Is there meta-level in smart spaces?

Ekaterina Gilman and Jukka Riekki
Dept. of Computer Science and Engineering and Infotech Oulu
P.O.Box 4500, University of Oulu, 90014, Oulu, Finland
firstname.secondname@ee.oulu.fi

Abstract— **Smart spaces are highly dynamic interactive environments supporting people in their activities. Management of such environments is a very challenging task due to the volatile nature of the interactions happening among resources and people acting in these environments. In this article, we study whether meta-level control of smart spaces helps to tackle this challenge. We suggest making a distinction between context-based adaptation activities of smart spaces and monitoring and controlling of these adaptation activities. We conduct simulation studies in order to validate the proposed concept.**

Keywords- smart space; management; reasoning; metareasoning

I. INTRODUCTION

Smart spaces are highly interactive physical environments equipped with computational facilities to support users in their daily activities. These spaces adapt their computational resources and services according to the situation, user needs and preferences. Due to dynamic and volatile nature of interaction between users and smart spaces, it is often not possible to predetermine the adaptation and service composition processes, especially for multi-user environments.

Management of smart spaces is challenging. In order to adequately support users, the following management issues should be tackled: 1) Management of computational facilities and infrastructure of the smart space, handling failures and equipment installations; 2) Management of services, their context-based composition and execution; 3) Management of users, their preferences, interaction with smart space, and explaining smart space behaviour to them. Management mechanisms are required in order to keep the smart space usable and useful.

Autonomic computing paradigm promises to tackle the management issues of large distributed systems. It relies on the idea that autonomic systems manage themselves with minimum human intervention. Generally, these systems are able to self-configure, self-optimize, self-heal and self-protect [1]. Exploring the properties of autonomic computing in the pervasive computing domain shapes the new autonomic pervasive computing vision, which aims to simplify the configuration, maintenance and management of pervasive environments such as smart spaces [2]. Self-configuration includes service configuration, deployment, integration and maintenance. Self-optimization refers to optimizing own characteristics and tuning them to suit better to users' needs. Self-healing refers to detecting and solving problems, like device failures. Self-protection addresses security issues for personal data and resources. These self-managing environments would automate most of the routine management tasks.

In this article, we suggest distinguishing the context-based adaptation activities of smart spaces from monitoring and controlling of these adaptation activities. This provides the benefits of the separation of concerns, like clear system design, customizability, and easier reuse [3], and allows achieving the self-optimization property of smart space. Smart spaces should monitor, analyze and control their behaviour and performance in order to maximize users' satisfaction – by configuring, composing, and executing services, while considering user preferences and feedback. Hence, we suggest treating the smart space as a feedback loop system, meaning that the smart space analyzes the feedback information (obtained from the users and from monitoring the different quantitative parameters) and controls itself to optimize the feedback values.

The contribution of our article includes a framework for meta-level control of smart spaces. First, we separate the tasks of context adaptation of smart spaces (object level) and adaptation of these adaptation tasks (meta-level). Second, we study how rule-based reasoning can be applied to achieve the meta-level of smart spaces. We conduct a simulation, exploring whether rule-based reasoning can support such separation of concerns in smart space.

The rest of the article is organized as follows. We study related work in section two, section three presents the meta-level framework for smart space, and section four describes the conducted experiment. We discuss findings and conclude the article in section five.

II. RELATED WORK

Plenty of research has been conducted for context-aware service composition, particularly focusing on describing services, their dependencies, algorithms and techniques to provide service compositions [4]. This research provides useful foundation for studying solutions supporting smart space management. For example, Abdulrazak and Helal [5] suggest a smart house in the box, which involves minimum engineering expertise in order to set up and configure. Their architecture enables device self-configuration in a smart

space, including the mechanisms of device description, discovery and installation.

Management of smart spaces (e.g., smart houses) is an extensively studied topic [6]. Jurmu et al. [7] suggest utilizing a leasing mechanism to manage smart space resources to bring resource usage awareness for other components of smart spaces. Feeney and Frisby [8] layout autonomic computing properties to smart spaces domain and implement self-healing property for autonomic smart space based on IBM's Autonomic toolkit. In comparison to their work, presented article is more related to self-optimization of smart space.

Work towards autonomic pervasive computing is presented by Ahmed et al. [9], studying self-healing property. Trumler et al. [10] suggests autonomic middleware (i.e. AMUN) for ubiquitous environments. Their middleware includes monitoring and control mechanisms to trigger the reconfigurations. Our proposal differs with the functional roles of these components.

Separation of concerns is a widely utilized technique [3]. Paspallis and Papadopoulos [11] suggest a framework for developing adaptive mobile applications by separating the work of application logic development from one enabling its adaptive behaviour. Our proposal goes one abstraction layer above this.

Meta-level provides great support for separation of concerns [12], [13]. Raja [14] studied how the meta-level control can improve agent interaction. We adapt some concepts from her work in the smart space domain. In our work, we apply meta-level principles [12] to smart space management in order to achieve its self-optimization property.

III. META-LEVEL FRAMEWORK FOR SMART SPACE

Traditionally, reasoning is understood as a decision making process within an Action-Perception loop (see. Fig. 1) [12]. This means that an agent decides based on perceived information what action to perform. This corresponds to the context adaptation activities of a smart space. The smart space perceives (Ground level, Fig. 1) its state, users, their activities and triggers corresponding service compositions, configuration and deployment to available resources to meet users' needs (Object level, Fig. 1).

We suggest placing the reasoning about adaptation activities the smart space performs to a separate level. This activity is called metareasoning (Meta-level, Fig. 1) [12], [13]. Metareasoning is analysis of how well the actions progress the tasks the services are performing and how well these tasks support users. In classical model, meta-level operates only with object level. However, to support the multi-user and dynamic nature of a smart space we modify this model slightly. Some ground level actions have to be perceived by meta-level in order to control the object level. The dotted line of Fig. 1 emphasizes the feedback loop of the smart space. This means that meta-level would be able to collect feedback to estimate and understand details of the

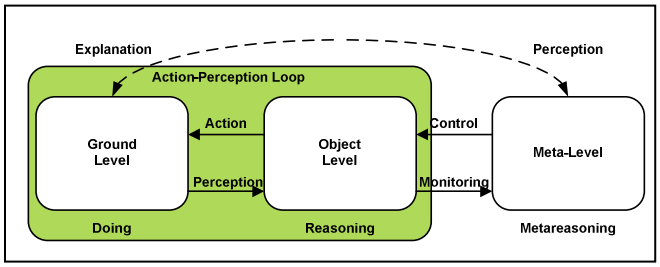

Figure 1. Metareasoning concept, modified from [12].

events at the ground level. Alternative strategies can be applied or even solutions can be created to achieve the goals in dynamic settings. Moreover, meta-level provides means to inform the users about system execution, for example, decision justification and failure explanation. Metareasoning consists of both the meta-level control of computational activities and the introspective monitoring of reasoning [12].

A. Smart Space Object-Level and Meta-Level Tasks

Let us layout the concept introduced above to the smart space functionality. At the object level, smart space handles the self-configuration property. The tasks are the following:

Service composition. Often, in order to complete a certain task, different services should be composed, considering environmental, computational and user context. This task includes the service discovery and building service composition plan.

Service deployment. This task is responsible for finding the suitable resources (devices) of smart space to execute a given service composition. Service deployment plan considers service requirements, resource capabilities, their availability, and user preferences.

Service execution. This task relates to executing the service deployment plan.

Meta-level is responsible for monitoring and controlling of the object-level tasks execution. In our case, meta-level monitors and controls smart space's self-configuration and tries to improve its efficiency, based on user satisfaction. Hence, meta-level is more responsible for the self-optimization property of smart space. The general meta-level tasks of smart space are the following:

Monitoring object-level task execution and gathering feedback information. This task is responsible for gathering quantitative and qualitative information of the object-level tasks in order to estimate whether these tasks progress well.

Controlling service composition and service deployment. These actions result in modifying the service composition or deployment strategy, e.g. because of poor system performance or due to user's feedback.

Controlling service execution. Meta-level can control the service execution, like interrupt or continue services.

B. Events Triggering Meta Level

Smart spaces are dynamic, asynchronous, multi-user, interactive environments. This means that events occur asynchronously. Hence, all smart space reactions are

triggered by user actions, changes in smart space, and context. Some events trigger behavioural adaptations of services; however, other specific events trigger the meta-level of smart space. Following [14], we list some of these events below and questions on which meta-level should respond.

A new user appears in smart space. When a new user comes into a space, he can violate privacy considerations of users already in the smart space, for example. Hence, smart space would probably need to reconfigure or redeploy the services for users. Some of meta-level questions are:

Q1. Whether to continue or stop current task execution?
Q2. Whether to redeploy the task to other resources?

New task or command arrives. User or service actions in the smart space can conflict with other services and users.

Q1. Whether to start processing the task immediately or postpone it's processing or even drop this task?
Q2. Which composition/deployment mechanism to use?
Q3. If the task causes conflicts, can it be deployed without conflicting or should the other running task be redeployed?

New resource appears. New device of smart space can provide better support for users.

Q1. Should a task being executed be redeployed to this device?

Resource/Service is removed from smart space or fails to perform the task. This event triggers the self-healing property of the smart space.

Q1. Whether redeployment of executing task is possible?
Q2. Which deployment strategy to use?

Performance is worse than expected. This event comes either from user feedback or from the smart space's own evaluation via a monitoring process.

Q1. Should the task be recomposed and redeployed?

Generally, these events can be considered as context for smart space and answering listed questions finally will lead to context-adaptation of smart space. However, this adaptation is considered at the higher abstraction level.

C. Framework Overview

Fig. 2 presents the components required to equip the smart space with meta-level control and monitoring facilities (see levels of Fig.1). Ground level is formed of components sensing the environment and users (Perceptors), and of components modifying the environment and delivering information to users (Actuators). Object level contains mechanisms to compose services, locate suitable resources and execute services (Service composer, Service deployer, and Service executor). Meta-level controls and monitors the execution of object-level tasks.

Meta-level contains Trigger component that gets events and triggers the Control component. The Control component controls the object-level tasks, such as component execution. In addition, it guards which user tasks can be postponed and put into Agenda, and which should be processed and executed immediately. A waiting queue buffers tasks if they cannot be performed at the moment, but

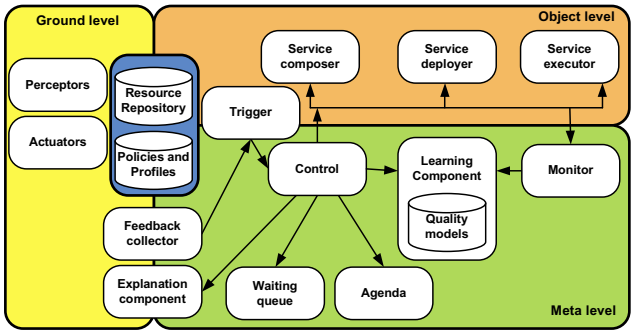

Figure 2. Necessary components for meta-level of smart space.

possibly can be handled when context changes (e.g. device becomes available). The Control component considers the Quality models, which are the best configurations of object-level tasks strategies and system performance (constructed with Monitor and Learning components). Also, the Control component gives its information to Explanation component to inform the user of the smart space about decisions. The Monitor component monitors the execution of these tasks and updates the Quality models. Resource repository, Policies, and Profiles are necessary components for all levels of smart space. Some meta-level components are on the edges, meaning that they go beyond the meta-level scope. For instance, Feedback collector component can be considered as Perceptor, however, due to its specific role in meta-level we treat it separately.

We aim to provide a plug-in solution in order to equip existing smart space infrastructures with meta-level facilities. Due to dependencies with components of other layers (see Fig. 2), certain requirements guide the design. First, formal service, resource and task descriptions have to be created. In this case, object level would be able to make service compositions and deployment plans. Meta-level, in turn, would be able to make decisions about tasks, their urgency, utility and priority. Second, mechanisms to control and monitor the object level of smart space should be created, such as interface descriptions. In this case, meta-level would be able to alter the composition mechanism. Third, mechanisms to collect feedback information as unobtrusively as possible should be developed. This is required as we aim to improve the performance of smart space, based on both users' opinion and quantitative measurements of object level tasks.

IV. Simulation Example

In order to validate the proposed concepts, we have conducted command-line simulation studies with Win-Prolog. The purpose of the simulated smart space is to support users according to their preferences and locations. That is, to execute right services at right devices, according to user profiles, feedback and location.

The simulated smart space consists of several rooms, equipped with computational resources to execute services.

Different services can be executed on different resources which provide certain quality. In our case, services don't have any QoS requirements, however, users have in their preferences specific quality requirements for service execution. Our current simulation considers only service deployment and service execution object-level tasks of smart space (see section 3.A).

We utilize a rule-based approach for system design. Russel and Wefald [15] prove that meta-level can be built using the same structures as object level. We apply rules for both object and meta-level tasks, hence all facts and rules of simulation are presented with Win-Prolog terms and clauses. Events, which trigger an inference process, are inputted manually with the command-line interface.

A. Context Model

The context model is designed in First Order Logic (Table 1). Service and device profiles describe the resources and applications the smart space offers. A user profile tells user preferences for locations and applications. Based on these profiles, object level deploys and executes services on proper resources. At the same time, meta-level addresses privacy, quality and time issues from these profiles.

We have also defined concepts required for meta-level to make decisions, see Table1. These concepts present the performance and satisfaction quality models. Meta-level considers them for decision-making. A best quality model concept tells how good deployment mechanisms are for the service in the specified location. These values are updated with the execution of these mechanisms. Based on this model, meta-level estimates the performance of the system and whether it is even possible to find a suitable resource for the service. A user satisfaction model tells how well the deployment plan satisfies the user. In this simulation, we retrieved this information directly from the user.

In this simulation, we considered four strategies for creating the service deployment plan based on three parameters: Privacy, Quality, and Availability (Table 2). Privacy and Quality requirements are specified in user profiles, Availability tells whether the resource is in use. We consider the cases when at least two requirements are satisfied. If only one or none of the requirements are satisfied (meaning that none of the strategies was able to find the resource for the service) then the user is asked whether to put the service into Waiting queue. These strategies are constructed with Win-Prolog clauses. Meta-level rules encode which strategy should be used for each case, see explanation in the next section.

B. Tasks of the Simulation

When a user comes into the smart space, he is served with all layers (see Fig. 2). Object level finds suitable resources and executes the services when meta-level monitors and controls these tasks. We followed the events triggering the meta-level, specified in section 3.B.

New user appears in smart space. When new user comes into the smart space, meta-level examines whether any privacy issues of other users in this space become violated. When it is so, the system asks violated user about interrupting the service execution and deploying it to another resource (see Fig. 3). The following rule is responsible for this task:

- If there is a service with private privacy value executing on a public resource in the smart space then suggest

TABLE I. CONTEXT MODEL OF SIMULATED SMART SPACE

Predicate
`service_profile(service_id, interface, format)` **Description:** Service profile tells which functionality and which formats a service supports. **Example:** `service_profile(music,play,mp3)`
`device_profile(device_id,interface,format, quality,publicity, latency_time)` **Description:** Device profile tells the functionality the device supports, data format it supports, the quality it promises, whether it is public or private, and how much time it takes to start executing the certain function (ms). **Example:**`device_profile(stereo,play,mp3,100, public,3000)`
`user_profile(user_name, location, service_id, action, privacy, priority, QoS, time, mode)` **Description:** User profile tells user preferences about the services to execute at certain location, their quality, privacy, priority, accepted start time delay and mode. Mode tells whether the service should be started automatically or manually (confirmed by user). **Example:**`user_profile(alice, room1, music, play, public, high, 100, 5000, 1)`

`best_quality_model(service_id,location, pqa_deployment_time,pnotqa_deployment_time, notpqa_deployment_time,pqnota_deployment_time)`
Description: Best quality model keeps track of best deployment time for different strategies.
Example:
`best_quality_model(music,room1,3000,3100,3120,4000)`

`user_satisfaction(user_name,service_id,action, location,device_id,desired_quality, achieved_quality,pqa_time,pnotqa_time,notpqa_time, pqnota_time,user_satisfaction_grade)` **Description:** User satisfaction tells whether user is satisfied with service deployment on the specific device and corresponding deployment latency. **Example:** `user_satisfaction(alice,music,play, room1,stereo,100,100,3002,0,0,0,5)`

TABLE II. SERVICE DEPLOYMENT STRATEGIES

Strategy	P	Q	A	Description
PQA	+	+	+	If a resource satisfies service privacy, and quality and it is available then locate the service to this resource.
PQNOTA	+	+	-	If a resource satisfies privacy and quality requirements, but it used by another user and that user's service priority is lower, then system asks whether it is possible to redeploy his service to another resource.
PNOTQA	+	-	+	If a device is available and it satisfies privacy, but not quality requirement, then the user is asked whether he agrees to execute the service on this device.
NOTPQA	-	+	+	If a resource is available and satisfies quality, but not privacy, then the system checks whether the user is alone in the smart space, if yes it suggests the user to execute the service on a public device.

Situation description: Alice executes private service photo_album on a public wall screen. When Bob comes into the same space, system notices that Bob's appearance violates Alice's preferences for this service.

```
To alice: Another user came into space. Would
you like to keep the photo_album running (run),
stop its execution (stop) or the system should
try to find a more suitable resource
(redeploy)?
|: redeploy
To alice:The required QoS for photo_album
service cannot be achieved at the moment, would
you agree to use mobile1 device instead ?
|: yes
To alice:The service photo_album will start
shortly at the mobile1 device.
```

Figure 3. Simulation screenshot for handling privacy in smart space.

user a choice between stop/run/redeploy this running service. Then act according to user reply.

When a user comes into smart space, all services from his profile (Table 1) are reported to meta-level to get processed.

New task or command arrives. Users specify in their profiles the priority of the services and their accepted start time delay. Meta-level estimates whether it is possible to create a deployment plan for certain service and whether to start doing it now, or put the service into Agenda in order to complete first with other more important services. The rules are the following:

- If a accepted delay of a service is less than the time to deploy it on any suitable resource, then this service execution is rejected and user gets informed about.
- If a service has low priority and has enough time to find resource and get executed, then put it into Agenda.
- If service is of medium priority and it's start time delay is much longer than time required for deployment, then put it into Agenda.
- If service is of high priority and it's start time delay is enough to create deployment plan and execute then process the service.

Deploying the service is an object-level task. However, prioritizing of the deployment strategies is a meta-level task. The prioritizing of strategies is important, as different requirements satisfy users unequally. For instance, if a perfect match is not possible (PQA strategy, Table 2) one user prefers the privacy satisfaction to quality (PNOTQA, Table 2). Also, some strategies require more time to execute and not all services can wait. In our simulation, meta-level considers user's feedback in order to make the right prioritizing for service deployment strategies for the user. Meta-level considers the time it takes to find and deploy the service with a certain strategy and user satisfaction represented as the integer number from 1 (absolutely not satisfied) to 5 (perfect). We use two schemas in our simulation to prioritize strategies: 1) user satisfaction schema; 2) strategy processing time schema. Meta-level considers the whole history of executing a certain service by the user and averages the user satisfaction according to the strategy used, also it averages the time it takes for certain

Situation description: According to Alice's profile, the music service has enough time to get started, hence meta-level of smart space selects satisfaction-based strategy in order to allocate the music service. Based on history analysis, the following sequence of allocating strategies is chosen: PNOTQA, NOTPQA, PQNOTA. This means that if PNOTQA does not find any resource, then NOTPQA strategy is executed and so on.

```
To alice: Would you like to activate service
music ?
|: yes
To alice:The required QoS for music service
cannot be achieved at the moment, would you
agree to use wall display instead ?
|: yes
To alice:The service music will start shortly
at the wall display.
```

Figure 4. Simulation screenshot for strategy selection.

service to create a deployment plan with the certain strategy (see Fig. 4). The rules are:

- If start time delay of a service is much higher than the maximum deployment plan creation time regardless of the strategy, then select the user satisfaction schema
- If start time delay of a service is just enough to get deployed, then select the time-based schema.

New service can cause conflicts with services being executed in a smart space already, hence meta-level should detect such situations and act accordingly. In our simulation, already executing service has the higher priority over new services. Conflict situations vary according to the smart space specifics and services it can provide:

- Two services cannot execute "play" action publicly simultaneously. The system will suggest to put one of the services to a private resource.
- One user cannot switch off a service being executed by another user.

New resource appears. When a new resource appears, meta-level considers whether this device suits for some awaiting services or whether it provides a better QoS for already running services. Rules to handle new devices are:

- If there is any service waiting for execution, then try to locate this service on this device.
- If the new device offers better QoS for one of the executing services then ask whether the user wants to use this device for the service.

Resource/Service is removed from smart space or fails to perform the task. Meta-level selects the strategy and triggers the redeployment (object-level) of services if the user agrees, otherwise service is stopped.

Performance is worse than expected. There are two kinds of performance evaluations. The first one is user-based performance evaluation, which depends only on user's opinion and varies significantly among users. This evaluation is obtained based on user feedback. The second one is performance evaluation, based on measuring of object-level performance. This one is based on quantitative criteria telling how well the task progresses. Function of these criteria forms the task performance. Each object-level task should have a model of the best performance, in other

words the criteria values, leading to the best performance of the task, should be known. Poor performance is defined with a threshold value. If current performance, computed from criteria values, differs from the best performance more than the threshold allows, then poor performance is detected. Criteria of object-level tasks are smart space specific.

When poor performance is detected, meta-level decides whether a task should be placed to a more capable device or select another composition or deployment strategy.

V. DISCUSSION

This article demonstrated the need to separate the smart space adaptation functionality and modification of these adaptation actions. A framework for such a meta-level for smart spaces was presented. Win-Prolog based example was conducted in order to verify the outlined concept.

Separation of the smart space adaptation to object level and meta-level brings the following advantages:

Separation of concerns. Isolating the meta-level of smart space from the object level brings flexibility in development and deployment of smart spaces, as meta-level module can be developed separately and plugged into already existing infrastructure. Also, such separation can delegate some smart space consistency support to the meta-level, hence release developers of services from considering such issues as conflicts, resource starvation, etc.

User-oriented design. Meta-level allows developing plenty strategies for object-level and integrating them into the space. This facilitates using different strategies with different user circumstances. User feedback would lead to better understanding of user needs and overall system adaptation. Moreover, meta-level can be responsible for informing the user about system decisions and performance.

However, meta-level processing consumes system resources and it is the matter of study how this trade-off can be balanced. Meta-level should be able to analyze the benefit/cost relations and make most profitable decisions.

The simulation example demonstrated that design of plug-in architecture for meta-level is challenging and needs further research. Meta-level must have the details about users, their preferences, resources and services of smart space. Some of these profiles can be constructed by meta-level with learning mechanism, but in this case the concept mapping should be done between meta- and object levels. Quality models can be constructed independently from users and infrastructure. Big challenge is interfaces to notify meta-level about occurrence of triggering events. Also, meta-level should be aware of object-level strategies and their interfaces in order to control them. We observed that rules serve meta-level well. Management of services is invisible to users, and they are informed about all decisions of smart space. Current simulation considered direct user feedback in order to make prioritizing of the strategies. In the future, we are interested in obtaining the user feedback with learning mechanisms by tracking user actions.

We consider the separation of smart space adaptation to object and meta-level as a flexible solution to achieve the self-optimizing property of smart space.

ACKNOWLEDGMENT

Ekaterina Gilman would like to thank GETA (Graduate School in Electronics, Telecommunications and Automation) for funding.

REFERENCES

[1] A.G. Ganek and T.A. Corbi, “The dawning of the autonomic computing era,” IBM Syst. J., vol.42(1), Jan. 2003, pp. 5-18, doi: 10.1147/sj.421.0005.

[2] C. Gouin-Vallerand, B. Abdulrazak, S. Giroux, and M. Mokhtari, “Toward autonomic pervasive computing,” Proc. 10th Int. Conf. on Information Integration and Web-based Applications \& Services (iiWAS '08), G. Kotsis et al. (Eds.) ACM, NY, USA, pp. 673-676, doi: 10.1145/1497308.1497440.

[3] P.K. McKinley, S.M. Sadjadi, E.P. Kasten, B.H.C. Cheng, "Composing adaptive software," Computer, vol.37 (7), July 2004, pp. 56- 64, doi: 10.1109/MC.2004.48.

[4] J. Rao and X. Su, "A Survey of automated Web service composition methods," Proc. the First Int. Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), San Diego, California, USA, July 2004, pp.43-54.

[5] B. Abdulrazak and A. Helal, "Enabling a plug-and-play integration of smart environments," Proc. Int. Conf. on Information and Communication Technologies (ICTTA '06), April 2006, pp.820-825, doi: 10.1109/ICTTA.2006.1684479.

[6] D. O’Sullivan and V. Wade, “A Smart Space management framework”, Technical Report, Trinity College Doublin, April 2002, pp.1-23.

[7] M. Jurmu, M. Perttunen, J. Riekki, “Lease-based resource management in smart spaces,” Proc. of Workshops of the 5th Annual IEEE Int. Conf. on Pervasive Computing and Communications (PerCom Workshops ‘07), White Plains, NY, USA, pp. 622-625.

[8] M. Feeney and R. Frisby, ‘‘Autonomic management of smart spaces,’’ Proc. 3rd International Workshop on Managing Ubiquitous Communications And Services (MUCS 2006), 2006, Cork, Ireland.

[9] S. Ahmed, S.I. Ahamed, M. Sharmin, and C.S. Hasan, "Self-healing for autonomic pervasive computing," In A.V.Vasilakos et al. (Eds.) Autonomic Communication, Springer, 2010, pp. 285-307, doi: 10.1007/978-0-387-09753-4_11.

[10] W. Trumler, J. Petzold, F. Bagci, and T. Ungerer. “AMUN: an autonomic middleware for the Smart Doorplate Project,” Personal Ubiquitous Comput., vol.10(1), Dec. 2005, pp. 7-11, doi: 10.1007/s00779-005-0029-4.

[11] N. Paspallis and G.A. Papadopoulos, “An approach for developing adaptive, mobile applications with separation of concerns,” Proc. the 30th Annual Int. Computer Software and Applications Conf. (COMPSAC’06), IEEE Computer Society, Washington, DC, USA, pp. 299-306, doi:10.1109/COMPSAC.2006.22.

[12] M. Cox and A. Raja, “Metareasoning: a manifesto,” Proc. Metareasoning: Thinking about Thinking workshop held within 23 AAAI Conf. on Artificial Intelligence (2008).

[13] M. Cox, “Metacognition in computation: A selected research review,” Artif. Intell., vol. 169(2), Dec. 2005, pp.104-141, doi: 10.1016/j.artint.2005.10.009.

[14] A. Raja, “Meta-level control in multi-agent systems,” Dissertation, Univ. of Massachusetts Amherst, September 2003.

[15] S. Russell, and E. Wefald, “Principles of metareasoning,” Proc. the First Int. Conf. on Principles of Knowledge Representation and Reasoning (1989), Morgan Kaufmann Publishers Inc., pp. 400-411.