Science of Computer Programming 78 (2013) 747–761

Contents lists available at SciVerse ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

Multi-agent oriented programming with JaCaMo

Olivier Boissier [a,*], Rafael H. Bordini [b], Jomi F. Hübner [c], Alessandro Ricci [d], Andrea Santi [d]

[a] ISCOD/LSTI, Ecole Des Mines 158 Cours Fauriel, 42024 Saint-Etienne Cedex, France
[b] INF-UFRGS, Federal University of Rio Grande do Sul, CP 15064, 91501-970 Porto Alegre RS, Brazil
[c] DAS-UFSC, Federal University of Santa Catarina, CP 476, 88040-900 Florianópolis SC, Brazil
[d] DEIS, Alma Mater Studiorum–Università di Bologna, Via Venezia 52, 47521 Cesena (FC), Italy

ARTICLE INFO

Article history:
Received 7 March 2011
Received in revised form 19 August 2011
Accepted 6 October 2011
Available online 29 October 2011

Keywords:
Multi-agent oriented programming
Autonomous agents
Shared environments
Agent organisations

ABSTRACT

This paper brings together agent oriented programming, organisation oriented programming and environment oriented programming, all of which are programming paradigms that emerged out of research in the area of multi-agent systems. In putting together a programming model and concrete platform called JaCaMo which integrates important results and technologies in all those research directions, we show in this paper, with the combined paradigm, that we prefer to call "multi-agent oriented programming", the full potential of multi-agent systems as a programming paradigm. JaCaMo builds upon three existing platforms: *Jason* for programming autonomous agents, *M*OISE for programming agent organisations, and CArtAgO for programming shared environments. This paper also includes a simple example that illustrates the approach and discusses some real-world applications that have been or are being developed with JaCaMo.

1. Introduction

Current trends in computer science are facing up the challenges of building distributed and open software systems operating in dynamic and complex environments, interacting with and acting on the behalf of humans. In this context, multi-agent technologies can provide concepts and tools that give possible answers to the challenges of practical development of such systems by taking into consideration issues such as decentralisation and distribution of control, flexibility, adaptation, trust, security, and openness.

At least four separate communities within the multi-agent research community have been dealing with specific dimensions of such practical software development, namely the research communities interested in agent oriented programming languages [1–3], interaction languages and protocols [4], environment architectures, frameworks and infrastructures [5], and multi-agent organisation management systems [6–9]. The results produced by these communities have clearly demonstrated the importance of each dimension for the development of complex and distributed applications.

Nevertheless, perhaps a bit surprisingly, currently the engineering of such systems is still hampered by the usage of programming approaches and related development platforms that are mainly focused on just *one* specific dimension, at best connected in an ad hoc way with the other ones. This means that developers currently have ad hoc programming solutions for the development of complex multi-agent systems, typically based on low-level implementation mechanisms, without suitable high-level abstraction and tools. However the benefits of adopting a comprehensive approach – integrating different dimensions – were recognised in the context of AOSE (agent oriented software engineering) and MAS modelling communities quite a long time ago [10–12].

* Corresponding author.
E-mail addresses: Olivier.Boissier@emse.fr (O. Boissier), r.bordini@inf.ufrgs.br (R.H. Bordini), jomi@das.ufsc.br (J.F. Hübner), a.ricci@unibo.it (A. Ricci), a.santi@unibo.it (A. Santi).

0167-6423/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2011.10.004

In this paper, we aim at bringing such a perspective down to the programming level, investigating a concrete programming model and platform that allows a comprehensive integration of three multi-agent programming dimensions, namely the agent, environment, and organisation levels, preserving a strong separation of concerns but, at the same time, exploiting such dimensions in a synergistic way. To this end, we took as a starting point three existing platforms that have been developed for years within the context of the research in the separate communities mentioned above, namely *Jason* [13] for programming agents, CArtAgO [14] for programming environments, and *M*OISE [15] for programming organisations.[1] Then, beyond a simple technological integration, we analyse how to integrate the related programming (meta-)models so as to come up with an approach that allows programmers to take advantage of such connections in order to simplify the development of complex systems. The result and main contribution of this paper is the JaCaMo conceptual framework and platform, which provides high-level first-class support for developing agents, environments, and organisations in synergy.

The remainder of the paper is structured as follows. In Section 2, we provide an account of the basic ideas on agent oriented programming and *multi-agent oriented programming*, including a discussion of related work. Then, in Sections 3 and 4, we describe the JaCaMo approach, first presenting the programming (meta-)model on which the framework is based – highlighting how the agent, environment, and organisation dimensions relate to each other – and then describing the platform architecture, along with the support for JaCaMo applications at runtime. To clarify the use of JaCaMo in practice and illustrate some of its features, we present in Section 5 some applications that were developed using our approach. Finally, in Section 6, we provide concluding remarks and some perspectives for future work.

2. Agent oriented and multi-agent programming: the state of the art and challenges

The idea of agent oriented programming was initiated by Shoham [1] as a new programming paradigm combining the use of mentalistic notions (such as belief) for programming (individual) autonomous agents and a societal view of computation. In the 90's, most of the work centred on a few languages that had seen significant theoretical work but had limited use in practice, and mostly centred on developing individual agents whether for a multi-agent system context or not. The work in this area in the 00's changed this picture substantially by producing many different agent programming languages based on varied underlying formalisms and inspired by various other programming paradigms. Furthermore, many such languages were developed into serious programming approaches, with working platforms and development tools. This led to some of these languages having now growing user bases and being used in many AI and multi-agent systems courses. Rather than referring to all those languages individually, we point the interested reader to [2,3,16] where many such languages have been presented in detail or comprehensive references given.

The important contribution of agent oriented programming as a new paradigm was to provide ways to help programmers to develop autonomous systems. For example, agent programming languages typically have high-level programming constructs which *facilitate* (compared to traditional programming languages) the development of systems that are continuously running and reacting to events that characterise changes in the dynamic environments where such autonomous systems usually operate. It is not only the case that agents need to take on new opportunities or revise planned courses of action because of changes in the environment; also agent programming facilitates programming agent behaviour that is not only reactive but also proactive in attempting to achieve long-term goals. The features of agent programming also make it easier, again compared to other paradigms, for programmers to ensure that the agents behave in a way that in the agent literature is referred to as "rational". For example, if a course of action is taken in order to achieve a particular goal (typically explicitly represented in the agent state), if the agent realises that the goal has not been achieved we would not expect the agent not to take further action to achieve that goal on behalf of its human designer unless there is sufficient evidence that the goal can no longer be achieved or is no longer needed.

While agent oriented programming in some two decades made impressive progress in supporting the programming of that kind of autonomous behaviour, it still was not able to achieve all that was expected of it in the context of multi-agent systems. The point is that multi-agent systems are normally used to develop very complex systems, where not only are many autonomous entities present, but also they need to interact in complex ways and need to have social structures and norms to regulate the overall social behaviour that is expected of them and, equally important, a shared environment can be an important and efficient source of coordination means for autonomous agents. Fortunately, while all the technicalities of language constructs that would facilitate the programming of (possibly intelligent) autonomous agents were being dealt with by some researchers in the autonomous agents and multi-agent systems research community, other researchers in that area were focusing precisely on the social/interaction and environment aspects of the development of multi-agent systems. Interestingly, as with agent oriented programming (AOP), the other researchers also coined what could appear to be other (separate) programming paradigms: organisation oriented programming (OOP) [17,18], interaction oriented programming (IOP) [19], and environment oriented programming (EOP) [20].

While they could indeed be independent programming paradigms, it has turned out – as we show in this paper – that the combination of all these dimensions of a multi-agent system into a single programming paradigm with a concrete

[1] Although we recognise that agent *interaction* (e.g., communication protocols) is also an important dimension of first-class abstractions, this is not yet incorporated into our platform, but we discuss how this dimension is currently handled and how we plan to address this in future work at the end of this paper.

working platform has had a major impact on the ability to program complex distributed systems for areas of application in highly dynamic contexts where significant autonomy and coordination is required. Of course these are very early days for a full *multi-agent oriented programming* (MAOP) paradigm, but even the first small demonstrators that we developed with this approach, now confirmed with more ambitious applications also discussed in this paper, showed that this has the potential to change completely the way complex (social) autonomous systems will be developed for the future generations of computational systems as envisaged through many of the current trends in computer science. It is in exploiting in clever designs and implementations – with appropriate abstractions and concrete language constructs – autonomous behaviours, social structure, and social and environment interaction, as well as a normative structure that complex systems can be developed and maintained in, in a much more natural way.

This paper discusses the first successful combination of the AOP, OOP, and EOP in a specific programming platform called JaCaMo. Clearly, there are many open problems in each of these dimensions, but it is important news that each has independently developed sufficiently to allow us to put together the JaCaMo platform, as discussed in this paper. For example, in AOP, modularity [21] and debugging [22] are two examples of aspects that still require significant research to ensure we have the best techniques that we can get. Similarly, in OOP there are many complex theories regarding both agent organisations and normative systems [23] that are discussed in the literature which it has not yet been possible to address in practical platforms. We expect that advances in those separate areas of research will be integrated into JaCaMo as soon as they are made sufficiently practical in computational terms.

Related work

As mentioned previously, many agent programming languages and platforms have been developed in the last two decades. Our work is related in particular to those BDI-based platforms that are explicitly oriented towards practical development and in particular to including explicit support not only for the agent level, but also for the other dimensions—environment and organisation in particular. In fact, to the best of our knowledge, JaCaMo is the first approach aimed at explicitly investigating the integration of all these dimensions from a design and programming point of view: existing approaches consider either only the agent–organisation dimensions or the agent–environment dimensions.

A main related work is that on the "Organization Oriented Programming Language" (2OPL) [24], a rule-based language that allows the programming of multi-agent organisations in terms of norms and which is meant to be exploited in synergy with agent programming languages — 2APL in particular. So far, the work has been given solid theoretical foundations but lacks a clear description of how the approach integrates with the agent level from a practical programming point of view. Besides, a main difference with respect to JaCaMo concerns the role of the environment. 2APL provides an explicit support for agents to interact with computational environments realised as Java objects, getting some benefits from the agent–environment integration. However, in 2OPL the environment is considered as a mere producer of "brute facts" without any significant role in designing/programming the MAS, so no effective integration between the organisation and environment dimensions is considered.

Another important related work is the Golem agent platform [25] which allows the programming of both cognitive agents and computational environments, structured as non-cognitive objects which are organised into "containers". Recent work on that approach presented the initial steps for extending the platform with norms to realise norm-governed multi-agent systems [26]. Other agent programming languages also provide some support for environments and some organisational notions such as roles, but without including a fully fledged organisational model or first-class environment abstractions. We do not aim to cover all of that work in this section.

Then, our work is clearly related to existing work in the AOSE literature that considers the use of organisation and/or environment as dimensions for engineering multi-agent systems. Many contributions have been made in those directions; interested readers can find comprehensive accounts in [27,28,5,12]. These include in particular work that aims at defining a unifying meta-model for developing MAS, integrating concepts belonging to different dimensions, for example the FAML meta-model [29]. Generally speaking, work in the AOSE area focuses on *modelling* aspects, so agent, organisation, and environment concepts are used as concepts to drive the analysis and high-level design of multi-agent systems without, however, exploring their value from a *programming* perspective. So in this work, while recognising the importance of having conceptual frameworks and methodologies that make it possible to exploit agent, organisation and environment concepts in order to model MAS, we argue that such concepts can have a key role also at the programming level, in particular in being then exploited as *first-class abstractions* in agent oriented programming languages and frameworks.

In this way, we aim at contributing to filling an evident gap that exists between the modelling level and the implementation level. Typically, MAS meta-models like FAML or the one described in Sterling and Taveter's book [28] are rather comprehensive, including concepts that are similar or analogous to the ones that will be described in the JaCaMo meta-model. For instance, the concept of *service* that appears in [28] and the concept of *facet* in FAML are strongly related to the notion of *artefact* which is part of the JaCaMo meta-model. The same applies for the notion of *role*. However, in Sterling and Taveter's book, for instance, when agent programming platforms and languages are considered for building concretely a multi-agent system, there is an apparent gap, since the platforms considered (*Jason*, 2APL, GOAL, Jack, JADE) are able to deal only (or almost only) with the agent level, so high-level organisation and environment concepts cannot be directly mapped. Also, typically low-level workarounds are used (such as modelling everything as an agent, even environment or organisational abstractions). So, we argue that the investigation of platforms like JaCaMo – providing an

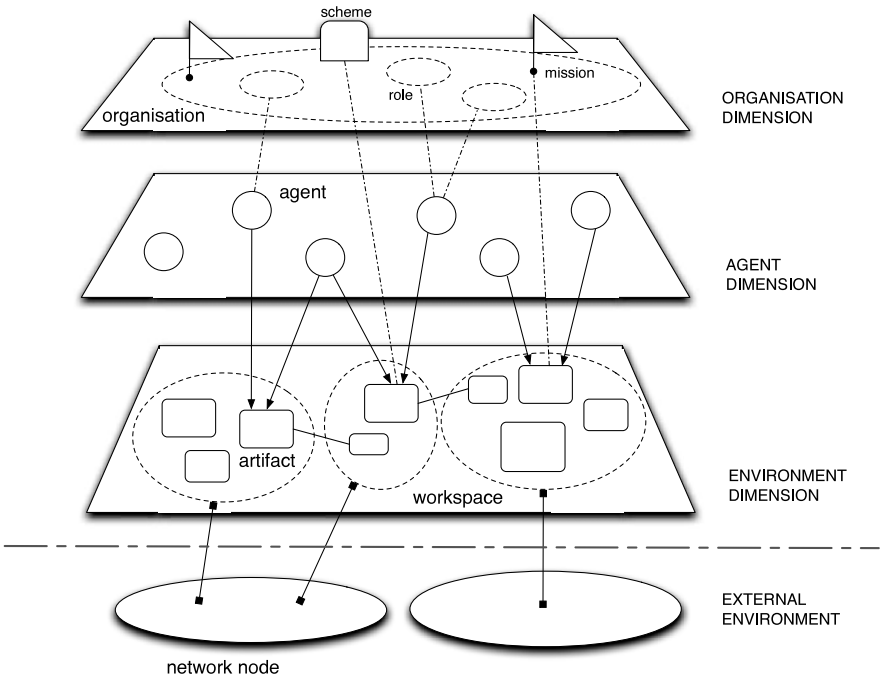


Fig. 1. Overview of a JaCaMo multi-agent system, highlighting its three dimensions.

explicit programming support for the basic, orthogonal dimensions of MAS, as proposed in the literature – is important also from a model-driven engineering perspective, so as to have platform specific models that are rich enough to allow a consistent mapping of high-level concepts defined at a platform independent level.

Finally, this work integrates and extends our previous contributions exploring a unifying perspective in programming agents, organisations, and environments — namely [9], which introduces the idea of designing an organisational management infrastructure in terms of artefact-based environments, and [30], which focuses particularly on the environment–organisation integration introducing the notion of the *embodied organisation*.

3. The JaCaMo approach

A JaCaMo multi-agent system (i.e., a software system programmed in JaCaMo) is given by an agent organisation programmed in $\mathcal{M}$OISE, organising autonomous agents programmed in *Jason*, working in shared distributed artefact-based environments programmed in CArtAgO (see Fig. 1).

Jason is a platform for the development of multi-agent systems that incorporates an agent oriented programming language. The logic-based BDI-inspired language AgentSpeak, initially conceived by Rao [31], was later much extended in a series of publications by Bordini, Hübner, and colleagues, so as to make it suitable as a practical agent programming language. These extensions led to the variant of AgentSpeak that was made available in *Jason* [13].

CArtAgO [14] is a framework and infrastructure for environment programming [20] and execution in multi-agent systems. The underlying idea is that the environment can be used as a *first-class abstraction* for designing MAS, as a computational layer encapsulating functionalities and services that agents can explore at runtime [32]. As they are based on the A&A meta-model [33], in CArtAgO such software environments can be designed and *programmed* as a dynamic set of computational entities called *artefacts*, collected into workspaces, possibly *distributed* among various nodes of a network.

Finally, the $\mathcal{M}$OISE framework [15] implements a programming model for the organisational dimension. This approach includes an organisation modelling language, an organisation management infrastructure [9], and support for organisation-based reasoning mechanisms at the agent level [15].

So JaCaMo integrates these three platforms by defining in particular a semantic link among concepts of the different programming dimensions – agent, environment and organisation – at the meta-model and programming levels, in order to obtain a uniform and consistent programming model aimed at simplifying the combination of those dimensions when programming multi-agent systems.

3.1. An overview of the JaCaMo programming meta-model

Fig. 2 shows the integrated programming meta-model of JaCaMo. The abstractions strictly related to each specific dimension are grouped by dashed lines. Being an integrated *programming* meta-model – that is, a meta-model focused

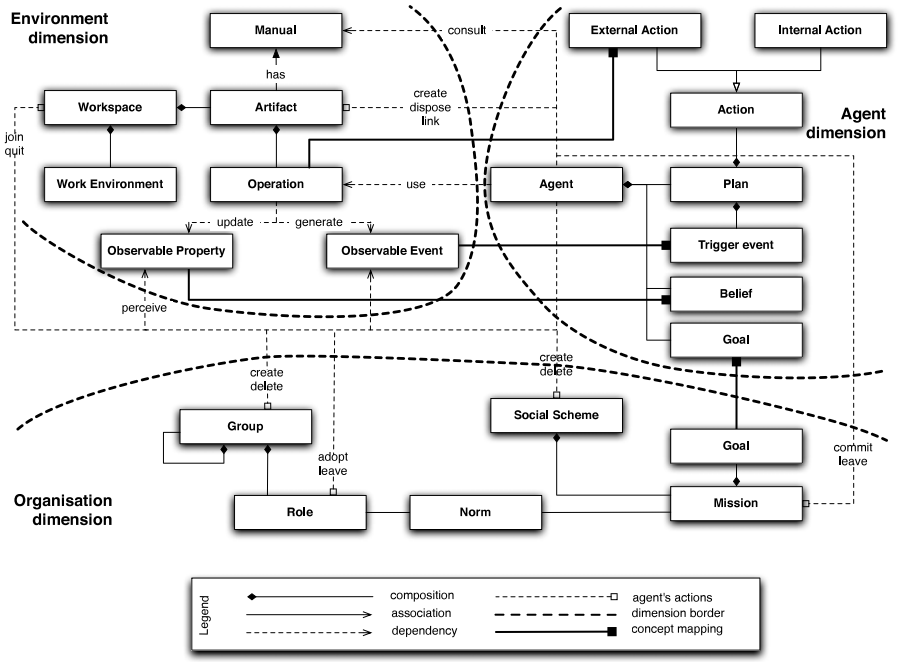


Fig. 2. JaCaMo programming meta-model; for readability, cardinalities are not reported.

on programming abstractions and constructs – it does not include concepts or abstractions which are not part of the programming languages or frameworks – for instance, the concept of *intention* which is part of *Jason* runtime but not of *Jason* programming language is not included. Besides presenting the main concept of JaCaMo programming, the main objective of this meta-model is to show explicitly the dependences, connections and conceptual mappings between the abstractions belonging to the different programming dimensions.

The abstractions belonging to the agent dimension, related to the *Jason* meta-model, are mainly inspired by the BDI architecture upon which *Jason* is rooted. So an agent is an entity composed of a set of *beliefs*, representing the agent's current state and knowledge about the environment in which it is situated, a set of *goals*, which correspond to tasks that the agent has to perform/achieve, and a set of *plans* which are courses of *action*, either internal or external, triggered by *events*, and that agents can dynamically compose, instantiate and execute to achieve goals. Events can be related to changes either to the agent's belief base or to its goals.

On the environment side, on the basis of CArtAgO and the A&A meta-model, each environment instance – the *work environment* entity in the picture – is composed of one or more *workspaces*, used to define the topology of the environment. Each workspace is a logical place containing a dynamic set of *artefacts*, which are the basic computational bricks defining the environment structure and behaviour, representing those resources and tools that agents can create, discover, perceive, and use at runtime. Each artefact provides a set of *operations* and *observable properties* defining an artefact's usage interface, used by agents to observe and operate on artefacts. Operation execution could generate updates to the observable properties and specific *observable events*. The last type of entity at the environment dimension is the *manual*, an entity used for representing the description of the functionalities provided by an artefact.

Finally, on the organisational side, on the basis of $\mathcal{M}$OISE, an organisation is described: (i) from a structural point of view, in terms of *group* and *role* entities; (ii) from a functional point of view, in terms of *social scheme*, *mission*, and *goal* entities — the social scheme decomposes the structure of the organisation's goals into sub-goals, which are grouped into *missions*; and (iii) from a normative point of view, in terms of *norms* which bind roles to missions, constraining an agent's behaviour as regards sets of goals that it will have to achieve when it chooses to enter a group and play a certain role in it.

3.2. Synergy among the JaCaMo programming dimensions

In Fig. 2, the synergies among the three programming dimensions are represented by connections terminating with a square (either filled or not). The connections terminating with a filled square are the most important part of our integrated meta-model as they explicitly represent the synergies and the conceptual mappings that we have identified during the definition of our integrated approach: such connections provide for free (i.e., transparently and without extra programming effort) the integration between the different dimensions, an integration that in other approaches must be programmed by users in an *ad hoc* manner.

The connection link between the agent and environment (A–E) dimensions is given by semantically mapping agents' external actions into artefacts' operations and artefacts' observable properties and events into agents' percepts (leading to beliefs and triggering events). This means that – at runtime – an agent can do an action α if there is (at least) one artefact providing α as an operation – if there is more than one such artefact, the agent may contextualise the action explicitly, specifying the target artefact. On the perception side, the set of observable properties of the artefacts that an agent is observing are directly represented as (dynamic) beliefs in the agent's belief base – so as soon as their values change, new percepts are generated for the agent that are then automatically processed (within the agent reasoning cycle) and the belief base updated. So in programming an agent it is possible to write down plans that directly react to changes in the observable state of an artefact or that are selected on the basis of contextual conditions that include the observable state of possibly multiple artefacts.

This mapping brings significant improvements to the action and percept model provided in general by agent programming languages:

- A *dynamic action repertoire* — The repertoire of an agent's action is dynamic and can be extended/reshaped dynamically by agents themselves by creating/removing artefacts dynamically. This is an improvement with respect to existing agent programming languages, where the set of (external) actions available to an agent is given by the set of actuators that are statically defined for the agent, typically implemented in an *ad hoc* way.
- A *more expressive action model* — By inheriting the semantics of the operation model defined for artefacts, the expressivity of the agent action model is increased in various ways [34]. Typically, actions in agent programming languages are modelled as *atomic events* and this tampers with the possibility of modelling concurrent actions (overlapping in time), which are essential for integrating and exploiting coordination mechanisms and languages — for instance, for modelling Linda suspensive primitive in as an action. By mapping actions into operations, actions inherit a *process-based* semantics [34], which makes it possible then to model long-term actions, overlapping in time, and then to easily design coordinating actions, providing some synchronisation capability.
- A *well-defined success/failure semantics* — In some BDI agent programming languages the burden of understanding whether an action done by an agent succeeded or not rests upon the agent itself (and the agent programmer), through reasoning about the beliefs (percepts). By mapping actions into operations, the action model is extended with an explicit and well-defined notion of success/failure for actions — an action succeeds if the corresponding operation execution on the artefact side completes with success. This in general simplifies agent programming and reduces agent program size, although agents might still need to reason about beliefs to ensure successful action execution in non-deterministic environments.

The connections that terminate with a non-filled square represent a set of predefined actions that agents can perform and which are mapped into operations in a set of predefined artefacts available in each JaCaMo application. These actions refer to the basic functionalities provided by the overall infrastructure, including the environment and organisation dimensions. This makes it possible in particular to avoid the introduction of *ad hoc* specific mechanisms for exploiting infrastructure services concerning organisation and coordination, for instance to adopt a role or to interact with a tuple space. Furthermore, since artefacts can be created and disposed of dynamically, this makes it possible (also for agents) to update and adapt the infrastructure itself at runtime.

This idea was explored in order to effectively connect the organisation and the environment dimensions (O–E), in particular by uniformly designing the organisational infrastructure as part of the (artefact-based) environment in which agents are situated [9]. In such an approach, the different concrete computational entities aimed at managing, outside the agents, the current state of the organisation in terms of groups, social schemes, and normative state are reified in the environment by means of *organisational artefacts*, encapsulating and enacting the organisation behaviour as described by the organisation specifications. From an agent point of view, such organisational artefacts provide those actions that can be used to proactively take part in an organisation (for example, to adopt and leave particular roles, to commit to missions, to signal to the organisation that some social goal has been achieved, etc.), and provide specific observable properties dynamically to make the state of an organisation perceivable along with its evolution. Besides, they provide actions that can be used by *organisational agents* to manage the organisation itself. The specific concrete organisational artefact types introduced in the JaCaMo programming model will be briefly described in the next section.

Overall, the O–E mapping provides some important outcomes which are important from a design and programming perspective:

- *uniformity* — the same action and perception model is used also to enable the interaction between agents and the organisation, without the need for introducing specific *ad hoc* primitives and mechanisms concerning the organisation;
- *distribution* — the organisation management infrastructure is distributed, in terms of collections of (interconnected) artefacts possibly belonging to different workspaces running on distinct network nodes;
- *dynamism* — organisational agents can change dynamically the shape of an organisation by acting on the set of organisational artefacts used by the agents to interact;
- *heterogeneity* — the O–E mapping also opens the way for scenarios in which heterogeneous agents – belonging to different platforms and programmed in different languages – could easily take part in a JaCaMo organisation, as soon as they have been equipped with the capability of working within artefact-based environments;

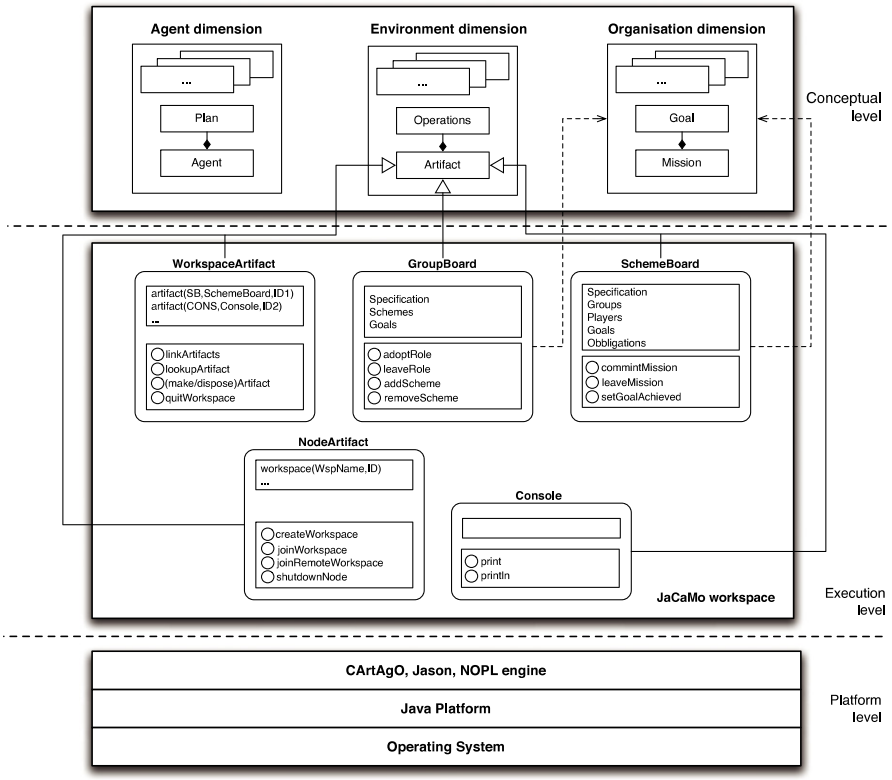


Fig. 3. The JaCaMo platform runtime, with the predefined set of artefacts available in all JaCaMo applications.

- *high-level reorganising capabilities* – the $\mathcal{M}$OISE-based specifications of the organisation themselves are part of the information made observable by organisational artefacts to agents, and this means that there is the potential for agents that understand the $\mathcal{M}$OISE specification formats to reason about the organisations in which they partake and therefore to change them at runtime; this allows for complex on-the-fly restructuring of computational systems to be done at a very high level.

Finally, as part of the agent and organisation (A–O) mapping, goals defined at the organisation dimension (those goals that are to be achieved by the organisation, i.e., the overall system) are mapped into individual agent goals (see Fig. 2), which agents may decide to adopt or not. This delegation of goals from the organisation to the agents is expressed by obligations. The state of goals from the organisation dimension (which agents should fulfil them and when) is maintained by an organisational artefact and is displayed as obligations for the agents (e.g., agent *a* is obliged to fulfil goal *x* in one week). An obligation is fulfilled when the corresponding goal is achieved by the agent before the deadline. At the agent side, when such an obligation is perceived, and the agent chooses to adopt it, a corresponding (individual) agent goal is created. It should be noted that, besides being free to adopt or not the goals from the organisation, the agent is also free to decide which courses of action should be used to achieve each goal, and that in turn might mean adopting further individual goals. As we can see, the mapping of goals prescribed by the organisation into agent individual goals is under the complete control of the agent's decision making process (so as to preserve its autonomy).

4. The impact on multi-agent system programming: the JaCaMo programming model and platform

A main objective of the integration of the dimensions described in Section 3 is to simplify the programming model adopted in the development of complex multi-agent systems. Accordingly, in this section we concretely describe the impact of the dimension integration on programming, by using a toy example called *Building-a-House* – included with the JaCaMo distribution available on the JaCaMo website [35] – which is simple yet effective in showing the integration of the dimensions at the programming level.

Before focusing on the example and on the programming model, first we give a brief overview of the structure of a general JaCaMo application at runtime and of the supporting JaCaMo infrastructure. The overall picture of runtime support provided by JaCaMo is shown in Fig. 3. A JaCaMo (distributed) application runs on top of possibly multiple JaCaMo infrastructure nodes. At the platform (infrastructure) level, each infrastructure node integrates the *Jason*, CArtAgO, and $\mathcal{M}$OISE platforms

as well as the required interfacing technology, running on top of a Java Virtual Machine which itself makes transparent the access to all resources of the operating system.

Then, at the execution level, a JaCaMo application is represented by an organisation composed by one or multiple workspaces, running on JaCaMo nodes. Agents running on a JaCaMo node can join and work concurrently in multiple workspaces, including remote ones (i.e., workspaces not hosted by the same node as they are executing at). Among the artefacts that populate a JaCaMo workspace, a fundamental role is played by those encapsulating infrastructural functionalities related to the agent, environment, and organisation management. They represent a direct reification of the concepts defined in the meta-model (the conceptual level in the figure). These include artefacts used to manage/inspect the structure of the environment itself – WorkspaceArtifact and NodeArtifact, providing functionalities for managing/inspecting respectively the set of artefacts inside a workspace and the set of workspaces inside the environment – as well as *organisation artefacts*, introduced in the previous section.

In particular, the basic organisational artefact types adopted in a JaCaMo application include:

- OrgBoard artefacts, used to keep track of the current state of deployment of the organisational entities overall (one instance for each organisation);
- GroupBoard artefacts, used to manage the life-cycle of specific groups of agents; for instance, if an agent chooses to adopt a role in a particular group, it will perform the adoptRole operation (action) on the GroupBoard artefact representing the group;
- SchemeBoard artefacts, used to support and manage the execution of social schemes; as an example, an agent can commit to a mission or tell the organisation that it has achieved one of the social goals that it was assigned to in a particular mission by executing actions such as commitMissions and setGoalAchieved on the specific SchemeBoard representing the scheme.

Besides the actions, organisational artefacts have specific observable properties for making the dynamic state of an organisation observable. For example, the GroupBoard artefact has an observable property concerning the available roles in a group; therefore agents can find out which are the existing roles and then reason about them so as to autonomously decide whether to adopt a role or not.

From a computational behaviour point of view, organisational artefacts encapsulate and enact the organisational behaviour specified in $\mathcal{M}$OISE. To this end, a specific language called NOPL ("Normative Organisation Programming Language") is adopted as a target language into which $\mathcal{M}$OISE specifications are translated [36]. So organisational artefacts embed a NOPL interpreter to provide the organisational infrastructure needed to manage a $\mathcal{M}$OISE organisation at runtime.

As a final remark, the set of organisational artefacts of the same organisation are connected together by means of the *linkability* features provided by CArtAgO [14], to execute operations among artefacts. This is necessary, "under the hood", to keep consistent the overall state of an organisation infrastructure that is distributed over various separate artefacts. Such distribution of JaCaMo artefacts application over multiple workspaces and network nodes are essential for scaling with organisation and application complexity.

4.1. The Building-a-House example

The example concerns a multi-agent system representing the inter-organisational workflow (IOW) involved in the construction of a house. An agent called Giacomo owns a plot and wants to build a house on it. In order to achieve this overall goal, first Giacomo will have to hire various specialised companies (the *contracting phase*), and then ensure that the contractors coordinate and execute the various tasks required to build the house (the *building phase*). For each company, there is a *company agent* – possibly running on a different network node – participating in the contracting phase and then, possibly, in the building one.

In this simple example, the organisation is composed of a single workspace called `HouseBuildingWsp`. Giacomo is responsible for creating and setting up the workspace, creating also the artefacts that will be used to interact with company agents. It is worth remarking that in larger applications – like the ones described in Section 5 – multiple workspaces possibly distributed on multiple nodes are often used, implementing then a distributed environment and organisation.

4.1.1. The contracting stage: agent–environment programming

In the contracting phase the objective for Giacomo is to hire one company (the one that offers the cheapest service) for each of the several tasks involved in building the house, such as site preparation, laying floors, building walls, building the roof, etc. The same company can be hired for more than one task, if they have more than one speciality and offer the cheapest service in more than one of the required tasks. An auction-based mechanism is used by Giacomo to select the best company from among the available ones for each of the tasks; one auction is run concurrently for each of the tasks. The auction starts with the maximum price that Giacomo can pay for a given task, and companies that can do that kind of task may offer a price lower than the current bid. After a given deadline (unknown to bidders), Giacomo clears the auctions and selects the contractors for building its house on the basis of the lowest bid at the time when the auction closed.

To exemplify the agent–environment programming in JaCaMo, auctions are here realised by means of an artefact providing auction functionalities. There will be one instance of such an artefact (created by Giacomo) for each of the

```
public class AuctionArt extends Artifact {

  void init(String taskDs, int maxValue)  {
    defineObsProperty("task",          taskDs);
    defineObsProperty("maxValue",      maxValue);
    defineObsProperty("currentBid",    maxValue);
   defineObsProperty("currentWinner", "no_winner");
  }

  @OPERATION  void bid(double bidValue) {
    ObsProperty cb = getObsProperty("currentBid");
   ObsProperty cw = getObsProperty("currentWinner");
    if (bidValue < cb.intValue()) {
      cb.updateValue(bidValue);
      cw.updateValue(getOpUserName());
    } else {
      failed("invalid_bid");
    }
 }
}
```

```
!discover_art("auction_for_SitePreparation").

my_price(1500).

i_am_winning(Art)  :-
  .my_name(Me) &
 currentWinner(Me)[artifact_id(Art)].

+currentBid(V)[artifact_id(Art)]  :
 not i_am_winning(Art) &  my_price(P) & P < V
 <- bid(math.max(V-150,P))[artifact_id(Art)].

+!discover_art(ToolName)
 <- joinWorkspace("HouseBuildingWsp");
   lookupArtifact(ToolName,ToolId);
    focus(ToolId).

+!contract("SitePreparation",GroupBoardId)
 <- adoptRole(site_prep_contractor)
     focus(GroupBoardId).

+!site_prepared
  <- ... // actions to prepare the site..
```

Fig. 4. *Left*: source code of the auction artefact. *Right*: source code of a company agent.

house-building tasks, concurrently used by Giacomo and the company agents. The left-hand side of Fig. 4 shows the source code of the auction artefact implemented using the CArtAgO API, and its right-hand side shows an excerpt of a company agent, implemented in *Jason*, that uses an instance of that artefact. In fact, we will not discuss here all the details of the implementation – the interested reader can refer to papers describing in detail the *Jason* and CArtAgO programming models – but just provide what is necessary for understanding the example and the benefits of the dimension integration from a programming point of view.

Auction artefacts have only a `bid` operation – used by company agents to submit a new bid – and there are four observable properties: the task name (`task`), the maximum value that the agent that created the auction is willing to pay for the service (`maxValue`), the current lowest bid (`currentBid`), and the agent that placed that bid (`currentWinner`). The bidding operation – which is seen on the agent side as a `bid` action – simply updates the current bid and winner if a better bid is submitted, or it fails if the bid is higher than the current one.

On the agent side, the company agent has the initial goal `!discover_art("auction_for_SitePreparation")` of discovering the auction for preparing the site[2]. To achieve this goal, the agent has a plan[3] triggered by a "new goal to achieve" event `+!discover_art(ToolName)`. In that plan, the agent first joins the workspace, retrieves the unique identifier of the artefact with the specified name (`lookupArtifact` action), and then uses it to start observing the artefact by executing the `focus` predefined action. By that being done, the artefact's observable properties are automatically mapped into the agent's belief base (as seen in Fig. 2). Changes in the belief base produce events that can be handled by plans. In this specific case, the company agent has a plan for reacting to changes in the `currentBid` (the one with `+currentBid(V)` triggering event) placing a new bid (`bid` action execution in the plan body[4]) if the agent is not the current winner[5] and also the current bid is higher than the minimum price that this agent can offer (stored as the initial belief `price(1500)`[6]).

4.1.2. The building stage: incorporating the organisation dimension

After the companies have been hired, in the building phase the contractors have to execute their own tasks on time and in coordination with each other. Some tasks depend on other tasks being completed first, while some tasks can be done in parallel with some others, as represented by the following workflow (where '`;`' is used for sequence and '`|`' for parallel composition):

```
a ; b ; c ; (d | e | f) ; (g | h | i) ; j
```

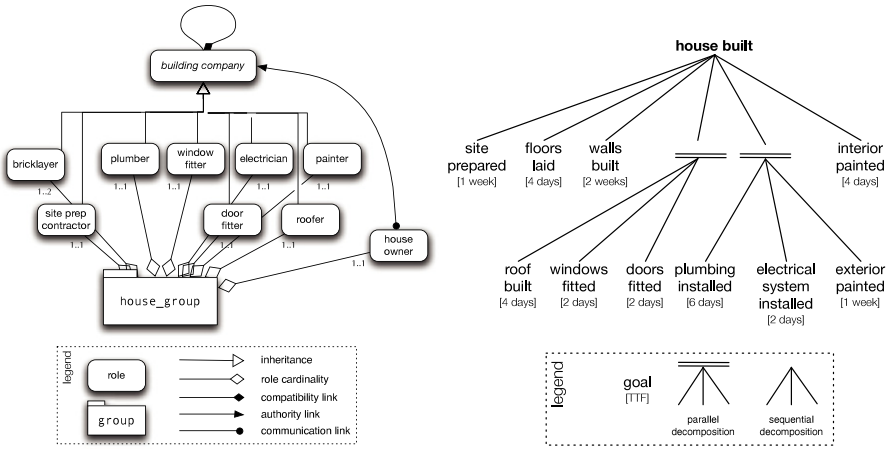
[2] Initial goals in *Jason* are represented by an exclamation mark followed by a first-order logic literal representing the goal to be achieved, specified at the beginning of the source code.

[3] Agent plans in *Jason* are described by rules of the type `event : context <- body`, where `event` represents the events that can trigger the execution of the plan, the plan `context` is a logical expression mainly involving current beliefs, indicating the conditions under which the plan can be selected for execution, and the plan `body` specifies a sequence of actions to perform and sub-goals to achieve when the plan is executed.

[4] The annotation `[artifact_id(Art)]` that appears next to a plan triggering event related to belief updates (e.g., `+currentBid(V)`) and to action execution (e.g., `bid`) is used in general to explicitly retrieve or specify information about the artefact which is the context of the observable property or the operation. This is necessary when multiple artefacts with the same observable properties and operations are used concurrently (as in this case).

[5] `i_am_winning(Art)` is a Prolog-like rule which is specified initially as part of the belief base of the agent. The conclusion of the rule is true if the current value of the `currentWinner` belief (acquired by the agent through an observable property) coincides with the agent's name (retrieved through the *Jason* internal action `.myName`).

[6] Initial beliefs are represented by Prolog-like facts, specified at the beginning of the agent program.



norm	role	mission / goals
n1	house owner	house built
n2	site prep contractor	site prepared
n3	bricklayer	floors laid, walls built
...		

Fig. 5. The structural specification (left), functional specification (right), and normative specification (bottom).

```
!have_a_house.    // initial goal

+!have_a_house    // plan to achieve the have_a_house goal
   <- !contract;
      !execute.

+!contract <- ... // plan to manage the first stage (auction)

+!execute         // create the artifact to manage the group
 <- makeArtifact("hsh_group","GroupBoard", ["src/house-os.xml", \ldots ], GrArtId);
    adoptRole(house_owner)[artifact_id(GrArtId)];
    !contract_winners("hsh_group");
    ...

+!contract_winners(GroupBoard)
 <- for ( currentWinner(Ag)[artifact_id(ArtId)] ) {
      ?task(Task)[artifact_id(ArtId)];
      .send(Ag, achieve, contract(Task,GroupBoard))
   }.
```

Fig. 6. Source code excerpt from the Giacomo agent.

At this stage, the organisation dimension is used to help the coordination and the monitoring of the companies that will build the house. Companies that won the auction will join the organisation playing specific roles and, by doing so, become responsible for some goals in the overall process of building the house. The roles and goals of this organisation are specified in Fig. 5 using the $\mathcal{M}$OISE notation.[7]

The structural specification defines a group (`house group`) where company agents will play the sub-role of the `building company` and the Giacomo agent plays the role of the `house owner`. The functional specification decomposes the organisation goals into sub-goals, defines the sequence in which each will be achieved, and gives a time-to-fulfil (TTF) for each sub-goal. These goals are assigned to roles by norms as defined in the table that appears at the bottom of Fig. 5. For instance, norm `n3` states that whenever an agent adopts the role `bricklayer`, it is obliged to achieve goals `floors laid` and `walls built`. Of course, this obligation is active only when the preceding goals have already been achieved (`site prepared`, in this example).

Once the auctions are finished, Giacomo adopts the role `house owner` in the group and asks the auction winners to adopt the corresponding roles. The code excerpt in Fig. 6 illustrates how these steps are programmed. Giacomo has the initial `!have_a_house` goal and a plan to handle it, by instantiating two sub-goals – `!contract` and `execute`, corresponding to the contracting stage and the building stage – that are achieved sequentially. As soon as the `!contract` goal is achieved, which means that the auctions are concluded, the plan managing the building stage is executed, in which a GroupBoard called the hsh_group artefact is created (specifying some parameters, including the $\mathcal{M}$OISE specification in XML),

[7] $\mathcal{M}$OISE specifications are richer than presented here; they were simplified in this paper to keep the focus on the main aspects.

the house_owner role is adopted by executing an `adoptRole` action and then a sub-goal is instantiated to contract winners. In the plan handling that sub-goal, the agent asks (by means of an `achieve` message sent by the execution of the `.send` communicative action) to all the company agents that won the auctions to sign the contract and finally adopt the role corresponding to the specific task. The required information for all that is made available as observable properties of the auction artefacts.

When the company agents receive the request sent by Giacomo, they adopt the roles by acting on the group artefact. Going back to the source code of the company agent (Fig. 4), this is done by the "`+!contract("SitePreparation", GroupBoardId) <- ...`" plan. The group artefact ensures that the specified organisation constraints are satisfied at all times.

The main purpose of the scheme artefact is to keep track of which goals are ready to be pursued (those whose preceding goals in the functional specification have already been achieved) and create obligations for the agents accordingly. Initially, only the `site prepared` goal can be pursued, and thus only the obligation `obligation(companyB, achieved(s1, site_prepared), "4/1/2011")` is created, where `companyB` is the agent playing the role `site prep contractor`, `s1` is the identification of the scheme instance, and 4/1/2011 is a week after the start of the building work. Such obligations are observed by the agents and corresponding goals are automatically created.[8] In the example, the goal `!site_prepared` is created within the `companyB` agent, which can then react by executing a plan of the form `!+site_prepared <- ...` (see Fig. 4). As soon as other goals become ready to be pursued, new obligations are created and the agents can then work towards the goals at the right moment. In the case of parallel goals, several obligations are created and the agents will work in parallel, as expected from the specification.

A main advantage of this approach is that by simply changing the scheme specification (which can be done by the designer or by the agents themselves) at a very high level, say to change the order or the dependences among goals, we will change the overall behaviour of the agent team without changing a single line of their code. We can see the scheme specification as the program for the social coordination and the scheme artefact as its interpreter. This artefact also manages the state of the obligations, checking, for instance, their fulfilment or violations. This feature is very useful for Giacomo who wants to monitor the execution of the scheme to ensure that the house is built correctly and on time.

4.2. Remarks

From an implementation point of view, in our approach we are able to maintain *separation of concerns* — aspects related to agents, organisations, environments are specified and programmed using specific separate abstractions and corresponding language constructs. This contributes to multi-agent systems development in terms of modularity, reusability, readability, extensibility, and software maintenance.

Furthermore, the semantic mappings facilitate the integrated use of all those concepts, as a uniform and consistent programming model. This also helps avoid common errors in coding that programmers often make when developing agent applications by implementing agents and environment/organisation separately.

Even if it is not possible, for many reasons, to provide an exact quantitative evaluation, we argue that the approach simplifies MAS programming, and makes it possible to have cleaner and typically shorter programs. This is possible because of the reasoning/interpreting/monitoring engines/infrastructures that are available in the three platforms incorporated into JaCaMo and the interfacings among them which also make automatic many things that programmers would normally have to worry about themselves (such as the perception of properties as regards the environment, and the mapping of agent actions to environment operations, to name just a few).

5. Using JaCaMo for real-world applications

In order to assess the applicability, advantages, and limitations of the approach, JaCaMo is being used in various projects that involve the development of real-world agent-based applications. In this section we provide a brief overview of some selected applications. The interested readers can find at the JaCaMo web site [35] an up-to-date list of the applications, each one described in detail. All these projects share some elements of complexity (distribution, openness, dynamism, need of flexibility, autonomy) which make it possible to effectively stress test JaCaMo's different programming dimensions and the usefulness of the integration.

5.1. Engineering smart co-working spaces

The first project concerns room management in smart co-working spaces (e.g., a school, an office); initial results can be found in [37]. The rooms provide facilities and services as typically found in ambient intelligence smart room contexts (see e.g. [38]), relating to light and temperature management, for instance. Besides, people can book and use rooms according

[8] This is done "under the hood" by predefined plans from a library made available with JaCaMo. This library facilitates the programming of (specially norm-abiding) agents and in this application we provided such plans to every agent.

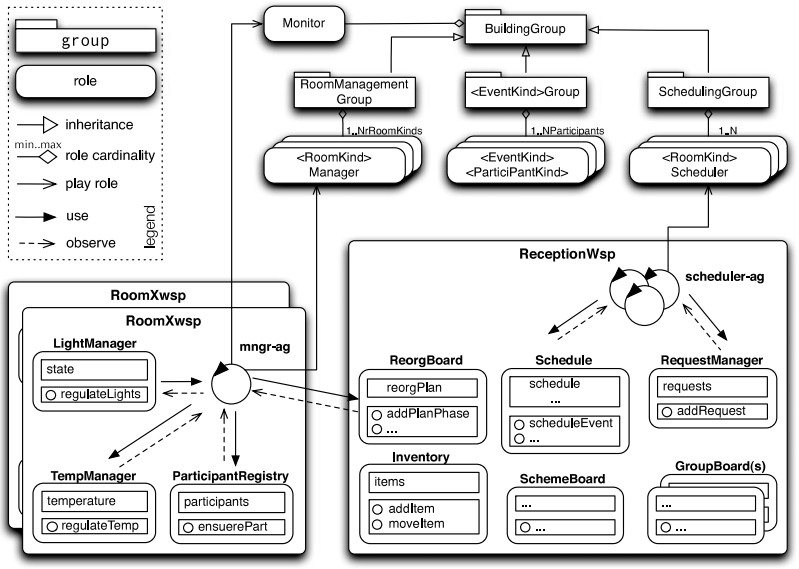


Fig. 7. Architecture of the JaCaMo application for the governance of room allocation in a smart co-working space.

to their needs and to the current and scheduled occupancy. The aim is to develop an autonomous and adaptive room management system that: (i) considers the events that are currently being held (e.g., regulating the room temperature in accordance with the number of participants, automatically turning off the lights for teaching events involving a projector); and also (ii) allows the (re)allocation of the rooms according to the user requests. Such a control software system required for the management of the scenario presented must be distributed and highly adaptable to cope with the inherent distribution of resources and rooms together with the highly dynamic nature of the activities in such spaces. Given the importance of situatedness and autonomy in this application, it must also be connected to the physical world by several sensors and actuators.

Fig. 7 provides an overview of a JaCaMo solution to the problem, focusing on the environment and organisational dimensions. Organisation programming is exploited here to specify and manage the overall management strategy of the system, involving different roles (from individual room managers to schedulers), their coordination, and in particular the handling of reorganisation processes required in such dynamic contexts. The artefact-based distributed environment is exploited to model and interface with the physical devices in the rooms (lights, temperature controllers, etc.), and to model and represent high-level shared data structures with related operations (such as registers keeping track of room participants and schedules), besides typical coordination artefacts (such as blackboards). Agents obviously encapsulate the control and decision making part of the application, in this case related to monitoring and controlling facilities in rooms as well as deciding the strategies to use for dynamic room allocation.

5.2. An agent-based machine-to-machine management infrastructure

This project concerns the realisation of an agile governance application for a machine-to-machine (M2M) management infrastructure. M2M refers to technologies allowing the realisation of automated and advanced services and applications (e.g., smart metering, traffic redirection, and parking management) that make great use of smart devices (sensors and actuators of different kinds, possibly connected through a wireless sensor and actuator network (WSAN)) communicating without human interventions. In this context, the Senscity[9] FUI project proposes an infrastructure for enabling the deployment of city-scale M2M applications that share a common set of devices and network services. Implementing such an infrastructure raises the problem of providing an agile governance with suitable scalability in different dimensions. In fact, it is hardly possible to define all of the requirements of M2M due to its heterogeneity and openness. For this purpose, in collaboration with Orange Labs and based on the communication and IT infrastructure proposed in the Senscity Project, a JaCaMo-based governance application has been developed in order to study how to properly adapt and evolve the Senscity infrastructure without experiencing scalability issues [39]. The governance infrastructure is evaluated using a smart parking management scenario, in which an M2M system monitors the parking occupation in order to reduce traffic and to guide drivers through the streets.

9 http://www.senscity-grenoble.com/.

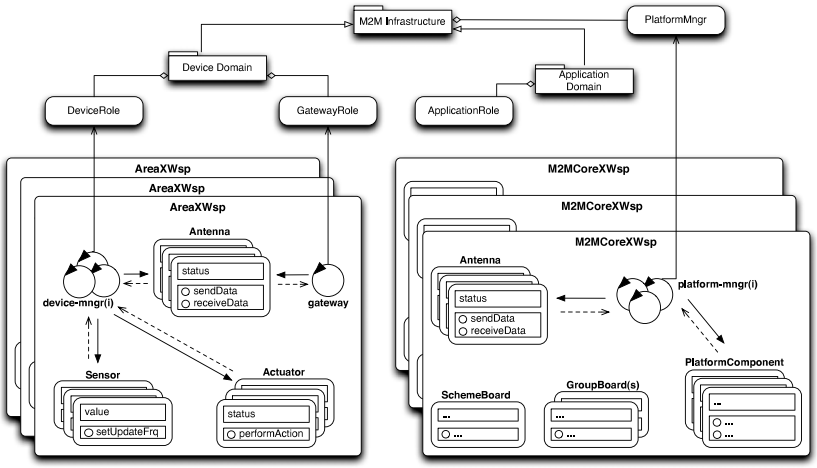


Fig. 8. Abstract architecture of the JaCaMo-based application for the Senscity M2M infrastructure.

The governance application developed in JaCaMo (see Fig. 8) is designed in terms of an organisation that provides to the agents a global strategy for managing and coordinating the functioning of the whole system, distributed among several workspaces. A workspace is used for each WSAN area group, in which an agent playing a *gateway* role collects data sent from all the agents playing a *device management* role, which manage and control M2M devices (sensors, actuators) through artefacts. A dynamic pool of agents playing a *platform manager* role regulate the functioning of each M2M infrastructural node in accordance with the current workloads experienced by the M2M infrastructure. This is achieved by cleverly using a set of dedicated artefacts that provide features such as transaction management for both sensor readings and complex actuator commands, and the possibility of enabling/disabling the interfacing of applications to the managed WSAN.

In the context of the smart parking scenario considered, the governance application allows the adaptation of the M2M infrastructure in several situations: (i) steep increase/decrease in requests coming from the different applications, (ii) changes in the topology of WSAN covered by applications (e.g. moving the area to monitor for the urban planning service from district scale to city scale), and (iii) integration of new client applications (e.g. smart parking integrated with a multi-modal transportation system). This is done by dynamically adapting the role cardinalities in the organisation specification, thus changing the size of the pools of agents playing roles in the organisation. Changes in the physical structure of the WASN managed by the M2M infrastructure are handled by dynamically deploying (destroying) a workspace for each WASN introduced (removed) and then registering adequately the new information in the governance infrastructure.

There is also an application in the area of ontology-based knowledge management for knowledge-intensive workflows in business processes. An existing application in that area which involves natural language processing and various AI techniques has been revamped using JaCaMo, leading to a clearer software architecture and increased potential for extending the system towards greater autonomy and adaptability. For the sake of space, we do not discuss this application in detail here; it has been described in [40].

6. Concluding remarks: current limitations and future work

The agent-based development of complex distributed systems requires design and programming tools that provide first-class abstractions along the main levels or dimensions that characterise multi-agent systems. To this end, in this paper we described a comprehensive approach for *multi-agent oriented programming* that integrates the agent, environment, and organisation levels, providing a programming model that aims at integrating in an effective and synergistic way related abstraction dimensions. To investigate the idea in practice, we introduced the JaCaMo platform, which integrates and extends existing approaches and accompanying technologies, namely *Jason* (agent dimension), CArtAgO (environment dimension) and $\mathcal{M}$OISE (organisation dimension). The whole approach was illustrated by a case study and some applications reported in this paper.

Future work falls into three main directions. First, we aim to investigate also *interaction* as a main dimension to be integrated synergistically with other ones. Currently, that dimension is handled through *(i)* the ad hoc direct communication support based on speech acts provided by *Jason*, and *(ii)* the mediated communication support based on communication and coordination artefacts [41] provided by CArtAgO. Besides this view, it is interesting to explore the use of environment abstractions also, to make direct communication support more flexible. For instance, by introducing appropriate personal communication artefacts, we can provide the agent with functionalities for communicating using different agent communication languages and managing complex conversations and ontologies. Second, it is possible to further explore the O–E (organisation–environment) connection in order to implement more advanced institution mechanisms such as the *count-as* relation [42], thus making it possible to provide, generally speaking, a direct semantic link

between the execution of actions (operations) on environment artefacts and their meaning and effect at the organisational level. Initial investigations in that direction can be found in [30]. Last, we aim at providing an integrated development environment that would facilitate the process of design, development, and execution of JaCaMo applications, potentially reusing and integrating existing *Jason*, CArtAgO, and *M*OISE tools and technologies.

Acknowledgements

Part of this work was funded by the CMIRA Programme of the Rhone-Alpes Region. Support was also given by CNPq (grants 307924/2009-2, 307350/2009-6, 478780/2009-5).

References

[1] Y. Shoham, Agent-oriented programming, Artif. Intell. 60 (1) (1993) 51–92.
[2] R.H. Bordini, M. Dastani, J. Dix, A.E. Fallah-Seghrouchni (Eds.), Multi-Agent Programming: Languages, Platforms and Applications Vol. I, Springer, 2005.
[3] R.H. Bordini, M. Dastani, J. Dix, A.E. Fallah-Seghrouchni (Eds.), Multi-Agent Programming: Languages, Tools and Applications Vol. II, Springer, 2009.
[4] M.P. Singh, Agent communication languages: rethinking the principles, Computer 31 (12) (1998) 40–47.
[5] D. Weyns, H.V.D. Parunak (Eds.), Environments for Multi-Agent Systems, in: Autonomous Agents and Multi-Agent Systems, vol. 14 (1), Springer, Netherlands, 2007 (special issue).
[6] O. Gutknecht, J. Ferber, The madkit agent platform architecture, in: Revised Papers from the Int. Workshop on Infrastructure for Multi-Agent Systems: Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems, Springer-Verlag, London, UK, 2001, pp. 48–55.
[7] D.V. Pynadath, M. Tambe, An automated teamwork infrastructure for heterogeneous software agents and humans, Autonomous Agents and Multi-Agent Systems 7 (1–2) (2003) 71–100.
[8] M. Esteva, J.A. Rodríguez-Aguilar, B. Rosell, J.L. Arcos, AMELI: an agent-based middleware for electronic institutions, in: Prog. of the 3rd Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems, AAMAS'2004, ACM, New York, 2004, pp. 236–243.
[9] J.F. Hübner, O. Boissier, R. Kitio, A. Ricci, Instrumenting multi-agent organisations with organisational artifacts and agents, Autonomous Agents and Multi-Agent Systems 20 (2010) 369–400.
[10] J. Ferber, O. Gutknecht, A meta-model for the analysis and design of organizations in multi-agents systems, in: Proc. of the 3rd Int. Conf. on Multi-Agent Systems, ICMAS'98, IEEE Press, 1998, pp. 128–135.
[11] Y. Demazeau, From interactions to collective behaviour in agent-based systems, in: Proc. of the 1st European Conf. on Cognitive Science, Saint-Malo, 1995, pp. 117–132.
[12] T. Stratulat, J. Ferber, J. Tranier, MASQ: towards an integral approach to interaction, in: AAMAS, 2009, pp. 813–820.
[13] R.H. Bordini, J.F. Hübner, M. Wooldridge, Programming Multi-Agent Systems in AgentSpeak Using Jason, in: Wiley Series in Agent Technology, John Wiley & Sons, 2007.
[14] A. Ricci, M. Piunti, M. Viroli, A. Omicini, Environment programming in CArtAgO, in: Bordini et al. [3].
[15] J.F. Hübner, J.S. Sichman, O. Boissier, Developing organised multi-agent systems using the MOISE+ model: programming issues at the system and agent levels, Agent-Oriented Software Engineering 1 (3–4) (2007) 370–395.
[16] M. Fisher, R.H. Bordini, B. Hirsch, P. Torroni, Computational logics and agents: a road map of current technologies and future trends, Computational Intelligence 23 (1) (2007) 61–91.
[17] O. Boissier, J.F. Hübner, J.S. Sichman, Organization oriented programming: from closed to open organizations, in: G. O'Hare, O. Dikenelli, A. Ricci (Eds.), Engineering Societies in the Agents World VII, ESAW 06, in: LNCS, vol. 4457, Springer, Berlin, Heidelberg, 2007, pp. 86–105.
[18] D.V. Pynadath, M. Tambe, N. Chauvat, L. Cavedon, Toward team-oriented programming, in: N.R. Jennings, Y. Lespérance (Eds.), ATAL, in: LNCS, vol. 1757, Springer, 1999, pp. 233–247.
[19] M.N. Huhns, Interaction-oriented programming, in: First International Workshop, AOSE 2000 on Agent-Oriented Software Engineering, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001, pp. 29–44.
[20] A. Ricci, M. Piunti, M. Viroli, Environment programming in multi-agent systems: an artifact-based perspective, Autonomous Agents and Multi-Agent Systems 23 (2011) 158–192.
[21] M.B. van Riemsdijk, M. Dastani, J.-J.C. Meyer, F.S. de Boer, Goal-oriented modularity in agent programming, in: Proc. of the 5th Int. Joint Conf. on Autonomous Agents and Multiagent systems, AAMAS'06, ACM, New York, USA, 2006, pp. 1271–1278.
[22] D. Poutakidis, L. Padgham, M. Winikoff, An exploration of bugs and debugging in multi-agent systems, in: N. Zhong, Z.W. Ras, S. Tsumoto Suzuki (Eds.), ISMIS, in: Lecture Notes in Computer Science, vol. 2871, Springer, 2003, pp. 628–632.
[23] G. Boella, L. Torre, H. Verhagen, Introduction to the special issue on normative multiagent systems, Autonomous Agents and Multi-Agent Systems 17 (1) (2008) 1–10.
[24] M. Dastani, D. Grossi, J.-J. Meyer, N. Tinnemeier, Normative multi-agent programs and their logics, in: KRAMAS-08, Proc., 2008.
[25] S. Bromuri, K. Stathis, in: D. Weyns, S. Brueckner, Y. Demazeau (Eds.), Engineering Environment-Mediated Multi-Agent Systems, in: LNCS, vol. 5049, Springer, Berlin, Heidelberg, 2008, pp. 115–134.
[26] V. Urovi, S. Bromuri, K. Stathis, A. Artikis, Initial steps towards run-time support for norm-governed systems, in: Coordination, Organizations, Institutions, and Norms in Agent Systems VI, in: Lecture Notes in Computer Science, vol. 6541, Springer, 2010, pp. 268–284.
[27] V. Sichman, in: J. Dignum (Ed.), Agent Organizations; Models, Architectures and Applications, Springer, 2010.
[28] L. Sterling, K. Taveter, The Art of Agent-Oriented Modeling, The MIT Press, 2009.
[29] G. Beydoun, G. Low, B. Henderson-Sellers, H. Mouratidis, J.J. Gomez-Sanz, J. Pavon, C. Gonzalez-Perez, Faml: a generic metamodel for mas development, IEEE Transactions on Software Engineering 35 (2009) 841–863.
[30] M. Piunti, A. Ricci, O. Boissier, J. Hübner, Embodying organisations in multi-agent work environments, in: IEEE/WIC/ACM Int. Conf. on Web Intelligence and Intelligent Agent Technology, WI-IAT 2009, Milan, Italy, 2009.
[31] A.S. Rao, AgentSpeak(L): BDI agents speak out in a logical computable language, in: W.V. de Velde, J.W. Perram (Eds.), MAAMAW, in: LNCS, vol. 1038, Springer, 1996, pp. 42–55.
[32] D. Weyns, A. Omicini, J.J. Odell, Environment as a first-class abstraction in multi-agent systems, Autonomous Agents and Multi-Agent Systems 14 (1) (2007) 5–30.
[33] A. Omicini, A. Ricci, M. Viroli, Artifacts in the A&A meta-model for multi-agent systems, Autonomous Agents and Multi-Agent Systems 17 (3) (2008) 432–456.
[34] A. Ricci, A. Santi, M. Piunti, Action and perception in multi-agent programming languages: from exogenous to endogenous environments, in: Proc. of Programming Multi-Agent Systems, ProMAS-'10, 2010.
[35] JaCaMo project web site – http://jacamo.sourceforge.net/, last retrieved: August 15th 2011.
[36] J.F. Hübner, O. Boissier, R.H. Bordini, From organisation specification to normative programming in multi-agent organisations, in: CLIMA XI, 2010, pp. 117–134.
[37] A. Sorici, Agile governance in an AmI environment, Master's Thesis, University "Politehnica" of Bucharest (September 2011).

[38] A. Kameas, Towards the next generation of ambient intelligent environments, in: Enabling Technologies: Infrastructures for Collaborative Enterprises, WETICE, 2010 19th IEEE Int. Workshop on, 2010, pp. 1–6.
[39] C. Persson, G. Picard, F. Ramparany, A multi-agent organization for the governance of machine-to-machine systems, in: IEEE/WIC/ACM Int. Conf. on Intelligent Agent Technology, IAT'11, IEEE Computer Society, 2011.
[40] C.M. Toledo, R.H. Bordini, O. Chiotti, M.R. Galli, Developing a knowledge management multi-agent system using the jacamo platform, in: Proc. of ProMAS 2011, held with AAMAS-2011, 2011.
[41] A. Omicini, A. Ricci, M. Viroli, C. Castelfranchi, L. Tummolini, Coordination artifacts: Environment-based coordination for intelligent agents, in: Proc. of the 3rd Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems, AAMAS'04, Vol. 1, ACM, New York, USA, 2004, pp. 286–293.
[42] J.R. Searle, The Construction of Social Reality, Free Press, 1997.