Towards Flexible Service-aware Adaptation Management in Ambient Networks

Lawrence Cheng, Kerry Jean, Roel Ocampo, Alex Galis
University College London, Electrical Engineering Department, Torrington Place, London, WC1E 7JE, UK.
e-mail: {l.cheng, k.jean, r.ocampo, a.galis}@ee.ucl.ac.uk

Abstract - Extending the current use of the event-condition-action (ECA) logic of active databases for defining self-adaptation policy has been suggested; key research areas are adaptation policy definition, dynamic conflict detection and resolution. However, developing service-aware self-adaptation systems in heterogeneous and rapidly changing wireless networks such as Ambient Networks (ANs) is a challenging issue. This paper identifies that existing approaches suffer from the lack of a flexible management system to handle conflicting service adaptation policies; and often conflicting policies are simply ignored, which results in situations in which certain network conditions are not satisfied. In this paper, we present the Ambient Virtual Pipe (AVP) platform, that uses an Action Object Base (AOB) that enables flexible management of potentially conflicting adaptation policies. Two conflict resolution approaches, known as the AVP Action Prioritisation approach and the AVP Composition approach, are also presented and discussed.

1. INTRODUCTION

An Ambient Network (AN) [1] is composed of a set of mobile nodes (known as AN nodes) that share a common control space, known as the Ambient Control Space (ACS) (Fig. 1). The ACS is a set of control layer functions and resource representations in AN [2].

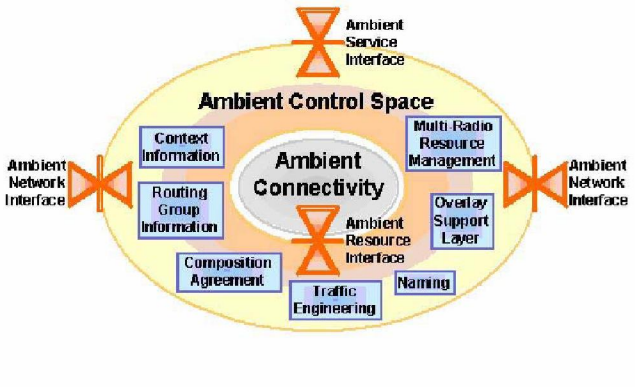

Fig. 1. Illustration of the Ambient Control Space (ACS)

The Ambient Network Interface (ANI) [7] connects ACSs of different ANs. It is through the ANI that network composition between ANs may negotiate (see later section). The Ambient Resource Interface (ARI) [17] is located inside a node, between the ACS and the connectivity layer that offers resource control mechanisms. The Ambient Service Interface (ASI) [18][19] is located between the ACS and the applications on a node. It allows applications and services to issue requests to the ACS concerning the establishment, maintenance, and termination of end-to-end service connectivity between functional instances connecting to the ASI.

Given the dynamic and heterogeneous nature of AN, one of the major challenges in AN is to develop a flexible service-aware adaptation system – that is capable of realising and self-adapting to the rapidly changing network conditions – in order to support services at run-time. To enhance flexibility, this self-adaptive system must provide an interface i.e. the ASI that allows Service Providers (SPs) to dynamically add new *service adaptation policies* for their new services. By service adaptation policies, we mean a collection of rules specifying the desired behaviour of a service [14][15]. The support for dynamic inclusion of service adaptation policies requires a flexible and manageable approach to policy management. Furthermore, the adaptation system must dynamically detect and resolve conflicts between simultaneously triggered policy actions.

Our interest in this paper is to detect and resolve application-layer conflicts i.e. conflicts between different services; rather than conflicts between component or hardware usage. We present in this paper the Ambient Virtual Pipe (AVP) platform, that is a flexible approach to service adaptation policy management. Potentially conflicting policy actions are organised in a hierarchical structure for easy inclusion of new policy actions, and easy detection and resolution of conflicting policy actions. The AVP platform adopts two flexible approaches towards policy conflict resolution i.e. the AVP Action Prioritisation approach and the AVP Composition approach. Lastly, we illustrate the enhanced level of flexibility of our resolution approaches through a scenario. As an initial attempt, the AVP platform handles service adaptation conflicts on local nodes only, we aim to address network-wide conflicts management as future work.

2. BACKGROUND

It was identified in [14][15][20][21][22] that the simple *event-condition-action* (ECA) logic of active databases can be used as the underlying technique for self-adaptation management. Essentially, the ECA approach can be used to express a policy rule: such as when an event happens, and as a result the condition returns TRUE, an action will be taken [20]. Based on the ECA approach, service adaptation policy may be defined as a service-specific policy that contains a set

0-7803-9746-0/06/$20.00©2006 IEEE

of *conditions* and their corresponding *actions*. By conditions, we mean a combination set of (network) events that are pre-defined by the SPs, which are consistently being monitored at system runtime. Once the conditions are reached, the corresponding set of reactions i.e. the actions, will be exercised.

Service-aware self-adaptation can be achieved through either implementing service adaptation policies (i.e. conditions and actions) into the service applications or the targeted systems [8][9][10][11] i.e. an *internalised* approach; or relying on a third party adaptation system that adapts accordingly by using service-specific knowledge and policies that are provided and defined by SPs [12][13][14] i.e. an *externalised* approach. Internalised approaches generally require specific self-adaptation code to be built into the applications [8][9][10][11]. This approach may be more sophisticated and self-contained, but at the expense of a higher development cost (more complex coding structure) incurred by building adaptation code into the applications, and a higher level of difficulty to add new service adaptation policies at application runtime [14]. Externalised approaches use a generalised framework through which different SPs may specify their service requirements and their service adaptation policies at application runtime, which gives a higher level of flexibility; but requires sophisticated techniques in the adaptation platform to interpret the context of service adaptation policies in order to detect and resolve potential conflicts between policies.

It should be note that, conflicts between (potentially conflicting) policy actions are created when the conflicting policy actions are actually being *executed simultaneously*. This is because, as we will discuss further in later section, all policy actions of the same nature (when executed simultaneously) are potentially conflicting. By the same nature, we mean the service-specific adaptation policy actions are of the same Service Type. For example, a SP that provides a QoS-assured delivery service may have different policies that use `tc` or DiffServ as its QoS controller respectively. These policies are potentially conflicting (i.e. use different technologies) but they would only cause problems if they were *executed simultaneously*. Thus, we believe that to enhance flexibility and dynamicity, conflict detection and resolution should be done *dynamically* by the adaptation platform at execution time, rather than requiring SPs to *pre-determine* all potential conflicts between its policies in advance (as required in [14]).

3. ADAPTATION POLICY MANAGEMENT

The AVP platform is a service-aware adaptive system that is distributed on all AN nodes, and it is responsible for creating virtual supportive overlays i.e. the AVPs in ANs, to support self-adaptation operations of (other) services in ANs. Note that the AVP platform follows an externalised adaptation approach. This is because (as explained in earlier section) externalised approaches provide a higher level of flexibility to cope with dynamic inclusion of new policies. The AN ASI is defined in the AVP platform as a set of open APIs to SPs, through which SPs can dynamically specify service adaptation policies when new services are launched, or when network condition changes. Simultaneously triggered policies are checked for conflicts at the time when they are executed, and detected conflicts are resolved by the AVP platform automatically prior to execution. In addition to support for service-aware self-adaptation, the AVP platform uses a ContextWare architecture [4][5] to monitor real-time network context information (context information would be used for overlay re-configuration [3]), and uses a programmable architecture [6] to support rapid supportive overlay creation [3][16]. Due to space limitation and the scope of this paper, we focus on the AVP's service-aware adaptation approaches. Details of the AVP platform on rapid overlay creation and the ContextWare architecture used by the AVP platform can be found in [3][16] and [4][5] respectively.

3.1 Assumptions

We assume SPs are capable of generating self-adaptation policies by using their service-specific knowledge; also, SPs must provide their own implementation i.e. the actual code of their policy actions. The AVP platform is designed to be a *generic* adaptation platform to support different types of service-aware adaptations. As such, it cannot be made responsible for defining, or creating the actual adaptation code for service-specific adaptation policies on behalf of SPs.

3.2 Adaptation Policy Definition

We define a policy to be composed of two parts: a (set of) condition(s) (`CON`), and the corresponding actions(s) (`ACT`). Because the condition(s) are service-specific, in order to maximise the level of flexibility for SPs to define their own conditions, and at the same time allowing the AVP platform to interpret the policies' context, the AVP platform requires the SP to specify their conditions using common basic data structures, relational and conditional operators e.g. `>`, `>=`, `==`, `&&`, `||`… etc. and in the form of BOOLEAN conditions. A set of conditions may be defined as one or more conditions combined through the use of relational and conditional operators. The pseudo code for an example set of conditions is shown in below.

```
(CON_1 == CON_2) && CON_3
```

3.3 Action Object Base (AOB)

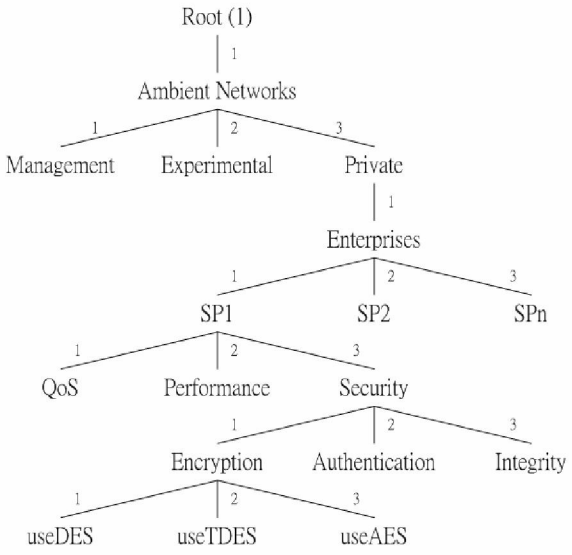

Fig. 2. An example AOB

In order to enable flexible and manageable service adaptation policy management, policy (or action) specification must be managed in a manageable format. We propose to use a tree-like data structure - known as the Action Object Base (AOB) (Fig. 2) - in which each action is considered as an action object that has an unique identification number i.e. an Action Object ID (AOID). Action Objects are references to

adaptation code developed by SPs that are to be executed when certain conditions are reached. AOIDs are organised in a hierarchical structure to allow maximum flexibility for SPs to define their own new actions (see later for discussion).

According to Fig. 2, an example AOID would be the `useTDES` (use Triple DES) action defined by `SP1` i.e. `.1.3.1.1.3.1.2`. Note that each AOID is placed in the AOB according to its *Service Type*. For example, `useTDES` is a action of the `Encryption` service of the `Security` service of `SP1`. As we will see in later sections, by categorising actions based on their Service Type, we ease the action conflict detection process.

Each SP must specify policy actions in their policies in the form of AOIDs according to the SP's AOB. Note that the AVP platform does not allow *opposite action* to be specified in an AOB. By opposite action, we mean the reverse action of an already defined action. For example, `useTDES` is a defined AOID in the above example AOB, the SP must not specify `doNotUseTDES` as another AOID. This requirement is needed to ease conflict detection (see later section). This requirement does not limit the applicability of the AOB approach, because - as will be discussed shortly - opposite action may be specified in a policy using the `NOT` operator. An example policy is shown below:

```
POLICY_1:
if       CON_1 && CON_2;
EXE:     AOID_1;
EXE:     !AOID_2;
```

This policy basically says if condition 1 and 2 (i.e. `CON_1 && CON_2`) are met, execute action 1 (i.e. `EXE: AOID_1`), but *do not* execute action 2 (i.e. `EXE: !AOID_2`). Note that the last action is defined by using the `NOT` operator.

There are several advantages of using an AOB to store actions as object references in a tree-like structure according to their Service Types. Firstly, this approach provides a way for each SP to uniquely allocate their actions according to the corresponding AOID, and thus allows *multiple* SPs to define their own actions in a way that is manageable by a third party adaptation platform (such as the AVP platform). This arrangement enables *logical separations* between *different* SPs' policies. By logical separating different SPs' service-specific adaptation policies, potential conflicts between different services provided by different SPs are avoided. Conflicts are created between actions defined by the *same* SP only; thus our approach is more manageable. Secondly, unlike the approach defined in [14], where action (or policy) organisation is not addressed, with a structured organisation such as an AOB, information on policies (or actions) can be easily updated by editing only the relevant subset of the tree (see later section on action prioritisation). For example, `SP1` can easily add a new action (e.g. `useRC4`) or other relevant information to its AOB as `.1.3.1.1.3.1.4`. This means the AVP approach is more manageable. Lastly, and more importantly, as we will see in later sections, this hierarchical arrangement of actions allows the adaptation system to quickly identify conflicts between adaptation actions.

As described, the use of AOBs enables dynamic introduction of adaptation policy actions; which in turn increases the flexibility of the AVP platform because SPs may dynamically add new adaptation policy actions to accommodate new network conditions. However, one important issue in self-adaptation is the detection of policy conflicts. Conflict detection must be performed at system runtime in the AVP platform because new policy actions may be added or executed at system runtime. Next, we will present how the AVP platform detects potential conflicts between simultaneously triggered policy actions.

4. REAL-TIME CONFLICT DETECTION

4.1 Opposite Action Conflicts

One major cause of conflicts between policies is when two opposite actions are executed at the same time, known in this paper as *Opposite Action Conflicts*. For example, suppose a SP is providing a secured AN service of which AN nodes may communicate securely (i.e. through encrypted channels). The AN is responsible for organising symmetric keys distribution within the AN. Assume initially there are two AN nodes in this AN, and both nodes support only DES. A policy (`POLICY_1`) may specify "if the number of participating AN nodes is less than or equal to five, use DES as the default encryption algorithm" [1]. The pseudo code for `POLICY_1` is listed below:

```
POLICY_1:
if       (numberOfUsers <= 5);
EXE:     useDES;

POLICY_2:
if       (AES == 1 || TDES == 1)
EXE:     !useDES;
EXE:     useAES;
```

Suppose another two new nodes wish to join the AN. These new nodes are determined to use a more secure encryption service e.g. AES, TDES (Triple DES)… etc. If these new nodes were to join the existing AN, the SP would have to add and execute a new policy (`POLICY_2`) that, says, "if either AES or TDES is available (or both), do not use DES as the default encryption algorithm, but use AES". The new policy `POLICY_2` is shown above in pseudo code. If the conditions defined in the two policies are reached at the same time – which will be the case if the new nodes join the AN, and the SP installs the new policy (`POLICY_2`) to the adaptation system and tries to execute it - there will be a conflict between the defined actions. A loop will be resulted (i.e. use DES, do not use DES… etc.). As explained earlier, conflicts (between policy actions) must be detected at system runtime to support dynamic inclusion of new policies to achieve a higher level of flexibility. The AVP platform detects opposite action detection through the AOB regulations: recalling from previous sections, opposite action cannot be specified in an AOB. Thus, an Opposite Action Conflict can be easily detected if the simultaneously triggered actions contain the same AOID, one with a `NOT` operator but not the other.

4.2 Identical Service Type Action Conflicts

This type of conflict is created when two (or more) policies of different conditions exercise different actions of the *same* Service Type. Recalling from previous sections, AOIDs are placed in an AOB according to their Service Types. The `useTDES` action is of the `Encryption` service of the `Security` service provided by `SP1`. As such, if two actions of the same Service Type are executed at the same time, a conflict may be resulted. The below example shows an

[1] This is just an example to show conflicts between different policies of opposite actions; this example does not replicate a real-life scenario. For simplicity, we assume a weak(er) encryption algorithm is acceptable if the number of participants is small.

Identical Service Type Action Conflict in pseudo code. Note that the actions i.e. `useDES` and `useTDES` are of the same Service Type (`SP1.Security.Encryption`). If the conditions of the policies are met at the same time, conflicts will be created between `POLICY_1` and `POLICY_2`, `POLICY_1` and `POLICY_3`: should the AN use DES or TDES as the default encryption algorithm?

```
POLICY_1:
if      (numberOfUsers <= 5)
EXE:    useDES;

POLICY_2:
if      (TDES == 1)
EXE:    useTDES;

POLICY_3:
if      (numberOfUsers <= 5 || DES == 0)
EXE:    useTDES;
```

Identical Service Type Action Conflicts can be easily identified in the AVP system by detecting actions of the same Service Type when policy actions are triggered. This is because we have made the necessary provisioning in the AOB definition that requires SPs to allocate their actions into appropriate places in the AOB according to their Service Types. When actions are simultaneously triggered, the AVP platform checks to see whether the actions are different but of the same Service Type. If they are different but of the same Service Type, a conflict is detected.

5. CONFLICTS RESOLUTION

5.1 Existing Approaches

The *first-in-first-to-execute approach* [14] accepts the first policy that was triggered, late comers were simply dropped. This approach is simple to implement, but at the expense of ignoring potentially more ideal policies that happen to be late comers. The *prioritisation approach* used in [15][23] to overcome conflicts between active database rules is a potential candidate solution: each SP would be required to prioritise its adaptation policies. Policies of lower priorities would be rejected. In this case, this approach would have a higher level of flexibility than the first-in-first-to-execute approach, but at the expense of a higher overhead for policy priority management (see later section).

Although both approaches are simple to implement, they have significant drawbacks. The prioritisation approach creates a scalable problem when SP creates and adds their new policies of different priorities. This is because if the assigned priorities of existing policies specified to the adaptation platforms are to be reassigned (because a new conflicting policy with a higher/lower priority is to be added, or the SP decided to rearrange the priorities due to new policies… etc.), the priorities of *all* existing policies specified by the SP to the adaptation platform must also be updated. This is largely due to a lack of an *organised* policy management system. As we will see later, this scalable issue can be largely resolved if an organised policy management structure i.e. AOB is used. More importantly, under both approaches, "disqualified" policies (either of lower priorities or simply arrive late) are simply eliminated. If adaptation policies are generated and executed in response to changes in network conditions, ignoring policies by either of the approaches means there are chances that the new network condition(s) would not accommodated.

Beside overhead and performance, we believe it is also important to realise and to enforce the actions of policies when they are to be executed, rather than simply ignoring them. As such, the AVP platform adapts a dual-technique approach, that consists of action prioritisation and composition respectively, which provides the necessary provisioning for SPs to choose which technique to use according to their preferences.

5.2 *The AVP Action Prioritisation Approach*

The AVP platform assigns priorities to policy actions with respect to their Service Types through the use of AOB in order to resolve policy conflicts. Priorities are assigned to actions by SPs. Note that the AOB requires all policy actions to be categorised into the AOB according to their Service Types. Thus, should a new action with a new priority is to be added, and as a result, the priorities of the existing actions must be updated accordingly; only the priorities of the actions of the same Service Types as the new action i.e. a subset of AOB would be updated. Unlike existing prioritisation approaches, the AVP Action Prioritisation approach is more manageable because there is no need for a *global* update on priorities of *all* policy actions. Thus, scalability is enhanced.

5.3 The AVP Composition Approach

Although the AVP Action Prioritisation approach is quick to deploy once action prioritisation is done, it may not be ideal because some policies (of lower priorities) will be dropped. Instead of dropping one ore more of the policies of lower priority, the AVP platform is designed with a facility to accommodate this kind of situation. This facility is known as the *composition* function. Through composition, the AVP platform resolves policy conflicts by creating a new AN to support one (or more) of the conflicting policy(ies). When a conflict between simultaneously policy actions is detected by the AVP platform, the to-be-added policy (and the new joining nodes) is ignored by members of the *existing* AN i.e. the new nodes will not be able to join the existing AN. The AVP platform on the joining nodes will begin the composition process, which enables the new joining nodes themselves to create a new AN to fulfil their own interest. In this case, the interest of both the existing AN nodes (in the original AN) and the new joining AN nodes (now in the new AN) are both satisfied.

Composition is a process in AN in which the ACSs of two or more AN nodes negotiate through the ANI, and results in a single control space. The process ends with a settlement on a Composition Agreement (CA) between all participating nodes. Essentially, the SP must specify - for each of its service e.g. the `SP1.Security.Encryption` service - a *Composition Agreement Template*, which will be used between participating nodes during CA negotiation. This template defines the framework for the agreement to be established between two (or more) AN nodes. This template specifies the initial policy and roles to be applied in the final concrete agreement. This template has two main distinct parts: a generic part and a AN service-specific part. The generic part consists of generic information such as Time-to-Live of the CA, and room for the identifiers of the AN nodes involved in the agreement in the form of signature (and corresponding public key certificates). The AN specific part defines management information that are required by the corresponding AN ACS Functional Entity (FE) that is involved, in this case the Overlay Support FE (OSFE) (Fig. 1). In a simple example, for a security service, the AN service-specific part of the new CA established between the new joining AN nodes would contain essential security

overlay establishment information that are agreed between the two new joining AN nodes such as key algorithm to be used, the key, key size… etc., and most important the adaptation policy. The composition process ends when an agreement on the CA is reached. Once a CA is settled between the new joining nodes, the AVP platform dynamically generate an AN based on the criteria defined in this new CA. The AVP platform uses a programmable approach [6] to rapidly generate AN to suit network needs[2]. Rapid AN creation through the AVP platform is outside the scope of this paper, but details can be found in [16]. Fig. 3 summaries the differences between the approaches.

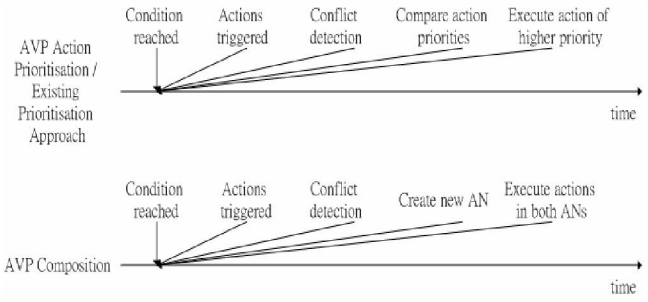

Fig. 3. Different conflict resolution approaches

6. SCENARIO

Our scenario was demonstrated in [24] using three laptops. A secure video delivery service is required by a set of AN nodes. We illustrate the differences between existing conflict resolution approaches and the AVP Composition approach in terms of *flexibility* i.e. to accommodate specific-service needs when service adaptation policy conflicts are detected in a dynamically changing wireless network.

6.1 Prototyping

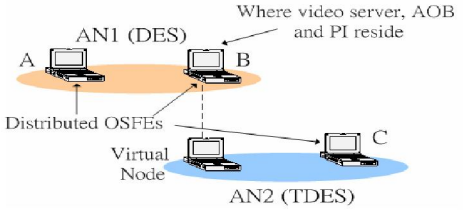

Fig. 4. AN establishment

The initial AVP prototype is built in Java, and consists of an Policy Interpreter (PI) that reads the policy inputs from the SPs, and stores the policies in an AOB (in the form of a Linked List). Each policy action defined in our example AOB was assigned with a priority. As an initial prototyping, the video server, AOB and the PI are located on one of the laptops in AN1 only (Fig. 4). The AN composition negotiation components i.e. the OSFEs are distributed on each laptop (Fig. 4).

6.2 Scenario Description

Two AN nodes (i.e. node A and B) realise `POLICY_1`. We set the policy action (`uesDES`) to a high(er) priority in our AOB.

```
POLICY_1:
if        (numberOfUsers <= 5)
EXE:      useDES;
```

[2] By rapid, we mean the AVP platform is capable of creating new (security) overlay with ~40% less in performance overhead than some traditional approaches [16].

Next, a new node (i.e. node C) that is capable of supporting TDES wishes to share the same secured video delivery service. It has a different policy (`POLICY_2`), in which its action has a low(er) priority.

```
POLICY_2:
if        (TDES == 1)
EXE:      useTDES;
```

The PI on node B adds the new policy action entry to an appropriate location in its AOB i.e. according to the action's Service Type. The two actions (i.e. `useDES` and `useTDES`) are triggered. The PI detects an Identical Service Type Action Conflict.

6.3 Analysis

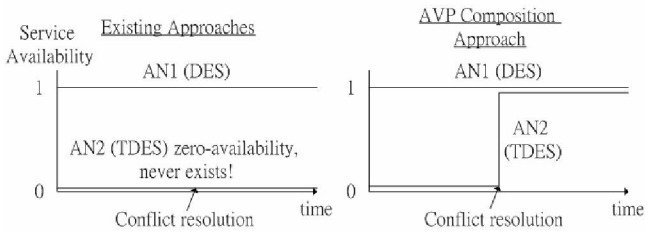

Fig. 5. Service availability under different approaches

Fig. 4 shows the resultant overlays establishment when the AVP Composition approach completes. The service availability under different approaches are shown in Fig. 5. Note that we are not justifying the performance of the approaches; but rather to highlight the differences between the approaches in terms of *flexibility* i.e. to accommodate different service needs in a changing network environment according to new and conflicting service adaptation policies. Our investigation suggests that the prioritisation approach is much simpler to deploy, but at the expense of ignoring policies that could be ideal and essential to accommodate network changes, and new joining nodes will not be able to enjoy the service. The AVP Composition approach is more *flexible* because all policies i.e. all responses to changes in network conditions are included, but at the expense of a higher level of overhead for composition.

We believe both approaches are essential, and thus the AVP platform supports both approaches. SPs may decide between prioritisation and composition. One possible criteria (besides flexibility or overhead) to be considered would be the number of participants in the AN. If the AN consists of a small number of participants, the SPs may prefer to follow the prioritisation approach for quick conflict resolution, in this case the AN would be less flexible, but only a limited number of nodes would be affected. In contrast, the composition approach may be more ideal for an AN that consists of a large(r) number of participants.

7. CONCLUSION & FUTURE WORK

In a relatively more static, smaller scale, and possible homogeneous network environment (e.g. a LAN), SPs may define all policies in advance, and resolve all potential conflicts at adaptation logic development time [8][9][10][11]. However, because wireless networks such as ANs are dynamically changing and composed of heterogeneous nodes, there is a need to provision for new service adaptation policies to be dynamically added to the adaptation system, and conflicts between simultaneously triggered policy actions to be detected and resolved at system run-time, in order to support dynamic introduction of new services and adaptation

to new network environments

As explained in this paper, a manageable approach for dynamically introducing and managing new service adaptation policies, dynamic detection of conflicts between simultaneously triggered policy actions, and flexible real-time conflict resolution are made feasible through the AVP platform. The AVP platform provides a novel and flexible approach towards manageable self-adaptation policy management by hierarchically organising potentially conflicting policy actions according to their Service Types through the use of AOB, in order to ease the process of real-time conflict detection. Additionally, the AVP platform provides two flexible approaches to overcome drawbacks of existing policy conflict resolution mechanisms. i.e. the AVP Action Prioritisation approach and the AVP Composition approach. The AVP Action Prioritisation approach makes use of the well organised structure of AOB for scalable prioritisation assignation to policy actions. Our scenario shows that the AVP Composition approach is flexible in the sense that specific service needs due to conflicts created between simultaneously triggered service adaptation policy actions and changing network conditions are not ignored but satisfied.

As part of our future work, we aim to complete development on our prototype by the end of 2006, and perform evaluation in scalability and efficiency. Currently, our approach handles conflicts on local node only. We aim to develop our approach further into a network-wide solution.

8. ACKNOWLEDGEMENT

This paper describes work undertaken in the context of the Ambient Networks (Phase 2) - Information Society Technologies project, which is partially funded by the Commission of the European Union.

9. REFERENCES

[1] The Ambient Networks (AN) Project, http://www.ambient-networks.org

[2] F. Pittmann, et al., "Ambient Networking: Concepts and Architecture", IST-2002-507134-AN/WP1/D08, http://www.ambient-networks.org/publications/D1-8_PU.pdf

[3] L. Cheng, A. Galis, R. Ocampo, K. Jean, "Self-Management in Ambient Networks for Service Composition", Intellcomm 2005. http://www.ee.ucl.ac.uk/~lcheng/Papers/INTELLCOMM_2005.pdf

[4] R. Ocampo, L. Cheng, Z. Lai, A. Galis, "ContextWare Support for Network and Service Composition and Self-adaptation", IEEE-MATA 2005. http://www.ee.ucl.ac.uk/~lcheng/Papers/MATA_2005.pdf

[5] R. Giaffreda, et al., "Ambient Networks ContextWare", IST-2002-507134-AN/WP6/D6-3, EU-IST Ambient Network Project, http://www.ambient-networks.org/publications/D6_3_Ambient_Networks_ContextWare_Second_Paper_on_Context-Aware_Networks_PU.pdf

[6] A. Galis, et al., "Programmable Networks for IP Service Deployment", ISBN: 1-58053-745-6, June 2004, Artech House Books.

[7] B. Ahlgren, L. Eggert, B. Ohlman, A. Schieder, "Ambient Networks: Bridging Heterogeneous Network Domains", PIMRC, the 16th Annual IEEE International Symposium on Personal Indoor and Mobile radio Communications, Berlin, Germany, 2005. http://www.ambient-networks.org/publications/AN_Bridging_Heterogeneous_Network_Domains.pdf

[8] J. W. Cangussu, K. Cooper, C. Li, "A Control Theory Based Framework for Dynamic Adaptable Systems", ACM Symposium on Applied Computing (ACM-SAC) 2004, March 2004.

[9] B. Ensink, V. Adve, "Coordinating Adaptations in Distributed Systems", 24th International Conference on Distributed Systems and Networks (ICDCS) 2004, March 2004.

[10] A. D. Joseph, A. F. deLespinasse, J. A. Tauber, D. K. Gifford, M. F. Kaashoek, "Rover: A Toolkit for Mobile Information Access", 15th Symposium on Operating Systems Principles, 1995.

[11] B. D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn, K. R. Walker, "Agile Application-Aware Adaptation for Mobility", 16th ACM Symposium on Operating Systems Principles, 1997.

[12] D. Garlan, S. W. Cheng, A. C. Huang, B. Schmerl, P. Steenkiste, "Rainbow: Architecture-based Self-Adaptation with reusable Infrastructure", IEEE Computer, 37(10), October 2004.

[13] W. E. Walsh, G. Tesauro, J. O. Kephart, R. Das, "Utility Functions in Autonomic Systems", International Conference on Autonomic Computing (ICAC) 2004, May 2004.

[14] A. C. Huang, P. Steenkiste, "Building Self-Adapting Services Using Service-Specific Knowledge", 14th IEEE International Symposium on High-Performance Distributed Computing (HPDC) 2005, July 2005.

[15] J. Chomicki, J. Lobo, S. Naqvi, "A Logic Programming Approach to Conflict Resolution in Policy Management", 7th International Conference on Principles of Knowledge Representation and Reasoning (KR 2000), April, 2000, http://citeseer.ist.psu.edu/cache/papers/cs/15489/http:zSzzSzwww.monmouth.eduzSz~chomickizSzpapers-kr2000.pdf/chomicki00logic.pdf

[16] L. Cheng, A. Galis, "Simple Key Exchange for Active Networks", IEEE-International Conference on Networks (ICON) 2005, http://www.ee.ucl.ac.uk/~lcheng/Papers/ICON_2005.pdf

[17] Ambient Network Consortium, "AN Framework Architecture", IST-2002-507134-AN/WP1-D05, http://www.ambient-networks.org/publications/D1_5_AN_Framework_Architecture_PU.pdf

[18] S. Schmid, F. Hartung, M. Kampmann, S. Herborn, J. Rey, "SMART: Intelligent Multimedia Routing and Adaptation based on Service Specific Overlay Networks", Eurescom Summit, 2005. http://www.ambient-networks.org/docs/SMART_Intelligent_Multimedia_Routing_and_Adaptation_based_on_Service_Specific_Overlay_Networks.pdf

[19] T. Petersen, et al., "SMART – Final Architectural Design", IST-2002-507134-AN/WP5/D03, http://www.ambient-networks.org/publications/D5_3_SMART_Final_Architectural_Design_PU.pdf

[20] N. Paton, O. Diaz, "Active Database Systems", ACM-Computing Surveys (CSUR), Volume 31, Issue 1, March 1999, pp. 63-103.

[21] S. Ceri, P. Fraternali, "Designing Database Applications with Objects and Rules: The IDEA Methodology", Addison Wesley, 1st edition, June 1997, ISBN: 0201403692.

[22] R. Agrawal, R. Cochrane, B. Lindsay, "On Maintaining Priorities in a Production Rule System", in proceedings of the 17th Conference on Very Large Databases, pp. 479-489, 1991.

[23] H. Jagadish, A. Mendelzon, I. Mumick, "Managing Conflicts between Rules", ACM-SIGACT/SIGMOD 1996, pp. 192-201.

[24] L. Cheng, R. Ocampo, K. Jean, A. Galis, "P2P Context-aware Management for Ambient Networks", demo presentation at Wireless World Initiative Symposium (WWI 2005), Paris.