HomeManager: Testing Agent-Oriented Software Engineering in Home Intelligence

Ambra Molesini[1], Enrico Denti[1], and Andrea Omicini[2]

[1] ALMA MATER STUDIORUM—Università di Bologna
viale Risorgimento 2, 40136 Bologna, Italy
ambra.molesini@unibo.it, enrico.denti@unibo.it
[2] ALMA MATER STUDIORUM—Università di Bologna a Cesena
via Venezia 52, 47023 Cesena, Italy
andrea.omicini@unibo.it

Abstract. Ambient Intelligence is an interesting research application area for Multi-agent Systems, in general, and for Agent-oriented Software Engineering, in particular. In this paper, we focus on the methodological support that agent-oriented methodologies can provide to such kind of systems: we discuss HomeManager, an application for the control of an intelligent home designed through the SODA agent-oriented methodology. There, the house is seen as an intelligent environment made of independent, distributed devices, each equipped with an agent to support the user's goals and tasks.

1 Introduction

The concept of Ambient Intelligence (AmI) introduces a vision of the Information Society where the emphasis is on greater user-friendliness, more efficient services support, user-empowerment, and support for human interaction [1]. In this scenario, an "intelligent environment" is a space that contains myriad devices that work together to provide users access to information and services [2,3]. So, people are surrounded by intelligent intuitive interfaces that are embedded in all kinds of objects and an environment that is capable of recognising and responding to the presence of different individuals in a seamless, unobtrusive and often invisible way. As noted in [4], AmI implies several challenges concerning the world of physical resources (sensors, actuators), the existence of many forms of task or goal interactions, a natural physical and functional distribution of control, a requirement for actions to be taken in given time intervals, and the integration of potentially conflicting preference specifications.

Other features include *(i)* the adoption of suitable coordination models to manage the physical resources, *(ii)* the support for the security techniques – access control, intrusion detection, etc. –, *(iii)* the support for richer interactions with the users, and *(iv)* a deeper understanding of the structure of the physical space. Due to these requirements, autonomy, distribution, adaptation and proactiveness emerge as key features of the entities populating the intelligent environment [5]. Such features naturally lead to consider agents and Multi-Agent Systems (MASs) as effective metaphors for system design and implementation in AmI scenarios [6].

J. Filipe, A. Fred, and B. Sharp (Eds.): ICAART 2009, CCIS 67, pp. 205–218, 2010.

Several works exploiting MASs in the context of home intelligence (or smart home) – an AmI specific scenario – can be found in the literature—among these, IHome [4], C@sa [5] and MavHome [7]. There, the house is seen as a MAS where agents perceive the environment by means of sensors, and act upon it by means of actuators. However, these works mainly present the application from the implementation viewpoint, providing only few guidelines from the methodological viewpoint: so, the underlying process guiding designers from the requirements analysis to the design choices – in particular, to select an architecture among all the possible ones – is somehow hidden and difficult to understand.

In this paper we adopt a different approach, focussing specifically on the system design process. In particular, we discuss how SODA [8], an agent-oriented software engineering (AOSE) methodology, can support the analysis and design of HomeManager, an agent-based application for the control of an intelligent home. Section 2 presents our reference scenario and discusses its main requirements, while Section 3 is devoted to the analysis, the design and an outline of the first prototype of HomeManager. Discussion and comparison with some relevant related work are reported in Section 4. Conclusions and future work follow in Section 5.

2 Scenario

We consider a home automation application, whose goals are limiting the overall energy consumption while supporting people's activities inside the house [9]. The application is supposed to mediate between the desire for higher comfort and the service cost, respecting both technical constraints and users' specifications; it should also be controllable both locally and remotely – e.g. via short messages on PDAs or mobile phones – in a transparent way.

Two basic user categories can be identified: people stably living in the house, and visitors. The latter cannot operate on the system, but should be enabled to exploit some support functionalities. Inhabitants, instead, may have different authorisation levels. So, at least three user types can be identified:

- *administrators* can specify the management policies of any all the systems aspects, and decide (other) users' privileges;
- *standard users* can express their preferences and give commands via SMS, but cannot act on power consumption nor can they change the privilege level of other users;
- *visitors* are occasional, unregistered users, which can only receive basic assistance.

All users can specify their preferences about the available devices—e.g. the desired temperature in a given room, the automatic switch-on of TV when entering the room, etc.; the system should do its best to meet such requirements, adopting suitable criteria to solve possible conflicts. For instance, different priorities could be assigned to different users based on temporal precedence (i.e., who asked first), or following ad-hoc policies imposed by administrators (e.g., a hi-fi stereo and a TV set cannot be switched on in the same room at the same time). The resulting plan will take care both on users' preferences and policies, and power consumption policies: for the sake of simplicity we

assume that administrators can choose whether to specify either the maximum power consumption allowed, or the period of the day where to concentrate most of the power consumption. Of course, many other strategies could be devised, without affecting the structure of the envisioned scenario.

2.1 Problem Analysis

The environment to be controlled is a family house: its rooms are supposedly provided with input/output sensors for identifying each (registered) user upon entry/exit into/from that room, via suitable identification techniques; unrecognised users should be offered the chance to identify explicitly via suitable system terminals. If they remain unidentified, they should be treated as visitors, with only a minimal set of actions allowed (such as, for instance, turning on/off the room light or the radio). These assumptions make it possible to know the exact position of each user at any time, which is needed for the system to perform correctly (e.g., to turn on the radio or TV in the right room).

Given that the house is a private building, we choose not to limit the number of people that can stay in a room at any time, and to leave free access to any room even without identification: the idea behind this assumption is that the system should assist its users in a constructive and pro-active way, rather than complicating their life with annoying constraints. This choice does not affect generality, since adding further security requirements has no impact on the other system requirements—specifically, on the user preferences and on the system planning choices.

As outlined in the requirements, the system should support users in three ways:

- satisfying the users' requests whenever possible, and trying to anticipate their needs by suitably pre-programming the available devices according to their preferences (e.g. room temperature);
- allowing users to give commands both directly and remotely, via short messages;
- minimising the house total power consumption.

Of course, several kinds of conflicts could arise: different users might express conflicting desires (for instance, one might want to turn on the heater, another the air conditioner), or their requests might involve more power than it is actually available. The conflict resolution policy is to be determined by administrators, and should be able to define priorities in a clear way. In this case, it should at least:

- decide whether to keep into higher consideration either the power consumption or the user requests;
- give different priorities to the different users—for instance, assigning higher priority to the (more tired) family member who comes back home later from work;
- keep into account the temporal order of users' requests;
- allow the system to postpone some non-critical activities when needed—for instance, in order to keep the overall power below the limit.

In order to carry out its management policy, the system must obviously operate on the available devices—to turn them on/off, adjust their parameters, etc; our hypothesis is

that such devices are never accessed directly, but, by asking them to perform some services via some suitable interaction protocol. Each device is supposed to handle its low-level details, so that the higher-level coordination system needs not be aware of them, also enhancing scalability.

2.2 SODA

SODA (Societies in Open and Distributed Agent spaces) [10,11] is an agent-oriented methodology for the analysis and design of agent-based systems, which adopts the Agents & Artifacts (A&A) meta-model [12] and introduces *layering* as the main tool for scaling with the system complexity, applied throughout the analysis and design process [13].

The SODA abstractions (explained below) are logically divided into three categories: *i)* the abstractions for modelling/designing the system active part (task, role, agent, etc.); *ii)* the abstractions for the reactive part (function, resource, artifact, etc.); and *iii)* the abstractions for interaction and organisational rules (relation, dependency, interaction, rule, etc.). Moreover, the SODA *process* is organised in two phases, structured in two sub-phases: the *Analysis phase*, which includes the Requirements Analysis and the Analysis steps, and the *Design phase*, including the Architectural Design and the Detailed Design steps. Each sub-phase models (designs) the system exploiting a subset of SODA abstractions: in particular, each subset always includes at least one abstraction for each of the above categories – that is, at least one abstraction for the system active part, one for the reactive part, and another for interaction and organisational rules.

Figure 1 shows an overview of the methodology structure: as we will show in the case study (Section 3), each step is technically described as a set of relational tables (listed in Figure 1 for completeness).

3 Home Manager

In this section we present the analysis (Subsection 3.1), the design (Subsection 3.2) and the first prototype (Subsection 3.3) of HomeManager. Due to the obvious limitations in space, only a few tables are actually reported in the article.

3.1 Analysis

Requirements Analysis. The first SODA step is the Requirements Analysis, where the system requirements and the world of the legacy systems are identified. Figure 2 shows the system requirements at the core layer – SODA supports a layering principle: a system can be seen at different levels of detail – so that only three coarse-grained requirements are present, one for each "macro" category identified: namely users, plans/policies, and devices. The legacy systems devised out are the devices placed in the rooms: each is in some relation with the three requirements above. This coupled relationship between requirements and legacy systems is tied to the particular kind of system, as in the home automation scenario the devices represents a very important portion of the system to be controlled.

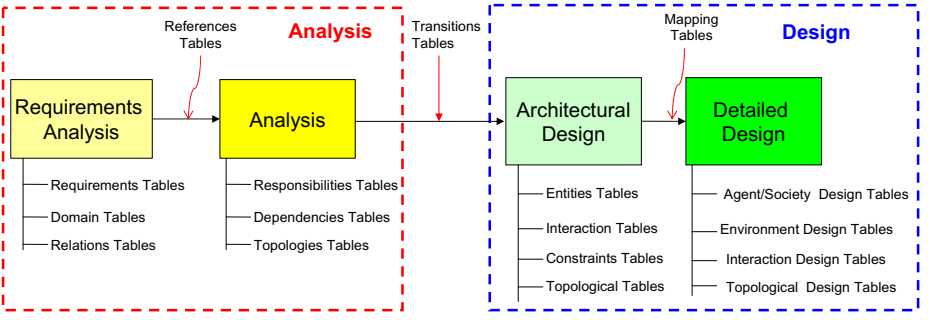

Fig. 1. An overview of the SODA process

Requirement	Description
Users management	management of the user data and preferences
Plans management	management of the plan
Devices management	management of the devices

Fig. 2. Requirement table

Analysis. The transition between Requirements Analysis and Analysis is expressed by the References Tables: as an example, Figure 3 reports the mapping between the requirements and the tasks they generate. Even though such tasks are more fine-grained than requirements, they are still represented at a high level of abstraction, so they could be in-zoomed in a more detailed layer.

Requirement	Task
Users Management	show_data, show_preferences, update_preference, add_user, canc_user, update_user
Plans Management	show_plane, show_policy, load_policy, accomplish_policy, elaborate_plan, execute_plan, detect_conflict, solve_conflict, check_request, check_activity
Devices Management	show_status, add_device, remove_device, check_people

Fig. 3. Reference Requirement-Task table

During the Analysis step also the environmental functions that derive both from requirements and legacy-systems and from the topological structure of the environment are individuated. This latter is already identified by both the physical structure of the controlled house and the position of the devices inside the rooms. In this step the dependencies among the tasks and functions are devised out, too. These relationships are exploited to determine the interaction spaces of the entities, the rules that govern interactions, and the related social aspects like organisation management, coordination and system security.

The reason why dependencies are derived from requirements, legacy systems and topology is that some access control rules are often expressed implicitly as topological "requirements"—as for instance in "The double room has a private bathroom". If we analyse such a requirement, we see that there are two connected rooms – a double room and a bathroom – that represent a portion of the environment structure, and recognise a clear, yet implicit, access control rule: "only the people allowed to enter the double room can access the bathroom". In turn, this can be seen as a prohibition for visitors to enter the bathroom, leading to an access control rule like "the role Visitor is not allowed to enter in the double room bathroom", ready to be translated into an RBAC (Role-Based Access Control) [14] rule.

Coordination plays a key role in this kind of applications, because the right "implementation" of the plan derives from the coordination of the different entities that mediate between the users preferences and the limitation of the energy consumption. The next subsection focuses on the design of such rules.

3.2 Design

Architectural Design. During the Architectural Design, the system is designed in terms of roles, resources, actions, operations, interactions, rules and spaces. These entities derive form the abstractions outlined in the previous step: roles are responsible for the achievement of tasks by executing actions, while resources offer the functions outlined in the previous step by providing the corresponding operations. Typically the task-role and function-resource mappings are not one-to-one, as a role/resource is able to complete/accomplish several different tasks/functions. For example, the role "Preference Manager" is responsible for the "show_preference" and "update_preference" (Figure 3) tasks. The SODA spaces are easily derived from the topology, and map one-to-one the physical rooms that compose the controlled house.

As noted in the Analysis, the core of the system is the coordination among the entities, which is captured by the SODA abstract entities "interactions" and "rules" derived from the dependencies. So, in order to design such aspects, a reference model for coordination [15] has to be chosen. Given the intrinsic features of this application, where a multiplicity of user preferences have to be considered altogether, prioritised and enforced at any time, while users enter/exit the house rooms, an approach based on mediated interaction seems more suitable than, for instance, a bid-based approach. In fact, mediated interaction makes it possible to delegate the coordination medium for policy store and enforcement, thus reducing the time needed to enforce the policy, and freeing roles from taking care of the coordination burden. Also, any policy change affects only a specific place, with no impact on roles—and therefore transparently with respect to them.

This model leads to design interactions as if no policies exist, delegating their enforcement and the access control to the design of the coordination laws expressed by the SODA rules. More concretely, if a user asks HomeManager to switch on the dishwasher, the request is managed by the role designed for achieving this task ("request dispatcher") by sending the request to the "dishwasher manager" role. Such a delivery, however, is not performed directly: rather, it is managed by a coordination medium, that is supposed to react according to the policy and thus dispatching the request to the

dishwasher manager or send a notification to the request dispatcher if the action is not allowed by the policy (for instance, if the energy consumption is close to the top limit—see the "MaxEnergy Rule" in Figure 4). The coordination medium is "transparent" for request dispatcher and dishwasher manager, as it represents a sort of "infrastructure layer" not directly perceived by roles.

For space reasons, Figure 4 shows only an excerpt of the Rule table that lists and describes all rules—namely, some representative rules for each "macro category". So, the first block includes the rules devoted to the general house policy – energy consumption and users priority –, the second represents the rules concerning the use of devices, while the third block lists the access control rules.

Rule	Description
MaxEnergy Rule	The electrical power cannot exceed 3 KW
Priority Rule	The preferences of the priority user have to be satisfied
Default Rule	Turn on lights when Visitor goes inside room
Heating Rule	Heating is more priority then other devices
Visitor Rule	Visitor cannot access to the system
Role Rule	User cannot modify its role
User Rule	User can modify only its data and preferences
...	...

Fig. 4. Rule table

Moving from Architectural Design to Detailed Design, the engineer should now operate a carving operation, so as to choose the most adequate representation level for each architectural entity: this leads to depict one (detailed) design from the several potential alternatives outlined by the layering. In our case, since the core layer is placed at a high level of abstraction, and no in-zoom operation is present, the carving operation is simply not necessary, and the core layer becomes the only layer of the Detailed Design.

Detailed Design. Detailed Design is expressed in terms of agents, agent societies, composition, artifacts, aggregates and workspaces for the abstract entities; interactions, in their turn, are expressed by means of concepts such as uses, manifests, speaks to and links to.

In HomeManager, agents play the roles individuated in the previous step, while resources are mapped onto suitable environmental artifacts—a kind of artifact [16] aimed at wrapping the physical devices that constitute the MAS external environment, or realising functions derived by requirements but not available in the external environment. Since the design level is very abstract, we devise out just one agent society. Moreover, spaces and their connections are then mapped one-to-one onto workspaces and

the respective workspace connections. Interactions are mapped onto different detailed abstractions in order to capture the different "nature" of the interaction itself: more precisely, agent-to-agent interaction is mapped onto *speaks to*, agent-to-artifact onto *uses*, artifact-to-agent onto *manifests*, and artifact-to-artifact onto *links to*.

In addition, in SODA each agent is associated to an individual artifact [16] that handles the interaction of a single agent within a MAS, and is used to shape of admissible interactions of MAS agents. Such individual artifacts play an essential role in engineering both organisational and security concerns: in fact, access control rules (third block of rules in Figure 4) are mapped precisely onto agents' individual artifacts. The rules in the first and second block are mapped onto social artifacts [16], as they rule the social interactions—even though indirectly, since they technically mediate interactions between individual, environmental, and possibly other social artifacts.

Social artifacts in SODA play the role of the coordination artifacts – i.e., coordination media – that embody the rules around which agent societies are built. Among the possible different social artifacts "organisations" (discussed in Section 4), we choose the one where each workspace is associated to a social artifact that rules the agents and the devices inside a room so as to manage the user priorities and control the access to the devices. We also easily individuate an aggregate of such social artifacts that can "enforce" the global rules of the house—like, for example, the "MaxEnergy Rule" in Figure 4. Figure 5 shows only an excerpt of the Artifact-UsageInterface table, which actually lists all the operations provided by each artifact in detail.

Artifact	Usage Interface
Room manager	receive_command send_command send_notification set_policy, get_policy get_plan, get_user...
Home manager	set_homepolicy, get_homepolicy send_command get_homeplan, get_userpos...
Device manager	switch_on switch_off...
...	...

Fig. 5. Artifact-UsageInterface table

As mentioned above, this is a simplified version of the system we actually designed and implemented (Subsection 3.3). In fact, in a real scenario, each room should be expected to contain a multitude of agents and devices, which could not be reasonably managed by a single social artifact for obvious performance and reliability reasons—preventing bottlenecks, avoiding delays in the system responses, etc. So, the social artifact should rather be seen as an abstract view of an aggregate of other social artifacts, each enforcing some given kind of rules.

3.3 Prototype

Following the above analysis and design process, we realised a prototype simulator [9] for HomeManager based on the TuCSoN infrastructure [17,18]. TuCSoN is well

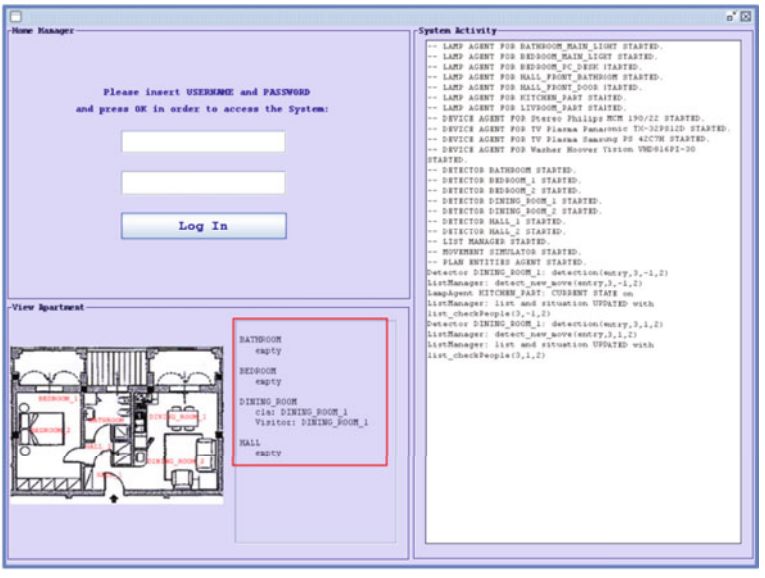

Fig. 6. The HomeManager main window

suited for this kind of application, since it provides "tuple centres" as coordination media, which allow a one-to-one map with the SODA social artifacts. In short, a tuple centre is a tuple space enhanced with behaviour specification: it is still perceived by agents as a standard tuple space [19], but its behaviour can be specified by suitable coordination laws. Since the behaviour specification of TuCSoN tuple centres is expressed in the ReSpecT language [12], the SODA rules enforced by the social artifacts take the form of suitable ReSpecT reactions.

In our prototype we simulated the control of a house as composed of four rooms: the main entrance, the dining room, the bedroom and the bathroom (Figure 6). Each room has several devices, like a washing machine in the bathroom, a hi-fi and a tv set in the dining room, another tv set in the bedroom, as well as various devices for light management. The main interface (Figure 6) is organised in three areas [9]: the login area (top-left) where users authenticate and modify the HomeManager parameters; the view-house area (bottom-left) which reports the map of the house (in the left side) and the current position of the people inside the house (right side): this information, highlighted in the figure by the red rectangle, is currently expressed as a textual description, but will be reported directly on the map in a future version; finally, the system activities area (right side) shows both the activities of the system and the messages sent by each device when working.

So, for instance, when an (authorised) user goes into the dining room, HomeManager reacts by signalling the presence of the user in the room, turning on the light if necessary, and applying the user's preferences – temperature, devices turned on/off – to that room. If there are two or more people inside the same room with different preferences, HomeManager reacts according to the preferences of the most important (priority) user balancing her/his preference with the energy consumption. A simple

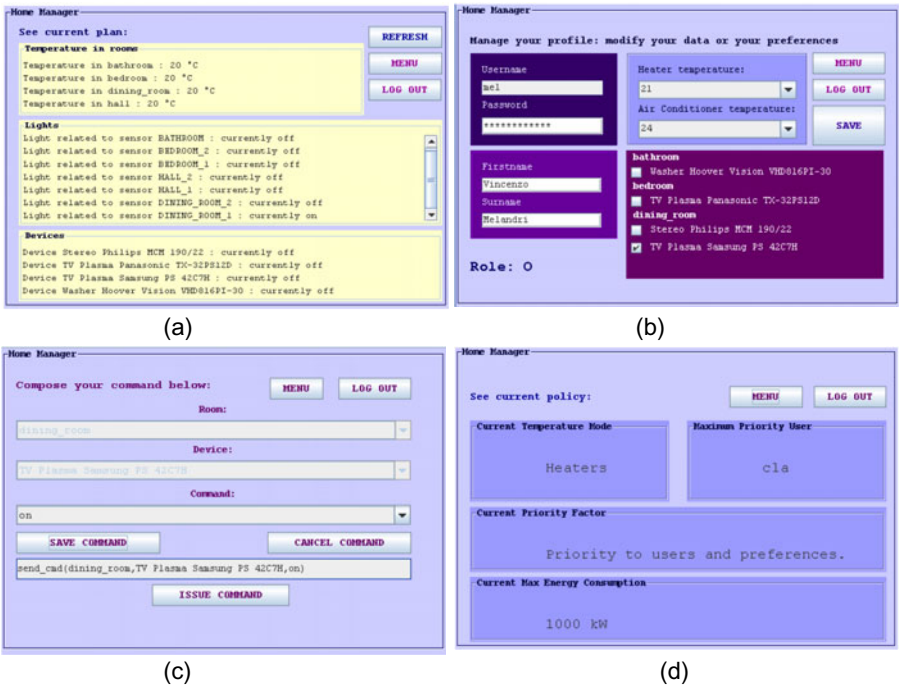

Fig. 7. Some screenshots of HomeManager: a) policy-management interface, b) profile-management interface, c) command interface and d) current-policy interface

default policy is applied to visitors (users with no profile), which just switches the light on/off when he/she enters the room, as no assumptions can be made about the visitor's preferences.

Figure 7 shows four further views of HomeManager. View (a) shows a screenshot of the policy-management interface: for each room, the GUI reports the temperature, the state of the lights and of the other room devices. View (b) shows the profile-management interface: there, each user can change his/her profile and preferences about the temperature and devices. However, he/she cannot change his/her role in the house, as this operation is reserved to administrators. The screenshot in view (c) is about the command interface: the authorised user operates on the system by choosing the room and the device (first two rows), and then issuing the desired command. HomeManager replies by starting a new action plan for the room where the device is placed. The GUI in view (d) shows the current policy for the selected room: in particular, the GUI reports the current temperature mode, the maximum priority user, the current priority factor and the current max energy consumption.

4 Discussion

The intelligent home scenario represents a good application domain for testing the MAS approach in general and the agent-oriented methodologies in particular. In fact, this

scenario introduces several different challenges from the methodology viewpoint, such as, for instance, the management of the security aspects and the energy consumptions limitation, that should be balanced with the system fast reaction and user's comfort, wellness and entertainment.

In order to obtain the best balancing among these requirements, the methodology should support the design of suitable coordination models so that the agents can collaborate and fulfil the house control task. As highlighted in Subsection 3.2, mediated coordination seems the best solution for this kind of application. This choice leads to structure the design of interactions and coordination rules in a specific way, yet without imposing any specific "architecture" for the system. In particular, the model considers a "coordination medium", but makes no hypotheses on the entities that should "implement" this role.

In SODA the coordination rules are naturally mapped onto social artifacts, so the coordination artifacts are delegated to enforce the coordination rules; in IHome [4], instead, the centrally-managed resources (IHome also considers decentralised resources) are controlled by an "agentified" coordination medium. Coordination is managed by agents also in Cas@ [5] and Intelligent Room [3].

Even though both approaches present strengths and drawbacks, from the design-for-change perspective delegating coordination to a coordination artifact seems a better choice, as it leads to encapsulate a coordination service in a specific entity, allowing user agents to abstract from how the service is implemented. As such, a coordination artifact is perceived as an individual entity, but can be actually distributed on different nodes of the MAS infrastructure, depending on its specific model and implementation [20]. In addition, coordination artifacts are meant to support coordination in open agent systems, characterised by unpredictable events and dynamism. For this purpose, they have to support a specific form of artifact malleability—namely, they should allow their coordinating behaviour to be adapted and changed both by engineers (humans) willing to sustain the MAS behaviour, and by agents responsible for managing the coordination artifact. So, coordination artifacts can be seen as engineering abstractions for designing, building and supporting at runtime coordination in agent societies, suitably instrumenting their dynamic working environment [20].

Of course, one may also adopt agents working so as to mimic coordination media, but such an approach seems less flexible in this context, for a change in a house policy would imply a modification of the coordination rules and probably also of the coordination protocols between the coordinator agent and the coordinated agents, spreading the impact of the change nearly everywhere. Instead, a coordination artifact helps keeping such changes inherently encapsulated.

With respect to the choice of the social artifact "organisation", it should be noted that the design solution mentioned in Subsection 3.2 is not the only possible. In fact, we exploited a social artifact – or an aggregate of social artifacts – as the room manager, so that all the room managers form an aggregate that coordinates the whole house: however, a different architecture, not-so-tied to the physical environment structure, could also be possible. For instance, an organisation based on the system functionalities could be based on a social artifact devoted to coordinate the entertainment devices – tv set,

hi-fi, … –, another for heating and air conditioning, a third one for lights, etc.—all forming an aggregate to coordinate the whole house.

Currently, we do not have enough experimental results convincing us to prefer an architecture to another. We chose the first – a social artifact manager for each room – for the prototype mainly because it naturally bounds the "scope" of each social artifact to the physical structure of the environment, making it easier to define the coordination rules holding inside each room.

5 Conclusions and Future Work

HomeManager is an agent-based application for controlling an intelligent home. We discussed its analysis and design with special regard to the coordination and access control rules which motivate the architectural design choices, from the coordination model to the social artifact organisation. In particular, we stressed the relevance of a mediated coordination model and of delegating coordination to a coordination artifact, so that agents can exploit coordination as any other service, without being concerned about the possible changes in the house policies.

The experiment was an interesting testbed for AOSE in general, and for the SODA methodology in particular, highlighting some benefits as well as some limitations that we mean to address in the near feature. In particular, the tabular representation is clearly more suitable for an automatic tool than for a human designer, due to the large amount of tables to be filled in at each stage: so, we plan to develop tools for supporting designers in doing so in a consistent and complete way. Another interesting extension could be the definition – or the adoption – of a language for specifying SODA rules in a more precise and formal way, overcoming the implicit limitations of the natural language which is currently adopted in the tabular representation. We also plan to evaluate whether to enrich SODA with methods for the internal design of agents and artifacts—which are not yet covered, since SODA currently does not deal with intra-agent (and more generally with "internal") issues.

Further work will be devoted to improve the current version of the prototype, implementing all the access control checks and comparing the two architectures discussed in Section 4. Moreover, as a form of protection against thiefs, HomeManager could inform administrators by SMS/email if an unidentified visitor is in the house: RBAC policies of the role visitor should then be revised accordingly, splitting the visitor role in sub-roles such as "closely related", "friends", "authorised visitor", and "unknown". Of course, this would imply the installation of some user (possibly biometrical) authentication mechanism. Moreover, we plan to consider some fuzzy-set theory methods to find a better compromise between the profiles of different people, instead of the simple priority policy which is currently adopted, and to test our prototype in some real environment.

Acknowledgements. Authors would like to thank Dr. Claudia Fontan for her contribution to the project and her work in the prototype implementation.

This work has been supported by the *MEnSA* project (*Methodologies for the Engineering of complex software Systems: Agent-based approach*) funded by the Italian Ministry of University and Research (MIUR) in the context of the National Research 'PRIN 2006' call.

References

1. Ducatel, K., Bogdanowicz, M., Scapolo, F., Leijten, J., Burgelman, J.C.: Scenarios for ambient intelligence in 2010. final. IPTS – European Commission's Joint Research Centre (2001)
2. Brumitt, B., Meyers, B., Krumm, J., Kern, A., Shafer, S.A.: Easyliving: Technologies for intelligent environments. In: Thomas, P., Gellersen, H.-W. (eds.) HUC 2000. LNCS, vol. 1927, pp. 97–119. Springer, Heidelberg (2000)
3. Kulkarni, A.: Design Principles of a Reactive Behavioral System for the Intelligent Room. ACM/IEEE Bitstream: The MIT Journal of EECS Student Research, 22–26 (2002)
4. Lesser, V., Atighetchi, M., Benyo, B., Horling, B., Anita, R., Vincent, R., Wagner, T., Xuan, P., Zhang, S.X.: The UMASS intelligent home project. In: Etzioni, O., Müller, J.P., Bradshaw, J.M. (eds.) 3rd International Conference on Autonomous Agents (Agents 1999), pp. 291–298. ACM, New York (1999)
5. De Carolis, B., Cozzolongo, G.: C@sa: Intelligent home control and simulation. In: Okatan, A. (ed.) International Conference on Computational Intelligence (ICCI 2004), Istanbul, Turkey, pp. 462–465. International Computational Intelligence Society (2004)
6. Zambonelli, F., Omicini, A.: Challenges and research directions in agent-oriented software engineering. Autonomous Agents and Multi-Agent Systems 9(3), 253–283 (2004)
7. Cook, D.J., Youngblood, M., Heierman, E.O.I., Gopalratnam, K., Rao, S., Litvin, A., Khawaja, F.: MavHome: An agent-based smart home. In: 1st IEEE International Conference on Pervasive Computing and Communications (PerCom 2003), Washington, DC, USA, pp. 521–524. IEEE CS, Los Alamitos (2003)
8. SODA: Home page (2008), `http://soda.apice.unibo.it//`
9. Fontan, C.: Tecnologie ad agenti per una casa intelligente. Master's thesis, Università di Bologna, Italy (2006)
10. Omicini, A.: SODA: Societies and infrastructures in the analysis and design of agent-based systems. In: Ciancarini, P., Wooldridge, M.J. (eds.) AOSE 2000. LNCS, vol. 1957, pp. 185–193. Springer, Heidelberg (2001)
11. Molesini, A., Omicini, A., Denti, E., Ricci, A.: SODA: A roadmap to artefacts. In: Dikenelli, O., Gleizes, M.-P., Ricci, A. (eds.) ESAW 2005. LNCS (LNAI), vol. 3963, pp. 49–62. Springer, Heidelberg (2006)
12. Omicini, A.: Formal ReSpecT in the A&A perspective. ENTCS 175(2), 97–117 (2007)
13. Molesini, A., Omicini, A., Ricci, A., Denti, E.: Zooming multi-agent systems. In: Müller, J.P., Zambonelli, F. (eds.) AOSE 2005. LNCS, vol. 3950, pp. 81–93. Springer, Heidelberg (2006)
14. RBAC: American National Standard 359-2004 – Role Base Access Control home page (2004), `http://csrc.nist.gov/rbac/`
15. Papadopoulos, G.A., Arbab, F.: Coordination models and languages. Advances in Computers 46, 330–401 (1998)
16. Omicini, A., Ricci, A., Viroli, M.: *Agens Faber*: Toward a theory of artefacts for MAS. ENTCS 150(3), 21–36 (2006)
17. TuCSoN: Home page (2008), `http://tucson.apice.unibo.it/`

18. Omicini, A., Zambonelli, F.: Coordination for Internet application development. Autonomous Agents and Multi-Agent Systems 2(3), 251–269 (1999)
19. Gelernter, D., Carriero, N.: Coordination languages and their significance. Communications of the ACM 35(2), 97–107 (1992)
20. Omicini, A., Ricci, A., Viroli, M.: Coordination artifacts as first-class abstractions for MAS engineering: State of the research. In: Garcia, A., Choren, R., Lucena, C., Giorgini, P., Holvoet, T., Romanovsky, A. (eds.) SELMAS 2005. LNCS (LNAI), vol. 3914, pp. 71–90. Springer, Heidelberg (2006)