Profile Aggregation and Policy Evaluation for Adaptive Internet Services[*]

Claudio Bettini Daniele Riboni
DICo, University of Milan, Italy
via Comelico 39, I-20135 Milan, Italy
{bettini,riboni}@dico.unimi.it

Abstract

Mobile and ubiquitous computing emphasize the need for highly adaptive delivery of Internet services. While several systems and even products exist that guarantee shallow or deep adaptation, they are usually based on profile information and customer relationship management modules stored and operating at the service provider. On the contrary, in this paper we assume that profile information, including user personal data and preferences, device capabilities, network bandwidth, location and other contextual information, as well as policy rules that can dynamically change this data, are provided by different sources. We describe a formal framework for aggregating this information and for solving conflicts between policy rules. We provide both a theoretical study of the properties of our techniques and a practical evaluation study obtained through a prototype implementation.

1. Introduction

Adaptation has been recognized as a central issue for current and future Internet services. Several systems and even products exist that offer shallow or deep adaptation mostly based on user profiling obtained by the analysis of past user interactions with a service. In these systems profile data are stored and managed at the service provider. Adaptation tailored to mobile and ubiquitous computing is still in its infancy and most systems are limited to transcoding services that are activated by detection of specific values in HTTP header requests. Some location based services should also be mentioned among services offering a form of adaptation.

However, in our view mobility calls for much more advanced personalisation techniques which involve both presentation and content adaptation and that can be obtained only by the use of information which is naturally distributed among different sources. This includes but is not limited to user personal data and preferences, device capabilities, device status, available bandwidth, location, local time, speed, and other contextual information, as well as past user interactions with the service, and the content of the service request.

In our framework [1] we use the word *profile* in a generalised sense, which includes the specification of the above information, and we extend CC/PP [16] to provide an adequate representation formalism.

Mobility also emphasises the need for modelling the dynamics of some of the profile attributes. For example, changes in the available bandwidth (due, e.g., to a switch to a different mobile device and infrastructure) should correspond to a change in the value of the attributes specifying the desired bitrate for streaming video services. A natural approach to the modelling of this dynamics is augmenting the profile attributes with policies; that is, rules that set or change certain profile attributes based on the current values of other profile attributes.

While distributed profile and policy specification enables the enhanced form of adaption that we envision for mobile computing, a technical problem can be easily identified. Different entities (user, network provider, device manufacturer, service provider, etc..) may provide partial and possibly conflicting profile data. For example, the location reported both by the user GPS module and by the network operator infrastructure may be inconsistent. Similarly, conflicts can arise when policies given by different entities, or even by the same entity, determine conflicting values for a profile attribute.

The contribution of this paper can be summarised as follows: (i) we analyse sources of conflicts in profiles and policies provided by different sources, and propose resolution strategies; (ii) by encoding policies, explicit values assignments, and priorities into logic programs we provide both a clear semantics for the intended model of a set of aggregated policies, and an evaluation procedure; (iii) we show experimental results performed on a prototype system.

[*] This work has been partially supported by Italian MIUR (FIRB "Web-Minds" project N. RBNE01WEJT_005).

A lot of related work exists, and will be addressed in Section 6.

The rest of the paper is structured as follows: In Section 2 we briefly illustrate the general architecture we are considering. In Section 3 we explain how profile data and policies are represented. In Section 4 we provide a solution for profile aggregation and policy conflict resolution and we show some interesting properties. Section 5 briefly reports on a prototype implementation and experimental results. Section 6 discusses related work, and Section 7 concludes the paper.

2. Architecture

In principle, profile data should include any information useful for offering a "better" response to a request; i.e., the information characterizing the user, the device, the network infrastructure, the context and the content involved in a service request.

However, as mentioned in the introduction, these pieces of information are owned by various entities located in different logical and physical places. We identified three main entities involved in the task of building an aggregated profile, namely: the user and his devices (called *user* in the rest of the paper), the network operator (called *operator*), and the *service provider*. Every entity is associated with a Profile Manager (called *UPM*, *OPM*, and *SPPM* respectively) devoted to manage profile information and policies. In particular:

- The UPM stores information related to the user and his devices. These data include, among other things, personal information, user preferences, context information, and device capabilities. The UPM also manages policies defined by the user, which can dynamically determine content and presentation parameters.
- The OPM is responsible for managing attributes describing the current network context (e.g., location, connection profile, and network status).
- Finally, the SPPM is responsible for managing service provider proprietary data including information about users derived from previous service experiences and service provider policies.

All profile manager modules also have to manage and enforce access control policies and authentication. While a detailed description of the mechanism is outside the scope of this paper, some details will be given later in this section.

Our architecture is applicable to a scenario where a set of service providers interact with a set of users, and each user may use different devices to request the same services. Service providers willing to retrieve the complete profile must query all the relevant profile managers. Figure 1 provides a general overview of the proposed architecture.

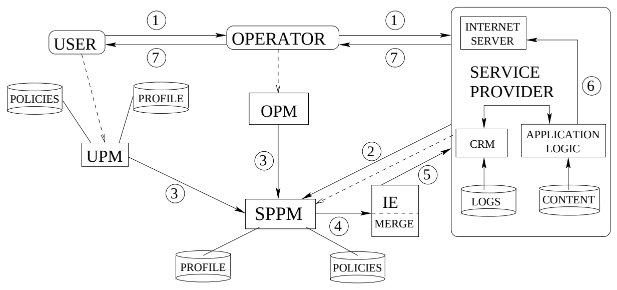


Figure 1. Information Flow upon User Request

The main steps involved in a typical service request are the following: In Step 1, a user issues a request to a service provider through his device and the connectivity offered by a network operator. In order to retrieve the profile information needed to perform adaptation, the service provider queries the SPPM (Step 2), which in turn queries the User Profile Manager (UPM) and the Operator Profile Manager (OPM) to retrieve profile data and user's policies (Step 3). Then, in Step 4 the SPPM forwards collected and local profile data and policies to the Inference Engine (IE). In Step 5, the IE first aggregates profile data; then, it evaluates service provider and user policies against the merged profile, solving possible conflicts. The resulting profile attributes are then returned to the Service Provider. These attribute values are used by the application logic to properly select content and customize its presentation (Step 6). Finally, in Step 7 the formatted content is sent to the user.

The dynamic nature of some profile attribute values claims for a mechanism for keeping up-to-date the profile information used by the service provider during a session, in order to allow the service provider to adapt the service to the new context. Our choice is to include in our architecture a trigger mechanism to obtain asynchronous feedback on specific events (e.g., available bandwidth dropping below a certain threshold, user location changed by more than 100 metres). Triggers can be defined over attributes managed by UPM and OPM. When a trigger fires, the corresponding profile manager sends the new values of the modified attributes to the SPPM module, which should then re-evaluate the profile attributes.

The architecture may be also easily extended by introducing other profile managers (e.g., profile managers owning context services) and by extending the per-user UPM model to a peer-to-peer network of UPM modules.

A very relevant issue for our architecture is privacy. The distribution of profile data is restricted by an authorization mechanism implemented at each profile manager. Authorizations specify which entities can access which components of a profile and the associated policies. For example, a set of `UPM` authorizations could allow the user client interface and user trusted agents to update personal data and policies, allow the user GPS module to update location data, and allow a set of service provider profile managers to read the profile attribute values in certain CC/PP components. Note that profile managers are considered trusted agents by their corresponding entities. We adopt a formalism for authorization rules which is very close to database access control models [15] and we also investigate extensions to classical approaches [4]. This ensures a clear formal semantics, well-known properties, and the application of materialization techniques which provide enhanced performance. Authentication and encryption issues are addressed with standard protocols.

3. Specification of Profiles and Policies

In order to aggregate profile information, data retrieved from the different entities must be represented using a well defined schema, providing a mean to understand the semantics of the data. For this reason, we chose to represent profile data using the Composite Capabilities/Preference Profiles (CC/PP) structure and vocabularies [16]. CC/PP uses the Resource Description Framework (RDF) to create profiles describing device capabilities and user preferences. In CC/PP, profiles are described using a 2-level hierarchy in which *components* contain one or more attribute-value pairs. CC/PP components and attributes are declared in RDFS vocabularies; values can be either *simple* (string, integer or rational number) or *complex* (set or sequence of values, represented as `rdf:Bag` and `rdf:Seq` respectively). It must be observed that different vocabularies can identify different attributes or components using the same name. In order to avoid ambiguities each attribute must be identified by means of its name, its vocabulary, its component, and its component vocabulary. Thus, in our framework attributes are identified using the notation

$$Vocabulary1.Component/Vocabulary2.Attribute$$

where: $Vocabulary1$ refers to the vocabulary the component belongs to; $Component$ is the ID of the component containing the attribute; $Vocabulary2$ refers to the vocabulary the attribute belongs to; and $Attribute$ is the ID of the attribute. In order to improve readability, throughout the paper the attributes syntax is simplified by omitting the vocabulary and possibly the component they belong to.

Currently, CC/PP is mainly used for describing device capabilities and network conditions; well known CC/PP-compliant vocabularies are UAProf [19] and its extensions. These vocabularies provide an exhaustive description of device capabilities and network state; however, they do not take into account various data that are necessary to obtain a wide-ranging adaptation and personalization of services. Vocabularies describing information like user's interests, content and presentation preferences, session variables, and user's context are also needed. We have been working in this direction mostly considering very confined domains, with the goal of experimenting our framework on test applications. Since a detailed discussion on vocabularies and their sharing policies is out of the scope of this paper, from now on we assume there exists a sufficiently rich set of profile attributes that is accessible by all entities in the framework.

As anticipated in the introduction, policies can be declared by both the service provider and the user. In particular, service providers can declare policies in order to dynamically personalize and adapt their services considering explicit profile data. For example, a map service provider can choose the appropriate points of interest and the resolution of the map, depending on the user's interests and his device capabilities. Similarly, users can declare policies in order to dynamically change their preferences regarding content and presentation depending on some parameters. For instance, a hypothetical user of the map service may prefer to receive image content when browsing from his desktop, while choosing audio instructions when driving his car. Both service providers and users' policies determine new profile data by analyzing profile attribute values retrieved from the aggregated profile.

As usual, the choice of a representation language is a compromise between simplicity, expressiveness, and efficiency. The policy language must also support the definition of a mechanism for handling conflicts that could arise when user and service provider policies determine different values for the same attribute. Our choice for a policy language has privileged low complexity, well-defined semantics and well-known reasoning techniques. Indeed, our policies are specified as a set of *first-order definite clauses* [18] with negation-as-failure and no function symbols, forming a general logic program. Each policy rule is composed by a set of conditions on profile data (interpreted as a conjunction) that determine a new value for a profile attribute when satisfied. A policy in our language is composed by a set of rules of the form:

$$\textbf{If } C_1 \textbf{ And} \ldots \textbf{And } C_n \textbf{ Then } A_k(V_k),$$

where A_k is a predicate corresponding to a CC/PP attribute, V_k is either a value or a variable, and C_i is either a subgoal like $A_j(V_l)$ or $not\, A_j(V_l)$. Note that the semantics of not in our rules is *negation as failure*.

For example, the informal user policy:

”When I am in the main conference room using my palm device, any communication should occur in tex-

tual form"

can be rendered by the following policy rule:

"**If** Location(MConfRoom) **And** Device(PDA)
Then *PreferredMedia(Text)* "

The language also includes various built-in comparative predicates, i.e., <, <=, >, >=, <>, ==, with their standard semantics in the domain of reals. Due to the special purpose of our logic programs, where atoms like $P(a)$ represent the fact of a being the value of the profile attribute P, we need to ensure that at most a single ground atom for each predicate must be present in the program model. This is due to the implicit assumption that each profile attribute can have a single value (even if it could be a composite value). For this reason, we have extended the syntax of general logic programs in order to declare priorities between conflicting rules (i.e., rules having the same head predicate). In particular, in our language rules are labeled, and expressions of the form $R_1 \succ R_2$ state that rule R_1 has higher priority than rule R_2. The relation $\succ$ on the sets of rules having the same head predicate is a strict partial order. Priorities are declared by the same entity that declares the policy, and if they are not given, a default ordering is used. Since priorities are introduced in the language only for managing conflicts between rules, we restrict priorities to be assigned to rules having the same head predicate (i.e., rules setting a value to the same attribute). The formal semantics of a set of policy rules and priorities is given by the unique model of the logic program in which it can be encoded (details in Section 4).

In order to facilitate the exchange of policy rules between the components of the architecture, policies are wrapped in RuleML [5], adopting the XML Schema defined for Datalog [8] with negation. Since that Schema does not allow the definition of priorities, we use the order of appearance of rules as an encoding of priorities. User friendliness issues are outside the scope of this paper, but we just mention that web based interfaces are currently used by service providers and users to insert and modify their own policies. User policies may also be taken or adapted from a library of predefined policy rules, as well as partially learned from user behavior.

4. Conflict resolution

The Inference Engine module, which receives all the profile data and policy rules collected from the profile managers, is in charge of profile aggregation and policy evaluation. In this section we explain in detail the adopted conflict resolution strategies.

4.1. Profile Aggregation

Even if no policies are given, conflicts can arise when different values are given by different profile managers for the same attribute. For example, the `UPM` could assign to the *Coordinates* attribute a certain value x (obtained through the user's device GPS), while the `OPM` could provide for the same attribute the value y, obtained through triangulation. This kind of conflict resolution is performed in our architecture by the `Merge` submodule of the `IE`. We have defined a simple language for allowing service providers to specify resolution rules at the attribute level. This means that, for instance, a service provider willing to obtain the most accurate value for user's location can give preference to the value supplied by the `UPM` while keeping the value provided by the `OPM` just in case the value from the `UPM` is missing. Priorities are defined by *profile resolution directives* which associate to every attribute an ordered list of profile managers.

Example 1 *Consider the following profile resolution directives:*

1. setPriority */* = (SPPM, UPM, OPM)
2. setPriority NetSpecs/* = (OPM, UPM,SPPM)
3. setPriority UserLocation/Coordinates = (UPM, OPM)

In (1), a service provider gives highest priority to its own profile data, and lower priority to data given by the other entities. Clearly, if no value is present in the service provider profile the value is taken from other profiles following the priority directive. Directives (2) and (3) give the highest priority to the operator for network-related data and to the user for the single Coordinates *attribute, respectively. The absence of* SPPM *in directive (3) states that values for that attribute provided by the* SPPM *should never be used.*

The semantics of priorities actually depends on the type of the attribute. When the attribute is *simple*, the value to be assigned to the attribute is the one retrieved from the first entity in the list that supplies it. When the attribute is of type `rdf:Bag`, the values to be assigned are the ones retrieved from all entities present in the list. If some duplication occurs, only the first occurrence of the value is taken into account (i.e., we apply union). Finally, if the type of the attribute is `rdf:Seq`, the values assigned to the attribute are the ones provided by the entities present in the list, ordered according to the occurrence of the entity in the list. All duplicates are removed keeping only the first occurrence.

4.2. Policy formal semantics and evaluation

Since policies can dynamically change the value of an attribute that may have an explicit value in a profile, or that may be changed by some other policies, they introduce non-trivial conflicts. They can be determined by policies and/or by explicit attribute values given by the same entity or by different entities.

4.2.1. Conflicts and resolution strategies A categorization of possible conflicts is useful for determining the system behavior. We summarize the desired behavior of the system, in the presence of possible conflicts, considering each case as follows:

1. *Conflict between explicit values provided by two different entities when no policy is given for the same attribute.* In this case, the priority over entities for that attribute determines which value prevails. This kind of conflict is totally handled by the `Merge` submodule of the Inference Engine.

2. *Conflict between an explicit attribute value and a policy given by the same entity that could derive a different value.* A simple example of a conflict of this type is the use of policies to override default attribute values when specific events occur and/or specific conditions are verified. In this case, a policy given by an entity, deriving a value for an attribute, intuitively has higher priority over an explicit value for that attribute given by the same entity. Thus, the value derived from the policy must prevail.

3. *Conflict between an explicit attribute value and a policy given by a different entity that could derive a different value.* Conflicts of this type can occur, for instance, when a provider is not able or does not want to agree with a user explicit preference, and sets up a policy rule to override the values explicitly given by the user. This kind of conflict can be taken care of considering the priority rules adopted in Section 4.1 for explicit attribute values. Considering the priority over entities for that attribute, if the entity giving the explicit value has priority over the other, then the policy can be ignored, otherwise the policy should be evaluated and if a value is derived, it prevails over the explicit one.

4. *Conflict between two policies given by two different entities on a specific attribute value.* Similarly to conflict (3), the priority over entities for that attribute states the priority in firing the corresponding rule. If a rule fires, no other conflicting rule from different entities should fire.

5. *Conflict between two policies given by the same entity on a specific attribute value.* There is no intuitive way to solve such a conflict. Hence, we assume that the entity gives a priority over these rules, using the syntax provided by the policy language, and if this is not given, a default ordering will be used. The priority over rules for that attribute is used to decide which one to evaluate first. If a rule fires, no other conflicting rule from the same entity should fire.

4.2.2. Implementing conflict resolution We now show how the conflict resolution strategies we have devised can be implemented. In the simple case when no policies are given for a certain attribute, conflicts are easily solved by the `Merge` submodule as explained above, and the resulting attribute value is directly passed to the Service Provider application logic. However, when policies are present, the resolution strategies must be integrated in the evaluation of logical rules.

A set of policy rules can be encoded in a logic program where each rule has the following form:

$$A(X) \leftarrow A_1(X_1),\ \ldots,\ A_k(X_k),\ not\, A_{k+1}(X_{k+1}),\ \ldots,\ not\, A_n(X_n) \quad (1)$$

where $A, A_1, \ldots, A_n$ are predicate symbols corresponding to profile attribute names, and $X, X_1, \ldots, X_n$ are either variable or constant symbols and denote attribute values. Note that our language allows positive and negative premises, with negative ones denoting the absence of a value for a specific attribute, but constrain the head of a rule to be positive. Moreover, safety imposes that if X is a variable appearing in the rule head, the same variable must appear in the rule body.

We ensure that the logic program corresponding to the policy rules defined by each entity is acyclic [2], by performing a simple test at the insertion of each new policy rule, and rejecting the rule if it generates a cycle. Note that this condition does not prevent rule chaining.

In addition to logic rules that encode policy rules, we have to encode in the logic program the implicit and explicit priorities that will be necessary to solve conflicts. For this purpose, a second argument, that we call *weight*, is added to each predicate, and the logic program encoding the policy rules is transformed in a program P by modifying each rule of the form (1) into:

$$A(X, \overline{w}) \leftarrow A_1(X_1, W_1),\ \ldots,\ A_k(X_k, W_k),\ not\, A_{k+1}(X_{k+1}, W_{k+1}),\ \ldots,\ not\, A_n(X_n, W_n) \quad (2)$$

where $W_1, \ldots, W_n$ are variables with values in non-negative integers, and $\overline{w}$ is a non negative integer determined by Algorithm 1. Note that rules labels as well as priorities over rules are used only in this pre-processing phase, and therefore are removed from the logic program.

The weight of a rule is defined as the weight assigned to the predicate in its head. Intuitively, rules on attributes for which a prevailing fact exists (see point 3 in Section 4.2.1) are not assigned any weight and discarded, all facts are

Algorithm 1 Setting the Weight parameter.

```
Let ({E_3{, E_2{, E_1}}}) be the priority over entities for the attribute $A$; $\mathbf{E_r}$ be the entity among $E_1, E_2, E_3$ providing the value obtained by the Merge module for $A$; $\mathbf{R_{E_j,A}}$ be the set of rules declared by $E_j$ for $A$; and $\mathbf{R_{E_j,A,k}}$ be the $k^{th}$ rule $\in R_{E_j,A}$ in increasing order of priority, according to $E_j$.

\* Facts have always weight 0 *\
Weight($Fact_A$) := 0
$w$ := 0
\* Repeat $\forall E_j,\ r \leq j \leq 3$ *\
for $j = r$ to 3 do
    $K_j := \|R_{E_j,A}\|$
    \* Repeat for each rule declared by $E_j$ on $A$ *\
    for $k = 1$ to $K_j$ do
        $w := w + 1$
        Weight($R_{E_j,A,k}$) := $w$
    end for
end for
```

given weight 0, and other rules are assigned increasing weights accordingly to priorities over entities and priorities specified by each entity.

Algorithm 1 ensures that (i) no pair of rules exist having the same head predicate symbol and the same weight; (ii) rules having the same head predicate but higher weight have higher priority, according to our conflict resolution strategy, over those with lower weights.

From a logic programming point of view we can also observe that *if the starting set of rules is acyclic, then the above transformation preserves acyclicity*. Indeed, it is easily seen that by simply adding a second argument to each predicate, independently from the value assigned to it, a cycle can exist only if one was present in the input program.

4.2.3. Evaluation The intuitive evaluation strategy is to proceed, for each attribute A, starting from the rule having $A()$ in its head with the highest weight, and continuing considering rules on $A()$ with decreasing weights till one of them fires. If none of them fires, the value of A is the one specified by the fact on A, or none if such a fact does not exist.

In order to give a standard formal semantics to our policies and to enforce the above evaluation strategy, we still need to encode in the logic program the fact that we do not allow two different values for the same attribute in the output. This means that the logic program should have a unique model and this model should contain at most one single atom for each predicate. For this purpose, program P is once more modified as follows.

Transformation 1 *Each rule (2) is modified by adding the subgoal:* $\mathbf{not\ A(Y, \overline{w}+1)}$, *where* Y *is a variable with the same domain as* $X, X_1, \ldots, X_n$, *leading to:*

$$A(X,\overline{w}) \leftarrow A_1(X_1,W_1),\ \ldots,\ A_k(X_k,W_k),\\ not\, A_{k+1}(X_{k+1},W_{k+1}),\ \ldots,\ not\, A_n(X_n,W_n),\\ not\, A(Y,\overline{w}+1). \quad (3)$$

We call P' the resulting program.

Example 2 *Consider conflicts between an explicit attribute value provided by the operator, two policies given by the same entity (e.g.; the user), and a policy given by the service provider, possibly deriving different values for the same attribute* A_1*; in this example, the user declared two policies over the same attribute, and he gave highest priority to the policy user2. Suppose that the priority over entities for the* A_1 *attribute is (SPPM, UPM, OPM). The Inference Engine preprocessor receives in input from the SPPM the following logic program:*

(op) $A_1(a) \leftarrow$
(user) $A_3(b) \leftarrow$
(p1-user) $A_1(X) \leftarrow A_2(X)$
(p2-user) $A_1(X) \leftarrow A_3(X)$
(p1-sp) $A_1(X) \leftarrow A_4(X)$
p2-user $\succ$ *p1-user*

The fact (op) represents the value provided by the OPM for A_1*, The fact (user) represents the value provided by the UPM for* A_3*, the first policy (p1-user) and the second policy (p2-user) are declared by the user, and the last policy (p1-sp) is declared by the service provider. Applying the Algorithm 1, the lowest weight (0) is assigned to the facts. The UPM has higher priority over the OPM, and so the preprocessor, following the priorities defined by the user over his rules, gives weight 1 to the head of the user policy with lowest priority (p1-user) and weight 2 to the head of the policy (p2-user). Finally, the highest weight (3) is assigned to the head of the policy (p1-sp), as it was declared by the entity with highest priority (the service provider in this case). Note that, if the OPM had highest priority than the UPM and SPPM, no rule would have been assigned any weight, and hence all rules would have been discarded.*

Hence, the above logic program is modified as follows:

(op) $A_1(a,0) \leftarrow not\ A_1(Y,1)$
(user) $A_3(b,0) \leftarrow\ not\ A_3(Y,1)$
(p1-user) $A_1(X,1) \leftarrow A_2(X,W),\ not\ A_1(Y,2)$
(p2-user) $A_1(X,2) \leftarrow A_3(X,W),\ not\ A_1(Y,3)$
(p1-sp) $A_1(X,3) \leftarrow A_4(X,W),\ not\ A_1(Y,4)$

In this case, the value of A_1 *is determined as* b *by the firing of rule (p2-user).*

We now show an essential and non-trivial property of Transformation 1.

Theorem 1 *Given an acyclic program P with weights assigned by Algorithm 1, the logic program P' obtained by Transformation 1 is acyclic.*

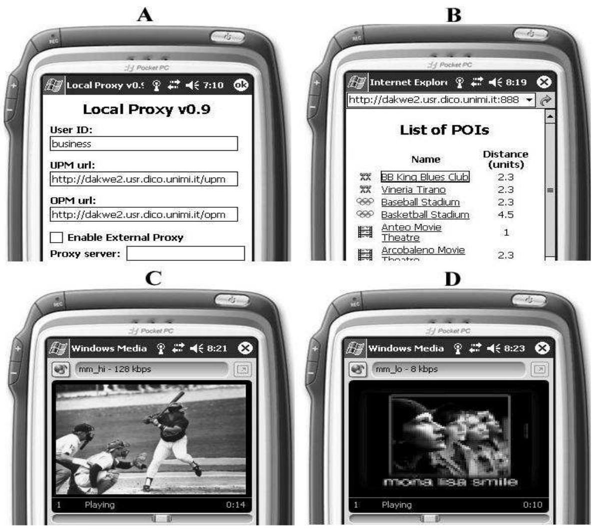
Since acyclic logic programs are a subclass of locally stratified programs [2], the program P' is locally stratified.

Despite these formal properties guarantee the uniqueness of the intended model, and hence provide a clear semantics to our prioritized rulesets, they do not guarantee in general an efficient evaluation procedure. However, in our case a direct evaluation algorithm can be devised that is linear in the number of rules, since each rule has to be evaluated only once. The algorithm is not illustrated here for lack of space, however, in Section 5 we show that the ad-hoc Inference Engine we developed can evaluate our rulesets in linear time. Moreover, since – for most internet services – adaptation will probably be performed considering a small subset of CC/PP attributes, we chose to adopt a form of goal-driven reasoning, implementing our Inference Engine using a backward-chaining approach. Hence, the cost of the evaluation will be generally less than linear in the size of policies, since a number of irrelevant rules will be ignored.

Despite we proved that all the transformations we applied preserve acyclicity, and acyclicity of rules is guaranteed in each entity ruleset, it is still possible that a cycle is created when policies are joined. The presence of such a cycle can be tested quite efficiently as a preprocessing step before rule evaluation.[1] However, for efficiency reasons, this test can also be integrated in the evaluation algorithm. Once a cycle is recognized, different strategies can be adopted. Our approach is to disregard the rules which were involved in the cycle, and continue the evaluation.

5. Prototype and experimental results

We have already developed part of the architecture [1]; while a description of the software implementation is out of the scope of this paper, we will briefly describe a prototype web application we developed for experimenting on our framework. The goal was to build a testbed, in order to evaluate the effectiveness of the profile integration mechanism against a real service. The web application is a location-based micro-portal addressed to tourists visiting a imaginary town. The portal shows an ordered list of *Points Of Interest* (POIs), i.e. hyperlinks that point to a web page providing multimedia information of a physical place. The list of POIs and the multimedia content to be delivered are selected by taking into account profile data such as the user location coordinates, device capabilities, network conditions,



Local Proxy v0.9

User ID: business

UPM url: http://dakwe2.usr.dico.unimi.it/upm

OPM url: http://dakwe2.usr.dico.unimi.it/opm

Proxy server:

http://dakwe2.usr.dico.unimi.it:888

List of POIs

Name	Distance (units)
BB King Blues Club	2.3
Vineria Tirano	2.3
Baseball Stadium	2.3
Basketball Stadium	4.5
Anteo Movie Theatre	1
Arcobaleno Movie Theatre	2.3

mm_hi - 128 kbps

mm_lo - 8 kbps

Figure 2. Some screenshots of the prototype.

context information, as well as user and service provider policies. We will now consider the case of John, a business man browsing using his PDA while moving across the city. John's PDA has a local proxy installed and running that will attach the URL of the profile managers to each service request (see Fig.2-A). When he connects to the portal, the list of POIs is shown (see Fig.2-B). The micro-portal continuously adapts to the context changes, showing different POIs and multimedia content. These changes are driven by the IE upon the evaluation of policies declared by both John and the service provider; policies generate different directives based on the different integrated profile obtained from the Merge module. As an example, following John's interests, at first on top of the list of POIs appear the "Baseball Stadium", and the multimedia content is delivered in high-resolution, since John happens to be in a Wi-Fi hot spot of its provider (see Fig.2-C). In the afternoon the weather gets worse and thus, applying John's policy, the application logic brings on top a list of indoor spots. John selects the nearest movie theatre (that appears on top of the list), and trailers are delivered in low resolution since he is now only covered by a GPRS network service (see Fig.2-D).

Given the simple nature of the prototype scenario, adaptation was performed considering a small number of rules, and essentially no appreciable delay due to conflict resolution could be observed in response time. In order to estimate the feasibility of the evaluation of policies based on logic programming for more sophisticated services, we performed an experiment using the ad-hoc Inference Engine we developed and artificial rulesets of various cardinalities. Each rule in the rulesets had three random subgoals, one

1 Detection of cycles in a directed graph G can be performed in time $O(N + V)$ where N and V are respectively the number of nodes and arcs in G [10].

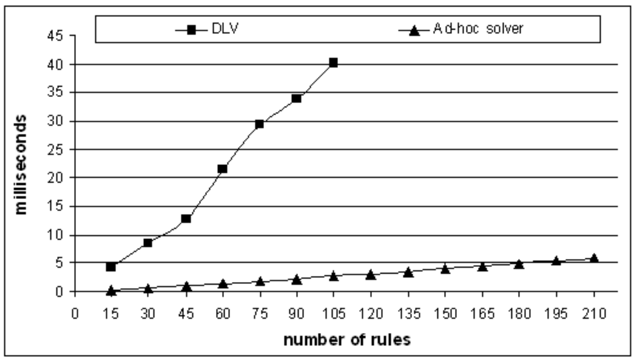


Figure 3. Rulesets evaluation time.

of which was negative. For each attribute, each ruleset contained three conflicting rules. Rulesets were properly built in order to avoid recursion, and to allow a random rule to fire for each set of conflicting rules over an attribute.

Figure 3 shows experimental results of executing the rulesets on a two-processor Xeon 2.4 GHz workstation. Evaluation times are averages of ten runs, each using a different random ruleset: Results show that a ruleset of 45 rules is evaluated in around 1 millisecond, while a ruleset of 180 rules is evaluated in around 5 milliseconds. It should be noted that evaluation times exhibit a linear increase with the number of rules in the ruleset. In order to compare our Inference Engine with a widely adopted solver, we performed the same experiments using the *DLV* system [17] (rules syntax has been slightly modified in order to be acceptable by the DLV parser). As expected, evaluation times are considerably higher with *DLV*, since the class of logic programs it considers (i.e., *Disjunctive Datalog* programs) is more complex than ours.

In conclusion, rulesets evaluation time seems to be acceptable for the class of services we consider; Therefore, we expect that response time will be dominated by the network latency.

6. Related Work

With respect to the issue of integrating multi-source profile data, our approach is similar to the one adopted by DELI [7] and Intel CC/PP SDK [6]. These frameworks adopt the profile aggregation approach of UAProf [19], consisting in associating a *resolution rule* to every attribute. Whenever a conflict arises, the resolution rule determines the value to be assigned to the attribute by considering the order of evaluation of partial profiles. This corresponds to assigning priorities to partial profiles, as opposed to assigning priorities to single attributes as we do. Furthermore, since resolution rules are defined in the CC/PP vocabulary, service providers cannot have control over the aggregation mechanism. Our approach to profile aggregation (see Section 4.1) overcomes the above mentioned limits providing, in our opinion, a more flexible and powerful aggregation mechanism. Moreover, policies are not considered in these frameworks.

Considering work on policy conflict resolution, we should mention the $\mathcal{PDL}$ language and the *monitor* concept introduced in [9]. $\mathcal{PDL}$, as many other policy languages, is based on the event-condition-action paradigm, however its semantics is given in terms of nonmonotonic logic programs as in our approach. An interesting extension of $\mathcal{PDL}$ that allows the specification of preferences regarding the application of monitors is proposed in [3]. However, with respect to $\mathcal{PDL}$ and its extensions we allow chaining of rules since we believe this is essential in our context to enable composition of policies specified by different entities.

A possible approach for implementing a form of prioritized conflict resolution is to adopt PLP [11], a Prolog compiler for logic programs with preferences. Programs are compiled by PLP into regular logic programs, that is the class of logic programs in which we encode our policies. However, the need of a Prolog compilation phase poses major problems in terms of response time, especially considering that policies can dynamically change and thus the compilation should be performed run-time.

An interesting class of engines is the one of *production rules systems* (e.g., engines based on the RETE algorithm [12] such as OPS-5, CLIPS, and Jess). These engines encode a built-in conflict resolution strategy that in certain systems can be modified. For instance, in Jess rules can be prioritized, and default conflict resolution strategies can be overridden. However, the use of priorities is discouraged, since it can have a negative effect on performance. Furthermore, production rules adopt the forward-chaining approach, which do not seem to be optimal in our case, as explained in Section 4.2.3.

Datalog engines like DLV and Mandarax would be suitable for evaluating our rulesets after the preprocessing phase (Transformation 1 in Section 4.2.3). Nevertheless, the experimental results shown in Section 5 confirm the intuition that even an optimized Datalog engine is slower than an ad-hoc implementation, since the restrictions we impose to our rulesets can be profitably exploited for improving the evaluation time.

A mechanism of rule evaluation against user context data similar to ours is adopted in the Houdini framework [14]. The focus of Houdini is on providing user context information to service providers while preserving the privacy of data. While the policy rule languages have similar expressiveness, requiring acyclicity but allowing rule chaining, in Houdini rules are declared by the user only and evaluated by a proper user-trusted module. Being primarily focused on adaptation, our policy mechanism is different: policies are declared by multiple entities in order to determine customization parameters, and our focus is on conflict resolution strategies.

The IBM CommonRules project investigates rule-based business processes for e-commerce and is based on *courteous logic programs* [13] which are closely related to the ones in which we encode our policy rules. However, due to the complex application domain addressed in that project, their rule language is more expressive and evaluation cannot be achieved in linear time as in our case.

7. Conclusions

In this paper we addressed the problem of profiles and policies conflict resolution in a framework devoted to support a comprehensive adaptation and personalization of internet based services in a mobile environment. We are extending this work in several directions. One of these concerns the issue of privacy policies and their enforcement by the profile managers which is particularly critical for the success of the framework. Other interesting issues regard intra-session adaptation and related techniques for partial re-evaluation of policy rules.

References

[1] A. Agostini, C. Bettini, N. Cesa-Bianchi, D. Maggiorini, D. Riboni, M. Ruberl, C. Sala, and D. Vitali. Towards highly adaptive services for mobile computing. In *Proceedings of IFIP TC8 Working Conference on Mobile Information Systems (MOBIS)*, 2004.

[2] K. R. Apt and M. Bezem. Acyclic programs. *New Generation Computing*, 9(3/4):335–365, 1991.

[3] E. Bertino, A. Mileo, and A. Provetti. Policy monitoring with user-preferences in PDL. In *Proceedings of Workshop on Nonmonotonic Reasoning, Action, and Change (NRAC'03)*, pages 37–44, 2003.

[4] C. Bettini, S. Jajodia, X. Wang, and D. Wijesekera. Provisions and obligations in policy rule management. *Journal of Network and Systems Management*, 11(3), 2003.

[5] H. Boley, S. Tabet, and G. Wagner. Design rationale of RuleML: A markup language for semantic web rules. In *Proceedings of the International Semantic Web Working Symposium (SWWS)*, pages 381–401, 2001.

[6] M. Bowman, R. D. Chandler, and D. V. Keskar. Delivering customized content to mobile device using CC/PP and the Intel CC/PP SDK. Technical report, Intel Corporation, 2002.

[7] M. Butler, F. Giannetti, R. Gimson, and T. Wiley. Device independence and the web. *IEEE Internet Computing*, 6(5):81–86, September-October 2002.

[8] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer-Verlag, 1990.

[9] J. Chomicki, J. Lobo, and S. A. Naqvi. Conflict resolution using logic programming. *IEEE Transactions on Knowledge and Data Engineering*, 15(1):244–249, 2003.

[10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGrawHill, 1990.

[11] J. Delgrande, T. Schaub, and H. Tompits. A framework for compiling preferences in logic programs. *Theory and Practice of Logic Programming*, 3(2):129–187, 2003.

[12] C. L. Forgy. RETE: A fast algorithm for the many pattern/many object pattern matching problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[13] B. Grosof. Prioritized conflict handling for logic programs. In *Proceedings of the International Logic Programming Symposium (ILPS)*, pages 197–211, 1997.

[14] R. Hull, B. Kumar, D. Lieuwen, P. Patel-Schneider, A. Sahuguet, S. Varadarajan, and A. Vyas. Enabling context-aware and privacy-conscius user data sharing. In *Proceedings of the 2004 IEEE International Conference on Mobile Data Management*, pages 187–198. IEEE, 2004.

[15] S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems*, 26(2):214–260, 2001.

[16] G. Klyne, F. Reynolds, C. Woodrow, H. Ohto, J. Hjelm, M. H. Butler, and L. Tran. Composite Capability/Preference Profiles (CC/PP): Structure and vocabularies 1.0. W3C Recommendation, W3C, January 2004. http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/.

[17] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. Technical Report cs.AI/0211004, arXiv.org, November 2002.

[18] P. Norvig and S. Russell. *Artificial Intelligence. A Modern Approach*. Prentice Hall Series in Artificial Intelligence, 2003.

[19] OpenMobileAlliance. User agent profile specification. Technical Report WAP-248-UAProf20011020-a, Wireless Application Protocol Forum, October 2001. http://www.openmobilealliance.org/.