Agent Oriented AmI Engineering

Raian Ali[1] Sameh Abdel-Naby[1] Antonio Maña[2] Antonio Muñoz[2] and Paolo Giorgini[1]

[1] University of Trento - DIT, 38100 Trento, Italy.
[2] University of Malaga - E.T.S.I.Informatica, 29071 Malaga, Spain.
{raian.ali, sameh, paolo.giorgini}@dit.unitn.it, amg@lcc.uma.es

Abstract. Ambient Intelligence (AmI) refers to an environment that is sensitive, responsive, interconnected, contextualized, transparent, intelligent, and acting on behalf of humans. This environment is coupled with ubiquity of computing devices that enables it to transparently sense context changes, to react accordingly, and even to take the initiative towards fulfilling human needs. Security, privacy, and trust challenges are amplified with AmI computing model and need to be carefully engineered. From software engineering perspective, the shift towards AmI can be seen abstractly similar to the shift from object paradigm towards agent one. Objects provide functionality to be exploited, while agents possess functionality and know how and when to use and offer it autonomously. Agent paradigm is suitable for implementing AmI considering AmI as an open complex system. Moreover, we argue that agent paradigm is equally useful for engineering all aspects of such systems from the early phases of software development life cycle.

1 Introduction

Notebooks, PDAs, and third generation cellular phones are now computing devices equipped with wireless connectivity features allowing them to access different data networks anytime anywhere. The evolution in size and capabilities of those computing devices, along with those in wireless communications have effectively enabled people to be always online. This increased mobility in its raw form is not more than going beyond the classical desktop into a portable one. People are still requested to deal with different computers, and to adapt themselves to them. The next step would be to relieve people of even being aware of computer existence [1]. Computing is going to be seeded in the environment as an integral part of it, instead of being a set of external entities, used explicitly by trained humans.

Many challenges are related to enhancing the uselfulness of the current advances in computing devices and communication ubiquity. One of them is that current software development methods were created mainly for what we can call request/response software. There is a lack of sufficient models, development experience and even of imagination about how the new software systems can exploit the new technology advances [2]. The expectations of new software is that it will support features like location and context awareness, personalization, adaptability, organic growth, mobility, and some other features that impose the need of more comprehensive software engineering methods and new innovative modeling languages [3].

AmI focuses on making our environment sensitive to our needs and responsive smartly to people and environment context changes [4]. Objects around us in office, home, club, and other daily life locations, are expected to play their roles autonomously on behalf of us humans. AmI implies the ability of environment to learn and adapt by time to people characters and profiles, so ambient intelligence is always growing in organic style together with humans. This ambient is intertwined with invisible computing, it aims to give people what they need transparently without they explicitly ask or even know. AmI will relieve humans of being busy of at least the most repetitive actions they might take during their daily life.

It is well known that agent paradigm is a promising paradigm for implementing complex open systems like e-commerce, air-traffic, enterprise resource planning, and so on [5]. The characteristics of these domains fit well to what agent and multi agent systems can do. Software Agent is a software element that realizes the concept of agency, and acts on behalf of people or other agents. Agent paradigm was firstly dealt with inside AI community. Recently, and after the long hard experience of artificial intelligence, researchers could find other areas to exploit fruitfully agent paradigm. Agent paradigm has received a special interest in software engineering community as a paradigm shift from the object oriented one [6][7]. The shift is based on seeing the world as a society of distributed intelligence units, called agents, that have characters and can decide. This way of viewing the world differentiates itself from the object oriented one that conceptually view the world as a collection of objects. Objects provide encapsulation of data together with the procedures related, they are used by main well defined central control, and do not have their own autonomy.

One of the challenges that face building an AmI is the lack of models and software engineering practice that help analysing system requirements, designing the system to be built, verifying and testing the implemented one. Until now the research is in its first stages, and the need for suitable development methodologies has been already recognized. For engineering AmI, we might need different software engineering methods from those that are suitable for developing request/response systems, where system behavior is well known and determined strictly, and where human-computer interaction is desktop driven one. AmI shifts this way of interaction into contextual, direct, and invisible human environment interaction, hiding the computers in the background of this environment. The disappearance of computers and coupling environment appliances with computing devices will arise like any new technology a variety of challenges. The system domain is no longer some sort of business or organization has a clear business process and tasks. Users are no longer those clerks or students in a library system; instead users are now those normal people in houses, offices, campus and other daily life environment. The request/response scenario is replaced here by continuous sensitive, reactive, intelligence surrounding computing.

We believe that agent paradigm is not only useful for implementing AmI systems, but rather we see it appropriate in all phases of the software development life cycle. As in object-oriented and component-oriented worlds, AmI ecosystems are composed of independent pieces of software with well defined interfaces, but the main difference is that in AmI each of these pieces has a different owner and has its own goals. This is another fundamental aspect that reinforces the appropriateness of agent oriented approaches for AmI. We are aware that current agent oriented software engineering methodologies, which are still inside the academic areas, have to be checked again for engineering AmI. If we succeed to analyze and design such systems by the use of agent driven software engineering, we might come up with final agent based system that is robust, scalable, and intelligent enough to satisfy AmI needs.

The remainder of this paper is structured as follows; next section shows AmI as multidisciplinary complex system. Section 3 outlines the agent paradigm. Section 4 introduces the agent oriented software engineering research. Section 5 discusses the possibility of exploiting agent paradigm for engineering AmI, for this purpose in subsection 5.1 we address the the potential agent paradigm has with regards to AmI systems engineering, and in the last but not least subsection we focus on how agent paradigm can be exploited to face security challenges in AmI ecosystems. In Section 5 we conclude.

2 The Multidisciplinary AmI

Approaching an ambient that is perceptive, intelligent, and active will involve multiple disciplines to contribute creating the final scene. Several researches are being done in AmI area, with some differences in emphasis and direction. Multiple terminologies are being used as this research is in its first steps. In the rest of this section, we will investigate the vision of AmI, and try to capture a variety of disciplines that need to meet in order to achieve this vision.

Philips vision of AmI [8] is based on shifting computers into the background, and supporting the ubiquitous computing with more awareness capabilities. The vision is based on three elements, 1) the ubiquity, which refers to those computing devices intertwined with human environment anywhere, and functioning anytime, 2) the transparency of such computing systems, so they are hidden in the background, 3) and the intelligence; they should act instead of being only responsive to human commands. Such system relieves people of thinking about many repetitive needs and takes the initiative of doing what should be done in the correct moment and approach.

MIT vision of AmI [9] similarly views it as an unobtrusive integration of computing with our daily life. Such computing provides humans with relevant information and performs necessary tasks when needed on their behalf. Such ambient will be continuously careful, doing the suitable tasks in a transparent, invisible and intelligent way. Traditionally, computers work as an apparent messenger or mediator between humans and environment. In AmI, this relation is replaced by direct non-disruptive relation between humans and the environment they are located within. In short, AmI computing is no longer visible.

The vision of invisible disappearing computers was addressed by Weiser [10]. The vision expected ubiquitous existence of computing and communication capabilities anytime and anywhere. AmI focuses on assisting the intelligence and awareness of this ubiquity of interconnected computing devices, so computing starts to take the initiative on behalf of human. AmI is meant to orchestrate the variety of environment objects in a way they might interoperate to do more complex tasks as well. Ubiquity of computing is the basis an AmI is built on. However, the terms ubiquitous computing, pervasive computing, ambient computing, ambient intelligence are now used interchangeably with some differences in the context and emphasis.

AmI is now about integrating computing devices with the environment we all live in; it is then sitting on the opposite side of virtual reality which brings world inside computers [10]. This makes computers invisible and relieves people mind of even knowing about their existence. To arrive this point, computers has to adapt to user needs and character by contrast of the traditional scene in which user is supposed to adapt to computer systems. This is now of great importance because people spend increasingly more time to interact with computing systems. To people, it is becoming a source of stress being obligated to remember when and what and how to do tasks. With AmI, artefacts encapsulate implicitly the role of computer mediation. Artefacts will look as they have their own character, autonomy, and intelligence, they are more agents than normal objects.

Consequently, AmI is by nature a multidisciplinary paradigm [11]. Distributed intelligence is needed to cover this intelligent ambient, it is now composed of distributed intelligence units that we might call Agents. New hardware design is needed for embedding computing devices invisibly inside the surrounding physical environment. AmI system is situated within a highly dynamic environment that is open for changes, these changes need to be sensed and interpreted in a way that is timely fashion and relevant to what might serve user needs. The input now is coming implicitly, and continuously from a variety of sensors, cameras, and other kind of peripherals. Such environmental information need to be modelled and reasoned about in order to take the correct contextual decision.

Computer disappearance was considered by Weiser as one of the most profound technology features [10]. Apart from the physical disappearance of computing devices, there is that mental disappearance toward peace of mind in human life. To achieve such peace of mind, the interaction between human and computer is updated to direct interaction between human and environment [1]. New novel ideas of interaction design have to be invented to move from the explicit interaction to an implicit one [12]. The implicit interaction includes the notion of implicit input known more commonly as Context [13].

Context awareness [14][15] is an essential feature an AmI system has to tackle in order to act in adaptive and intelligent way. This context, that might be spatio-temporal, environmental, personal, social, and so on, needs to be modeled, captured, analysed and reasoned about [2]. Reasoning about context needs a model and formalization acts as a knowledge base, and enables inferring more high level knowledge. For example blood pressure and body temperature besides user current activity and location might reveal user current mood, this mood can be provided implicitly as an input, so AmI might take some actions as a response.

AmI is expected also to have the ability of learning and keeping track of human historical behaviour. AmI embodies a high degree of personalization to human profiles and life styles. Software personalization is a standalone research now, but we might hardly consider AmI as a useful system if it behaves in the same way with different kind of people and characters. The social mobility of humans is another important issue an AmI application needs to consider. People normally play more than one social role, they should be accordingly supplied by tailored services and information considering their social context [16].

AmI arises many social issues that need to be studied and analysed before AmI can get acceptance in practice. The ubiquity of computing might relieve people mind in one hand and might have negative impacts as well. People will feel that they lost control, and might not trust technology. People have already lost some privacy providing that cellular phones enable other party of at least knowing their location, and the same for using credit cards. Instead of commanding computing, computing in AmI is supposed to control several aspects of people everyday life. An essential principle in this regard is that human do not feel that they lost control, and to enable them configuring their needs in a simple way, may be through some privacy patterns. However, we see many interesting practical domains that can benefit from AmI scenarios, such as the health care domain, in particular those specialized of caring old people, and supporting persons with dementia problems, where AmI might play the role of caregiver.

3 Agent Paradigm

Agent-based computing is currently becoming an important research area. This increased interest is motivated by the need for software can act on behalf of its user, software that is able to realize the concept of agency. Giving a definition for agent is not straightforward; there is no consensus about the main characteristics an agent should have to deserve this name. A well accepted definition of software agent is found in [6]:

"An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives."

An agent is supposed to have its own control over its state and behavior, to percept the environment around and to affect it in turn. Being in an environment and sensing it implies the necessity that agent can react to environmental context changes. Moreover, agent is supposed to activate goals without external prompt and to tailor suitable plans to achieve them. The key characteristics an agent must have that are highly agreed upon include: autonomy, proactiveity, reactivity, situatedness, directedness, and social ability.

Being autonomous, an agent behaves independently according to the state it encapsulates. For example, an agent, by contrast to an object, can decide the way of how to respond to the incoming messages from other agents. Agents interact with each other without losing control if they do not allow that. Proactivity means that agent is able to take the initiative without external order. Agents have goals and act in order to achieve them. This is more complicated than reacting in timely fashion to direct environment stimulus. Situatedness means the ability of agent to settle in an environment that might contain other agents, to perceive it, and to respond to changes that happen in it. An agent might make changes and effect this environment in turn. Directedness means that agent has a goal, this goal represents the reason of the actions an agent has to take. An agent does not exist in vacuum; instead it lives in a society of other collaborative or possibly competitive groups of agents. Agents have the social ability to interact with other agents. This interaction might be motivated by collaborative problem solving.

A long discussion can be found in the literature about what formulates an agent and what differentiates it from object. We are here not concerned about such discussion, rather we believe that using agent as a kind of abstraction might enable us of viewing the world as an organization of autonomous entities, directed by goals, able to sense the environment changes and can learn by time. The use of agent paradigm as kind of abstraction might better help of analyzing and designing complex open systems, and presents more natural way to start with, and hopefully this will lead to more robust and flexible software systems in correspondence.

An agent is supposed to live in a society of agents; multi-agent system (MAS) is known as a system composed of several agents collectively capable of reaching goals that are difficult to achieve by an individual agent or monolithic system. The relation can be alternatively competitive one, like for example multiple agents responsible for advertising products in an open market on behalf of different producers, or a society of agents in an e-auction. Again, defining MAS is not that straightforward. MAS might help us decomposing the problem into components that are able to interact and deal with unpredictable situations that can happen in complex systems like AmI.

A MAS represents a natural way of decentralization, where there are autonomous agents working as peers, or in teams, with their own behavior and control. Each of these agents looks to the world from its own perspectives and has its own goals and intentions. Such MAS is expected to work well with open complex systems, and to scale well by time. It is one promising computing paradigm for implementing many application domains such as e-commerce, enterprise resource planning, and traffic control, and so on [5]. We consider AmI as a system that fits by its nature to agent and multi-agent system paradigm as we are going to discuss later.

4 Agent Oriented Software Engineering (AOSE)

Software engineering is different from other engineering disciplines in its dependability on engineer skills of analysing the problem, designing a suitable solution, and coming up with the final system [17]. Although software engineering is qualitative in nature, a serious research is being done to find more and more scientific methods, models, and criteria that assist developing the intended software. Problems are everywhere in software development process, engineering a software is an engineering for abstraction. For example, understanding precisely what a software is supposed to do and transforming this knowledge into abstract models readable by both of engineers and stakeholders is far of being easy as it seems.

Many large industrial projects failed because the final software was not the one needed or expected. The models used to describe software requirements and design need to be compact and expressive enough to replace usefully the natural language. The models need therefore to be precise enough to not lose the real concepts they are supposed to represent. The models might be formal or transformable into formal ones, so reasoning can be done over them with the purpose of discovering any anomalies, incompleteness, or inconsistencies. Software engineering methodology is concerned not only about inventing and using modelling languages that can express what the system has to fulfil, and the software design, but rather it has to provide a process model for creating such models in turn.

Software agent that persistently observes the environment, interprets it, acts, and might communicate with other agents is a promising computing paradigm for implementing open complex systems. AOSE methodologies tend to analyse and design such kinds of complex systems in order to arrive finally to an agent-based implementation. There are several research groups working in developing their own AOSE methodologies [7]. The orientation towards agent does not mean that these methodologies use agency concepts and agent mentalistic notions along with all phases of developing software, rather the goal is to analyze and design in a way that leads to multi agent system. Only Tropos [18], as an AOSE methodology, uses the notions of agent and the related mentalistic notions from the early analysis down to the actual implementation.

As the use of computing is becoming an essential part of individuals' daily life together with business and organizations, and as we increasingly need to combine between different computing ends and parties, the need for software that is dynamic, flexible, adaptable, situated is more critical. The need for software evolution is becoming faster than software development process itself. Solving these challenges is based to a large extent on the way such software has to be engineered. Agent oriented software engineering is trying to arrive methods that enable developing a software can resist against evolving requirements, a software that is flexible enough to adapt and change fluently according to the new environments and requirements.

Fortunately, agent oriented software engineering, by contrast to object oriented software engineering and structured analysis and design, is not restricted or deeply influenced by some existing programming paradigm [19][20]. Agent oriented software engineering research is now taking the initiative towards programming languages and infrastructures that serve the concepts suitable for software development instead of using those of existing programming languages in reverse unnatural way. Being limited to programming languages has enforced those previous software engineering practices to focus on the solution domain, since the concepts used are not those describing naturally requirements and problem domain. Agent oriented software engineering is growing together with agent oriented programming and agent infrastructure, this might fill the gap between problem and solution domains. Hopefully such consistency will make software development process faster, and lead to software can be easily evolved and maintained, and can adapt to different environments and requirements.

5 Exploiting agent paradigm for engineering AmI

Agent paradigm fits well for implementing AmI scenarios due to the coincidence between agent characteristics and AmI needs. Agent paradigm as a kind of abstraction is also capable of giving a good contribution with regards to AmI systems development, including analysis and design phases, besides the security issues. In this section we will state our initial view of the agent oriented AmI engineering and securing.

5.1 Agent-oriented AmI development

As we explained previously, AmI shows a degree of complexity and multiple inter-related disciplines that require using special engineering paradigm. This need is coming from the new nature of such systems, where behavior is not known in details, or adequately controllable. AmI is distinguished by its dynamicity, openness, and complex inter-relations amongst environment components. Compared with object oriented software engineering practice, agent paradigm offers a higher level of abstraction suitable for engineering complex systems [21]. Agent paradigm enables engineering software at the knowledge level; at this level we talk of mental states, of beliefs instead of machine states, of plans and actions instead of programs, of communication, negotiation and social ability instead of direct interaction and I/O functionalities, of goals, desires, and so on [22].

Tackling the complexity of developing complex software can be done through some techniques such as 1) Decomposing the problem into smaller sub-problems that can be managed more easily. 2) Using abstract models to represent system focusing on some concepts and relations, and omitting others unrelated. Such models should be compact and expressive in order to usefully summarize and even formalize what can be alternatively expressed by the natural languages. 3) Defining and managing the inter-relationships between problem solving components as they were an organization of some hierarchy [23].

As shown in [17], agent paradigm is not only useful as software construct but rather it can be used as a new way for analyzing and designing complex systems. Using the decomposition, abstraction and organization techniques to tackle the complexity of such systems can be done following agent paradigm from the early phases. Decomposing complex systems into related subsystems, each with its own thread of control, and own objectives to be achieved autonomously can be seen as a society of interacting agents. Agent paradigm provides a sort of abstraction to model problem domain in terms that are too constituent with solution domain. Subsystems are viewed as autonomous agents, agent social ability implies the interrelation at high level amongst those autonomous subsystems. This interaction might model cooperation, coordination, or negotiation amongst agents. The evolution of inter-relations between components of complex systems and the different aggregation these components can be classified at different levels of abstraction match closely to agent and multi-agent system paradigm. As for the dynamic organization structure, agent paradigm has the expressivity to represent these concepts due to its explicit structure and flexible mechanisms. A methodology called Gaia [24] was developed to reflect such ideas providing a methodological way for engineering some kinds of complex systems.

Another attempt for using agent paradigm as conceptualization construct is based on BDI agent architecture, the world is viewed as a society of actors each has its own autonomy, and might depend on each others for task to be performed, goal to be achieved or resource to be provided [21]. Agent beliefs are the world model at the conceptual level, agent desires are translated into goals to be achieved, while the intention an agent might commit is considered as a plan. The multiple plans an agent might follow to achieve the same goal give some degree of flexibility for dealing with different contexts. Goals are analyzed through means-end analysis to conclude the actual actions by which goals are achieved. These actions are the actual requirements of the intended final software [25]. Tropos is another methodology was developed on the basis of these ideas, it uses agent mentalistic notion along all the phases of software development [18].

For engineering AmI, like for example smart campus, we need to decompose it into autonomous subsystems, and to abstract using knowledge level conceptualization rather than the fine grained one used by OO which is useful for predicted behavior and relatively static systems. With AmI we are not talking about an organization with one well defined behavior, business process, and straight control. Here the ambient is always changing and in an unpredictable way sometimes, so we need high degree of adaptability to cope with AmI going to serve everyday life scenarios with a lot of alternatives. Considering AmI as complex open system, we believe that agent paradigm and agent mentalisitc notions can contribute well for analyzing, and designing AmI scenarios rather than only implementing them.

5.2 Securing AmI ecosystems

We have shown that Agent-systems can bring important benefits especially in application scenarios where highly distributed, autonomous, intelligente, self organizing and robust systems are required. Furthermore, the high levels of autonomy and self-organization of agent systems provide excellent support for the development of systems in which dependability is essential. Both Ubiquitous Computing and Ambient Intelligence scenarios belong to this category. However, despite the attention given to this field by research community the agent technology has failed to gain a wide acceptance and has been applied only in a few specific real world scenarios. Security issues play an important role in the development of multi-agent systems and are considered to be one of the main issues to solve before agent technology is ready to be widely used outside the research community. However, we will show in this section that solutions are available for most of these problems. Furthermore, very promising technologies are currently under development (in some cases in a quite advanced phase) for the remaining problems. Consequently, our view is that the main reason why agent-oriented approaches have not gained wider acceptance is the lack of appropriate application scenarios. Precisely, AmI ecosystems are perfect scenarios for the application of agent approaches. With regards to security, agents present the most appropriate solution because they facilitate concealing disparate security requirements from different points in order to achieve each parts' goals in a collaborative setting. Of course, as mentioned above, we need to solve the most important security issues for general multi-agent systems.

Some of the general software protection mechanisms can be applied to the *protection of agents*. However, the specific characteristics of agents mandate the use of tailored solutions. First, agents are most frequently executed in potentially malicious pieces of software. Therefore, we can not simplify the problem as is done in other scenarios by assuming that some elements of the system can be trusted. Then, the security of an agent system can be defined in terms of many different properties such as confidentiality, non repudiation, etc. but it always depends on ensuring the correct execution of the agent on agent servers (a.k.a. agencies) within the context of the global environments provided by the servers [26].

Some protection mechanisms are oriented to the protection of the host system against malicious agents. More relevant approach is Sandboxing, a sandbox is a container that limits, or reduces, the level of access its agents have and provide mechanisms to control the interaction among them. Another technique, called proof-carrying code, [27]. For this purpose, every code fragment includes a detailed proof that can be used to determine wether the security policy of the host is satisfied by the agent. Therefore, hosts just need to verify that the proof is correct (i.e. it corresponds to the code) and that it is compatible with the local security policy.

Other mechanisms are oriented towards *protecting agents against malicious agencies*. Sanctuaries [29] are execution environments where a mobile agent can be securely executed. Most of these proposals are built with the assumption that the platform where the sanctuary is implemented is secure. Unfortunately, for agent-based systems this assumption is not applicable.

Several techniques can be applied to an agent in order to verify self-integrity in order to avoid that the code or the data of the agent is inadvertently manipulated. Anti-tamper techniques, such as encryption, checksumming, anti-debugging. anti-emulation and some others [30] [31] share the same goal, but they are also oriented towards the prevention of the analysis of the function that the agent implements.

Additionally, some protection schemes are based on self-modifying code and code obfuscation [32]. In agent systems, these techniques exploit the reduced execution time of the agent in each platform. Software watermarking techniques [33] are also interesting. In this case the purpose of protection is not to avoid the analysis or modification but to enable the detection of such modification. The relation between all these techniques is strong. In fact, it has been demonstrated that neither perfect obfuscation nor perfect watermark exists [34].

In summary, all these techniques provide short-term protection; therefore, in general they are not applicable for our purposes. However, in some scenarios, they can represent a suitable solution, especially, when combined with other approaches. Theoretic approaches to the problem have demonstrated that self-protection of the software is unfeasible [35].

In some scenarios, the protection required is limited to some parts of the software (code or data). In this way, the function performed by the software, or the data processed, must be hidden from the host where the software is running. Some of these techniques require an external offline processing step in order to obtain the desired results. Among these schemes, function hiding techniques allow the evaluation of encrypted functions [36]. This technique protects the data processed and the function performed. For this reason it is an appropriate technique for protecting agents. However, it can only be applied to the protection of polynomial functions.

The case of online collaboration schemes is also interesting. In these schemes, part of the functionality of the software is executed in one or more external computers. The security of this approach depends on the impossibility for each part to identify the function performed by the others. This approach is very appropriate for distributed computing architectures such as agent-based systems or grid computing, but has the important disadvantage of the impossibility of its application to off-line environments

Finally there are techniques that create a *two-way protection*. Some of these are *hardware-based*, such as the Trusted Computing Platform. With the recent appearance of ubiquitous computing, the need for a secure platform has become more evident. Therefore, this approach adds a trusted component to the computing platform, usually built-in hardware used to create a foundation of trust for software processes [37]. Other techniques are software-based, for instance Protected Computing [38] approach. Protected Computing approach is based on the partitioning of the software elements into two or more dependent parts, then a part of this code will be remotely executed in a different agent.

6 Conclusions

AmI implies a shift from appliances that provide some functionality and can be utilized by external entity, towards appliances that have their own autonomy and know how to behave on behalf of humans without explicit request.

Abstractly speaking, this shift can be seen similar to the shift from object oriented towards agent oriented software paradigm. When we described AmI as a multidisciplinary paradigm, we discovered the similarity between AmI needs and Agent paradigm primitives. In AmI, each ambient appliance will behave like an agent that has character and can decide. Each appliance needs to be autonomous, reacting to environment changes, and taking the initiative towards fulfilling human needs in the correct moment and way. Appliances also need to settle down within an open environment of other appliances and need to communicate with them, so it needs some kind social ability.

Moreover, we consider AmI, that is intended to serve unpredictable everyday life scenarios and includes a spread of interacting appliances, as an open complex system that needs to be engineered using more advanced techniques than those tailored for well defined systems. We believe that agent paradigm is promising for developing AmI systems during all the development life cycle phases and not only for implementing them. Agent oriented software engineering methodologies have a good potential with respect to AmI; so the next step could be adapting some existing methodologies, or creating a new one, in order to engineer AmI at all phases of development life cycle from requirement gathering until the final implementation.

References

1. N. Streitz and P. Nixon. The Disappearing Computer. Communications of The ACM, March 2005/Vil. 48, No.3.
2. Krogstie, J., et al., Research Areas and Challenges for Mobile Information Systems. International Journal of Mobile Communication, 2004. 2(3).
3. Krogstie, J. Requirements Engineering for Mobile Information Systems. In Proceedings of the Seventh International Workshop on Requirements Engineering: Foundations for Software Quality (REFSQ'01). 2001. Interlaken, Switzerland.
4. Eu Project Report. ISTAG Scenarios for Ambient Intelligence 2010. ftp://ftp.cordis.lu/pub/ist/docs/istagscenarios2010.pdf
5. M. Wooldridge and N. R. Jeanings, Intelligent agents: Theory and practice, Knowl. Eng. Rev., vol. 10, no. 2, 1995.
6. M. Wooldridge. Agent-based Software Engineering. In IEE Proceedings on Software Engineering, 144(1), pages 26–37, February 1997.
7. Paolo Giorgini and B. Henderson-Sellers (Eds.) Agent-Oriented Methodologies, Idea Group Inc., 2005
8. Philips Research. Ambient Intelligence Research in ExperienceLab. http://www.research.philips.com/technologies/syst_softw/ami/
9. MIT Ambient Intelligence Research Group. http://ambient.media.mit.edu/
10. Mark Weiser. The Computer for the Twenty-First Century. Scientific American, pp. 94-10, September 1991.
11. Remagnino, P. and Foresti, G.L. Ambient Intelligence: A New Multidisciplinary Paradigm. IEEE Transactions on Systems, Man and Cybernetics, Part A, Volume 35, Issue 1, Jan. 2005 Page(s):1 - 6.
12. A. Schmidt. Implicit Human Computer Interaction Through Context. Personal Technologies Volume 4(2&3), June 2000. pp191-199.
13. A. Schmidt, K.A. Aidoo, A. Takaluoma, U. Tuomela, K. Van Laerhoven, W. Van de Velde. Advanced Interaction in Context. The International Symposium on Handheld and Ubiquitous Computing (HUC99), Karlsruhe, Germany, 1999 & Lecture notes in computer science; Vol 1707, ISBN 3-540-66550-1; Springer, 1999, pp 89-101.

14. Anind K. Dey and Gregory D. Abowd. Towards a Better Understanding of Context and Context-Awareness. In the Proceedings of the CHI 2000 Workshop on The What, Who, Where, When, and How of Context-Awareness, The Hague, Netherlands, April 1-6, 2000.
15. Jolle Coutaz , James L. Crowley , Simon Dobson , David Garlan, Context is key, Communications of the ACM, v.48 n.3, March 2005
16. K. Lyytinen , Y. Yoo The Next Wave of Nomadic Computing: A Research Agenda for Information Systems Research. Sprouts: Working Papers on Information Environments, Systems and Organizations. Vol. 1, Issue 1, Article 1 - 2001.
17. N.R. Jennings. On Agent-Oriented Software Engineering. Artificial Intelligence 117 (2) 277-296 (2000).
18. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, A. Perini. TROPOS: An Agent-Oriented Software Development Methodology. Journal of Autonomous Agents and Multi-Agent Systems. Kluwer Academic Publishers Volume 8, Issue 3, Pages 203 - 236, May 2004.
19. J. Mylopoulos, L. Chung, and E. Yu. From Object-Oriented to Goal-Oriented Requirements Analysis. Communications of the ACM, 42(1):3137, Jan. 1999.
20. J. Mylopoulos, Information modeling in the time of the revolution, Information Systems, v.23 n.3-4, p.127-155, May 1, 1998
21. E. Yu. Agent Orientation as a Modelling Paradigm. Wirtschaftsinformatik. 43(2) April 2001. pp. 123-132.
22. A. Perini, P. Bresciani, F. Giunchiglia, P. Giorgini, J. Mylopoulos. A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. In the Proceedings of the Fifth International Conference on Autonomous Agents, Montreal, Canada - May 29 - June 01, 2001.
23. G.Booch "Object-Oriented analysis and design with applications" Addison Wesley (1994)
24. M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. In Journal of Autonomous Agents and Multi-Agent Systems. 3(3):285-312. 2000.
25. A. Dardenne, A. van Lamsweerde and S. Fickas. Goal-Directed Requirements Acquisition. Science of Computer Programming Vol. 20, North Holland, 1993, pp. 3-50.
26. S. Berkovits, J. Guttman, V. Swarup. Authentication for Mobile Agents. Mobile Agents and Security. Springer-Verlag Publishers Volume 1419, 1998, pp 114-136.
27. G. Necula G. Proof-Carrying Code. Proceedings of 24th Annual Symposium on Principles of Programming Languages. 1997.
28. A. Gunter Carl, P. Homeier, S. Nettles. Infrastructure for Proof-Referencing Code. Proceedings of the Workshop on Foundations of Secure Mobile Code. March 1997.
29. Yee, Bennet S. A Sanctuary for Mobile Agents. Secure Internet Programming. 1999.
30. I. Schaumller-Bichl1, E. Piller. A Method of Software Protection Based on the Use of Smart Cards and Cryptographic Techniques. Proceedings of Eurocrypt. Springer-Verlag. LNCS 0209, pp. 446-454. 1984.
31. J.P. Stern, G. Hachez, F. Koeune, J.J. Quisquater. Robust Object Watermarking: Application to Code. Proceedings of Info Hiding, Springer-Verlag. LNCS 1768, pp. 368-378. 1999.
32. C.Collberg, C. Thomborson. Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection. University of Auckland Technical Report 170. 2000.
33. P. Wayner. Dissapearing Cryptography. Information Hiding, Stenography and Watermarking. Morgan Kauffman. 2002.

34. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, K. Yang. On the (Im)possibility of Obfuscating Programs. Proceedings of CRYPTO. Springer-Verlag. LNCS 2139. pp. 1-18. 2001.
35. O. Goldreich. Towards a theory of software protection. Proceedings of the 19th Ann. ACM Symposium on Theory of Computing, pp. 182-194. 1987.
36. T. Sander, C.F. Tschudin. On Software Protection via Function Hiding. Proceedings of Information Hiding. Springer-Verlag. LNCS 1525. pp 111-123. 1998.
37. S. Pearson, B. Balacheff, L. Chen, D. Plaquin, G. Proudler. Trusted Computer Platforms. Prentice Hall. 2003.
38. A. Maña, A. Muñoz Mutual Protection for Multiagent Systems. Proceedings of the Third International 3rd International Workshop on Safety and Security in Multiagent Systems. 2006.