Self-Adaptation to Mobile Resources in Service Oriented Architecture

Nour Ali
University of Brighton
Brighton, UK
N.Ali2@brighton.ac.uk

Carlos Solis
Paddy Power
Dublin, Ireland
Carlos.Solis@paddypower.com

***Abstract*—Mobile or pervasive systems continuously change their environments and resources (e.g. battery or bandwidth). Mobile applications require different services when they enter or exit environments and as their resources change. In this paper, we propose a service oriented architectural approach that supports self-adaptation to changes in resources and location topology when mobility occurs, by reconfiguring the software architecture at runtime. The location topology and mobility primitives are inspired from ambient calculus. Our approach considers ambients to be autonomic elements that can manage elements located in them to their environment and provide them with new services suited to the available resources, when mobility occurs. Ambients implement a hierarchical and decentralized MAPE-K loop to adapt the distributed and mobile service oriented architecture to the resource requirements. We have designed an algorithm based on swarm optimization technique in order to allow ambients to optimally plan the reconfiguration process according to available services and resources. Throughout the paper, we use a scenario to illustrate our approach and perform initial evaluations on the swarm algorithm.**

Keywords- autonomic computing, mobile ambients, self-adaptation, service oriented architectural model, mobile resources, discrete swarm particle optimization

I. INTRODUCTION

As mobile applications evolve and become more business critical it is essential to apply software engineering processes and techniques to assure their quality. In his overview paper, Wasserman discusses software engineering issues for mobile application development [1]. He points out that greater attention needs to be paid to the software architecture of mobile applications as quality attributes are very critical. For example, applications of mobile devices need to take into account mobile device resources such as power consumption.

A promising architectural style for designing mobile applications is the Service Oriented Architecture (SOA) [2]. Several of its principles such as dynamic composition, loose coupling, and well defined interfaces are suitable to define the change and heterogeneity encountered in mobile applications. However, to properly define mobile characteristics SOA concepts have to be combined with mobile ones.

Mobile applications continuously interact with devices, networks, their environments, or pervasive services. At runtime these interactions can continuously change and adapt due to changes to the environments or to resources. An emerging area of research is the one of architectural models at runtime [5] which suggests representing architectural information explicitly at runtime and use it to adapt systems to changing user needs and/or operating environments. In [6], we described a research agenda for architectural models at runtime of mobile systems.

In this paper, we propose a model based software architectural runtime approach for allowing mobile systems to self-adapt to resources when they enter or exit environments. Self-adaptation occurs by using the software architectural topology at runtime, which is used as a knowledge base, and selecting a service oriented architectural configuration that best suits the resources when mobility occurs.

Our approach is based on modelling ambients, which are inspired from Ambient Calculus [3], as part of the software architecture configuration. An ambient is a bounded place where computation happens. An ambient can be a mobile phone, a network, a laptop, a building or a room. Ambients can have mobility capabilities which allow them to enter sibling ambients and exit parent ones. Ambients have a tree hierarchy of ambients as ambients can have other subambients, e.g., a mobile phone is located in a room. Our approach uses ambients to represent the location hierarchy and mobility primitives (exiting or entering) of a service oriented architecture. It also extends the traditional concept of an ambient as in ambient calculus [3], with the capability of being an autonomic element that can manage elements located in its environment and self-adapt by reconfiguring the service oriented architecture, e.g., by disconnecting applications from a service. An ambient can manage the resources of the elements in its environment, e.g., a mobile device monitors and manages how its applications consume the battery or bandwidth. Also, an ambient can choose a software architectural configuration to minimize resource consumption.

Another contribution to our approach is its usage of swarm particle optimization technique [4] in order to select the best possible runtime configuration based on the resources available. This is important in a service oriented architecture as services can appear and disappear in a mobile environment and the usage of resources can change. Swarm particle technique is efficient as it does not need all possible configurations to be defined at design time. It also can determine the nearly-optimal configuration based on the available resources at runtime, specifying a number of iterations.

This paper is structured as follows: Section 2 presents an example scenario that is used throughout the paper to illustrate our approach. Section 3 introduces the Ambient Service Oriented Architectural Metamodel that represents service oriented architectural models at runtime. Section 4

explains how ambients are autonomic managers. Section 5 presents the algorithms that allow ambients to plan adaptations of the service oriented architecture based on resources. Section 6 demonstrates our approach through the example. Section 7 explains our implementation and evaluation. Section 8 discusses related work, and finally sections 9 highlights conclusions and our further work.

II. EXAMPLE SCENARIO

Our scenario is a mobile device of Peter that is running two applications: a cinema and a healthcare application. The cinema application allows Peter to view the movie listings and their times. The Health App allows Peter's medical team to monitor his health and involves sending medical data. The Health App is essential to be executing as Peter will be conducting an operation soon. The Health App needs internet connection so the mobile device needs to be connected either to the *Data Service* or to the *Wireless Network*.

The Cinema application performs differently when located in the cinema or outside. When the cinema application detects it is inside the cinema, it will connect to additional services. The application will connect to two extra services in the cinema: the *Video Streaming Service* which allows Peter to view trailers of listed movies at the time and the *Friends Service* to identify if any friends are in the cinema or not, and where they are located. Also, when Peter exits a Screening Room, the Cinema app will connect to a *Restaurant Service* to advice Peter of restaurants and their routes.

When Paul enters into the cinema, the cinema application on the mobile device will consume more resources (e.g., battery) due to the extra services that it needs to connect to. However, the Health App needs resources as it will need to be executing as long as possible. Therefore, based on the state of the resources at the time of entering the cinema, it should be evaluated whether the Video Streaming, Friends or Restaurant Services should be used by the Cinema App.

III. SELF-ADAPTABLE AMBIENTS SERVICE ORIENTED ARCHITECTURE AT RUNTIME

To provide support for service oriented architecture at runtime we have defined a metamodel (see Figure 1.). This metamodel represents the architectural knowledge we will maintain at runtime. The metamodel defines different kinds of ambients appropriate for mobile software as well as resources.

An ambient can *contain* different ambients inside its boundary, including an ambient tree hierarchy. An ambient can be a *Location* area, a *Device*, a folder or an *application Component*. A location area can be a house, building or a town area. For example, the cinema location can be an ambient which has different screen rooms as subambients. A mobile device is a Device ambient that can be located in a cinema location or a screen room. When a mobile device is located in the cinema, it is a subambient of the cinema (see Figure 5.).

Ambients are architectural elements. They can provide services to elements located in them and can participate in Service Contracts. Service Contracts define the terms, conditions, interfaces and choreography that interacting participants must agree. They specify how services are provided and consumed based on interactions and behaviours involving the architectural elements (ambients and components). An ambient can participate in service contracts with its siblings (ambients and components). Also, a parent ambient can participate in service contracts with elements inside it. For example, a subambient application can have a service contract with the ambient it is located in e.g., a mobile device. In this case, the parent ambient has to provide services to its subambients or other elements of its service architecture.

Ambients have resources that need to be monitored at runtime and can represent the CPU, memory, bandwidth or battery power. The resources of an ambient are calculated based on their hierarchy. Also, the resources of an ambient are shared with its sibling hierarchy. For example, a mobile device ambient that has a memory of 1GB, can have two application component ambients, e.g., a health application and the cinema application that have to share the 1GB between them.

Ambients can be mobile or not. Those that are mobile can request mobility services. For example, a mobile device is a mobile ambient. Also, ambients provide two mobility services: enter and exit. The enter service allows ambients to enter inside the boundary. The exit service allows subambients to exit parents' boundary. The execution of these services trigger the reconfiguration of the service oriented architecture. For example, when a mobile device enters into the cinema, new services are provided to the cinema application in the mobile device that were not when outside of the cinema. When the mobile device enters into a screen room, and then exits it, the cinema application client in the device can connect to the Restaurant Service.

Also, the resources that the mobile device consumes e.g., CPU and battery consumption are also affected because of entering or exiting. For example, the mobile device in the cinema can consume more battery as it will be using more services. Also, when the mobile device enters the cinema, the mobile device will be connected to the Wireless Service and therefore, this resource will change as a result of entering into the cinema.

In our approach, a software architect has to decide whether an ambient is an autonomic element that is able to manage and adapt the service architecture. In some scenarios, ambients just represent the topology of a location in the architecture e.g., a city. In other cases an ambient can self-adapt its services and applications to resources, e.g., a mobile device ambient can disable an application. Therefore, we uniquely provide in our metamodel a concept called Autonomic Ambient. Autonomic ambients are able to create/delete service channels, enable/disable applications, etc. In the next section, we describe how Autonomic Ambients are supported.

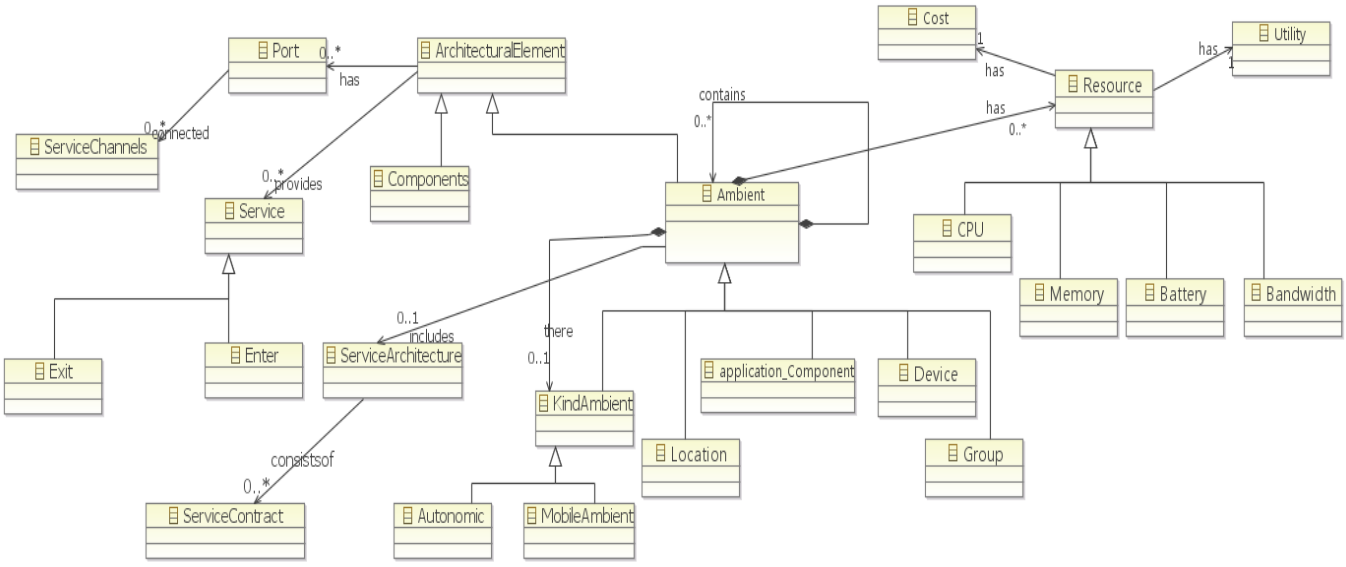

Figure 1. Metamodel for Mobile Resources Architectural Model at Runtime

IV. AUTONOMIC AMBIENTS

As stated in the previous section, ambients can be autonomic elements that can adapt how their elements can use resources. However, not all ambients are autonomic managers. Some of them just represent boundaries (see Figure 1.). To provide with autonomic capabilities, autonomic ambients follow the IBM MAPE-K (Monitoring, Analysis, Planning, and Execution- Knowledge) loop [7].

A MAPE-K Loop distinguishes between the autonomic manager that implements the MAPE-K loop and the managed element which is observed and affected by the autonomic manager. In our approach, ambients can be autonomic managers that can manage subambients or elements in their environments and coordinate them with other ambients.

An ambient new MAPE loop is executed, each time a change in the ambient hierarchy (topology) occurs triggered by a mobility capability (enter or exit). In the following, we explain each step of our adopted MAPE loop:

Knowledge: The knowledge in our approach is represented as the **Ambient-Service Oriented Architectural Model** represented with the ambients (and their tree hierarchy), services, service contracts, resources of each ambient, the service channels as represented in the metamodel (see Figure 1), runtime data e.g., the resources that need to be monitored, an approximation of the cost of using services and applications and their utility.

Monitor: Captures runtime information represented in the Ambient-Service Oriented architectural model. This includes the topology of the software architecture which consists of the hierarchy of the ambients, the connections between elements (service channels), the services, and the current values of resources represented in the model e.g., battery or bandwidth. Each time a mobility service is executed (enter or exit), the topology and configuration of the software architecture and resources of the system will change.

Analyze: In this phase, we analyze any threats/failures that can be caused to the functioning of any mobile application due to the resource limitations in the environment. As not all threats can be analyzed, the constraints defined in the service contracts are checked and compared with the actual state of the monitored architectural information at runtime in order to detect any essential threats. As resource information only exists at runtime, applying resource analysis at design would not be valid as changes in a topology or the applications in a device cannot be anticipated. In our approach, mobility of ambients (exit or enter) will cause a reconfiguration of the architecture due to changes in resources and service provision.

Plan: Identifies a suitable set of actions that are needed for adapting the software architecture in order to comply with the resource constraints or priorities. This will include planning the reconfiguration of the software architecture by removing or adding service channels, adding or removing services, etc. As an ambient selects the plan, the plan usually will affect the elements inside its boundary (its subambients, service channels, etc) or its sibling ambients (e.g., removing a service channel between it and its sibling ambient). Parent ambients act as coordinators between the other ambients and the elements inside them. Therefore, the ambient analyzes based on its current location and the resources available the best service oriented architecture configuration available. For choosing a nearly- optimal configuration, we have designed a swarm particle algorithm (see section V).

Execution: The reconfiguration plan of the service oriented architecture is executed in order to adapt which implies removing or adding services, service channels, components, etc. Before executing the plan, the state of all architectural elements affected (components, services, ambients, etc) are ensured to be safe to perform the changes.

In our approach, instead of having one centralized MAPE-K Loop, each ambient in a tree hierarchy can have a

MAPE-K Loop that can manage and coordinate the adaptation of its elements.

V. PLANNING ADAPTATION THROUGH DISCRETE SWARM PARTICLE TECHNIQUE

When mobility capabilities are executed, the software architecture of the mobile system needs to reconfigure. This reconfiguration has to satisfy the constraints and requirement specified in the service contracts and resource needs of the applications. For example, when the mobile device enters into the cinema, the Cinema App will need to connect to extra services. The provision of these services, however, will need to take into account the resources that are needed by the Health App.

When an ambient enters/exit it is able to know according to its location in the ambient hierarchy which are the reachable services (these are services provides by its parent ambient or its sibling ambients). These services are connected to applications deployed on the ambient, and which consume resources. When an ambient enters/exits the ambient has to analyze which is the best reconfiguration suitable for the resources based on the choices of combining the services.

To enable to analyze and choose the service oriented according to the resources needs, an algorithm needs to be designed. This algorithm receives as input the services that could be connected to the applications, the service contract constraints related to the resources, the current value of the resources e.g., the battery value when entering or exiting, and the resource utility and cost for each reachable service and applications deployed on the ambient.

The ambient calculates the sum of resource consumption for each service and application when connected and disconnected and compares it to the resource constraint defined in the service contract, creating different combinations of services and applications. The best combination of services and applications is the one chosen.

To analyze and plan the new configuration, a naïve or exact algorithm can search all the possible combinations of services and resources. However, this algorithm is only valid when the number of combinations is small (i.e., the number of possible services or resources to take into account) as its computational cost is O(n3). Also, the algorithm is not appropriate for searching valid combinations when there are constraints to be satisfied and there is an NP-hard problem [8]. Due to these limitations, we have decided to use optimization techniques and design a more efficient algorithm based on discrete swarm particle optimization [4]. This algorithm finds the best-optimal solution in a number of iterations by considering a set of candidate solutions which are explored by particles. Particles can change their positions according to their predisposition to change. When particle redisposition they can evaluate and compare solutions to a best selected solution and reposition them in the next iteration according to their distance to the best solution.

In our approach, we have defined a candidate solution (or configuration) as a string of bits where each bit can represent a resource, a service or a mobile application (see Table 1 for possible solutions). For example, if our problem has 2 resources, 5 services and two applications, the solution will consist of 9 bits, where the two first bits will be representing resources. Each particle will represent possible combinations, where 1 represents the selection of that bit and 0 its non-selection. Each of these solutions (or combinations) has to be evaluated based on a utility function.

The Utility Function that we have defined (see Algorithm 1) takes a possible combination and calculates the goodness of the solution. First, we check if the combination is valid (e.g., either we use the WLAN or the Data Service, not both together). If it is a valid combination, the algorithm calculates the *resourceCost* for each resource based on the selected services. If the *resourceCost* is less than the indicated cost (which is the value of the current resource e.g., the state of the battery at that time), the utility is the sum of the utilities for each services and resources in a valid combination.

Algorithm 1: Utility function

```
Function utility(bits: boolean[]) : real is
    utility = 0
    if bit combination is valid selection then
        for each resource r ∈ bits do
            if r.bit is 1 then
                resourceUtility = 0
                resourceCost = 0
                for each service s ∈ bits do
                    if r.bit is 1 then
                        resourceUtility = resourceUtility +
                        utility(s,r)
                        resourceCost = resourceCost + cost(s,r)
                if resourceCost < r.cost then
                    utility = utility + resourceUtility
                else
                    utility = utility+ 0.5*resourceUtility
    return utility;
```

Algorithm 2 presents the swarm algorithm used to choose the configuration of the service oriented architecture based on the resources. Particles are randomly initialized in the solution space. The algorithm checks whether the Utility of that position (candiadate solution) of a particle is higher than the bestUtility, if it is, then the utility of that solution becomes the bestGlobal. Particles then calculate their new position, i.e., a new candidate solution to explore. The value of bestGlobal after the iterations is the solution (configuration) chosen by the algorithm.

Algorithm 2: Discrete Particle Swarm Optimization Algorithm

Particles is a set of particles
$P_i.V_d$ is the predisposition (probability) of $particle_i$ of changing bit d to 1
$P_i.X$ is the position of $particle_i$, which is a bit array
$P_i.X_d$ is bit d in $P_i.X$
bestGlobal is the best position achieved so far by any particle, it is a bit array
$bestGlobal_d$ is bit d in *bestGlobal*
ϕ_1 and ϕ_1 are two positive random numbers which sum $VMAX$
$VMAX$ is a constant which value is 4 (Allows some probability of change using sigmoid function)

repeat
  for $P_i \in Particles$ **do**
    if $utility(P_i) > P_i.bestUtility$ **then**
      $P_i.bestUtility = \text{utility}(P_i)$
    if $utility(P_i) > bestGlobal$ **then**
      $bestGlobal = \text{utility}(P_i)$
    for $d \in dimensions$ **do**
      $P_i.V_d = P_i.V_d + \phi_1 * (P_i.bestUtility_d - P_i.X_d) + \phi_1 * (bestGlobal_d - P_i.X_d)$
      $P_i.V_d \in -VMAX \ldots VMAX$
      $P_i.X_d = \rho_{id} < Sigmoid(V_{id}(t))$
until *end condition*

Then based on the combination chosen e.g., the Health App, the Video Streaming Service, etc a plan is created. For example, enabling/disabling the applications, or creating or deleting service channels that connect applications to the services.

VI. ILLUSTRATION OF THE APPROACH THROUGH THE SCENARIO

In this section, we illustrate our approach through the scenario described in section II to allow the reader to understand its different parts.

In this example, we have five ambients: Mobile Device, which is a mobile and autonomic ambient, Cinema and Screening Room (SR), which are Location Ambients, and HealthApp and CinemaApp, which are Application Ambients. The Cinema App can be connected to the VideoStreaming and the Friends Services when the Mobile Device enters into the Cinema. When the Mobile Device enters a Cinema and exits a ScreeningRoom, the Cinema App can connect to the Restaurant Service.

Paul installs the the CinemaApp and the HealthApp in his mobile device. When he installs the CinemaApp contracts between the CinemaApp and the MobileDevice are created. The Specification of one of the contracts between the CinemaApp and the MobileDevice is the CinemaAppRestaurantService (see Figure 2.). The Contract indicates that when the Mobile Device exits the ScreeningRoom, the Restaurant Service is connected. Also, that the RestaurantService is connected only when the Mobile Device is located in the Cinema. These contracts form part of the Knowledge Base that is monitored at runtime by the Mobile Device Autonomic Ambient. We also have two other constraints: that either the WLAN or the Data Services are selected but not both, and that the HealthApp has to be selected at all times.

Service_Contract CinemaAppRestaurantService
Trigger
  `RestaurantService` **when (exit(ScreenRoom));**
Constraint
  always {ParentAmbient==CinemaLocation};
End_Service_Contract

Figure 2. Specification of CinemaAppRestaurantService Contract

When the HealthApp is installed, the Mobile Device and the HealthApp create a HAppResources Contract that indicates that the Mobile Device has to provide the HealthApp with the resource requirements. As a result, the Mobile Device creates a composite service called *PrioritizeResources* that allows it to alter the services of other applications in order to optimize the resources for HealthApp (see Figure 3.). We have used the Service Oriented Architecture Modelling Language [9] to graphically represent contacts.

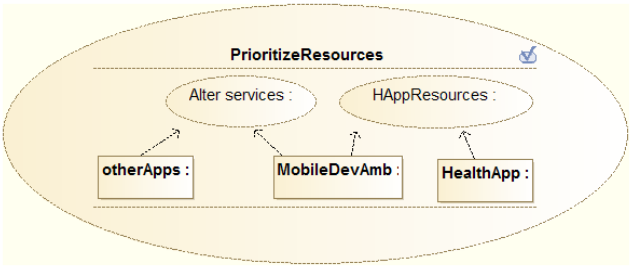

Figure 3. PrioritizeResources Composite Service Contract

Figure 4. shows the configuration of the service oriented architecture before the Mobile Device enters into the Cinema. The Mobile Device has the Autonomic Element that implements the autonomic MAPE-K Loop explained in section IV. The Autonomic element realizes the PrioritizeResources contract with HealthApp. The Cinema has the screening Room ambient and the three Services (Restaurant, VideoStreaming, and Friends) which are not reachable by the Mobile Device. The Cinema ambient, and the Mobile Device ambient provide the Enter Service which allows mobile ambients to enter into their boundary. In this case, the Mobile Device requests the enter service from the Cinema ambient. When the Cinema accepts the MobileDevice, the autonomic element will have monitored a change in the ambient hierarchy due to entering and detecting new reachable services.. Therefore, the algorithm defined in V will be executed in order to select an optimal configuration based on the resources available.

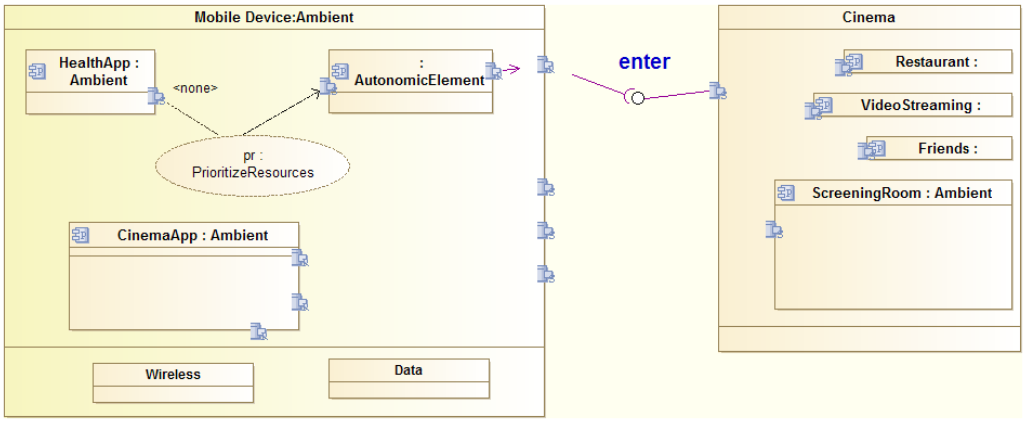

Figure 4. Configuration of the Service Oriented Architecture before entering the Cinema

TABLE I. gives examples of possible solutions that are created by the algorithm when the mobile device enters into the cinema. If you notice they consist of 5 bits, two for the resources, one for the HealthApp, one for the Video Streaming (VS) and the Friend Service (FS). The utility functions for the last two solutions in TABLE I. are going to be 0 because they are invalid combinations (the one in the fifth row has both the WLAN and Data selected and the one in the sixth row has the HealthApp unselected). For example, when the utility functions are calculated for all solutions by using the battery costs in TABLE II. and the value of the current battery is 100, the utility function results will also be 0 because the cost of these services is more than the current value of the battery. However, when the battery is 200 and WLAN is available at runtime, the chosen solution can be WLAN, HealthApp, VideoStreaming and Friends Services.

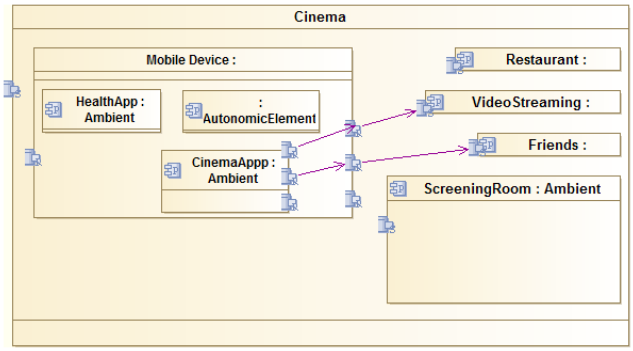

Figure 5. Configuration when the Mobile Device enters the Cinema and Restaurant and VideoStreaming are selected

TABLE I. POSSIBLE SOLUTIONS CONSIDERED BY THE ALGORITHM WHEN ENTERING INTO CINEMA

Data	WLAN	HAPP	VS	FS
1	0	1	0	0
1	0	1	1	0
0	1	1	0	0
0	1	1	1	0
1	1	1	1	1
1	0	0	1	1

The Mobile Device will then create the service channels that connect the CinemaApp to the VideoStreaming and the Friends Services (see Figure 5.). Similarly happens when the Mobile Device enters and exits the ScreeningRoom. However, when the device exits the ScreeningRoom the algorithm will create solutions based on 6 bits as Restaurant Service will be considered.

TABLE II. RESOURCE COSTS AND UTITLY

	Battery COST with DATA (mA)	BatterY COST WITH WLAN (mA)	Utility
Health App	70	50	100
VideoStreaming Service	60	60	50
Friends Service	70	50	10
Restaurant Service	50	30	10

VII. IMPLEMENTATION AND EVALUATION

To perform initial evaluations for our approach, we first implemented our metamodel in Ecore by using the Eclipse Modeling Framework (EMF) [10] (see Figure 6.).

We have also implemented our swarm particle optimization algorithm (algorithm 1 and 2) in Java and applied it to the above example and using the costs and utility in TABLE II. We conducted preliminary experiments

where we assumed that the current value of the battery is 200 mA and the length of the solution combinations is of 5 bits. We have indicated to the algorithm that the maximum number of iterations to perform is 1000. For all cases when the WLAN is available, the algorithm always selects the WLAN services. This demonstrates the accuracy of the approach.

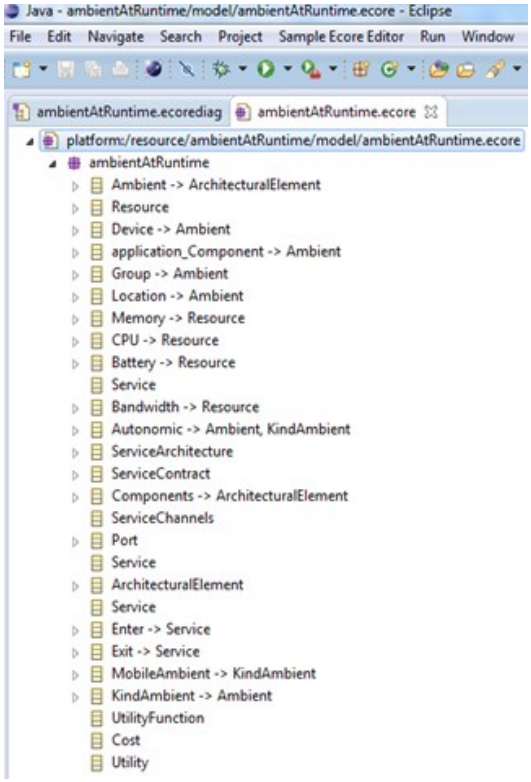

Figure 6. Metamodel implemented in Ecore

We also did experiments in order to compare the solutions obtained using the swarm particle algorithm with the optimal solutions (percentage of success), with different number of particles. Figure 7 shows results of 8 cases with different values for the number of particles, when the mobile device enters the cinema and with the objective of selecting the best configuration which is selecting the WLAN, the HealthApp, and VideoStreaming and Friends Services. We can observe that as the number of particles increases, the percentage of success increases. After using 25 particles, the number of iterations became around 50. The execution times used to find the best solution dramatically improved after 25 particles. The best execution time was 0.99 ms when 25 particles were used, with an average of 46.4 iterations and 96.4% success.

VIII. Related Work

In previous work, we developed an aspect-oriented and component based approach called Ambient-PRISMA [11] [12] where the software architecture automatically reconfigures due to agent mobility based on Ambient Calculus [3]. However, in this approach, the reconfiguration was always performed due to one single preplanned strategy. We also defined a metamodel called Ambient-SoaML to provide the design of service oriented architecture of mobile devices in a technology independent way [13] and a graphical modelling tool [14]. In all the above approaches, we did not consider the usage of the architectural model after design and did not make usage of architectural models at runtime.

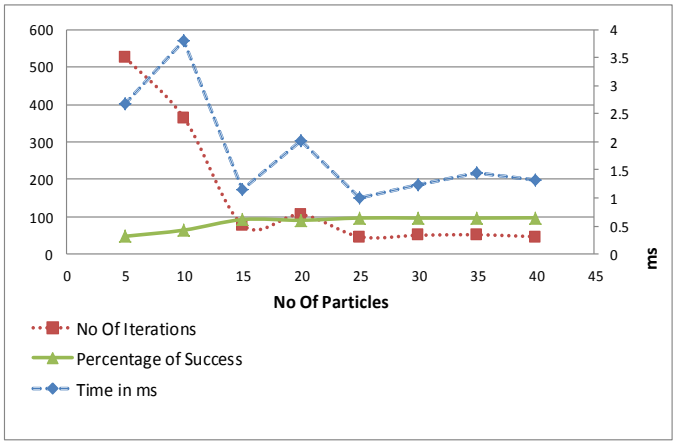

Figure 7. Experiments when incresing number of Particles

Obtaining the runtime models has many advantages: models can be used to fix design errors, to include new design decisions, for reasoning, or to observe the runtime behaviour. The usage of models at runtime have been explored for creating simple graphical user interfaces for legacy systems [15], or their use for adapting requirements models [16].

Many approaches for self-adaptation use a Variability Tree at design time e.g., Ali et al [17]. Fluch et al. [18] use architectural models that describe variability at runtime to choose among components to plug in applications. Rouvey et al. [19] also define an approach where they require the availability at runtime of all the valid configurations. None of these approaches considers self-adapting to the resources of the mobile environment. Our approach is different to the mentioned approaches as it specifically considers mobile environments as well as it does not need predefined all the valid configurations, as well as a variability model.

Gustavo et al. [20] present a software architectural approach that has the objective to self-adapt component based configurations to mobile resources. Their approach is based on calculating the differences between previous and new configurations by navigating through a variability model and using a genetic algorithm. Our approach is different in that our swarm algorithm creates the possible configurations based on the reachable services at runtime and the constraints defined in the service contracts without the need for a Variability Tree in the knowledge base. There are other several differences between our approaches: 1) Our approach

is based on a service oriented architecture that assumes that services appear and disappear at runtime, whereas they use a component based architecture that is fixed 2) We have mobility primitives and resources as first class citizens for adaptation. Since our metamodel includes mobility capabilities and ambients, our approach analyzes the configuration when mobility occurs 3) Their approach has a centralized autonomic model, whereas in ours ambients control adaptation of their own sub-architecture.

IX. CONCLUSIONS AND FURTHER WORK

In this paper we have presented a novel approach that provides support for the dynamic reconfiguration of service oriented architecture of mobile applications to the available resources, when mobility occurs. Our approach considers ambients to be autonomic elements that manage elements located in them and are able to observe and monitor the change in resources and services as mobility occurs. Ambients are able to plan the reconfiguration of the architecture by using a swarm particle optimization technique that can find the nearly-optimal configuration.

Although our initial results are promising, we still have to extend and work on many aspects of our approach. These are related to the validation, the analysis and planning of adaptation and the provision of tools for users. In terms of validation, we are working on validating our approach by considering a wider number of services and resources, real-time data and perform tests on different mobile devices. We also would like to perform additional experiments in order to understand the optimal number of iterations and particles our algorithm should go through in order to find the best solution on a mobile device.

In terms of tooling, we are extending our own graphical modeling tool presented in [14] by using the Graphical Modeling Framework and including the new primitives presented in this paper such as the resources. This will allow users to define initial configurations and generate their code. In addition, one of our main objectives is to allow software engineers to also monitor the system by using the graphical notation in order to collaborate on the decisions in planning the reconfiguration or perform changes to initial requirements. For example, users can change the utility of the resources provided at runtime.

ACKNOWLEDGMENT

This work has been partly funded by the University of Brighton Rising Star Scheme awarded to Nour Ali.

REFERENCES

[1] Wasserman, A. I., Software engineering issues for mobile application development. In FSE/SDP workshop on Future of Software Engineering Research (FoSER '10). ACM, pp.397-400.

[2] Singh, M., P., Huhus, M. N., “Service-Oriented Computing Sematics, Processes, Agents”, John Wiley & Sons, ISBN: 0-470-09148-7.

[3] L. Cardelli, “Abstractions for Mobile Computation.” In Vitek, J. and (Eds.), C. J., editors, Secure Internet Programming: Security Issues for Distributed and Mobile Objects, volume 1603 of LNCS, Springer Verlag, pp. 51-94.

[4] Kennedy, J.; Eberhart, R.C., "A discrete binary version of the particle swarm algorithm," Systems, Man, and Cybernetics, 1997. 1997 IEEE International Conference on Computational Cybernetics and Simulation.,vol.5, no., pp.4104,4108 vol.5, 12-15 Oct 1997

[5] Garlan, D., Schmerl, B. Using Architectural Models at Runtime: Research Challenges, First European Workshop on Software Architecture, LNCS 3047, Springer, 2004, pp. 200-205.

[6] N. Ali and C. Solis. 2014. Mobile architectures at runtime: research challenges. In Proceedings of the 1st International Conference on Mobile Software Engineering and Systems (MOBILESoft 2014). ACM, New York, NY, USA, 41-44.

[7] IBM. An architectural blueprint for autonomic computing, 2004

[8] Jules White, Brian Dougherty, Douglas C. Schmidt, Selecting highly optimal architectural feature sets with Filtered Cartesian Flattening, Journal of Systems and Software, Volume 82, Issue 8, August 2009, Pages 1268-1284, ISSN 0164-1212

[9] Service oriented architecture Modeling Language (SoaML) - Specification for the UML Profile and Metamodel for Services (UPMS) Revised Submission, OMG document: ptc/2009-04-01

[10] Eclipse Modeling Framework: http://www.eclipse.org/modeling/emf/docs/

[11] Ali, N., Ramos, I., Solís, C. Ambient-PRISMA: Ambients in mobile aspect-oriented software architecture. Journal of Systems and Software 83(6): 937-958 (2010).

[12] N. Ali, J. Pérez, C. Costa, I. Ramos, J. A. Carsí, Mobile Ambients in Aspect-Oriented Software Architectures. SET 2006: 37-48

[13] N. Ali and M. A. Babar, 'Modeling Service Oriented Architectures of Mobile Applications by Extending SoaML with Ambients', Proc. 35th Euromicro Conference SEAA, IEEE Computer Society, Patras, 27-29 August, 2009, 442-449.

[14] N. Ali, F. Chen, C. Solis, "Modeling Support for Mobile Ambients in Service Oriented Architecture," 2012 IEEE First International Conference on Mobile Services (MS), pp.1,8, 24-29 June 2012.

[15] Song, H., Gallagher, M., Clarke, S. Rapid GUI development on legacy systems: a runtime model-based solution. In 7th Workshop on Models@run.time (MRT '12), 2012, 25-30.

[16] Bencomo, N., Whittle, J., Sawyer, P., Finkelstein, A., Letier, E., Requirements reflection: requirements as runtime entities, 32nd International Conference on Software Engineering, 2010, pp. 199-202.

[17] R. Ali, C. Solis, I. Omoronyia, M. Salehie, B. Nuseibeh, “Social Adaptation at Runtime”, Evaluation of Novel Approaches to Software Engineering Communications in Computer and Information Science Volume 410, 2013, pp 110-127.

[18] Floch, J., et al., “Using Architecture Models for Runtime Adaptability,” IEEE Software, vol. 23, no. 2, 2006, pp. 62-70.

[19] R. Rouvoy, P. Barone, Y. Ding, F. Eliassen, S. O. Hallsteinsen, J. Lorenzo, A. Mamelli, U. Scholz:, MUSIC: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments. Software Engineering for Self-Adaptive Systems 2009: 164-182

[20] Gustavo G. Pascual, Mónica Pinto, Lidia Fuentes, Self-adaptation of mobile systems driven by the Common Variability Language, Future Generation Computer Systems, Volume 47, June 2015, Pages 127-144, ISSN 0167-739X, http://dx.doi.org/10.1016/j.future.2014.08.015.