A Multi-agent Solution for the Deployment of Distributed Applications in Ambient Systems

Ferdinand Piette[1,2(✉)], Costin Caval[1], Cédric Dinont[2], Amal El Fallah Seghrouchni[1], and Patrick Tailliert[1]

[1] Sorbonne Universités, UPMC Univ Paris 06, LIP6, Paris, France
[2] Institut Supérieur de l'Électronique et du Numérique, Lille, France
ferdinand.piette@yncrea.fr

Abstract. Ambient Intelligence (AmI) and Internet of Things (IoT) are promising fields for the application of Multi-Agent Systems (MAS). A specific MAS application, described through a video doorkeeper scenario in this paper, is the deployment and the configuration of distributed applications on a hardware infrastructure in ambient systems. It requires the modelling of the available infrastructure and of the deployable applications, respecting a domain ontology, which can then be used by reasoning tools to find the hardware entities that can support the running of the application in the existing infrastructure. It also requires a distributed architecture that allows this solution to be scalable and to provide mechanisms to enhance privacy. In this paper, we discuss this last point. We describe the use of goal-driven agents and show how the MAS architecture and organisation allow for the privacy of the infrastructure resources to be enhanced.

Keywords: Applicative paper · Multi-agent system · Ambient Intelligence · Goal-driven agents · Agent design · Privacy management · Deployment

1 Deployment of Smart Applications

AmI research focuses on the improvement of human interactions with smart applications [13]. These improvements are made possible by the proposal of frameworks and platforms that facilitate the development of context-aware and dynamic applications. These platforms offer mechanisms to build such applications by handling data and events [16,18] or by wrapping hardware and software capabilities into agents [9,14]. However, it is often assumed that an underlying interoperable hardware and energy infrastructure already exists [22]. Meanwhile, the Internet of Things (IoT) aims to provide a global infrastructure for the information society, enabling advanced services by interconnecting physical and virtual "things" based on existing and evolving interoperable information and communication technologies [17]. The main challenge of the IoT is to achieve full interoperability of interconnected devices while guaranteeing the trust, privacy

M. Baldoni et al. (Eds.): EMAS 2016, LNAI 10093, pp. 156–175, 2016.
DOI: 10.1007/978-3-319-50983-9_9

and security of communications [4]. However, a gap exists between AmI and the IoT. Indeed, because of the heterogeneity of such systems, it is difficult to have horizontal communication between connected devices. Present applications use devices that are vertically connected, from the device to an external server that collects and processes the data. The available commercial products are usually not directly interoperable. Moreover, this approach raises privacy questions: the user does not own his data any more, so privacy cannot be guaranteed. Hence, to fill this gap between IoT and AmI applications, adequate deployment mechanisms are required. We addressed the deployment problem in [23] by proposing to model the available hardware infrastructure and the needs of the applications using graphs that describe the various entities, their relations and properties. For deploying an application on the infrastructure, we proposed an extended graph matching algorithm for finding the hardware entities of the infrastructure that fulfil the requirements of the distributed application. However, this solution was centralised, which makes it unsuitable for real systems that need to take into consideration, among others, privacy and scalability. To address these issues, we propose a multi-agent-based distributed deployment software. Through its modularity, the multi-agent paradigm facilitates the local processing of data and guarantees the autonomy of the different parts of the hardware infrastructure, thus enhancing the privacy and robustness of the software.

This paper is organised as follows. Section 2 shows similar works that use agents for the deployment of applications and privacy management. Section 3 presents a scenario that illustrates the deployment of applications and introduces the different key aspects of our solution. The next sections show that multi-agent systems are a well-adapted paradigm to handle the distributed aspect and ensure resource privacy. We detail this multi-agent architecture of our solution (Sect. 4) and the behaviour of each kind of agent using a goal-directed approach (Sect. 5). At last, Sect. 6 explains some implementation specificities and presents the first results. We conclude by presenting the next steps of this work.

2 Related Work

Several works address the deployment problem. Braubach et al. [6] propose a deployment reference model based on a MAS architecture (e.g. agent services) for deploying MAS applications. As an agent is a software entity, the deployment of agents does not have to deal with the high heterogeneity of hardware entities. Some other works in the service-oriented architectures (SOA) community [3] reason on deployment patterns, that specify the structure and constraints of composite solutions on the infrastructure, in order to compose services. The cited paper refers not to the localisation of resources and installation of software, but rather to the binding of existing resources in order to provide the desired composition of services. This is realised using a centralised pattern-matching algorithm that takes into account the various requirements for the given service. Flissi et al. [15] propose a meta-model for abstracting the concepts of the deployment of software over a grid. All these works have shortcomings when considering

their use for deploying AmI applications on the IoT infrastructure. Some do not take into consideration the heterogeneity of the hardware and software, as well as the interaction between the two layers (i.e. software and hardware). Others do not tackle the privacy problem. And some propose centralised solutions that are not scalable for real life AmI applications. Our MAS approach takes these problems into consideration: scalability is handled thanks to the agent structure; the autonomy of agents, organisation and privacy policies provide resource privacy; and heterogeneity is supported by the description of the system and the reasoning mechanisms that find projections of applications on the infrastructure.

Privacy in multi-agent systems has already been well explored. Such et al. [27] categorise research on data privacy on different levels: collection, disclosure, processing and dissemination. Multi-agent system specificities have been used to propose different manners of handling the data privacy. Some works focus on norms [5,20] and privacy policies [12,28,29], checked by agent brokers to control the disclosure of the data. Other works [24,26] use social relationships like trust, intimacy or reputation to select the agents with which data can be shared. Trusted third parties are already used in [1,11,21] in order to anonymise the data or the metadata (e.g. IP address, receiver or sender identity), and also to check disclosure authorisations. At last, some works [2] focus on integrating secure communication in the agent platforms by using well known encryption protocols. All these works use MAS in order to provide data privacy. In our work, as explained in Sect. 4, we take advantage of MAS properties to handle the privacy of the hardware resources and of the structure of the system. The data privacy of the deployed applications is left to the developer who can use one of the cited methods.

3 Scenario

The scenario we use in this paper highlights the dynamic deployment of distributed applications. Mr Snow uses a *video doorkeeper* for dependant persons (e.g. visually impaired) application in his home. When someone rings at the door, the image of the entrance camera is displayed on a screen near Mr Snow, making sure he can properly see the person. He can then discuss with the person and decide whether or not to remotely open the door.

It is Saturday morning and Mr Snow is waiting for a parcel that will be delivered to his home at any time. While he is grooming himself in the bathroom, his neighbour, Mr Den, rings the door. The smart house, aware that Mr Snow is in his bathroom, selects the connected mirror of the bathroom, instead of any of the other display screens of the house, as a support to display the image stream of the entrance camera. Mr Snow, not being able to receive his guest, informs him, thanks to the microphone in the mirror, that he will meet him in an hour. After getting ready, Mr Snow goes to his neighbour. In the middle of their conversation, he is notified on his smartphone that an unkonwn man rings at his door again. He decides to display the image of this man on Mr Den's television to ask him if he recognises the guest since his smartphone screen is

to small. By default, Mr Snow does not have the right to use any devices that he does not own, but Mr Den has authorised him to access the television when he is at home. The doorkeeper application is redeployed dynamically to use the requested hardware entities. Neither Mr Snow nor his neighbour know the visitor. Mr Snow decides to activate the microphone of the camera which allows him to learn that the unknown person is the expected transporter, who he can now go and see in person.

The important point in this scenario is not the video doorkeeper application, but the way it is deployed dynamically in the environment, considering the user's context. The scenario shows two deployment situations: (1) the application was deployed for use in the user's own home infrastructure, but in a less usual place: the bathroom; (2) the application was deployed on the infrastructure of another user, as the necessary access rights had been granted. We can isolate five main needs of the system:

1. The system has to find which hardware entities to use in order to launch the desired application. These entities have to respect hardware requirements and contextual constraints such as the user location.
2. Once the hardware entities have been chosen, the system has to deploy the application or some part of it on the infrastructure. In the scenario, the application can be divided into two parts. The first one monitors the door bell of the house and is automatically deployed when a user chooses to launch the video doorkeeper application. When someone rings the door, the second part of the application, the one which will display the image stream of the camera on a screen near the user, is deployed. It is the deployment of the second part that interests us.
3. The system has to undeploy (part of) the applications. In the scenario, when the user ends the communication with the guest, the second part of the video doorkeeper application should be undeployed and the corresponding resources released.
4. The system has to monitor the environment: get contextual information about the user location or the current amount of bandwidth of a communication channel for instance. If an inconsistency between the hardware infrastructure properties and the requirements of an application is detected, another deployment of the application (or some parts of it) should be planned.
5. At last, a user has to manage the hardware entities he owns. He can also use hardware entities of others if he has the required permissions (as described in Sect. 4.2). However, he does not have access to information on the structure of the infrastructure he does not own.

The first need is already discussed in our previous work [23] and later improved in a distributed version of the algorithm that finds the hardware entities of the infrastructure that can support the deployment of an application, based on their descriptions. The second and third needs involve interactions with the real environment in order to configure the hardware entities. The fourth need also involves interactions with the environment to sense its properties and state. At last, the fifth need raises the question of the privacy of hardware resources:

how can the system guarantee that users cannot have information about the hardware entities of others?

The multi-agent system we describe in this paper is used to deploy applications on an available hardware infrastructure, to monitor the system and to maintain its consistency. As we describe below, the agents are in charge of the high level reasoning, while artifacts correspond to tools for interacting with the environment. The decentralisation of MAS is an asset to enhance resource privacy. Indeed, the description of the hardware infrastructure can be split into agents so that no global knowledge exists. Then the agents can be organised to apply sharing policies and guide the deployment of the applications. At last, cooperation between agents allows to find solutions of the deployment even if no local solution exists.

With these observations, we can establish the different roles of the system. These roles will be associated with the different entities of the MAS.

1. Interact with a user
2. Maintain the consistency of an application
3. Find a projection of the requirements of an application on the infrastructure graph
4. Interact with the environment to configure hardware entities and deploy applications
5. Sense the environment properties
6. Update the description of the environment
7. Manage sharing policies and resource privacy

4 Multi-agent Architecture

Our scenario highlights several necessary specificities of the deployment software. This software has to dynamically deploy and undeploy distributed AmI applications in an environment that is also dynamic: when a visitor rings the doorbell, the deployment of the video doorkeeper should start, considering the available hardware entities and the location of the user, in order to choose the most relevant screen for displaying the image of the camera. Given its the distribution and openness that characterize the AmI domain, privacy is a very important characteristic of the deployment software. Privacy is defined by Alan Westin [30] as the claim of individuals, groups or institutions to determine for themselves when, how and to what extent information about them is communicated. In this scenario, we focus on *resource privacy* since the data manipulated by the deployment solution concerns hardware resources. Mr Snow is the owner of the hardware entities in his house and he does not want unauthorised persons to use or even know of the existence of these resources. At last, autonomy and robustness of the system are also very important specificities: if my neighbour's system failed, mine should continue to work normally and should not be impacted.

As the required software demands distribution, privacy, context management, autonomy and robustness, we identified MAS as a suitable solution. Through its

modularity, this paradigm facilitates a local processing of the data and guarantees the autonomy of the different parts of the hardware infrastructure, thus handling aspects of privacy and robustness. To solve the dynamic deployment problem, we use the graph representation for the hardware infrastructure from our previous work [23]. Nodes represent hardware entities or relations between these entities and properties can be attached to each node. The requirements of the deployable applications are also described using such graphs. A graph matching algorithm can then be used on the available infrastructure graph to find the entities that can support the running of the application.

In the next sub-sections, we present the modelling of agents and the agent organisation for our deployment solution, while focusing on the encapsulation of resource privacy.

4.1 Agents and Artifacts

The deployment software involves the user deploying applications on an infrastructure. Three types of agent were therefore defined to represent and clearly separate each of the parties in handling the deployment: *User Agent*, *Application Agent* and *Infrastructure Agent*. A fourth type of agent was introduced for providing organisation capabilities and enhancing resource privacy: the *Infrastructure Super Agent*. For each type of agent we identified the main goals, that will be described in Sect. 5:

An ***Infrastructure Agent*** deals with a part of the global hardware infrastructure. It uses the graph representation of this available infrastructure [23] (hardware entities, relations and properties). This graph representation is never shared with other agents. To deploy an application, an *Infrastructure Agent* has to find a projection (possibly partial) of the hardware requirements of the application on the infrastructure (role 3, as identified above). The agent may need to cooperate with other agents to complete the projection if no local solution can be found. This infrastructure description is updated (role 6) when the agent receives information about the current state and properties of the real infrastructure. At last, as the infrastructure description is not shared with the other agents, the *Infrastructure Agents* have to manage the authorization levels and the sharing policy (role 7).

An ***Application Agent*** manages an entire application during its runtime (role 2). It has a graph-based description of the application that expresses its hardware requirements and the way the hardware entities should be used (configuration, software deployment, ...). If the hardware requirements of parts of the deployed application are no longer respected during its runtime, then the agent will plan another deployment of these parts by interacting with *Infrastructure Agents*. An example of such graph is represented in Fig. 1: the upper part represents the functionalities of the application and the bottom part shows their hardware requirements.

The ***User Agent*** is attached to a user and saves his preferences and his context. It is the interface between the user and the other agents (role 1). Each user is represented in the MAS by his own agent. This one handles the user's

requests for the deployment or undeployment of applications and creates the associated *Application Agents*.

The *Infrastructure Agents* can be grouped to form sub multi-agent systems. These groups are represented by an ***Infrastructure Super Agent***. From an outside point of view, a *Infrastructure Super Agent* is seen as a regular *Infrastructure Agent*. It acts as a proxy between the agents inside and outside of the group. It is then easier to abstract groups of agents and make then invisible from the outside. It results on a multi-scale organisation that helps to enhance resource privacy by hiding information about the structure of it sub-organisations.

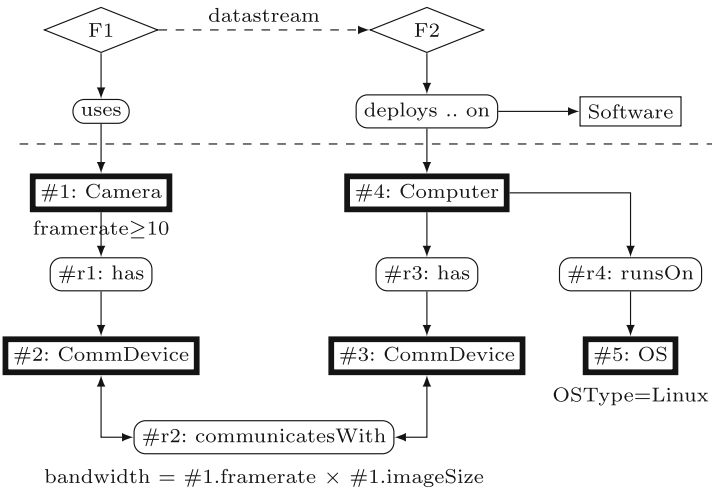

Fig. 1. Example of a basic application graph

In addition to these four classes of agent, we also propose two classes of artifact which are resources and tools that can be instantiated and/or used by agents in order to interact with the environment [25]:

Deployment artifacts [15] can be used by the *Infrastructure Agents* in order to effectively deploy/undeploy some parts of an application, or configure hardware entities so that they can be used by the application (role 4).

Monitoring artifacts provide useful contextual information to the MAS (role 5) such as the location of a user or the current available bandwidth of a communication channel. This information helps the agents keep their application or infrastructure descriptions up to date.

4.2 Sharing Policies

To improve privacy by controlling the use of resources, we also propose sharing policies. *User Agents* can be authorised, by the owner of some hardware infrastructure, to use some parts of its infrastructure, and cooperate with the associated *Infrastructure Agents* or *Super Agents*, to deploy applications. If a

User Agent is not authorised by the *Infrastructure (Super) Agent*, it cannot use the hardware resources proposed by this agent. These authorization levels are defined by the owner of each *Infrastructure (Super) Agent*. The *Administrator level* is defined to identify these owners. With this level, a user can manage the other authorization levels, configure or create sub-organisations of *Infrastructure Agents* (by implicitly instantiating *Infrastructure Super Agents*) and has access to every hardware entity managed by the agent. When an administrator creates a sub-organisation, he is automatically the administrator of the new *Infrastructure Super Agent*.

4.3 Deployment

The deployment is started by a user, through his *User Agent*. The latter then creates an *Application Agent* that will handle the deployment and the monitoring of the application. This agent chooses among the authorized *Infrastructure (Super) Agents* the ones it will ask for the deployment of parts of the application. The concerned *Infrastructure Agents* try to find the hardware entities that will support the deployment of the application. If a partial projection is found, the agent will cooperate with other authorized agents in order to complete the projection. Once such projection is found, the concerned agents effectively deploy the application through the deployment artifacts. The monitoring artifacts provide the *Infrastructure Agents* with information about the environment. When some properties change, these agents notify all the *Application Agents* that have inconsistent applications. These ones can decide to plan another deployment of some parts of the application.

Figures 2 and 3 show the interaction between an *Application Agent* and the *Infrastructure Agents* in order to get a projection of the application and to effectively deploy this application. As stated before, the current *Infrastructure Agent* ("Infra Agent Actor" in the figures) may need to request other agents ("Infra Agent Delegate") to handle a part of the deployment.

4.4 Scenario Illustration

Figure 4 shows the agent structure of the doorkeeper scenario. The description of the infrastructure is split into three *Infrastructure Agents*. The first one manages the hardware entities located in the living room of Mr Snow, like the television set. The second one manages the entities of the bathroom like the connected mirror. These two agents are grouped behind an *Infrastructure Super Agent* representing the house of Mr Snow. And the last one manages the house of the neighbour. Similarly, the *Infrastructure Agent* managing the house of Mr Snow's neighbour can be a super agent, regrouping several *Infrastructure Agents* (or other sub-super agents) to manage more finely the house. The advantage of such organisation is that it is easy to abstract groups of agents and make them invisible from the outside, resulting in a multi-scale organisation that helps improve privacy. Indeed, Mr Snow knows about his own *Infrastructure Agents* (bathroom and living room), but he does not have to know anything about the

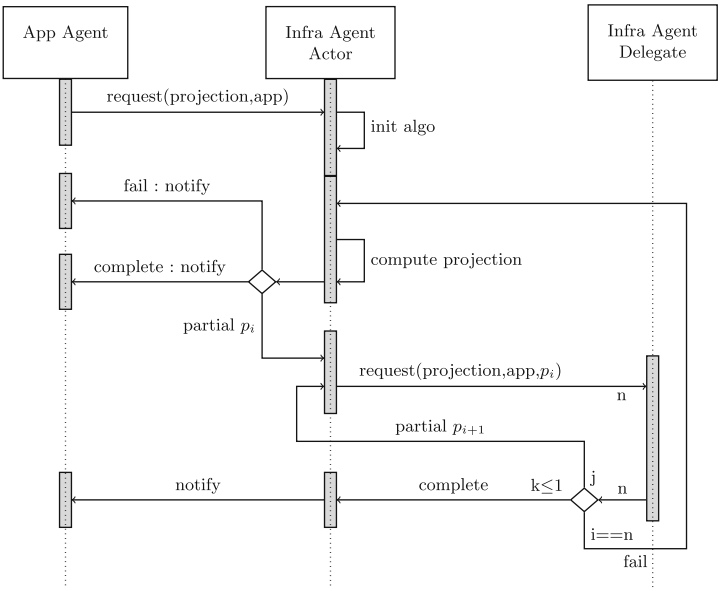

Fig. 2. Infrastructure Agent: get a projection of a part of the application

details of Mr Den's infrastructure organisation. If he wants to interact with his neighbour's house, he has to interact with Mr Den's *Infrastructure Super Agent* – provided that the right access rights were granted, as described below –, without knowing the real number of agents managing Mr Den's house, and reciprocally.

We also find two *Application Agents*. The first one manages the video doorkeeper application; when a visitor rings the doorbell, this *Application Agent* triggers the deployment of the video interaction functionality. The second one manages the application which provides the location of the Mr Snow inside his own house to his own *Infrastructure Agents*. The contextual location information is useful for deploying other applications. Indeed, the display screen of the video doorkeeper application has to be chosen near the user. Then, we have two *User Agents*. The first one is the interface between the deployment software and Mr Snow, and the second one is owned by Mr Snow's neighbour. At last, we have a certain number of deployment artifacts that can configure the display screens, the cameras, or deploy software on devices (TV box, connected mirror etc.).

In this scenario, three authorisation levels are defined: the administrator level, the regular user level and the guest level. With the regular user level, the agent has access to the resources of the *Infrastructure (Super) Agent* but it cannot reconfigure authorisation levels or agent organisation. With the guest level, the agent has a restricted access to the resources. Only the resources considered as non critical by an administrator are allowed to be shared. These authorisation levels are not limited to three and can be modified by the administrator of the *Super Agent*. In the video doorkeeper scenario, Mr Snow's *User Agent* is a Regular user for his home *Infrastructure Super Agent*, but it is just a Guest

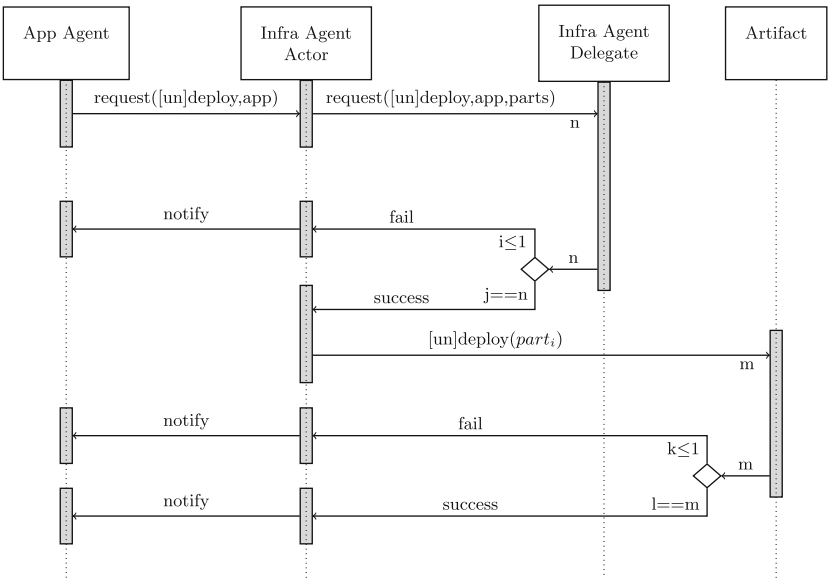

Fig. 3. Infrastructure Agent: deploy a part of an application

to his neighbour's home *Infrastructure Super Agent*. As such, it has only access to the television of Mr Snow's neighbour. This ensures the privacy of the other resources of Mr Den. The *Application Agents* have the same authorisation level as the *User Agent* that creates them. They can interact with the authorised *Infrastructure Agents* in order to effectively deploy their application.

4.5 Summary

In this section, we defined the multi-agent system architecture. *User*, *Application* and *Infrastructure Agents* were defined in order to provide a clear separation. *Infrastructure Super Agents* were introduced to allow the creation of hierarchical organisations of *Infrastructure Agents*. Sharing policies were defined to control the use of the hardware resources of the infrastructure. The authorizations based on these sharing policies define the organisation and the possible acquaintances for the agents of the MAS.

The agent decomposition encapsulates a part of the privacy mechanism. Indeed, the graph representation of the available hardware infrastructure managed by an *Infrastructure Agent* is only known by this agent and is never shared with others. Moreover, the architecture used helps keep a clear separation between the applicative part, managed by the *Application Agents*, and the hardware part, monitored by the *Infrastructure Agents*. As agents only have a local view of the system, the privacy is enhanced. Privacy policies can allow or prevent the sharing of resources to *User Agents*. This results in privacy by design.

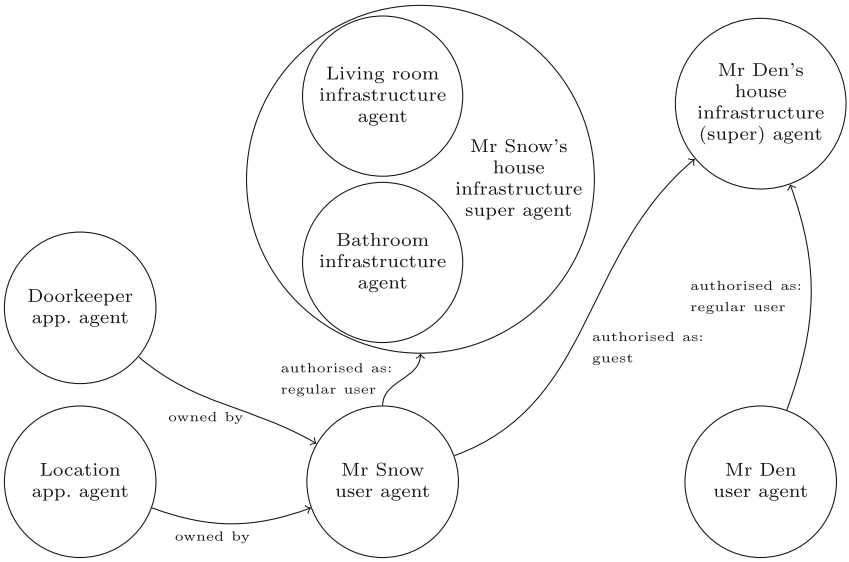

Fig. 4. Agent organisation

5 Agents' Behaviour

The four types of agent presented in the previous section were designed using a goal-based model due to its benefits to the autonomy and robustness of the application [10]. Goals are specified by describing their associated plans: higher level *goal plans* describing relationships between goals and lower level *action plans* for concrete actions. This goal-based representation is based on the Goal-Plan Separation (GPS) approach [8], where each agent has a main goal plan (i.e. plan without any actions, so only decisions, perceptions and goal adoptions) that describes the top level behaviour, which can be pursued using other goal plans or directly action plans (i.e. plan without any goal adoptions). This approach helps handle agent complexity through a multi-level description, from top level abstract behaviours with goals to concrete action plans. Using goal-plans also has the advantage of specifying the relationships between goals in a plan format.

Plans are represented using a flowchart notation we adapted for modelling goal-driven agents (Fig. 5). The notation contains the main elements that allow for the behaviours of agents to be defined. Event perceptions (wait), decision nodes and iterators (ForEach) can be used in any type of plan. Action nodes are specific to action plans and goal adoptions (parallel or synchronous) are specific to goal plan. Parallel executions are launched when adopting goals. For this application, we considered a simple goal model (similar to a *perform* goal [7]) where a goal is successful ("S") when the plan executing for it ends with "End ok". This allowed us to keep a simple goal life-cycle appropriate for using in our application, while still benefiting from the features of the goal-based design.

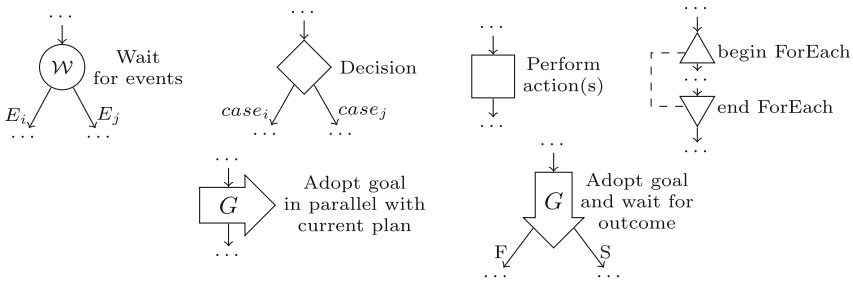

Fig. 5. Flowchart nodes for efficiently describing the plans of goal-driven agents

We continue by describing in detail the agents of the system. Since the *Infrastructure Super Agent* is only a proxy between the agents of the group it represents and the other agents outside this group, its implementation is not detailed here. In what follows, P_{Xi-j} are the plans for a goal G_{Xi}.

5.1 User Agent

The *User Agent* acts as an interface between the user and the deployment MAS. The main goal plan of the *User Agent* (Fig. 6) waits for user input and, depending on the received request, adopts the necessary goal, corresponding to the agent functions identified in Sect. 4.1. The goal plan of G_{U1} (Fig. 7) creates an *Application Agent*, wait for a confirmation and adopt a goal that monitor the *Application Agent*.

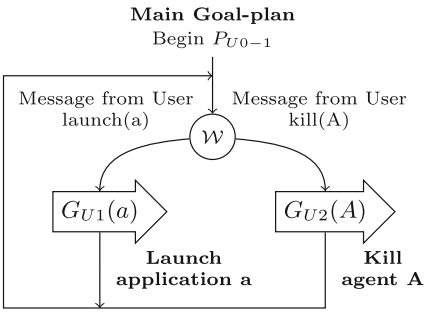

Fig. 6. *User Agent*: main goal plan

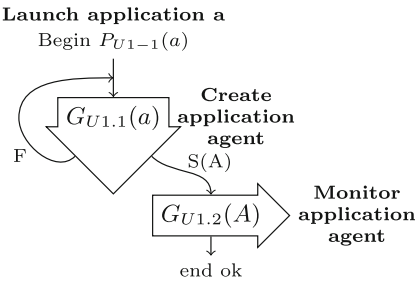

Fig. 7. *User Agent*: goal plan for G_{U1}

5.2 Application Agent

The *Application Agent* is created by a *User Agent*. It tries to deploy a precise application by cooperating with one or more known *Infrastructure (Super) Agents*, from which it does not need to have any infrastructure details.

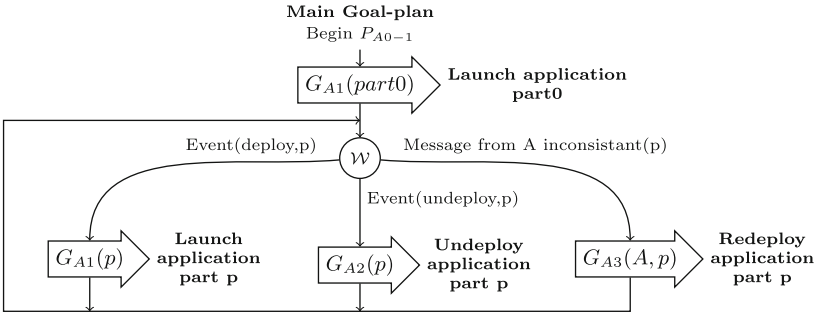

Fig. 8. *Application Agent*: main goal plan

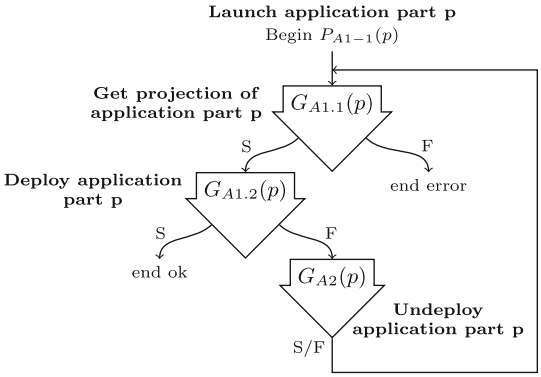

Fig. 9. *Application Agent*: goal plan for G_{A1}: "Launch application part"

Upon its creation, an *Application Agent* execute its main goal plan (Fig. 8). The goal G_{A1} that deploy an initial functionality (Fig. 9) is adopted and the agent waits for internal events for new deployments or undeployments.

The deployment is done in two steps: first the agent obtains a deployment solution from *Infrastructure Agents* via $G_{A1.1}$ and then it requests the deployment according to this solution through $G_{A1.2}$. The *Application Agent* sends a list of the requirements described in the application graph to the *Infrastructure Agent* and the solution it receives contains the list of requirements that could be fulfilled. Note that the reply does not contain any actual infrastructure details, which is important for the privacy of the infrastructure. It can be seen (Fig. 10) that the agent may need to call multiple *Infrastructure Agents* in order to obtain a complete deployment solution. Indeed an *Infrastructure Agent* tries to find in its own infrastructure the hardware entities that match the requirements of the application. However, if these requirements only partially match, the *Infrastructure Agent* will return a partial solution to the *Application Agent*. In this case, the latter will call another *Infrastructure Agent* that will continue to match the requirements of the application. Once a solution has been found, the

Application Agent interacts again with the concerned *Infrastructure Agents* to effectively deploy the functionalities of the application: plan $P_{A1.2}$ simply sends messages and waits for a confirmation (Fig. 11). The plan for the undeployment of a part of the application (G_{A2}) is similar to the plan that deploy a part of the application ($G_{A1.2}$). The plan for the redeployment of a part of the application (G_{A3}) only undeploy this part first and deploy it again. After a functionality was deployed, the agent monitors it through G_{A0} and wait for a message from the *Infrastructure Agents* that tells that a part of the application is inconsistency (e.g. changing infrastructure availability, changing user location).

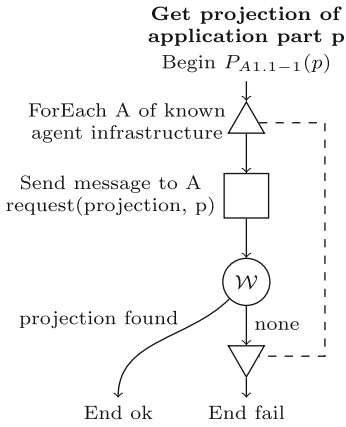

Fig. 10. *Application Agent*: plan for $G_{A1.1}$: "get projection of application part"

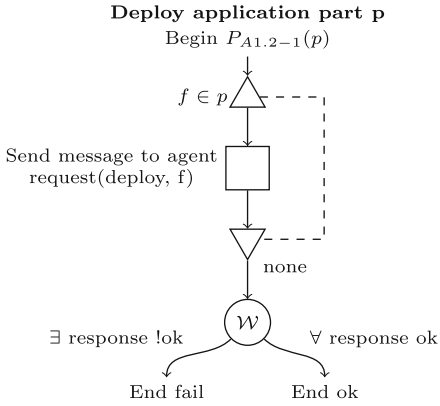

Fig. 11. *Application Agent*: plan for $G_{A1.2}$: "deploy application part"

Note here that the *Application Agents* only handle the application deployment. The application itself is in charge of its own actions, data and privacy.

5.3 Infrastructure Agent

An *Infrastructure Agent* receives requests from *Application Agents* that it tries to satisfy (Fig. 12). Only requests originating from known *User Agents* are treated, in other words only applications from agents that were granted one of the levels of authorisation are accepted.

When it receives a request for a deployment solution, the *Infrastructure Agent* uses the graph matching algorithm to determine if it can fulfil the requirements of the request (Fig. 13) using the devices it manages. The algorithm takes into consideration the levels of authorisation of the involved *User Agents*. If it cannot produce a complete solution, the *Infrastructure Agent* requests the help of other agents in its group, but without informing the *Application Agents*. In this way, the components of the infrastructure remain private. If a complete solution is eventually produced and the *Infrastructure Agent* is given the order to deploy

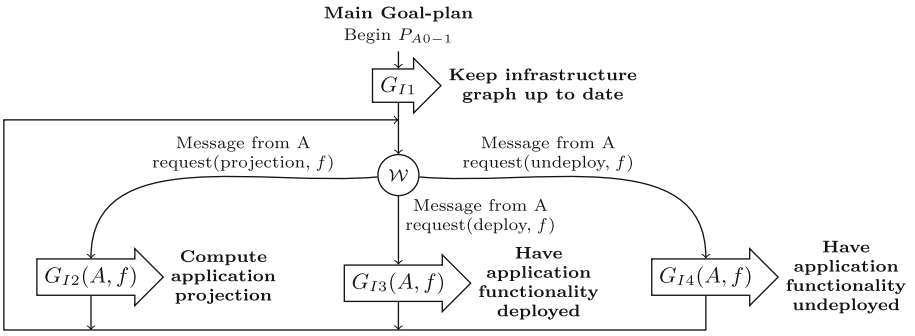

Fig. 12. *Infrastructure Agent*: main goal plan

the application, it will dispatch the deployment tasks to its own deployment artifacts as well as to any other *Infrastructure Agents* that were included in the final solution. In case any of these requests fails (e.g. an artifact malfunctions), the whole application is undeployed and the *Application Agent* is informed, which will cause it to restart the deployment procedure.

In parallel with the request handling, the agent also adopts G_{I1} which listens for agent and artifact information in order to manage the graph the devices corresponding to the *Infrastructure Agent*. In case of an inconsistency (e.g. Mr Snow leaves Mr Den's home, so any display he used there are no longer relevant for the application), the agent informs the *Application Agents* that it will need to redeploy the concerned parts of their applications.

6 Implementation and Experimentations

A demonstration model of the deployment software has been developed in an apartment replica attached to our laboratory. This home replica implements various scenarios applied to home care for dependent persons, including the presented scenario. These scenarios are using commercial connected devices tweaked to be horizontally connected, thanks to the deployment software.

Our goal is to run the MAS on different devices like smartphones or embedded systems with few resources. Most of existing regular MAS platforms like Jade for instance are memory-consuming and Java-oriented platforms [19]. They are not suitable for our purpose. That is why we designed our own MAS platform in JavaScript. Indeed, web technologies are fully interoperable and the agents can easily be run on devices like smartphones or the Raspberry Pi. Visualisation and interfaces are also JavaScript web applications. The agents embed a monitoring and debugging web server that proposes interfaces for interacting with it. The effective deployment is handled by deployment artifacts. The demonstration model handles *ssh* and *puppet* artifacts in order to deploy and run software on UNIX systems (computers, micro-computers, Unix-based devices etc.). We also implemented a specific deployment artifact that configures the frame rate of IP

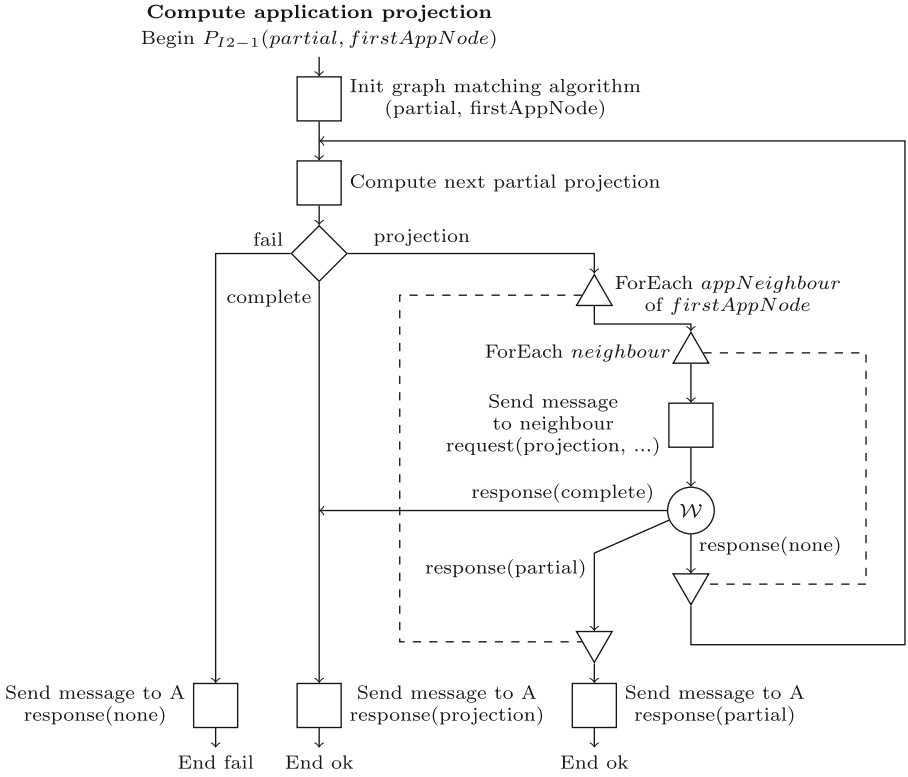

Fig. 13. *Infrastructure Agent*: plan for G_{I2}: "compute application projection"

cameras. In this implementation we mostly used IP devices. We also integrated EnOcean devices. These devices, however, are handled by a hard-coded gateway that extends the IP network to EnOcean devices. Next stage will be to handle multiple means of communication by automatically deploying gateways or proxies between the devices when needed. At last, the agent implementation was, in first place, not obvious. The multi-level GPS approach made it intuitive to develop.

For these experiments, we generated random infrastructure and application graphs for which we varied the number of nodes, the average number of edges and the average number of properties each node has. We only considered the infrastructure-application pairs for which at least one complete projection solution exists. In the graphs depicted below, each point is the median execution time obtained by running the algorithm on 100 randomly-generated infrastructure-application pairs. A random graph generation was introduced in order to evaluate the MAS performances independently of the application domain. Then, the properties of the application and infrastructure graph in the context of smart-homes were extracted and correlated with the general results.

Figure 14 shows the variation of the computation times with the sizes of the application and infrastructure graphs for only one infrastructure agent. The execution time shows a moderate increase with respect to the infrastructure size variation. However, it grows fast with the application graph size.

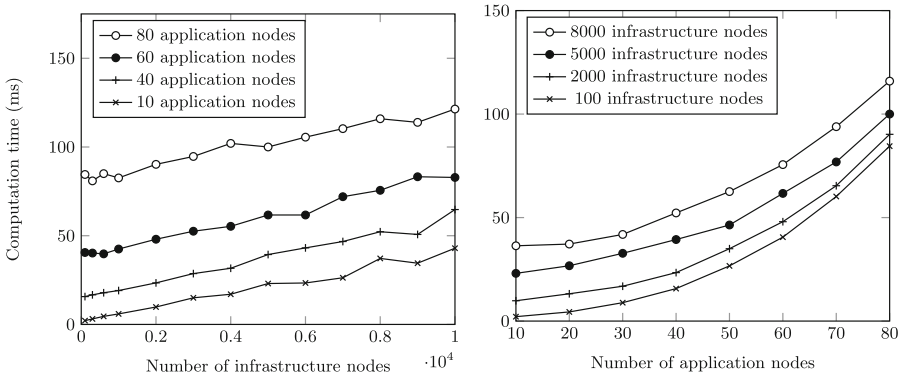

Fig. 14. The execution times of the projection algorithm executed for various infrastructure and application sizes.

Figure 15 represents the variation of the execution times with size of the infrastructure and application graph shared amoung different number of agents. We can note that it is not efficient to let an agent manage only few infrastructure nodes. In that case, a lot of time is lost in the cooperation process.

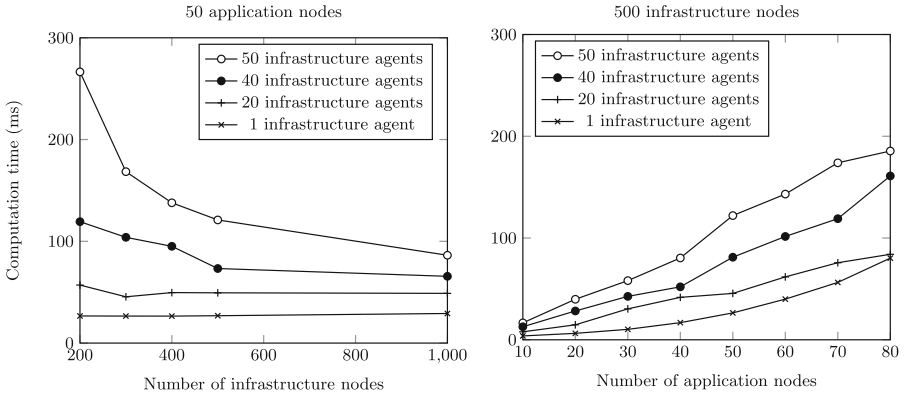

Fig. 15. The execution times of the graph-matching algorithm for various size of infrastructure, application and number of agents.

These experiments show that the algorithm can be used with real applications and smart environments. Indeed, the computation time grows fast with the size of the application graph, however, applications do not have an important number of nodes. Contrariwise, the global infrastructure graph can grow

rapidly, but we have seen that the computation time evolves reasonably. For example, the application from the scenario in Sect. 3 contains 23 nodes, while the infrastructure is made of more than 50 nodes shared amoung 3 agents. The average number of edges between the nodes is between 2 and 3.

This realisation helps us to figure out the difficulties of handling the heterogeneity of hardware entities. We are now able to handle applications through an AppStore for Smart Homes. These applications can be automatically deployed in a real environment, using the available hardware devices, and including mechanisms to ensure privacy management of the resources. This provides a concrete base for the implementation of a complete middleware for the deployment of distributed applications in a smart environment.

7 Conclusion and Future Work

In this paper, we presented a multi-agent solution for reasoning on the dynamic deployment of distributed applications in ambient systems. We described the modelling of the system and presented the specifications of the goal-based agents. We illustrated the MAS using a context-aware video doorkeeper scenario. In this scenario, a doorkeeper application is dynamically deployed in order to route the video stream of the entrance hall camera to a relevant screen, near the user, thanks to contextual information about his location. Even if smartphones are nowadays the favorite interface of the users, we are convainced that multimodal interactions have to be proposed. The devices and the interfaces has to be selected considering the context. Other scenarios to help people with reduce mobility has been inplemented using the apartment replica we used.

The MAS proposed in this paper contains four classes of goal-directed agent to handle a clear separation between the hardware and software layers and to ensure resource privacy in ambient systems. In order to preserve the privacy of the resources, the graph models of the infrastructure are handled locally by the concerned agents. The use of MAS made it possible to introduce privacy measures at architecture and organisation level, on top of which we were able to add a user-defined privacy policy mechanism. This was an important criterion for the choice of the agent paradigm since in the domain of Ambient Intelligence there are often different infrastructure owners that need to ensure the privacy of their resources. The separation between the applicative and the infrastructure layers, together with the decentralised approach also enhance the robustness of the solution. The clearly delimited entities, with either virtual (the applications) or physical (users, infrastructure elements) correspondents, guided the agentification. The use of a goal-based representation for agents together with the Goal-Plan Separation approach facilitated the modelling task. The specific plan notation was efficient in describing the agent plans both during design and for presentation purposes.

In terms of future work, for the deployment software, data privacy in the deployed applications should also be taken into consideration in addition to the resource privacy discussed here. We would like to facilitate the local processing and storage of the data by defining data privacy policies which should be

facilitated by the modularity of the MAS. This would impact the reasoning on the deployment: the hardware entities would have to be filtered with respect to this new data privacy policy. In the interest of the engineering of multi-agent systems, we are studying the goal-based modelling approach with GPS agents and the plan notation for the extension towards a development methodology for robust software.

References

1. Aïmeur, E., Brassard, G., Fernandez, J.M., Onana, F.S.M.: Privacy-preserving demographic filtering. In: Proceedings of the 2006 ACM Symposium on Applied Computing, SAC 2006, pp. 872–878. ACM, New York (2006)
2. Alberola, J., Such, J., Garcia-Fornes, A., Espinosa, A., Botti, V.: A performance evaluation of three multiagent platforms. Artif. Intell. Rev. **34**(2), 145–176 (2010)
3. Arnold, W., Eilam, T., Kalantar, M., Konstantinou, A.V., Totok, A.A.: Automatic realization of SOA deployment patterns in distributed environments. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 162–179. Springer, Heidelberg (2008). doi:10.1007/978-3-540-89652-4_15
4. Atzori, L., Iera, A., Morabito, G.: The internet of things: a survey. Comput. Netw. **54**(15), 2787–2805 (2010)
5. Barth, A., Datta, A., Mitchell, J., Nissenbaum, H.: Privacy and contextual integrity: framework and applications. In: 2006 IEEE Symposium on Security and Privacy, pp. 15–198, May 2006
6. Braubach, L., Pokahr, A., Bade, D., Krempels, K.-H., Lamersdorf, W.: Deployment of distributed multi-agent systems. In: Gleizes, M.-P., Omicini, A., Zambonelli, F. (eds.) ESAW 2004. LNCS (LNAI), vol. 3451, pp. 261–276. Springer, Heidelberg (2005). doi:10.1007/11423355_19
7. Braubach, L., Pokahr, A., Moldt, D., Lamersdorf, W.: Goal representation for BDI agent systems. In: Bordini, R.H., Dastani, M., Dix, J., Fallah Seghrouchni, A. (eds.) ProMAS 2004. LNCS (LNAI), vol. 3346, pp. 44–65. Springer, Heidelberg (2005). doi:10.1007/978-3-540-32260-3_3
8. Caval, C., El Fallah Seghrouchni, A., Taillibert, P.: Keeping a clear separation between goals and plans. In: Dalpiaz, F., Dix, J., Riemsdijk, M.B. (eds.) EMAS 2014. LNCS (LNAI), vol. 8758, pp. 15–39. Springer, Heidelberg (2014). doi:10.1007/978-3-319-14484-9_2
9. Chen, H., Finin, T.W., Joshi, A., Kagal, L., Perich, F.: Intelligent agents meet the semantic web in smart spaces. IEEE Internet Comput. **8**(6), 69–79 (2004)
10. Cheong, C., Winikoff, M.: Hermes: designing goal-oriented agent interactions. In: Müller, J.P., Zambonelli, F. (eds.) AOSE 2005. LNCS, vol. 3950, pp. 16–27. Springer, Heidelberg (2006). doi:10.1007/11752660_2
11. Cissée, R., Albayrak, S.: An agent-based approach for privacy-preserving recommender systems. In: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2007, pp. 182:1–182:8. ACM, New York (2007)
12. Crépin, L., Demazeau, Y., Boissier, O., Jacquenet, F.: Sensitive data transaction in hippocratic multi-agent systems. In: Artikis, A., Picard, G., Vercouter, L. (eds.) ESAW 2008. LNCS (LNAI), vol. 5485, pp. 85–101. Springer, Heidelberg (2009). doi:10.1007/978-3-642-02562-4_5

13. Ducatel, K., Bogdanowicz, M., Scapolo, F., Leijten, J., Burgelman, J.: Scenarios for ambient intelligence in 2010 (2001)
14. Fallah Seghrouchni, A., Olaru, A., Nguyen, N.T.T., Salomone, D.: Ao dai: agent oriented design for ambient intelligence. In: Desai, N., Liu, A., Winikoff, M. (eds.) PRIMA 2010. LNCS (LNAI), vol. 7057, pp. 259–269. Springer, Heidelberg (2012). doi:10.1007/978-3-642-25920-3_18
15. Flissi, A., Dubus, J., Dolet, N., Merle, P.: Deploying on the grid with deployware. In: 8th IEEE International Symposium on Cluster Computing and the Grid, CCGRID 2008, pp. 177–184, May 2008
16. Hellenschmidt, M., Kirste, T.: A generic topology for ambient intelligence. In: Markopoulos, P., Eggen, B., Aarts, E., Crowley, J.L. (eds.) EUSAI 2004. LNCS, vol. 3295, pp. 112–123. Springer, Heidelberg (2004). doi:10.1007/978-3-540-30473-9_12
17. ITU-T: Overview of the internet of things, recommendations (2012)
18. Johanson, B., Fox, A., Winograd, T.: The interactive workspaces project: experiences with ubiquitous computing rooms. IEEE Pervasive Comput. **1**(2), 67–74 (2002)
19. Kravari, K., Bassiliades, N.: A survey of agent platforms. J. Artif. Soc. Soc. Simul. **18**(1), 11 (2015)
20. Krupa, Y., Vercouter, L.: Contextual integrity and privacy enforcing norms for virtual communities. In: Boissier, O., El Fallah Seghrouchni, A., Hassas, S., Maudet, N. (eds.) MALLOW. CEUR Workshop Proceedings, vol. 627 (2010). CEUR-WS.org
21. Menczer, F., Street, W., Vishwakarma, N., Monge, A., Jakobsson, M.: IntelliShopper: a proactive, personal, private shopping assistant. In: Proceeding 1st ACM International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS) (2002)
22. O'Hare, G.M.P., Collier, R., Dragone, M., O'Grady, M.J., Muldoon, C., Montoya, J.A.: Embedding agents within ambient intelligent applications. In: Bosse, T. (ed.) Agents and Ambient Intelligence, Ambient Intelligence and Smart Environments, vol. 12, pp. 119–133. IOS Press (2012)
23. Piette, F., Dinont, C., El Fallah Seghrouchni, A., Taillibert, P.: Deployment and configuration of applications for ambient systems. In: The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), Procedia Computer Science, vol. 52, pp. 373–380 (2015)
24. Ramchurn, S.D., Huynh, D., Jennings, N.R.: Trust in multi-agent systems. Knowl. Eng. Rev. **19**(1), 1–25 (2004)
25. Ricci, A.: Agents and coordination artifacts for feature engineering. In: Ryan, M.D., Meyer, J.-J.C., Ehrich, H.-D. (eds.) Objects, Agents, and Features. LNCS, vol. 2975, pp. 209–226. Springer, Heidelberg (2004). doi:10.1007/978-3-540-25930-5_13
26. Such, J.M., Espinosa, A., GarcíA-Fornes, A., Sierra, C.: Self-disclosure decision making based on intimacy and privacy. Inf. Sci. **211**, 93–111 (2012)
27. Such, J.M., Espinosa, A., Garca-Fornes, A.: A survey of privacy in multi-agent systems. Knowl. Eng. Rev. **29**, 314–344 (2014)
28. Tentori, M., Favela, J., Rodriguez, M.D.: Privacy-aware autonomous agents for pervasive healthcare. IEEE Intell. Syst. **21**(6), 55–62 (2006)
29. Udupi, Y.B., Singh, M.P.: Information sharing among autonomous agents in referral networks. In: Joseph, S.R.H., Despotovic, Z., Moro, G., Bergamaschi, S. (eds.) AP2PC 2007. LNCS (LNAI), vol. 5319, pp. 13–26. Springer, Heidelberg (2010). doi:10.1007/978-3-642-11368-0_2
30. Westin, A.F.: Privacy and Freedom. Atheneum, New York (1967)