3]	Harel clone	Armarx	Armarx	Physical	Events	None

Table 3 tries to provide a view of the examined applications with respect to the models used for design. Thus, on the second column the reader will find the modeling language used. It can range from a simple diagram of the statechart, i.e. FSM, UML and Harel, to a Domain Specific Language (DSL), i.e. agentTool [49], AMOLA [85] and TSTATES [75]. The reader can connect the applications to the paradigms that we have seen in the previous sections. Thus, *FSM* and *TSTATES* correspond to finite state machines (Def. 1), *agentTool* (supporting the MaSE methodology), *UML*, *UML clone* and *Harel clone* correspond to Harel statecharts (Def. 2), and, *AMOLA* to ASEME statecharts (Def. 3).

On the third column of Table 3, the statechart execution engine is mentioned, i.e. , AnyLogic (https://www.anylogic.com), JADE [5], Monas [69], NetLogo [81], Rhapsody [35], Robolog [61]. The term “proprietary” refers to engines that were not found online or in the literature. This column can give the reader an idea on the different platforms that are compatible with the models of the previous columns.

The fourth column refers to the development method used. There, we find *Fujaba* [63] (a UML-based method with an open source CASE tool providing developers support for model-based software engineering and re-engineering), the *Gaia2JADE* process [58] (an AOSE method for developing agents using Gaia for analysis and JADE for implementation), AOSE methodologies, i.e. *MaSE* [16], *Agent UML* [40] *(AUML)*, *ASEME* [91] and *AGEME* [82] (an ASEME clone), and Armarx [98] (a statecharts-based method for robotic applications development).

The fifth column reports on the software or physical agent type (there are two robots). The sixth column reports on the communication method used. Most of the systems used the standardized FIPA[1] *ACL*. Message-based communication that is not compliant with the FIPA standard is indicated as *Event*. One system employs a *Blackboard*. The blackboard architecture [37] allows coordination through access to shared information. Agents read and write information on the blackboard in topics. they can also subscribe to topics and be informed when a new message is written in that topic. The blackboard architecture is quite common in robotic systems [94, 106]. In a few single agent systems or systems where agents can fully access the environment (check out Table 4) there is no communication between the agents.

The last column of Table 3 indicates if the FIPA standard has been used for developing the system. This standard is followed by the JADE framework. Other platforms usually employ a specific standardized aspect, e.g. the ACL message structure [29] or the AUML method for modeling agent interactions [62].

Table 4 shows the environment type of these applications using the categories proposed by Russel and Norvig [73]. The environment may be *Fully* or *Partially* observable (column two). In column three we identify the agents in the developed systems as cooperative (*Coop*) or competitive

[1] FIPA (Foundation for Intelligent Physical Agents) is an IEEE Computer Society standards organization for agent-based technology, http://www.fipa.org.

Table 4 Real world agent-based systems modeled using statecharts. Environment types for the examined applications

Work	Observable	Agents	Determinism	Episodic	Dynamic	Time model
[61]	Fully	Comp	Stochastic	Seq	Dynamic	Continuous
[29]	Partially	Coop	Stochastic	Seq	Dynamic	Continuous
[57]	Partially	Coop	Stochastic	Seq	Dynamic	Continuous
[28]	Partially	Comp	Stochastic	Seq	Dynamic	Continuous
[14]	Partially	Coop	Stochastic	Seq	Dynamic	Discrete
[79]	Partially	Coop	Stochastic	Seq	Dynamic	Continuous
[27]	Partially	Coop	Stochastic	Seq	Dynamic	Continuous
[86]	Partially	Single	Stochastic	Episodic	Dynamic	Discrete
[80, 100]	Partially	Coop	Deterministic	Episodic	Static	Continuous
[82, 83]	Partially	Coop	Stochastic	Seq	Dynamic	Continuous
[75]	Partially	Coop	Stochastic	Episodic	Dynamic	Discrete
[94]	Partially	Coop	Stochastic	Seq	Dynamic	Continuous
[68]	Fully	Single	Deterministic	Seq	Static	Discrete
[88]	Partially	Coop	Stochastic	Seq	Dynamic	Continuous
[1]	Partially	Coop	Stochastic	Seq	Dynamic	Continuous
[39]	Partially	Coop	Stochastic	Seq	Dynamic	Continuous
[62]	Partially	Coop	Stochastic	Seq	Dynamic	Continuous
[20]	Partially	Coop	Stochastic	Seq	Dynamic	Continuous
[3]	Partially	Single	Stochastic	Seq	Dynamic	Continuous

Comp is an abbreviation for *Competitive*, *Coop* for *Cooperative* and *Seq* for *Sequential*

(*Comp*). We even have some *Single* agent systems. If the next state of the environment is completely determined by the current state and the action executed by the agent, then it is *Deterministic*, if not, then it is *Stochastic* (see column four). If the action of an agent depends solely in the state of the environment and not in the agent's memory then we call the environment *Episodic*, otherwise *Sequential* (column five). *Static* environments do not evolve over time while *Dynamic* environments do (column six). Finally, time can be *Continuous*, as in the real world, or it may "tick" periodically (usually in simulations) and be *Discrete*. From the table it seems that in their vast majority statecharts-based agent applications concern cooperative agents that have a partial view on the stochastic, sequential, dynamic environment where they operate in a continuous time model.

Discussion

The statecharts main added value is the capability of the language to capture both the static (activities and variables) and dynamic aspects of a system [34, 36]. Thus, one can have a unique design model and use it to generate code for diverse platforms.

Statecharts and FSMs are pivotal to the development of MAS. For example, Hahn et al. developed a meta-model for defining a platform independent model for agents (PIM4Agents) [33]. According to that work, agent decision and protocol concepts are transformed to JADE *FSMBehaviours*. Thus, the core concepts of agency (communication and decision-making/autonomy) are related to FSMs.

One of our findings by working with statecharts is that agent behavior specification is not a trivial task. The development of the simplest possible player in Robocup took a statechart with 99 states in a hierarchy with a depth of 17 [69, 94]. In this direction, the modern model driven engineering (MDE) approaches (see for example a special issue in this matter [44]) can aid the development process, as model-transformation can help initialize the statechart(s). This demonstrated the added value of the ASEME methodology as it allows for the automatic transformation of Gaia liveness formulas to a statechart [87], which is at least a "good start", as opposed to starting the design directly with a statechart CASE tool, as was the case of StatEdit [60], or using a flat statechart model with no hierarchy, such as the plan diagrams of MaSE [16].

For example, 28 students taking the Autonomous Agents class at the Electrical and Computer Engineering School of the Technical University of Crete were asked to develop a Robocup soccer humanoid player in one of the 2-hour laboratory sessions of the class. The students worked in small teams of two or three people per team. The students first went through a quick tutorial on using the KSE CASE tool, which demonstrated the development of a Goalie behavior for the Nao robot. This included the Gaia formulas for the goalie role, and its IAC (Intra-Agent Control) model. Then, they were asked to use the existing functionalities of the Goalie (scan for the ball, kick the ball, approach the ball,

Table 5 Four categories of this paper's references

Type of research	Related references
AOSE/EMAS-statecharts research	[2, 5, 15–18, 23–26, 32, 33, 40, 45, 48, 49, 54–56, 58, 60, 65–67, 69, 70, 75, 76, 78, 85, 87, 89, 91, 94, 96, 98, 99, 104]
AOSE/EMAS-statecharts applications	[1, 3, 14, 20, 27–29, 39, 57, 61, 62, 68, 79, 80, 82–84, 86, 88, 100]
General research	[4, 6, 8, 9, 12, 13, 21, 22, 37, 38, 41–44, 46, 47, 52, 53, 59, 63, 64, 71–74, 77, 81, 90, 93, 95, 97, 101–103, 105]
Statecharts specific research	[7, 10, 11, 19, 30, 31, 34–36, 50, 51, 92, 106]

etc) to develop an *Attacker* behavior using KSE. Thus, the students did not have to develop the robot functionalities. They defined the attacker role's liveness and then edited the generated statechart, i.e. they defined variables and transition expressions. All student teams were able to deliver an *Attacker* behavior and enjoyed watching their players in a game (for more information the reader can consult [91, 94]).

Proposing radical extensions to the language of statecharts may seem to facilitate or enable new features, e.g. as in the case of propositional statecharts that we examined earlier, however, it renders them incompatible with existing CASE tools and they may become difficult for mainstream software engineers to learn and use [72].

Some times, and especially in works that do not adopt the orthogonal components of statecharts (i.e. *and-type* states), it is not obvious how one develops an agent realising more than one protocols simultaneously, and/or how to combine them with other agent capabilities.

The ASEME inter and intra-agent control models, being derived by Gaia formulas, do not use the possibility of the state transitions to traverse levels or the history connectors. If the developers choose to introduce these features to the statechart they lose the connection of the design phase models (the statecharts) to the analysis phase models (i.e. the role model and the Gaia formulas). This situation can impact the tracing of software features to their requirements and has been reported as the "round-trip" problem [77]. The acquired experience after modeling a number of systems for software and robotic agents shows that, on one hand, the choice to not use state transitions traversing levels or the history connectors does not hinder the possibility to model complex systems, and, on the other hand, important engineering concepts, such as comprehension, modularity and reusability, are enhanced. This has been reported by the more recent work on Armax statecharts for modeling robotic behaviour [98].

AOSE/EMAS and statecharts related research is an area that has been active during the last 20 years. We have found a number of works in theory and applications. Table 5 shows these works in its first two rows. We employed these two rows to create the histogram depicted in Fig. 8. We used the centers of the bins (shown in the x axis labels) to draw the trend line shown in the figure (using linear regression). The latter indicates that we must expect to see more works in this area in the coming years.

Related Work

Regarding related work, this survey is the first of its kind, i.e. to outline the use of statecharts in AOSE/EMAS. However, it is related to works covering the areas for which statecharts are used, i.e. for representing plans, agent interaction protocols and intra-agent control.

A recent survey on cooperative Multi-Agent Planning (MAP) considers multiple agents that work cooperatively to develop an action plan to satisfy their collective goals. This area is mostly concerned not on modeling but on studying the reasoning side of planning [95]. Thus, related to this work are mostly works that use Finite State Machines for representing plans [96]. Interestingly, Harel statecharts have not yet been employed as such by that community. This might be an interesting introduction, but one that needs to support reasoning through well-defined logical semantics. This isn't so easy, as different statechart engines can have differences in execution semantics [11].

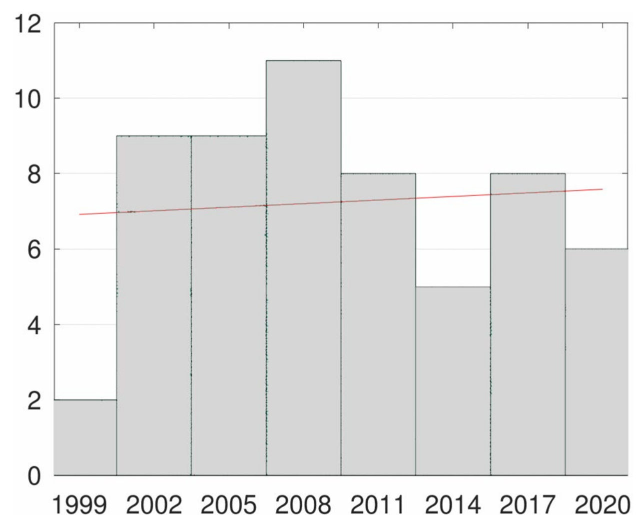


Fig. 8 A histogram of the AOSE/EMAS-statecharts related research items from 1998 to 2021. A trend line is also shown

If we try to locate recent agent interaction protocol definition techniques or methods surveys, we will find that AOSE/EMAS researchers propose their models for such interactions but surveys are rare. However, researchers such as Paurobally et al. [70] have argued on other pros and cons of statecharts. Based mostly on that work, we present some of the statecharts advantages (+) and disadvantages (−) over other techniques:

+ States and processes can be treated equally allowing an agent to refer and reason about the state of an interaction
+ Statechart notation is more amendable for extension thanks to their simple semantics
+ Visual models are easier to conceive and display [26]
+ Engineers familiar with UML can start working with them immediately [72]
− Participating roles are not shown explicitly
− Compound transitions are not shown in detail
− There is a question of completeness

It is beyond the scope of this work to present all other techniques, however we can name a few that employ a visual language, such as Agent UML (or AUML) [41], Petri Nets [52], or the Business Process Modeling Notation (BPMN) [21, 46]. In BPMN, the interaction of two or more participant roles is called a *choreography*. It was very recently reported that the BPMN language has specific limitations that prevent some communication scenarios from being modeled [22]. An example is the case of an agent changing its orders. However, BPMN also has specific advantages, on one hand it can be used to define interactions between humans and agents (or Human-Agent Collectives [43]), on the other hand, its models are executable. BPMN has also been used for early validation of a MAS analysis models before even the design phase is concluded [90].

Finally, the intra-agent control model exists in all agent architectures. An architecture defines how the different components are connected and interact. Among the available candidates are service oriented architectures and microservices [93] that define a completely stateless and loose connection between the agent's components. BPMN has also been used, along with logic-based proposals, such as Multi-Context Systems based on a modular architecture where components interact using bridge rules for filtering information exchange [74].

Future Directions

The future holds many challenges. Ozik et al. [67] consider that the use of statecharts for agent-based modeling is still at its "nascent stages". This is also reflected by the fact that real-world systems are not so numerous as the undertaken research. They propose the development of design and process patterns for capturing idiomatic state-based agent behaviours.

Regarding the use of statecharts, agents and autonomous systems continuously face the possibility of an unexpected (at design time) event to happen while they are in operation. Unexpected means that either a known event happens that the system is unable to handle at its current state (unexpected at that time), although it is related to its operation, or an unknown event happens (totally unexpected), see Marron et al. [51] for a detailed definition. Although there are some "hacks" for ad-hoc catering for this issue, such as having a default handler for incoming messages not handled by a defined behaviour that replies with information about the services offered by the agent, this is a valid research direction.

In the area of design for autonomy (empowerment, self-management and self-regulation) it is very interesting to research how an inter- or intra-agent control can self-evolve over time. Evolution may be triggered through introspection or through the desire to maximize or fine-tune an agent's performance. This is a very important research direction for the area of robotics. For example, a robot may have a failing limb, it may need to fine tune its grasp to manage its best with the available functionality. Or to fine-tune the robot's situational awareness [3]. Another kind of evolution is to evolve the statechart itself. Researchers are already delving into this area with results only for flat statecharts until now [31].

Recently, researchers proposed the concept of the property statecharts [54] for expressing and enforcing safety criteria in statecharts. Safety properties have been defined in AOSE, and the Gaia methodology's role model [101], however, statechart-based design models have not yet fully realised this feature, especially those leading to object-oriented implementations. Property statecharts are monitoring the events generated by the execution of normal statecharts and safeguard conditions. An important emerging area, further enabled by the emergence of blockchain and 5G [97], is that of the *smart contracts*, Mens et al. [54] have given an example, where an agent A signs a Service Level Agreement (SLA) with agent B. The SLA dictates that whenever A receives a request from B, then A must reply within 1 hour. The property statechart gets in the monitoring state whenever A receives a request from B. If the A's state for sending a reply to B is not exited within 1 hour the contract is considered violated. It would be very interesting to adapt this idea to safety properties of agents.

In the same work [54], the authors proposed a method for transforming user stories to statecharts in the context of Behavior-Driven Development (BDD) [103]. BDD allows users to specify representative scenarios and their expected outcomes. This is an emerging area in EMAS as researchers are already proposing the inclusion of BDD ideas in

agent-oriented methodologies [99]. According to Wautelet et al. [99], if an agent task is complex, it is transformed into a JADE *CompositeBehaviour* (a behaviour type that is composed of other behaviors). Here, there is room for the use of statecharts for defining this composite behaviour. This is also an existing trend in modern statechart CASE tools, e.g. for the Yakindu tool [50]. The reported challenges are related to the enhancement of statecharts with additional scenario-based modeling-related features, like strict event ordering, and specification of liveness properties for execution control and verification [50]. In this context, the liveness formulas transformation to statecharts [87] might be an excellent contribution from the EMAS/AOSE community to the statecharts one.

To realize implementations of agents in the modern open systems [42], agents need to use predefined protocols to interact. However, when diverse stakeholders come in, they need to work the protocols with their own algorithms and/or goals. Currently, protocols focus on defining sequences of exchanged messages. Adopting the point of view of the ASEME methodology [85, 91], where protocols are regarded not as simple communication protocols that determine how data is transmitted (as in telecommunications and computer networking), but as their higher-level abstractions used by humans, where protocols define *codes of behaviour* (or *procedural methods*), we can use statecharts for defining them. Thus, a protocol does not only answer the question of what messages are allowed but also what activities the participants need to engage with within the protocol. In this context, an important direction is towards defining new design patterns, that on the one hand will allow the developers to re-use existing protocol parts and logic defined in the open system; and on the other hand to customize key functionality or capabilities according to their needs and/or goals.

Thus, when defining open systems, or even proprietary systems, the use of statechart repositories would lead to the simplification of the statechart-based agent development. Consider for example, the Robocup Player agent that we referred to in the discussion above. It would be much easier to develop this agent if some parts of its statechart or intra-agent control model were reused from local or public repositories. A step forward would be to have the developers not reuse just activities of states (as they did in the above experiment) but whole statechart components (including transition expressions) as modules. Modules have also been referred to as capabilities in the AOSE community [8, 91]. Modular programming has been identified as the ultimate aim of agent programming languages and developing frameworks, be they declarative or imperative [12]. In the future, these modules may be assembled on the fly (at runtime) in a statechart realising the optimal behaviour of a system. Towards this direction, is the recent Gamma Statechart Composition Framework [32], an integrated modeling tool that aims to support the composition of heterogeneous statechart components. Its Gamma Composition Language supports the interconnection of components and reuses existing code generators.

If one checks the latest applications, a certain trend is towards application for Industry 4.0 (intelligent manufacturing) [62] and robots [3]. The internet of things applications will need to capture states of the environment and agents will want to manage IoT services. According to Nagadi et al. [62] the agent-based development process integration with the Internet of Things (IoT) reference model, and especially with its communication model, will enable to merge the right information with the relevant functionalities, thus affecting the performance of a Smart Manufacturing System (SMS).

Moreover, the smart grid is an active area and one that will certainly require MAS, as energy prices fluctuate and consumers need to continuously monitor the network for booking their consumptions [20, 39, 84].

Conclusion

Statecharts and AOSE are quite close, the former have influenced and empowered AOSE methods and techniques and the latter have proposed and tested a number of statecharts language extensions. The future holds more prospects for both areas but also for their cooperation. Numerous applications in the last 20 years show that this relationship is not only theoretic but also practical.

We provided a background on statecharts and finite state machines, and gave appropriate definitions to help to disambiguate these terms. Then, we explored the use of statecharts in AOSE and EMAS methodologies and methods. We identified the challenges that the researchers faced and how they coped with them, some times by defining the transition expressions language, and others by imposing restrictions or introducing new elements (such as the synch states). Then, we illustrated several features that statecharts have enabled in AOSE.

Subsequently, we explored the various agent-based real world applications we found in the literature and attempted to give a distilled view on several of their characteristics in tables.

We presented several future directions, mainly for the intra-agent's control, on one hand monitoring events that are essential for the agent's successful operation and for detecting failing capabilities, and, on the other hand, for safeguarding restrictions and contracts. Statecharts are a still evolving paradigm [3, 10, 50, 51, 54] and modern AOSE works use it [25, 67, 91].

In the future, this work can be expanded with more theoretical work from the statechart research community, also incorporating formal models and verification techniques.

Declarations

Conflict of interest The author declares that he has no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by the author.

References

1. Abbas HA, Shaheen SI, Amin MH. Self-adaptive large-scale SCADA system based on self-organised multi-agent systems. Int J Autom Control. 2016;10(3):234–66.
2. Arai T, Stolzenburg F. Multiagent systems specification by UML statecharts aiming at intelligent manufacturing. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002): Part 1, pp. 11–8, 2002. https://doi.org/10.1145/544741.544745.
3. Asfour T, Waechter M, Kaul L, Rader S, Weiner P, Ottenhaus S, Grimm R, Zhou Y, Grotz M, Paus F. Armar-6: a high-performance humanoid for human-robot collaboration in real-world scenarios. IEEE Robot Autom Mag. 2019;26(4):108–21.
4. Austin JL. How to do things with words. Cambridge: Harvard University Press; 1975.
5. Bellifemine FL, Caire G, Greenwood D. Developing multi-agent systems with JADE. New York: Wiley Series in Agent Technology Wiley; 2007.
6. Bordini RH, Braubach L, Dastani M, Seghrouchni AEF, Gomez-Sanz JJ, Leite J, O'Hare G, Pokahr A, Ricci A. A survey of programming languages and platforms for multi-agent systems. Informatica. 2006;30(1).
7. Brand D, Zafiropulo P. On communicating finite-state machines. J ACM (JACM). 1983;30(2):323–42.
8. Braubach L, Pokahr A, Lamersdorf W. Extending the capability concept for flexible BDI agent modularization. In: Bordini RH, Dastani MM, Dix J, El Fallah Seghrouchni A, editors. Programming multi-agent systems. Berlin: Springer; 2006. p. 139–55.
9. Bresciani P, Perini A, Giorgini P, Giunchiglia F, Mylopoulos J. Tropos: an agent-oriented software development methodology. Auton Agent Multi-Agent Syst. 2004;8(3):203–36. https://doi.org/10.1023/B:AGNT.0000018806.20944.ef.
10. Broad A, Argall B. Path planning under interface-based constraints for assistive robotics. In: Proceedings of the 26th International Conference on Automated Planning and Scheduling, ICAPS'16, p. 450–8. AAAI Press; 2016.
11. Crane ML, Dingel J. UML vs. classical vs. rhapsody statecharts: not all models are created equal. In: Briand L, Williams C, editors. Model driven engineering languages and systems. Berlin: Springer; 2005. p. 97–112.
12. Dastani M. Programming multi-agent systems. Knowl Eng Rev. 2015;30(4):394–418. https://doi.org/10.1017/S0269888915000077.
13. Dastani M, van BirnaRiemsdijk M, Meyer JJC. Programming multi-agent systems in 3APL. In: Multi-agent programming. Springer: Berlin; 2005. p. 39–67.
14. DeLoach SA. Developing a multiagent conference management system using the O-MaSE process framework. In: Luck M, Padgham L (eds.) Agent-Oriented Software Engineering VIII: 8th International Workshop, AOSE 2007, Honolulu, HI, USA, May 14, 2007, Revised Selected Papers, pp. 168–181. Berlin: Springer; 2008. https://doi.org/10.1007/978-3-540-79488-2_13.
15. DeLoach SA, Garcia-Ojeda JC. O-MaSE: a customisable approach to designing and building complex, adaptive multi-agent systems. Int J Agent-Oriented Softw Eng. 2010;4(3):244–80.
16. DeLoach SA, Wood MF, Sparkman CH. Multiagent systems engineering. Int J Softw Eng Knowl Eng. 2001;11(03):231–58.
17. Dumas M, Governatori G, ter Hofstede AH, Oaks P. A formal approach to negotiating agents development. Electron Commer Res Appl. 2002;1(2):193–207. https://doi.org/10.1016/S1567-4223(02)00016-9.
18. Dunn-Davies H, Cunningham R, Paurobally S. Propositional statecharts for agent interaction protocols. Electron Notes Theor Comput Sci. 2005;134:55–75.
19. Eilenberg S. Automata, languages, and machines. New York: Academic Press; 1974.
20. Elshaafi H, Vinyals M, Grimaldi I, Davy S. Secure automated home energy management in multi-agent smart grid architecture. Technol Econ Smart Grids Sustain Energy. 2018;3(1):4.
21. Endert H, Küster T, Hirsch B, Albayrak S. Mapping BPMN to agents: an analysis. IN: Agents, web-services, and ontologies integrated methodologies. 2007, pp. 43–58.
22. Fleischmann A. Limitations of choreography specifications with BPMN. In: International Conference on Subject-Oriented Business Process Management, pp. 203–16. Springer; 2020.
23. Fortino G, Garro A, Mascillaro S, Russo W. Using event-driven lightweight DSC-based agents for mas modelling. Int J Agent-Oriented Softw Eng. 2010;4(2):113–40.
24. Fortino G, Rango F, Russo W. Statecharts-based JADE agents and tools for engineering multi-agent systems. In: Setchi R, Jordanov I, Howlett RJ, Jain LC, editors. Knowledge-based and intelligent information and engineering systems. Berlin: Springer; 2010. p. 240–50.
25. Fortino G, Rango F, Russo W, Santoro C. Translation of statechart agents into a BDI framework for MAS engineering. Eng Appl Artif Intell. 2015;41:287–97.
26. Fortino G, Russo W, Zimeo E. A statecharts-based software development process for mobile agents. Inf Softw Technol. 2004;46(13):907–21.
27. Gago ISB, Werneck VMB, Costa RM. Modeling an educational multi-agent system in MaSE. In: Liu J, Wu J, Yao Y, Nishida T, editors. Active Media Technol. Berlin: Springer; 2009. p. 335–46.
28. Ganzha M, Paprzycki M, Pirvanescu A, Badica C, Abraham A. JADE-based multi-agent E-commerce environment: initial implementation. Analele Universitatii din Timisoara Seria Matematica-Informatica. 2005;42:79–100.
29. Giese H, Burmester S, Klein F, Schilling D, Tichy M. Multi-agent system design for safety-critical self-optimizing mechatronic systems with uml. In: OOPSLA 2003—Second International Workshop on Agent-Oriented Methodologies, pp. 21–32. 2003.
30. Girault A, Lee B, Lee EA. Hierarchical finite state machines with multiple concurrency models. IEEE Trans Comput Aided Des Integr Circuits Syst. 1999;18(6):742–60.
31. Goldsby HJ, Cheng BH, McKinley PK, Knoester DB, Ofria CA. Digital evolution of behavioral models for autonomic systems. In: Proceedings of the 5th IEEE International Conference on Autonomic Computing (ICAC 2008), pp. 87–96. Los Alamitos: IEEE Computer Society; 2008.
32. Graics B, Molnár V, Vörös A, Majzik I, Varró D. Mixed-semantics composition of statecharts for the component-based design of reactive systems. Softw Syst Model. 2020;19(6):1483–517.

33. Hahn C, Madrigal-Mora C, Fischer K. A platform-independent metamodel for multiagent systems. Auton Agent Multi-Agent Syst. 2009;18(2):239–66.
34. Harel D. Statecharts: a visual formalism for complex systems. Sci Comput Progr. 1987;8(3):231–74.
35. Harel D, Kugler H. The rhapsody semantics of statecharts (or, on the executable core of the uml). In: Ehrig H, Damm W, Desel J, Große-Rhode M, Reif W, Schnieder E, Westkämper E (eds) Integration of Software Specification Techniques for Applications in Engineering: Priority Program SoftSpez of the German Research Foundation (DFG), Final Report, pp. 325–54. Berlin: Springer; 2004. https://doi.org/10.1007/978-3-540-27863-4_19.
36. Harel D, Naamad A. The statemate semantics of statecharts. ACM Trans Softw Eng Methodol (TOSEM). 1996;5(4):293–333.
37. Hayes-Roth B. A blackboard architecture for control. Artif Intell. 1985;26(3):251–321. https://doi.org/10.1016/0004-3702(85)90063-3.
38. Henderson-Sellers B, Giorgini P, editors. Agent-oriented methodologies. Hershey: Idea Group Publishing; 2005.
39. Hippolyte JL, Howell S, Yuce B, Mourshed M, Sleiman HA, Vinyals M, Vanhée L. Ontology-based demand-side flexibility management in smart grids using a multi-agent system. In: 2016 IEEE International Smart Cities Conference (ISC2), pp. 1–7. IEEE; 2016.
40. Huget MP. Agent UML notation for multiagent system design. IEEE Int Comput. 2004;8(4):63–71.
41. Huget MP, Odell J. Representing agent interaction protocols with agent UML. In: International Workshop on Agent-Oriented Software Engineering, pp. 16–30. Springer; 2004.
42. Huynh TD, Jennings NR, Shadbolt NR. An integrated trust and reputation model for open multi-agent systems. Auton Agent Multi-Agent Syst. 2006;13(2):119–54.
43. Jennings NR, Moreau L, Nicholson D, Ramchurn S, Roberts S, Rodden T, Rogers A. Human-agent collectives. Commun ACM. 2014;57(12):80–8. https://doi.org/10.1145/2629559.
44. Kardas G, Gomez-Sanz JJ. Special issue on model-driven engineering of multi-agent systems in theory and practice. Comput Lang Syst Struct. 2017;50:140–1. https://doi.org/10.1016/j.cl.2017.07.002.
45. Kienzle J, Denault A, Vangheluwe H. Model-based design of computer-controlled game character behavior. In: Engels G, Opdyke B, Schmidt DC, Weil F, editors. Model driven engineering languages and systems. Berlin: Springer; 2007. p. 650–65.
46. Kir H, Erdogan N. A knowledge-intensive adaptive business process management framework. Inf Syst. 2021;95. https://doi.org/10.1016/jis2020101639.
47. Kleppe AG, Warmer J, Bast W. MDA explained: the model driven architecture: practice and promise. Boston: Addison-Wesley; 2003.
48. König R. State-based modeling method for multiagent conversation protocols and decision activities. In: Carbonell JG, Siekmann J, Kowalczyk R, Müller JP, Tianfield H, Unland R, editors. Agent technologies, infrastructures, tools, and applications for E-services. Berlin: Springer; 2003. p. 151–66.
49. Loach SAD, Wood M. Developing multiagent systems with agenttool. In: Castelfranchi C, Lespérance Y, editors. Intelligent agents VII agent theories architectures and languages. Berlin: Springer; 2001. p. 46–60.
50. Marron A, Hacohen Y, Harel D, Mülder A, Terfloth A. Embedding scenario-based modeling in statecharts. In: MODELS workshops, pp. 443–52. 2018.
51. Marron A, Limonad L, Pollack S, Harel D. Expecting the unexpected: developing autonomous-system design principles for reacting to unpredicted events and conditions. 2020.
52. Mazouzi H, Seghrouchni AEF, Haddad S. Open protocol design for complex interactions in multi-agent systems. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002): Part 2, AAMAS '02, pp. 517–26. New York: Association for Computing Machinery; 2002. https://doi.org/10.1145/544862.544866.
53. McBurney P, Parsons S. Dialogue games for agent argumentation. In: Simari G, Rahwan I, editors. Argumentation in artificial intelligence. Boston: Springer; 2009. p. 261–80. https://doi.org/10.1007/978-0-387-98197-0_13.
54. Mens T, Decan A, Spanoudakis NI. A method for testing and validating executable statechart models. Softw Syst Model. 2019;18(2):837–63.
55. Moore SA. On conversation policies and the need for exceptions. In: Dignum F, Greaves M, editors. Issues in agent communication. Berlin: Springer; 2000. p. 144–59.
56. Moraitis P, Petraki E, Spanoudakis NI. Engineering JADE agents with the Gaia Methodology. In: Carbonell JG, Siekmann J, Kowalczyk R, Müller JP, Tianfield H, Unland R, editors. Agent technologies, infrastructures, tools, and applications for E-services, lecture notes in computer science, vol. 2592. Berlin: Springer; 2003. p. 77–91. https://doi.org/10.1007/3-540-36559-1.
57. Moraitis P, Petraki E, Spanoudakis NI. Providing advanced, personalised infomobility services using agent technology. In: Bramer M, Ellis R, Macintosh A, editors. Applications and innovations in intelligent systems XI. London: Springer; 2004. p. 35–48.
58. Moraitis P, Spanoudakis N. The GAIA2JADE process for multi-agent systems development. Appl Artif Intell. 2006;20(2–4):251–73. https://doi.org/10.1080/08839510500484249.
59. Moulin B, Chaib-Draa B. An overview of distributed artificial intelligence. In: O'Hare GM, Jennings NR, editors. Foundations of distributed artificial intelligence. New York: Wiley; 1996.
60. Murray J. Specifying agent behaviors with UML statecharts and statedit. In: Polani D, Browning B, Bonarini A, Yoshida K, editors. RoboCup 2003: Robot Soccer World Cup VII. Berlin: Springer; 2004. p. 145–56.
61. Murray J, Obst O, Stolzenburg F. Robolog koblenz 2000. In: Stone P, Balch T, Kraetzschmar G, editors. RoboCup 2000: Robot Soccer World Cup IV. Berlin: Springer; 2001. p. 469–72.
62. Nagadi K, Rabelo L, Basingab M, Sarmiento AT, Jones A, Rahal A. A hybrid simulation-based assessment framework of smart manufacturing systems. Int J Comput Integr Manuf. 2018;31(2):115–28. https://doi.org/10.1080/0951192X.2017.1407449.
63. Nickel U, Niere J, Zündorf A. The FUJABA environment. In: Proceedings of the 22nd International Conference on Software Engineering, ICSE '00, pp. 742–5. Association for Computing Machinery; 2000. https://doi.org/10.1145/337180.337620.
64. Nilsson N. Teleo-reactive programs for agent control. J Artif Intell Res. 1994;1:139–58.
65. Nwana HS, Ndumu DT, Lee LC, Collis JC. Zeus: a toolkit for building distributed multiagent systems. Appl Artif Intell. 1999;13(1–2):129–85.
66. Obst O. Specifying rational agents with statecharts and utility functions. In: Birk A, Coradeschi S, Tadokoro S, editors. RoboCup 2001: Robot Soccer World Cup V. Berlin: Springer; 2002. p. 173–82.
67. Ozik J, Collier N, Combs T, Macal CM, Northe M. Repast simphony statecharts. J Artif Soc Soc Simul. 2015;18(3):11. https://doi.org/10.18564/jasss.2840.
68. Papadimitriou GL, Spanoudakis NI, Lagoudakis MG. Extending the kouretes statechart editor for generic agent behavior development. In: Iliadis L, Maglogiannis I, Papadopoulos H, editors. Artificial intelligence applications and innovations. Berlin: Springer; 2014. p. 182–92.
69. Paraschos A, Spanoudakis NI, Lagoudakis MG. Model-driven behavior specification for robotic teams. In: Proceedings of

the 11th International Conference on Autonomous Agents and Multiagent Systems—Volume 1, pp. 171–8. International Foundation for Autonomous Agents and Multiagent Systems; 2012.
70. Paurobally S, Cunningham J, Jennings NR. Developing agent interaction protocols using graphical and logical methodologies. In: Dastani MM, Dix J, El Fallah-Seghrouchni A, editors. Programming multi-agent systems. Berlin: Springer; 2004. p. 149–68.
71. Rao AS, Georgeff MP. Modeling rational agents within a BDI-architecture. KR. 1991, pp. 473–84.
72. Riemenschneider CK, Hardgrave BC, Davis FD. Explaining software developer acceptance of methodologies: a comparison of five theoretical models. IEEE Trans Software Eng. 2002;28(12):1135–45.
73. Russell S, Norvig P. Artificial intelligence: a modern approach. 3rd ed. Hoboken: Prentice Hall; 2010.
74. Sabater J, Sierra C, Parsons S, Jennings NR. Engineering executable agents using multi-context systems. J Logic Comput. 2002;12(3):413–42.
75. Sakellariou I. Agent based modelling and simulation using state machines. In: Second International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2012), pp. 270–9; 2012.
76. Sánchez P, Álvarez B, Martínez R, Iborra A. Embedding statecharts into teleo-reactive programs to model interactions between agents. J Syst Softw. 2017;131:78–97.
77. Selic B. The pragmatics of model-driven development. IEEE Softw. 2003;20(5):19–25.
78. Seo HS, Araragi T, Kwon YR. Modeling and testing agent systems based on statecharts. In: Núñez M, Maamar Z, Pelayo FL, Pousttchi K, Rubio F, editors. Applying formal methods: testing, performance, and M/E-commerce. Berlin: Springer; 2004. p. 308–21.
79. Sethuraman A, Yalla KK, Sarin A, Gorthi RP. Agents assisted software project management. In: Proceedings of the 1st Bangalore Annual Compute Conference, COMPUTE '08. New York: Association for Computing Machinery; 2008. https://doi.org/10.1145/1341771.1341777.
80. Shirabad JS, Wilk S, Michalowski W, Farion K. Implementing an integrative multi-agent clinical decision support system with open source software. J Med Syst. 2012;36(1):123–37.
81. Sklar E. Netlogo, a multi-agent simulation environment. Artif Life. 2007;13(3):303–11. https://doi.org/10.1162/artl.2007.13.3.303.
82. Smarsly K, Hartmann D. Agent-oriented development of hybrid wind turbine monitoring systems. In: Proceedings of ISCCBE International Conference on Computing in Civil and Building Engineering and the EG-ICE Workshop on Intelligent Computing in Engineering; 2010.
83. Smarsly K, Law KH. Advanced structural health monitoring based on multi-agent technology. Computation for Humanity: Information Technology to Advance Society. 2012.
84. Spanoudakis N, Akasiadis C, Kechagias G, Chalkiadakis G. An Open MAS Services Architecture for the V2G/G2V Problem. In: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, pp. 2198–200. 2019.
85. Spanoudakis N, Moraitis P. An agent modeling language implementing protocols through capabilities. In: Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology-Volume 02, pp. 578–82. IEEE Computer Society; 2008.
86. Spanoudakis N, Moraitis P. Engineering an agent-based system for product pricing automation. Eng Intell Syst. 2009;17(2):139.
87. Spanoudakis N, Moraitis P. Gaia agents implementation through models transformation. In: Yang JJ, Yokoo M, Ito T, Jin Z, Scerri P, editors. Principles of practice in multi-agent systems. Berlin: Springer; 2009. p. 127–42.
88. Spanoudakis N, Moraitis P. Engineering ambient intelligence systems using agent technology. IEEE Intell Syst. 2015;30(3):60–7.
89. Spanoudakis NI. Statecharts and AOSE: the past and the future. In: Multi-Agent Systems—17th European Conference, EUMAS 2020. Springer; 2020.
90. Spanoudakis NI, Floros E, Mitakidis N, Delias P. Validating mas analysis models with the ASEME methodology. Int J Agent-Oriented Softw Eng. 2018;6(2):211–40.
91. Spanoudakis NI, Moraitis P. The ASEME methodology. Int J Agent-Oriented Softw Eng (in press)
92. Syriani E, Sousa V, Lúcio L. Structure and behavior preserving statecharts refinements. Sci Comput Progr. 2019;170:45–79.
93. Thönes J. Microservices. IEEE Softw. 2015;32:1.
94. Topalidou-Kyniazopoulou A, Spanoudakis NI, Lagoudakis MG. A case tool for robot behavior development. In: Chen X, Stone P, Sucar LE, van der Zant T, editors. RoboCup 2012: Robot Soccer World Cup XVI. Berlin: Springer; 2013. p. 225–36.
95. Torreño A, Onaindia E, Komenda A, Štolba M. Cooperative multi-agent planning: a survey. ACM Comput Surv. 2017;50(6):1–32. https://doi.org/10.1145/3128584.
96. Tožička J, Jakubův J, Komenda A, Pěchouček M. Privacy-concerned multiagent planning. Knowl Inf Syst. 2016;48(3):581–618.
97. Varga P, Peto J, Franko A, Balla D, Haja D, Janky F, Soos G, Ficzere D, Maliosz M, Toka L. 5g support for industrial iot applications-challenges, solutions, and research gaps. Sensors. 2020;20(3):828. https://doi.org/10.3390/s20030828.
98. Wächter M, Ottenhaus S, Kröhnert M, Vahrenkamp N, Asfour T. The armarx statechart concept: graphical programing of robot behavior. Front Robot AI. 2016;3:33.
99. Wautelet Y, Heng S, Kiv S, Kolp M. User-story driven development of multi-agent systems: a process fragment for agile methods. Comput Lang Syst Struct. 2017;50:159–76. https://doi.org/10.1016/j.cl.2017.06.007.
100. Wilk S, Michalowski W, O'Sullivan D, Farion K, Matwin S. Engineering of a clinical decision support framework for the point of care use. In: AMIA Annual Symposium Proceedings, vol. 2008, p. 814. American Medical Informatics Association; 2008.
101. Wooldridge M, Jennings NR, Kinny D. The gaia methodology for agent-oriented analysis and design. Auton Agent Multi-Agent Syst. 2000;3(3):285–312.
102. Wooldridge MJ. An introduction to multiagent systems. New York: Wiley; 2009.
103. Wynne M, Hellesoy A. The cucumber book: behaviour-driven development for testers and developers. Raleigh: Pragmatic Bookshelf; 2012.
104. Yu Y, Lapouchnian A, Liaskos S, Mylopoulos J, Leite JCSP. From goals to high-variability software design. In: Foundations of intelligent systems. Berlin: Springer; 2008. p. 1–16.
105. Zambonelli F, Jennings NR, Wooldridge M. Developing multiagent systems: the Gaia methodology. ACM Trans Softw Eng Methodol. 2003;12(3):317–70. https://doi.org/10.1145/958961.958963.
106. Zieliński C, Figat M, Hexel R. Communication within multi-FSM based robotic systems. J Intell Robot Syst. 2019;93(3):787–805.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.