Enlightened Agents in TuCSoN

Alessandro Ricci Andrea Omicini Enrico Denti

***Abstract*—In the network-centric computing era, applications often involve sets of autonomous, unpredictable, and possibly mobile entities interacting within open, dynamic, and possibly unreliable environments: Intelligent Environments are a typical case. The complexity of such scenarios requires novel engineering tools, providing effective support from the analysis to the deployment stage. In this paper we illustrate the impact of a general-purpose coordination infrastructure for multiagent systems – providing a model, a run-time, and suitable deployment tools – on the engineering of such applications. As a case study, we consider the intelligent management of lights inside a building: despite its simplicity, this problem endorses the typical challenges of this class of applications. The case study is built upon the TuCSoN coordination infrastructure, which provides engineers with both the abstractions and the run-time support for effectively managing the application complexity.**

I. Infrastructures for Network-Centric Computing

The technological progress – concerning chip density, processor speed and network bandwidth, to cite some – makes it possible to conceive new classes of applications: Ubiquitous [1], Proactive [2], and Network Centric [3] computing are among the most important areas of interest. Intelligent / Smart Environments [4], [5] are prominent cases of such application classes (see [6] for a comprehensive and up-to-date collection of references – projects, publications, and conferences), whose aim is to remodel the environments where people live and act, considering the new services that can be provided by embedding (intelligent) software in network-enabled subsystems, such as sensors, actuators, mobile devices and general-purpose computers.

Traditional software engineering approaches are known to be inadequate to face the complexity of these scenarios [7], [8], so new paradigms and models are required. The *agent* paradigm [9], originated in the distributed artificial intelligence context and now central also in software engineering research [10], [11], appears to be more effective and to provide better support in the application analysis, design, development and deployment stages.

However, the engineering of these systems, which are characterised by the integration and coordination of open and dynamic sets of (heterogeneous) agents, calls for other abstractions, such as the *society* concept [12]. This is the focus of several research efforts, introducing notions such as *social agency* [13], *socialware* [14], and *social laws* [15].

So, new engineering methodologies, models, infrastructures and technologies are needed to explicitly face the design of both the single agents and the society itself: social tasks, social laws, and collective behaviour should be considered first-class issues, constituting the system *social / global intelligence* [16]. This approach includes, for instance, the ability of specifying goals, constraints, and desired properties that are not specific of a given agent, but of a community of agents, as well as the ability to adapt / modify this *social glue* dynamically, at run-time. Furthermore, it should be possible to specify and enforce the coordination activities that involve an open (and possibly a-priori unknown) set of agents: since agent, thanks to their autonomy, can be perceived only by their observable behaviour, acting upon them means / requires interacting with them.

The case study considered in this paper – the management of lights inside a building – is taken here as a simple representative of this class of applications: our aim is to describe how to model, design and develop this application as a society of agents. In particular, we show how the social aspects of the system can be properly engineered by means of a suitable coordination model (for analysis and design) and infrastructure (for development and deployment). The coordination infrastructure of choice, TuCSoN, provides both a coordination model and a technology for development and deployment, including a run-time environment and IDE tools.

In the following, Section II introduces the case study, focusing on the application requirements. Section III discusses the application analysis and design, adopting the SODA agent-oriented software engineering methodology, while development and deployment are considered in Section IV, where the TuCSoN coordination infrastructure is introduced and exploited. The case study concludes in Section V, by discussing the impact of integrating a new service into the application: Section VI then discusses the benefits and drawbacks of TuCSoN in the engineering of the application. Conclusions are finally drawn in Section VII.

II. The Light Management Case Study

This work has been inspired by current research on agent infrastructures for intelligent environments [17], [18], [19]. Despite its simplicity, the case study below is representative of a broad class of applications, where autonomous and (possibly) mobile agents interact in an open environment, like the Internet.

In the case study presented here, agent coordination requires the management of resources (lights) shared by autonomous agents (visitors), whose desires and preferences call for different coordination policies, to be changed and adopted dynamically. In particular, the application goal is to manage the lights in the rooms of a building so as to automatically adapt their intensities to the visitors' preferences while respecting the existing constraints – and more generally to apply light policies that could change over the time.

A. Structure

Each room has its own light policy, ruling the behaviour of lights: light policies can be added / removed dynamically. Henceforth, we will refer to the visitors of the building as *visitor*

A. Ricci is with the DEIS, Università di Bologna
Via Rasi e Spinelli, 176, 47023 – Cesena (FC), Italy
email: aricci@deis.unibo.it
A. Omicini is with the DEIS, Università di Bologna
Via Rasi e Spinelli, 176, 47023 – Cesena (FC), Italy
email: aomicini@deis.unibo.it
E. Denti is with the DEIS, Università di Bologna
Viale Risorgimento, 2, 40136 – Bologna, Italy
email: edenti@deis.unibo.it

agents. Visitor agents freely move from room to room according to their purpose: their light preference is kept into account when either they stop moving, or change their preference explicitly. Each light source is characterised by the zone enlightened by its light: only visitors currently inside this zone contribute, by their preference, to determining the source lightness.

There are two kinds of light sources: *zone lights* and *directional lights*. Zone lights are room lamps whose influence zones do not to overlap. Directional lights, instead, are extra sources that can overlap zone lights, so as to produce a local light enhancement.

The intensity of a zone light is computed according both to the preferences of agents inside the specific zone, and other constraints such as energy management. We consider three different kinds of (simple) light policies:

- *mean value* – the light intensity of a lamp is computed as the mean value of the preferences of visitors in the influence zone of the lamp;
- *min value* – the light intensity of a lamp is computed as the minimum value of such preferences;
- *max value* – the light intensity of a lamp is computed as the maximum value of such preferences.

We also consider different types of visitor agents: agents moving randomly from room to room, agents looking for a room with their preferred light intensity, and agents looking for empty rooms where to stay alone. From the viewpoint of the multiagent system, however, all such visitors have the same (private) goal: getting the light intensity of the room they are visiting - or, better, of the lamp which they are influencing - as nearest to their preference as possible.

B. Key Issues

The management of a dynamic set of shared resources (lights) among an (open) set of autonomous agents (visitors) is a typical coordination problem. In our case, there is no a-priori knowledge about which agents, and how many, will visit the rooms, nor is there any assumption about their structure or behaviour: we can only observe them or interact with them, with no chance of knowing or changing their behaviour (or their code) in order to coordinate them.

Also, in this case study light management policies can be changed dynamically and must be enforced in a prescriptive way, without relying on agents' finding an agreement by themselves: this requirement implies the adoption of a coordination model enabling coordination policies to be specified outside the coordinated agents, providing the capability of enforcing them prescriptively, and supporting the dynamic specification / change of coordination laws – all without bothering the (unaware) agents.

From the single agent viewpoint, the case study underlines agent *autonomy*, since visitors express their light preference where and when they want autonomously, and move inside the building according to their own plans. In addition, it requires *(virtual) mobility*, since visitors - moving from room to room - meet different social contexts, find different light policies and, more generally, heterogeneous local interaction contexts, each with its own coordination laws and constraints.

III. Case Study: Analysis and Design

Analysis and design are carried out using the SODA [16] agent-oriented software engineering methodology.

A. Analysis sketch

In the SODA methodology the analysis stage is performed by identifying three models: the *role model*, the *resource model* and the *interaction model*.

The role model is aimed at identifying both the individual and the social roles. In our case study, a reasonable model is to consider three individual roles: the *Visitor*, the *Light Master*, and the *Light Installer*. The Visitor role represents visitors moving from a room to another, whose only responsibility is to express a preference about the light intensity of a room. In the context of this multiagent system, a Visitor has no specific competence. The Light Master role represents the one who changes the light policies of the rooms, while the Light Installer is the one who installs / removes room lights. In addition, there is one social role, the *Room Group*, which is composed by all the Visitors currently in a room: its social tasks is to set the room light intensity according both to the current light policy of the room, and to the preferences of the room Visitors. To be able to do so, the Room Group must have the right of accessing light sources to change light intensity.

With respect to the resource model, the following types of resources can be identified: the *Building*, the *Rooms*, and the *Light sources*. The Building is a set of rooms: as such, it provides services to get information about the building planimetry and similar issues. A Room is defined in a bi-dimensional space, and represents the place where light sources are located and visitors move to: its dimension is expressed in blocks. A room provides services to add / remove lights, to retrieve the light intensity in a specific point inside it, and to specify and select the current light policy. Light sources are characterised by their light intensity and influence zone, which is the (rectangular) patch of the room influenced by their light. Lights can be of two types, zone lights – the main, not-overlapping room lights – and directional lights: these have a fixed 2×2 block influence zone size, and can be installed only in addition to already-existing zone lights. Services provided by a light source concern setting / getting its light intensity.

The SODA interaction model includes interaction protocols for individual roles and resources, and interaction rules for social roles. Each Room is an interaction context, with its own protocols and rules. One further interaction protocol enables visitors to express their current preference about the desired light intensity in the point of the room where they are currently situated. Interaction rules are associated to the Room Group: the fundamental rule takes care of setting light intensity according to the current light management policy of the room. This involves collecting visitor preferences, applying the current light policy associated to the room, and then accessing the corresponding light sources to set the light intensity.

B. Design sketch

According to the SODA methodology, the design stage is aimed at defining three models – the *agent model*, the *society* *model*, and the *environment model* – as the design counterparts of the analysis models (role model, resource model, interaction model). More precisely, the agent model is the counterpart, at the design stage, of the individual roles of the role model, while the society model captures the social roles of the role model; the environment model, in its turn, represents the counterpart of the resource model.

So, here the agent model must define a mapping for the *Visitor*, the *Light Master*, and the *Light Installer* roles, while the *Room Group* role is to be mapped by the society model. It is the resource model's goal to define mappings for the *Building*, the *Rooms*, and the *Light sources* resources.

For each individual role, the agent model should define *(1)* the mapping class, *(2)* the cardinality of the class, *(3)* the agent locality degree (mobile vs. fixed agent), and (4) the agent's source. The Visitor role is mapped onto the `Visitor` class (an agent class in SODA terminology) : the cardinality of the class is $0 \div \infty$, locality is mobile, and the source is outside the system. The Light Master role is mapped onto the `LightManager` agent class: cardinality is one per room, locality is fixed, and the source is inside the system. Finally, the Light Installer role is mapped onto the `LightInstaller` agent class: cardinality, locality, and source are identical as above.

With respect to the society model, the Room Group social role is mapped onto the *Room Society* abstraction: the role's responsibility is to set the intensity of light sources according to visitors' preferences and to the light management policy currently defined for that room resource. The Room Society is designed around the *Light coordination medium*, which embeds the interaction rules of the Room Group role in terms of coordination laws.

The environment model is involved with resources. Building is mapped onto the `Building` class (an infrastructure class in SODA terminology), and provides information about planimetry and available rooms: its services can be accessed by visitor agents entering the building. This resource is located and accessible via the *Building coordination medium*: its cardinality is one. Rooms are mapped on the `Room` infrastructure class, which provides services to add / remove lights and to ask for the desired light intensity at a specific position in the room. The first service is accessible only to Light Installers, while the second is available to all Visitor agents. The `Room` class is accessible only via its own *Light coordination medium*. Light Sources are mapped on the `LightSource` class, which allows to set / get the light intensity: these services are available to the Room Group role only. So, Light Sources, too, are located and accessible via the *Light coordination medium*.

IV. Development and Deployment in TuCSoN

Application development and deployment has been carried out upon a mobile agent platform based on the TuCSoN coordination infrastructure.

A. TuCSoN *overview*

The TuCSoN coordination model and infrastructure is based on the notion of (logic) *tuple centre* [20], which is a Linda tuple space [21] empowered with the ability to define its behaviour in response to communication events according to the specific coordination needs.

It has been argued [22] that the openness and the wideness of the Internet scenario make it suitable to conceive the Internet as a multiplicity of independent environments (e.g. Internet nodes or administrative domain of nodes), and to design applications in terms of agents that explicitly locate and access resources in this environment. So, TuCSoN is based on a multiplicity of independent interaction spaces – *tuple centres* – that abstract the role of the environment. (Mobile) agents access tuple centres by name, either locally in a transparent way, or globally on the Internet in a network-aware fashion [23].

A local interaction space can be used by agents to access the local resources of an environment and as an *agora* where to meet other agents and coordinate activities with them. This is why tuple centres, as fully distributed interaction media, can be understood as *social abstractions* [24], allowing to constrain agent interactions explicitly and to enforce the coordination and cooperation activities that define the agent aggregation as a society.

TuCSoN supports both *subjective* and *objective* coordination [25] – that is, the logic of coordination can be either embedded in agents, directly coordinating themselves via generative communication in pure Linda-style, or, better, left outside agents, embedded in tuple centres. By doing so, the application social rules are captured in terms of tuple centres' coordination laws, expressed in the Turing-equivalent, logic-based ReSpecT language.

Interaction protocols can then be designed and distributed among agents and media, adopting the balance that is the most adequate to the specific application goals. One first consequence is the enhancement of agent autonomy, since agents can be designed focusing only on their own goals and tasks, disregarding dependencies with other agents, and with no need to track (open) environment evolution. Another key issue is the enactment of prescriptive coordination, thus constraining agent interactions so as to reflect sound behaviours. Finally, tuple centres spread over the network and living in infrastructure nodes visited by rambling agents enhance the decentralised nature of the multiagent system. So, the topological nature of the TuCSoN global interaction space systems [23] makes it possible to deal with the typical issues of distributed systems – namely to enforce flexible security policies, workload allocation policies, fault tolerance policies.

The TuCSoN technology [26], fully developed in Java, is based on two main ingredients: Java as the main standard Internet "glue technology", and tuProlog [27] as the logic-based language to support high-level inter-agent communication and coordination. Both an agent run-time system and an effective coordination technology, with a comfortable IDE and specialised tools, are provided to effectively support the system development process.

One first advantage of a Java based technology is that any Internet node can become a TuCSoN node by simply starting up the Java-based run-time system: the local coordination space is then immediately available both to local and remote agents. Moreover, as a logic-based technology, higher-level inter-agent communication and coordination is based on first-order logic:

the Java-based tuProlog virtual machine is a light-weight Prolog interpreter used to provide both the communication and the coordination language. So, both Java and tuProlog agents interact by exchanging logic tuples, even though this is transparent to Java agents. Logic tuples are used also for ReSpecT programs: both the ordinary tuple space and the specification space of a tuple centre are represented as first-order logic theories.

Besides featuring classes and tools to build Java and tuProlog agents, the TuCSoN technology provides special IDE agents, *GUI agents* and the *Inspector*, to enable developers to operate over tuple centres for deployment, debugging and monitoring purposes. GUI agents provide a (graphical) interface to access tuple centres by means of TuCSoN coordination primitives (*out*, *in*,*rd*,*inp*,*rdp*,*setSpec*, *getSpec*), while the Inspector is specifically tailored to the TuCSoN metaphors: since a tuple centre is characterised at any time by the set T of its (ordinary) tuples, the set W of its pending queries, and the set S of its reaction specifications [20], the Inspector makes it possible to view, edit and control tuple centres from the data, the pending query and the specification viewpoints – thus providing both a data-oriented (from the communication and the coordination viewpoints) and a control-oriented view over the space of agent interaction, along with the control capabilities required to manage it effectively.

B. Development

From the topology point of view, we assume that there is exactly one distinct TuCSoN node for each room of the building. Moreover, we also suppose that a TuCSoN node represents the building as a whole , so that its coordination media can manage information about the building topology and accessible rooms – in particular, storing the addresses of the room TuCSoN nodes.

With respect to agents, the `Visitor` class has been implemented as a TuCSoN mobile agent: there are different kinds of visitors, according to different ways of exploring the building. `LightManager` and `LightInstaller` class are TuCSoN agents, too. The *Light* coordination medium of each room has been mapped directly onto a tuple centre named `Light`, located at the TuCSoN node of the corresponding room. The interaction rules associated to `Light`, which represent light management policies and agent coordination rules, have been encoded in the ReSpecT language and used to program the behaviour of the `Light` tuple centre. `LightSource` and `Room` resources are wrapped in components accessible only indirectly, by suitably interacting with the local `Light` tuple centre: for instance, to set the intensity of a light source requires a tuple such as `set_light(`*`LampId`*`,`*`Value`*`)` to be output into `Light` by means of an `out` operation.

This approach underlines the role of local interaction spaces as agent interfaces to local resources, thus outlining the relevance of coordination media as *the* middleware enabling interactions, whatever they may be.

C. Deployment

Fig. 1 shows the run time evolution of our system, with rooms, light zone and some mobile agents visiting the building and expressing preferences about lights. The dark gray zones represent zones with an installed (zone) light source, currently

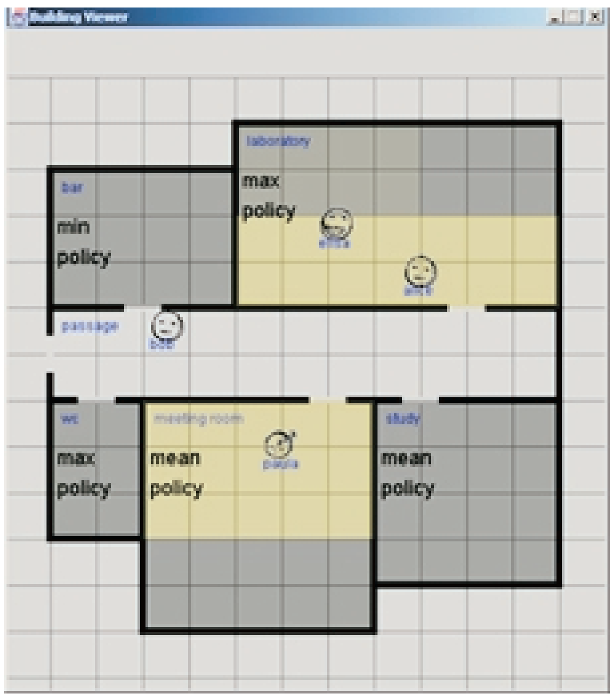


Fig. 1. A snapshot of MAS dynamics with rooms and visitors (mobile agents)

switched off. An indication of the currently installed light policy is shown for each room: in Fig. 1, for instance, the meeting room adopts the mean value policy.

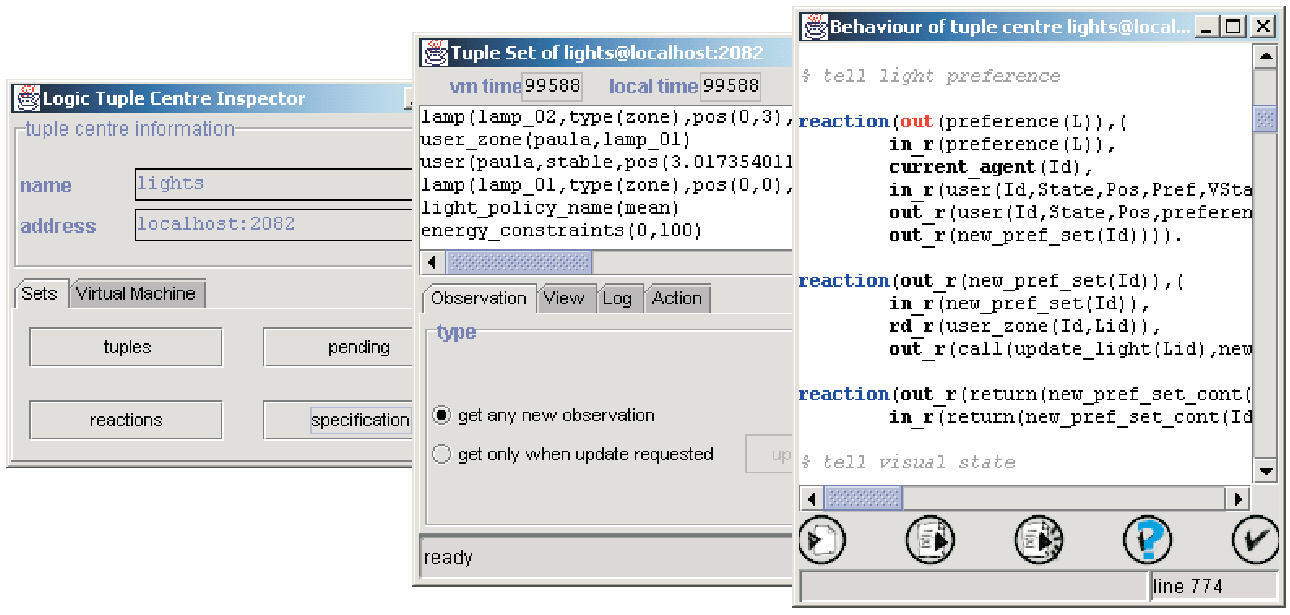
As a test, we can change at run time the light management policy of the meeting room to the min value policy (say, to save energy). We can do so either using an agent, or directly on our own via the Inspector tool, inspecting the `Light` tuple centre of the meeting room, opening the specification set and replacing the coordination rules involved in the light management (see Fig. 2). In this way, the social laws can be adapted dynamically, with no need of acting on visitor agents, and focusing only on the coordination activities, encapsulated in a suitable abstraction.

V. Incremental Development

Now, suppose we want to provide an intelligent service which, analysing the history of the dynamics inside a room, suggests where to place new directional light sources so as to enhance future visitors' overall satisfaction.

To this end, a new role must be introduced - the *History Analyser* - aimed at analysing the history of visitors' satisfaction and possibly provide suggestions. In the environment model, the Room resource must provide a new history service to trace agents' satisfaction, computed as the difference between actual light intensity in the place of the room where they are, and their preference. Of course this service must be accessible by History Analysers only. The interaction model should be updated with new protocols for History Analysers, allowing to start tracing and then to retrieve history information – both accessing the `Room` resource.

At the design stage, the History Analyser role is mapped on the `HistoryAnalyser` SODA agent class, whose cardinality is one and whose source is outside the system. The inter-



name: lights
address: localhost:2082

vm time 99588 local time 99588

```
lamp(lamp_02,type(zone),pos(0,3),
user_zone(paula,lamp_01)
user(paula,stable,pos(3.017354011
lamp(lamp_01,type(zone),pos(0,0),
light_policy_name(mean)
energy_constraints(0,100)
```

(●) get any new observation
( ) get only when update requested

```
% tell light preference

reaction(out(preference(L)),(
        in_r(preference(L)),
        current_agent(Id),
        in_r(user(Id,State,Pos,Pref,VSta
        out_r(user(Id,State,Pos,preferen
        out_r(new_pref_set(Id)))).

reaction(out_r(new_pref_set(Id)),(
        in_r(new_pref_set(Id)),
        rd_r(user_zone(Id,Lid)),
        out_r(call(update_light(Lid),new

reaction(out_r(return(new_pref_set_cont(
        in_r(return(new_pref_set_cont(Id

% tell visual state
```

Fig. 2. Inspecting `Light` tuple centre

face to the `Room` class via the *Light* coordination medium is enhanced to provide the new history service.

At the implementation stage, the History Analyser is mapped onto a TuCSoN mobile agent, and the new history services are provided by extending the previous behaviour of the `Light` tuple centre with new coordination laws. What is interesting here is that the new interaction protocols and the new room services are provided by extending the social rules embedded in `Light` incrementally, at run-time, adapting the behaviour of the agent social infrastructure dynamically.

VI. Benefits, Limits, and Comparisons

First, let us consider the benefits of tuple centres as objective coordination models. One first advantage is that the tuple centre model guarantees the consistence and the enforcement of the coordination laws representing light policies: as local interaction spaces, tuple centres enable the overall system consistence to be maintained despite the inherent openness and distribution of the environment.

Moreover, the social rules (light policies) encapsulated as coordination laws within tuple centres can be changed dynamically, via the Inspector and agents, in a seamless way: synchronisation issues involved in the run-time change of policies are handled automatically, thanks to the properties of the tuple centre model. Also, the dynamic change of the light policies is performed without affecting visitor agents, which can be completely unaware of that – a crucial point in the engineering of open environments.

So, tuple centres, as first-class abstractions where social rules can be dynamically specified and enforced, provide direct support for the incremental design and development of agent societies: as shown in the Deployment subsection of Section IV, the behaviour of a tuple centre can not only be inspected, but also enhancedby adding new coordination activities without necessarily knowing the existing ones.

Inspectability of tuple centres, along with the logic nature of the communication and of the ReSpecT languages, enables sound (automated) reasoning about the interactions occurring in a tuple centre and the involved coordination activities. For instance, the history service discusses in the case study requires the observation of agent satisfaction about light evolution, inspecting and reasoning about interactions, and finally adapting the environment (and the coordination policies – not shown in this paper) accordingly.

For comparison purposes, let us consider how the same problems could be modelled and engineered with a subjective coordination model. In that context, two kinds of solutions are usually adopted: either to use special agents as coordinators or mediators of light preferences, thus encapsulating the coordination policies into specialised agents; or to spread coordination policy management among agents – for instance, using protocols like contract net, where managers are light source agents and contractors are visitor agents.

The first case involves the creation of a specific coordinator agent – a typical situation in multiagent systems where coordination media are not modelled as first class abstractions. Typically, such a coordinator agent has a reactive, complex behaviour, since it has to manage all the many dependencies among the open set of agents, reacting to agents interactions, facing coordination activities at different levels, including concurrency and synchronisation issues, resource allocation, etc.

In this approach, there is no explicit coordination model, and every coordination aspect is faced at a mechanism (implementation) level: as a consequence, no engineering and structured reuse of coordination activities is possible. The description of the coordination activity has to be extracted from the agent source code, mangled with its computational behaviour in possibly any language: of course, this makes it quite hard to build a reusable framework for inspection, formal specification and verification of coordination activities. Moreover, adapting or changing coordination activities dynamically is very difficult: typically, it is necessary to shutdown the coordinator agent, change its behaviour and then spawn it again – which is unfeasible in an open, always-on environment. Alternatively, one can build a general purpose coordinator agent, able to accept a new coordination specification dynamically: yet, doing so means to admit precisely the need of considering the problem at a more abstract level – which opens the way towards the definition of a coordination medium.

Inspecting and changing dynamically the coordination activities is an even-worse problem in the second approach, where coordination activities are completely distributed among the open set of involved agents. In this case, maintaining the conceptual unity of coordination, and making changes while keeping the global consistence of the social tasks intact is a really hard task. Therefore, here agents must have the skills not only to use, but also to build the coordination process execution. This characteristic implies some drawbacks: the society is less open (because of the requirements on the agent skills), and the agent autonomy is reduced since they cannot pursue only their specific tasks, but have to participate to other activities, which are in some sense outside their actual goals.

With respect to performance, coordination infrastructures in general can be of help to improve system performance, in that they provide explicit structures to organise interactions rationally and to adapt them according to the dynamism of the environment. Here we have shown how an objective coordination model such as tuple centre, by providing first-class coordination abstractions that are not specific agents but part of the infrastructure, allows to enforce coordination while minimising negotiation, contracting stages and interactions in general, thus making the coordination process more fluid and automated.

VII. Conclusions and Future Work

The paper has shown the positive impact of using a coordination model and infrastructure to support the engineering of complex applications in the Network-Centric computing era. In particular, the SODA methodology and the TuCSoN coordination infrastructure have been used to engineer a simple case study about the intelligent management of lights in a building, taken as an example of Smart Environment application. We illustrated the benefits of carrying out the analysis, design and development of systems explicitly at two distinct levels – the agent level and the society level –, focusing in particular on the last one.

Future work will be devoted to the deployment of the TuCSoN technology in more complete and demanding applications in the Intelligent Environment context.

References

[1] Mark Weiser, "Hot topic: Ubiquitous computing", *IEEE Computer*, vol. 26, no. 10, pp. 71–72, Oct. 1993.

[2] David Tennenhouse, "Proactive computing", *Communications of the ACM*, vol. 43, no. 5, May 2000.

[3] Jim Waldo, "The Jini architecture for network-centric computing", *Communications of the ACM*, vol. 42, no. 7, pp. 76–82, July 1999.

[4] Victor Lesser, Micheal Atighetchi, Brett Benyo, Bryan Horling, Anita Raja, Regis Vincent, Thomas Wagner, Ping Xuan, and Shelley Zhang, "A multi-agent system for intelligent environment control", in *Proceedings of the Third International Conference on Autonomous Agents*. 1999, ACM Press.

[5] Michael Huhns, "Networking embedded agents", *IEEE Internet Computing*, vol. 3, no. 1, pp. 91–93, Jan. 1999.

[6] Microsoft, "Intelligent environments resource page", `http://research.microsoft.com/ierp/`.

[7] Nicholas R. Jennings and Micheal Wooldridge, "Agent-oriented software engineering", in *Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World : Multi-Agent System Engineering (MAAMAW-99)*, Francisco J. Garijo, , and Magnus Boman, Eds. 30– 2 1999, vol. 1647, pp. 1–7, Springer-Verlag: Heidelberg, Germany.

[8] Martin Griss and Gilda Pour, "Accelerating development with agent components", *IEEE Computer*, vol. 34, no. 5, pp. 37–43, May 2001.

[9] Michael J. Wooldridge and Nicholas R. Jennings, "Intelligent agents: Theory and practice", *The Knowledge Engineering Review*, vol. 10, no. 2, pp. 115–152, 1995.

[10] Nicholas R. Jennings, "On agent based software engineering", *Artificial Intelligence*, vol. 117, no. 2, pp. 277–296, 2000.

[11] Micheal Wooldridge and Paolo Ciancarini, "Agent-oriented software engineering: The state of the art", in *Handbook of Software Engineering and Knowledge Engineering*. 2001, World Scientific Publishing.

[12] Paolo Ciancarini, Andrea Omicini, and Franco Zambonelli, "Multiagent system engineering: the coordination viewpoint", in *Intelligent Agents VI — Agent Theories, Architectures, and Languages*, Nicholas R. Jennings and Yves Lespérance, Eds. February 2000, vol. 1767 of *LNAI*, pp. 250–259, Springer-Verlag.

[13] Munindar Singh, "Agent communication languages: Rethinking the principles", *IEEE Computer*, vol. 31, no. 12, 1998.

[14] Fumio Hattori, Ohguro Takeshi, Makoto Yokoo, Shigeo Matsubara, and Sen Yoshida, "Socialware: Multiagent systems for supporting network communities", *Communication of the ACM*, vol. 42, no. 3, 1999.

[15] Joav Shoham and Moshe Tennenholtz, "Social laws for artificial agent societies: Off-line design", *Artificial Intelligence*, vol. 73, 1995.

[16] Andrea Omicini, "SODA: Societies and infrastructures in the analysis and design of agent-based systems", in *1st International Workshop "Agent-Oriented Software Engineering" (AOSE 2000)*, Paolo Ciancarini and Michael J. Wooldridge, Eds., Limerick (Ireland), 10 June 2000, pp. 84–93.

[17] Rune Gustavsson, "Agents with power", *Communications of the ACM*, vol. 42, no. 3, pp. 41–47, Mar. 1999.

[18] Victor Lesser, "Reflections on the nature of multi-agent coordination and its implications for an agent architecture", *Autonomous Agents and Multi-Agent Systems*, vol. 1, pp. 89–111, 1998.

[19] Nelson Minar, Matthew Gray, Oliver ROUP, Raffi Krikorian, and Pattie Maes, "Hive: Distributed agents for networking things", in *Proceedings of the First International Symposium on Agent Systems and Applications and Third International Symposium on Mobile Agents ASA/MA*, 1999.

[20] Andrea Omicini and Enrico Denti, "From tuple spaces to tuple centres", *Science of Computer Programming*, vol. 40, no. 2, July 2001.

[21] David Gelernter, "Generative communication in Linda", *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 1, pp. 80–112, 1985.

[22] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli, "Mobile-agent coordination models for internet applications", *IEEE Computer*, vol. 33, no. 2, pp. 82–89, Feb. 2000.

[23] Andrea Omicini and Franco Zambonelli, "Coordination for Internet application development", *Autonomous Agents and Multi-Agent Systems*, vol. 2, no. 3, pp. 251–269, Sept. 1999, Special Issue: Coordination Mechanisms for Web Agents.

[24] Micheal Huhns and Munindar Singh, "Social abstractions for information agents", in *Intelligent Information Agents: Agent-Based Information Discovery and Management on the Internet*, Matthias Klusch, Ed. 1999, Springer-Verlag.

[25] Michael Schumacher, *Objective Coordination in Multi-Agent System Engineering – Design and Implementation*, vol. 2039 of *LNAI*, Springer-Verlag, Apr. 2001.

[26] "TuCSoN home page", `http://lia.deis.unibo.it/tucson/`.

[27] Enrico Denti, Andrea Omicini, and Alessandro Ricci, "tuProlog: A lightweight Prolog for Internet applications and infrastructures", in *Practical Aspects of Declarative Languages*, I.V. Ramakrishnan, Ed. 2001, vol. 1990 of *LNCS*, pp. 184–198, Springer-Verlag, 3rd International Symposium (PADL 2001), Las Vegas (NV), 11–12 Mar. 2001, Proceedings.