An Overall Process for Self-Adaptive Pervasive Systems

Antonio Bucchiarone*, Annapaola Marconi*, Marco Pistore*,
Stefan Föll†, Klaus Herrmann†, Christian Hiesinger†, Srdjan Marinovic‡
*FBK-IRST, Trento, Italy
{bucchiarone,marconi,pistore}@fbk.eu
†Universität Stuttgart, Germany
{stefan.foell,klaus.herrmann,christian.hiesinger}@ipvs.uni-stuttgart.de
‡Imperial College, London, UK
srdjan@imperial.ac.uk

***Abstract*—Self-adaptive pervasive systems often implement adaptation in a centralised manner, where one component holds all the necessary knowledge to identify when and how the system needs to adapt. In self-adaptive pervasive systems, composed of autonomous components with different authorities (such as security, distribution, etc.), this approach cannot be implemented as composing a centralised knowledge is not feasible and it also obstructs the system's ability to dynamically change its components. A simple alternative would be to allow each component to adapt independently but this can quickly give rise to conflicts, race conditions and oscillations between multiple independent adaptations. To avoid these problems, we propose to coordinate individual adaptations so that each component's adaptation goals are satisfied. Each component proposes an adaptation which is reviewed by other components who may propose their own adaptations that they may need to do. This continues until a complete adaptation plan is agreed upon. In cases where certain individual adaptations conflict with some components' goals, components are instructed to seek alternative proposals. The *Adaptation Manager* component is in charge of the negotiation process and it also has the authority to resolve certain conflicts between adaptations. Our approach is evaluated in the context of pervasive workflow systems where the failure probability and execution times are assessed.**

***Keywords*-Pervasive Systems; Adaptation; Workflows**

I. INTRODUCTION

Self-adaptive pervasive systems operate in environements characterised by a high degree of dynamic behaviour due to frequent changes in the environment and frequent changes of the environment in which they operate [1]. As a running use-case study of adaptive pervasive systems, this paper will adopt the system described in the ALLOW project [2], that focuses on modelling and building human-centric pervasive systems. The project is also heavily investigating the applications of workflows as a new programming paradigm for these systems. More precisely, adaptable pervasive systems (APSs) are modeled using Adaptable Pervasive Flows (APFs) [3], [4], [5] and these flows represent an extension of traditional workflow concepts [6] which make them more suited for adaptation and execution in pervasive execution environments.

Principal application adaptations are executed by replacing or rearranging tasks in a flow such that the flow's goals can still be fulfilled despite the fact that various failures and problems have occurred. Beyond that, a number of other adaptation aspects need to be controlled while a flow is running such as: the adaptation of user interfaces, security adaptation, and flow distribution adaptation. The adaptation strategies for these different aspects of the flow execution are stored and executed by the components in control of these aspects, such: security manager, distribution manager and so forth. Having each component adapt its own concerns independently of the rest of the system, makes the adaptation much more robust since the adaptation knowledge and strategies are decentralised. This in turn makes the components loosely-coupled and the system more robust in face of component failures. However, this also introduces a problem of coordinating these different independent adaptations to avoid conflicts, oscillations and race conditions among their adaptation actions and goals.

The main contribution of this paper is to propose a novel approach for coordinating a set of autonomous components that can adapt different execution aspects of a common pervasive system. Our solution allows the components to retain their autonomy and authority over the execution aspects that they are overseeing, while undesirable effects such as conflicts, oscillations and race conditions are avoided. We have implemented this approach for the pervasive flow systems and we use this implementation to validate our ideas. We refer to this approach as the Overall Adaptation Process (OAP). The OAP approach is based on two main principles: 1) every aspect publishes the changes it is going to make to the system, 2) other aspects say whether they agree with the changes and whether they themselves need to make further changes to accommodate the original proposed ones. In order to coordinate this negotiation loop between different components we introduce the *overall adaptation manager* whose internal logic guides the negotiation to an overall agreement.

The rest of the paper is structured as follows: Section II analyses the requirements that the proposed adaptation

 ISBN: 978-1-61208-109-0

approach, specified in Section III, ought to meet. Section IV discusses testing and validating the proposed approach, while the last two explore the relevant related works and conclude the paper.

II. Problem Analysis

The nature of pervasive systems defines extensive requirements for the design of self-adaptive systems. First, pervasive systems are becoming increasingly complex and are required to manage a large set of different aspects of the system. Each of these heterogeneous aspects considers a bounded part of the overall system and is associated with a dedicated goal for execution. This goal is defined by means of *acceptance criteria*. Second, adaptation is considered as a key principle to deal with the dynamics found in pervasive environments, which results from unforeseen changes in the system that may violate each aspect's acceptance criteria. Therefore, adaptations are necessary to bring the system back into a state that preserves the individual criteria of each aspect. Due to the above characteristics, the overall complexity of the system is only manageable by avoiding a single, centralized point of control. The reduction of complexity can be achieved through a loosely-coupled system design, where the aspect-specific adaptation strategies are independently encapsulated inside each aspect. Thus, aspects can be flexibly composed to suit the characteristics of different environments, i.e., some aspects may not be present in each setting.

However, this model of independently acting aspects also elevates certain issues. Although each aspect is only concerned with a part of the system, harmful side-effects resulting from adaptation may occur. In particular, the adaptation of one aspect may easily push the system into a state that triggers adaptations in other aspects. This causes the following problems:

Conflicts: A conflict arises if two or more aspects react to a relevant change by issuing contradictory adaptations concurrently. Contradictory adaptations are adaptations that cannot be applied in combination since this would violate the acceptance criteria of the system.

Oscillations: An oscillation can occur if conflicting adaptations are issued in sequence: If one aspect as_1 receives an event that violates its internal acceptance criteria it issues an adaptation which in turn violates the acceptance criteria of one or more other aspects. These aspects react by issuing adaptations to correct this violation, which in turn violates the criteria of as_2. This can lead to a long series of mutual adaptations without ever reaching a system configuration that satisfies all aspects.

Race conditions: Race conditions occur if two or more aspects apply adaptations and the final outcome depends on the order in which they complete and apply their individual adaptations. In particular, if we have aspects as_1 and as_2 issuing adaptations, the order as_1, as_2 may lead to a system configuration that satisfied all acceptance criteria while the order as_2, as_1 may result in a conflict or oscillation.

In order to avoid these problems, a coordinated way of interaction among the aspects is required. The goal is to restrict the freedom of each individual aspect in order to avoid the negative effects detailed above while still preserving as much of the autonomy as possible. The principal question that we are tackling in this paper is therefore: *How can loosely coupled autonomous adaptation aspects be coordinated while their autonomy is preserved*?

III. Cross-Aspect Adaptation

The novel approach for cross-aspect adaptation proposed in this paper is based on the insight that aspects need to coordinate at an early stage of adaptation before applying changes to the system. For this reason we advocate the introduction of an adaptation process split into phases of *adaptation negotiation* and *adaptation enactment*. Adaptation negotiation denotes the process of aspect interaction with the goal of finding an acceptable adaptation that all aspects agree on. Adaptation enactment refers to the physical adaptation of the system, based on the agreement achieved by the negotiation process before. Both phases are usually strongly integrated into one indivisible process in traditional systems. However, the decoupling of these constituent parts of our adaptation process enables adaptation aspects to reason over the effects of possible adaptation and influence other aspects in decision making before the actual change of the system.

In the next subsections, we describe our solution that facilitates coordinated adaptation among the aspects in the system. For this purpose, we first introduce in Section III-A the adaptation framework, which presents the necessary elements and their relationships for building self-adaptive systems according to our approach. Subsequently, we use these elements to define our overall adaptation process in Section III-B.

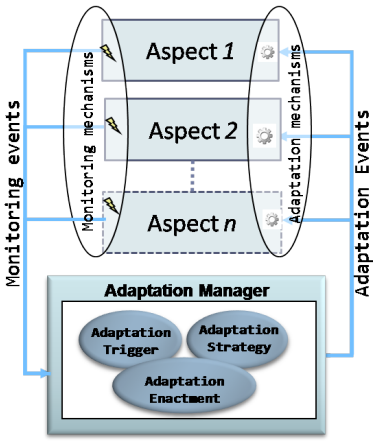

Figure 1. Overview of Cross-Aspect Adaptation Framework

A. Adaptation Framework

The adaptation framework serves as a blueprint for the design of composed adaptive systems. It defines a unified view on the elements of a self-adaptive pervasive system as illustrated in Figure 1. The elements are *Adaptation Aspects* representing the self-contained entities executing adaptations in the system and the *Adaptation Manager* that represents the control instance of a coordinated adaptation process.

An adaptation aspect covers a well-defined part of the pervasive system subject to dynamic adaptation. The *model* of an adaptation aspect as_i is expressed as a set of public variables $V(as_i)$ modelling the system properties relevant for adaptation. An assignment of the variables in $V(as_i)$ is called a *configuration* of the aspect and denoted as $v_{as,i}$.

Each aspect represents an autonomous part of the system, which monitors changes of the system relevant to its acceptance criteria. For this purpose, each aspect provides a set of *monitoring mechanisms* to detect events that provide information on the execution environment. These events are used to inform the Adaptation Manager about aspect-specific problems that demand an adaptation. Moreover, each adaptation aspect has a set of *adaptation mechanisms* that can be used to enact adaptations physically in the environment. The concrete adaptation mechanisms are aspect-specific and depend on the functionality provided by each individual aspect. While aspects change the system according to their adaptation needs, they are not required to know or understand the direct effects of their own adaptations on other aspects. This preserves their fundamental level of autonomy as aspects just have to implement their internal adaptation logic and know their own acceptance criteria.

Aspects are able to communicate with each other based on a common negotiation language. The negotiation language of aspects $AS_{1,n} = \{as_1, as_2, .., as_n\}$ is based on all variables $V(AS_{1,n}) = V(as_1) \cup V(as_2) ... \cup V(as_n)$ shared among them. The variables are used to communicate the changes an aspect is going to make for adaptation. An assignment of all variables is formally referred to as $v_{AS_{1,n}}$ and indicates a desired *system configuration*, which may result from a series of adaptations of a set of adaptation aspect. The change of an aspect's configuration has consequences on all aspects that are influenced or depend on its shared properties. Thus, the coordination of several aspects is based on the exchange of system configurations, allowing all aspects to inspect and influence the future adaptation of the system. As the exchanged system configurations describe hypothetical system states under negotiation, they can be seen as *proposals* of an overall adaptation that involves more than one aspect for adaptation.

In addition to the adaptation aspects, our framework also introduces the *Adaptation Manager* (AM) component whose purpose is to coordinate multiple independent adaptations. This introduction is motivated by the fact that a well-structured way of interaction among the aspects is required, which needs to be controlled by a mediator. The Adaptation Manager reacts to monitoring events arriving from the adaptation aspects. Based on the incoming events, the AM infers an adaptation trigger a_t that characterizes the cause of adaptation. The trigger may be the result of a single monitoring event, as well as be discovered from the correlation of multiple events coming from different adaptation aspects. The AM acts upon a set of adaptation strategies, each of which defines a coordination process for specific adaptation triggers. Formally, an adaptation strategy a_s is a tuple (a_t, a_p, a_e, p) where a_t denotes the adaptation trigger the strategy is devised for, a_p is the adaptation plan, a_e describes the adaptation enactment, and p is the priority associated to the strategy. The adaptation plan a_p associated with an adaptation strategy a_s defines the procedure of how to negotiate among the aspects in order to react appropriately on an adaptation trigger. For this purpose, the adaptation strategy reflects the dependencies among the adaptation aspects. The dependencies of aspects define the order, in which aspects can make their proposals of desired adaptations. For example, if the adaptation manager selects a strategy with the associated plan $\langle as_i, as_j, as_k \rangle$, it means that the aspects are contacted in the order as_i, as_j, as_k. Similarly, the adaptation enactment a_e is a list of aspects and captures the order in which different aspects should be invoked to enforce the adaptation. This allows complex adaptations, where not only the agreement for finding an adaptation, but also the implementation of adaptations requires a coordinated approach.

B. Overall Adaptation Process

The process of overall adaptation defines an interaction protocol in which the adaptation aspects iteratively take turns in a well-defined order and propose adaptations. In this way, adaptations are not occurring concurrently but they are serialized. Each aspect thus does not base its adaptation on the actual current system configuration but on the hypothetical system configuration that would exist if the previously proposed adaptations had been applied. This process is coordinated by the Adaptation Manager and consists of four individual phases, which are described in detail subsequently:

Initialization *(Phase 0)*: During the system execution, the set of adaptation aspects publishes monitoring events. The AM, using a correlation function C, infers an adaptation trigger a_t to launch the overall adaptation process. In order to react appropriately on a_t, the AM selects all adaptation strategies $\{\langle a_t, a_p, a_e, p \rangle\}$ which are devised for this trigger. Among this set, the strategy with the highest priority p is chosen for execution. Each aspect listed in the adaptation plan a_p will participate in the overall adaptation process. Let these aspects be denoted as $AS_{1,n} = \{as_1, as_2, ..., as_n\}$ in

 ISBN: 978-1-61208-109-0

the following. The initial configuration of the system before negotiation is thus given by the values of all variables in $V(AS_{1,n})$ found in the running system at the time a trigger is raised. The goal of the OAP is then to negotiate a future configuration of the system, so that the individual requirements of each aspect can be met.

Negotiation phase *(Phase 1)*: In the negotiation phase a consensus protocol is executed that involves interactions of the AM with the adaptation aspects. The AM controls the interactions according to the adaptation strategy and collects the adaptation proposals from the aspects. An adaptation proposals represents a hypothetical aspect configuration that would be the result of an adaptation action. It is important to note that the aspect configuration is merely virtual and describes the changes of a desired adaptation in an understandable way for the remaining aspects. This creates the opportunity for aspects to inspect the objectives of other aspect before their physical enactment. For this purpose, each aspect as_i provides a uniform interface $propose_{as_i}$ that can be invoked by the AM during the negotiation phase:

$$propose_{as_i} : (v_{AS_{1,n}}, a_t) \mapsto v'_{AS_{1,n}}$$

Each aspect participates by suggesting an adaptation, which in turn provides the input for aspects consulted subsequently. The proposal is made for a specific adaptation trigger and based on the proposals made by other aspects which came before. This process ensures to exchange adaptation proposals in a way, which takes the dependencies among the aspects into account. This iterative procedure results in an evolution of the future system configuration from the first version, which is the initial proposal of first aspect, to the final one, where each aspect has stated its proposal. This procedure can be seen as a concatenation of the proposal functions: $v'_{AS_{1,n}} = propose_{as_n}(...propose_{as_2}(propose_{as_1}(v_{AS_{1,n}}, a_t), a_t), a_t)$. The proposal phase is completed after the last aspect as_n from the set of participating aspects has contributed its proposal. The result is a new hypothetical system configurations, that needs to be validated for enforcement. As the AM forwards a proposals from one aspect to another, the aspect have not to be aware of each other directly.

Decision phase *(Phase 2)*: After each aspect has taken its turn, an agreement needs to be reached whether one coherent overall adaptation (result of all the aspect-specific adaptations) was found. For making this decision, each aspect needs to rate the outcome of the proposal phase with regards to its individual acceptance criteria. As the rating of an aspect may be influenced by proposals made *before* and *after* an aspect's own turn, the decision making is reached in a separate phase that allows all aspects to see the final negotiated configuration $v'_{AS_{1,n}}$. For rating, each aspect may choose from a set of aspect-specific *verdicts* $r_{as_i} = \{r_1, .., r_n\}$ that are also known to the AM. An aspect can communicate a quality measure via a verdict which allows the AM to infer the satisfaction of an aspect with the adaptation proposal. The rating is calculated using the function $rate_i$ of each aspect that can be formalized as:

$$rate_i : v'_{AS_{1,n}} \mapsto r \in r_{as_i}$$

so that the overall rating $r_{overall} = (r_1(v'_{AS_{1,n}}), ..., r_n(v'_{AS_{1,n}}))$ can be expressed as the tuple of all the aspects' verdicts. The AM inspects the overall rating and determines whether one coherent overall adaptation was found. For this purpose, the AM holds decision rules that determine if the proposal can be accepted:

$$decide : r_{overall} \mapsto \{accepted, rejected\}$$

Thus, the AM represents the final authority for enforcing adaptation decision. If the overall rating is not sufficient for enforcing an adaptation, the AM will start another negotiation phase as described before such that the solutions proposed in the previous round are avoided by the aspects who proposed the first initial proposal. In each loop, all aspects are required to *relax their adaptation criteria*. Such a relaxation is realized by applying a less restrictive threshold to some quality criteria an aspect uses internally to judge whether a given system configuration is sufficient. By means of criteria relaxation, the set of possible system configurations and thus the probability of finding valid overall adaptations is increased. The termination of the negotiation loop is ensured by a defined strategy associated with the adaptation plan, e.g., restricting the number of negotiating cycles.

Phase 3 – Enactment phase *(Phase 3)*: If a valid adaptation is found in phase 2, the hypothetical configuration on which the aspects have agreed needs to be physically implemented in the system. This again motivates a coordinated approach among the aspects, since an aspect may already require the configuration of another aspect to be physically existent in order to implement its target configuration. Therefore, the aspects again take turns in a well-defined order. The order is defined by the enactment a_e as part of the selected adaptation strategy a_s. Thereupon, each aspect implements its part of the overall system configuration. The required actions to enforce an adaptation for an aspect as_i are directly derived from the target configuration $v_{AS_{1,n}}$ of the agreed overall system configuration.

IV. EVALUATION

To evaluate the feasibility of our proposed overall adaptation approach, we applied our proposal in the context of the EU FP7 ALLOW project [4]. This project focuses on human-centric pervasive applications. Applications are

 ISBN: 978-1-61208-109-0

specified as adaptive pervasive workflows. To execute a workflow successfully, a workflow must adhere to a set of functional and non-functional constraints. In the ALLOW project, three different adaptation aspects are considered: 1) the *Flow Security Aspect* which ensures that the execution of a workflow is according to specified security policies. 2) the *Flow Structure Aspect* which may modify the structure of a workflow in order to meet the specified goal of the workflow. 3) the *Flow Distribution Aspect* which ensures that the execution of a workflow complies to specified performance criteria.

Without coordination adaptation the problems mentioned in Section II may occur. Assume, for example, the *Flow Security Aspect* replaces untrusted services with trusted ones. The new set of services does not suite the performance criteria of the *Flow Distribution Aspect*. Thus, the *Flow Distribution Aspect* replaces the trusted services with untrusted ones that have a better performance which would in turn trigger the *Flow Security Aspect* again. Consequently, the system would oscillate.

To measure the effectiveness of our approach, we compared a system with adaptation manager controlling the overall adaptation process to a system without adaptation manager. We define a system without adaptation manager as a system without any coordination between the individual aspects. We conducted two sets of experiments. In the first set, we assumed a workflow engine aborts the execution of a workflow if a task blocks the execution because it cannot be executed without violating the acceptance criteria of the individual aspects. In this set, we measured the number of completely executed workflows.

In the second set, we assumed a workflow engine waits until a task can be executed without violating the acceptance criteria of the individual aspects. In this set, we measured the average time required to completely execute workflows.

Our evaluation is based on simulation. In each simulation run, we generated a workflow and executed the workflow in each system under identical conditions. This means, that the environment a workflow is executed in generates the same events at the same point in time. In the simulation, we measure time in simulation cycles.

The environment influencing the execution of our workflows consists of a network (simulating the electronic world) and a context system (simulating the real world). Both, network and context system may generate events at arbitrary points in time during a simulation run. These events may result in an adaptation of the currently executed workflow. Network events give information about changes in the availability or the trust of services provided by the network. Events of the context system indicate that the goal of a workflow is endangered due to real-world events.

This information is used by our aspects as follows. The *Flow Security Aspect* ensures that all services which are bound to the tasks of a workflow are according to the specified security policies. If this is not the case, the *Flow Security Aspect* replaces untrusted services with trusted ones if possible. The *Flow Distribution Aspect* binds the fastest available service in order to improve the execution time of a workflow. Whenever an event endangers the goal of a workflows, the *Flow Structure Aspect* tries to replace some tasks of the workflow with a predefined set of tasks. Hereby, the *Flow Structure Aspect* can chose from several predefined sets.

The probability for changes in the network or for context events is defined by a failure probability. In each simulation cycle, each service changes its state according to this failure probability. Similarly, context events are generated with this probability in each simulation cycle.

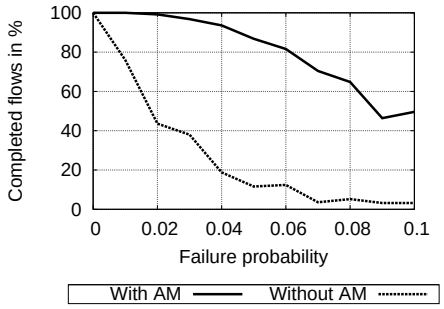

Figure 2. Engine aborting in case of a blocking task

Figure 2 shows the evaluation results for the first set of experiments. It can be seen that in the system with adaptation manager more workflows are executed completely than in the system without adaptation manager. This is due to fact that the system with adaptation manager is able to resolve conflicts and oscillations by means of its negotiation process. However, if the failure probability increases, the probability rises that no solution exists in the whole solution space. Thus, the percentage of successful executions decreases with an increase in the failure probability.

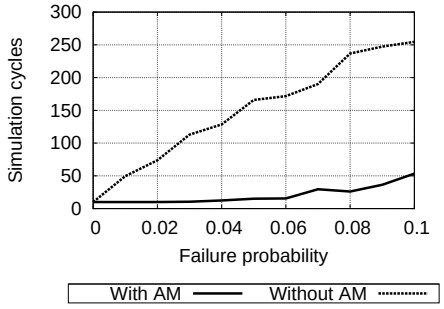

Figure 3. Engine waiting for tasks to complete

In Figure 3, the results for the second set of experiments are depicted. It can be seen that the number or simulation

 ISBN: 978-1-61208-109-0

cycles is much higher in the system without adaptation manager. This is because without coordination the aspects are not able to derive compromises and, thus, have to wait for a configuration that by chance fits all acceptance criteria. Again, as the failure probability increases, it may happen that no solution for a problem can be found. In this case, also the system with adaptation manager has to wait and, thus, the average time required to execute a flow increases.

V. Related Work

The motivations for, and a notion of coordinated adaptation has been investigated by Cheng et al. [7]. They have argued that independent adaptations must not introduce unwanted effects through which the system can be put into an unsafe state. These unwanted behaviours can be oscillations, races, deadlocks and so forth [8]. They propose that components should coordinate their evaluation metrics to reach a consensus, but they do not have a strategy to deal with situations when such consensus cannot be reached.

CARISMA system [9] explores bidding on a proposed set of adaptation policies as a way to adapt the whole system. When a set of proposals is collected every agent places a sealed bid for each proposal. The proposal with the highest sum of bids, wins. The agents implement a utility function that decides how much they are willing to bid on a particular proposal. The main difference with our work is that CARISMA has no further rounds, where agents would be able to revise their utility functions. This is a crucial requirement for our system since the workflow adaptation component can revise its adaptation plan based on a set of available and trusted services.

In [10] each application consists of different functional levels where each level has its own evaluation metrics. The model employs an independent component that monitors the performance of individual layers. If certain constraints are not satisfied, this monitor invokes different adaptation mechanisms in affected layers. The main difference with this approach is that the presented overall adaptation manager runs a negotiation loop.

VI. Conclusion

Self-adaptive pervasive systems that are composed of cooperating and autonomous components, typically employ each component to control and monitor a specific application execution aspect. Each of these components usually is capable of self-adapting a particular part of the underlying pervasive system or application in order to meet its own goals.

To avoid problems like conflicts, race conditions and oscillations, this paper proposes a solution based on a negotiation loop that attempts to find one adaptation that satisfies all components' individual constraints and needs. The negotiation loop provides a coordinated way for individual components to exchange adaptation proposals and infer whether such proposals break their individual requirements. This process is coordinated by a special component, the *Adaptation Manager*, that acts as a mediator to make sure that all components have a fair say.

We evaluated our approach in the context of pervasive workflow systems based on multiple simulations where we have compared a system with and without coordinated adaptation. Our results show that the convergence to a stable system is faster with the coordinated approach. For the future work we will investigate the following questions: how many proposals per round should be made,and how can the overall adaptation manager facilitate faster convergence to the overall solution.

Acknowledgment

This work is partially funded by the FP7 EU FET project Allow IST-324449.

References

[1] A. Soylu, P. D. Causmaecker, and P. Desmet, "Context and adaptivity in context-aware pervasive computing environments," *Ubiquitous, Autonomic and Trusted Computing, Symposia and Workshops on*, vol. 0, pp. 94–101, 2009.

[2] "ALLOW Project," http://www.allow-project.eu/.

[3] A. Bucchiarone, A. L. Lafuente, A. Marconi, and M. Pistore, "A formalisation of adaptable pervasive flows," in *WS-FM'09*, Bologna, Italy, 4 September 2009.

[4] K. Herrmann, K. Rothermel, G. Kortuem, and N. Dulay, "Adaptable Pervasive Flows - An Emerging Technology for Pervasive Adaptation," in *Workshop on Pervasive Adaptation (PerAda)*, September 2008.

[5] A. Marconi, M. Pistore, A. Sirbu, H. Eberle, F. Leymann, and T. Unger, "Enabling adaptation of pervasive flows: Built-in contextual adaptation," in *Proceedings of the 7th International Joint Conference on Service Oriented Computing (ICSOC'09)*, 2009.

[6] W. M. P. van der Aalst and K. M. van Hee, *Workflow Management: Models, Methods, and Systems*. MIT Press, 2002.

[7] S.-W. Cheng, A.-C. Huang, D. Garlan, B. Schmerl, and P. Steenkiste, "An architecture for coordinating multiple self-management systems," in *WICSA'04*, June 2004, pp. 243–252.

[8] V. Zamudio and V. Callaghan, "Unwanted periodic behaviour in pervasive computing environments," in *Pervasive Services, 2006 ACS/IEEE International Conference on*, June 2006, pp. 273–276.

[9] L. Capra, W. Emmerich and C. Mascolo, "CARISMA: Context-Aware Reflective mIddleware System for Mobile Applications," *IEEE Trans. Software Eng.*, vol. 29, no. 10, pp. 929–945, 2003.

[10] R. Kazhamiakin, M. Pistore, and A. Zengin, "Cross-layer adaptation and monitoring of service-based applications," in *MONA+*, 24 November 2009.

 ISBN: 978-1-61208-109-0