Journal of
Sensor and Actuator Networks

Article

User-Generated Services Composition in Smart Multi-User Environments

Vincenzo Catania, Giuseppe La Torre, Salvatore Monteleone, Daniela Panno and Davide Patti *

Department of Electrical, Electronic, and Computer Engineering, University of Catania, V.le A. Doria, 6, 95125 Catania, Italy; vincenzo.catania@dieei.unict.it (V.C.); giuseppe.latorre@dieei.unict.it (G.L.T.); salvatore.monteleone@dieei.unict.it (S.M.); daniela.panno@dieei.unict.it (D.P.)

* Correspondence: davide.patti@dieei.unict.it; Tel.: +39-095-738-2385

Received: 6 July 2017; Accepted: 30 August 2017; Published: 1 September 2017

Abstract: The increasing complexity shown in Smart Environments, together with the spread of social networks, is increasingly moving the role of users from simple information and services consumers to actual producers. In this work, we focus on security issues raised by a particular kind of services: those generated by users. User-Generated Services (UGSs) are characterized by a set of features that distinguish them from conventional services. To cope with UGS security problems, we introduce three different policy management models, analyzing benefits and drawbacks of each approach. Finally, we propose a cloud-based solution that enables the composition of multiple UGSs and policy models, allowing users' devices to share features and services in Internet of Things (IoT) based scenarios.

Keywords: user generated services; smart environments; IoT

1. Introduction

The possibility for everyone to create custom content and services, in a simple and reliable way, has been introduced in recent years and is currently supported by a plethora of web based platforms, such as IFTTT [1], OpenMashup [2], and APIANT [3] (formerly known as We-Wired Web). These solutions carry out the composition of services with two different approaches: (i) by aggregating data, existing services, and user's defined tasks; and (ii) by simply creating new views of existing content. The results of these compositions can be referred to as User-Generated Services (UGSs) and User-Generated Content (UGC), respectively [4,5].

The Internet of Things (IoT) vision [6], which emerged in recent years due to a convergence of different communication/computing technologies, is leading to the diffusion of everyday-life Internet-capable objects, such as smartphones, Smart TVs, and health monitoring devices. Thus, IoT represents the ideal ecosystem for composite UGSs and UGC, making users able to generate new content and services not only from scratch, but also gathering data from objects and the environment [7–9]. This process, however, may give rise to several security issues due to the composition of the service-related Access Control Policies (ACP). For example, when an application developer assembles existing services that have their own security policies, he must ensure that the resultant policies do not conflict with the existing ones [10].

This scenario highlights the increasing importance that User-Generated Content and Services are going to assume in the short term. In particular, it is important to outline that there are substantial differences between traditional Web Services based on the Service-Oriented Architecture (SOA) paradigm and User-Generated Services, even more when the latter are integrated in an IoT scenario [11]. In fact, while the first ones are "always on" and regulated by access control policies that usually do not need to be updated frequently, the second ones may be subjected to more frequent changes due to users' actions and context. Let us consider, for example, social networks: when users perform common tasks such as adding/removing content (photos, videos, etc.) or updating the list of friends, the related policies change accordingly. Therefore, we can affirm that, unlike what happens for classical services, policies associated with UGC or UGSs may change very frequently. In addition, the access to a UGS may be regulated by pieces of contextual information connected to the service owner.

Let us consider the following example of context-dependent access control policy: a user belonging to the group "colleagues" can automatically access my gallery via bluetooth when all the following rules are satisfied: (i) My battery charge level is at least 35%; (ii) I'm at my workplace; and (iii) I'm not driving. It is easy to imagine that the user's context may change very rapidly and in a way that implies the continuous change of the current active policy for a device. This is just another example showing how UGSs represent a new class of services that substantially differ from traditional ones.

In this paper, which extends our prior work [12], we first analyze how the composition of services whose access control depends on context-aware policies further complicates the policy enforcement on the resulting composite service. Then, we define three models of policy management that capture the different possibilities available for UGSs. Finally, we demonstrate how a cloud-based approach can be exploited to support such policy models using commonly available open-source components and standard web technologies. The implementation of this approach has been built on top of *webinos* [13] platform, to which the authors contributed.

2. Reference Scenario

As the number of devices such as smartphones, tablets, in-car-devices and so on, increases, the whole electronic industry ecosystem is switching from "multi-user per device" to "multi-device per user" environments. Adopting the terminology and concepts we introduced in the *webinos* project [7], the set of devices associated with a given user can be considered as virtually contained (registered) inside the user's *Personal Zone* (PZ) and might also be shared with others, that is, registered in multiple PZs.

Figure 1 shows two PZs belonging, respectively, to the users Alice and Bob. Alice's PZ contains a smartphone and a camcorder while Bob's contains only a Smart TV. Despite a single device that could be "shared" among several PZs, it is owned by a unique user that is responsible for its policies and whose management may also be delegated to other users/entities. An example policy for Alice's PZ could be the following:

Alice's relatives are allowed to see her smartphone's gallery pictures on their Smart TV when she is connected through Wi-Fi.

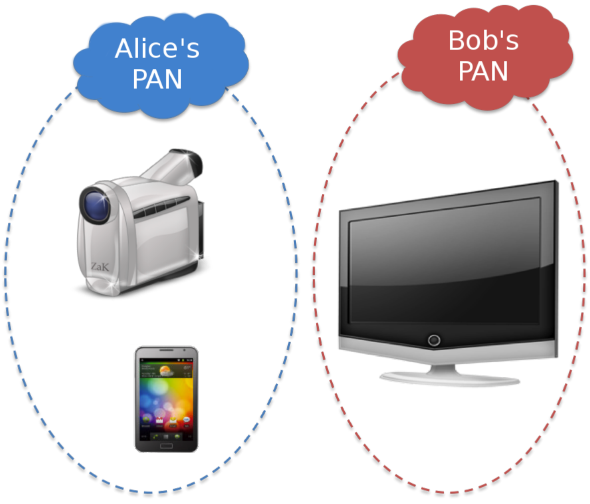

Figure 1. A simple representation of Personal Area Networks.

Each user can set policies for devices registered in his/her PZ. In the case that a device is shared among multiple users (i.e., PZs) and its behavior is regulated by multiple policies (e.g., one for each user), the one written by the owner takes priority over the others. For the sake of simplicity, in the following, we will use the name of the PZ owner to refer to a generic device registered in his/her PZ. For example, a request from Carol is to be intended as a request coming from one of the devices registered in her PZ.

Many different issues could arise when services regulated by context-aware policies are composed. These issues may be experienced for instance considering the scenario depicted in Figure 2, in which services *Videostream* and *Screenshot* are provided by Alice to Bob and by Bob to Carol. In general:

- *Videostream* allows users to receive a video stream from Alice's camcorder/smartphone;
- *Screenshot* allows users to take snapshots of contents reproduced on Bob's Smart TV.

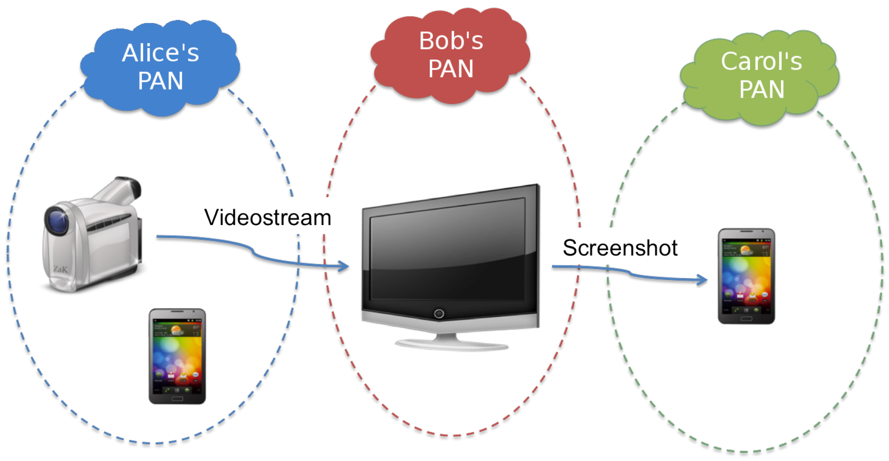

Figure 2. An example of User-Generated Services composition: the *Screenshot* service can make use of the *Videostream* service to provide content.

Let us consider the following:

Example 2.1. Alice's policies for *Videostream* service are:

1. Alice's parents are always allowed to use *Videostream* service;
2. Alice's relatives are allowed to use *Videostream* service only when Alice is in Catania;
3. Every other subject is not allowed to use *Videostream* service.

Moreover, Bob's policies for *Screenshot* service are:

1. Only Bob's house guests are allowed to use *Screenshot* service;
2. Every other subject is not allowed to use *Screenshot* service.

Both Alice's and Bob's policies specify contextual constraints: Alice is willing to offer her service toward her relatives only if she is in Catania (so this policy depends on Alice's context), and Bob's policy instead allows only people who are inside Bob's house (so this policy depends on subject's context). Let us imagine that Bob invites Carol (Alice's aunt) to his home to watch a video and suppose that Carol takes some screenshots of this video with her smartphone using the Smart TV provided service. When Bob's Smart TV receives a screenshot request from Carol, it checks its policies and allows Carol's smartphone to access the service, according to Bob's policy. This scenario might raise some problems if Carol tries to take screenshots of content owned by Alice. In this case, even supposing that Bob (his Smart TV) is aware of Alice's policies at the time that a request from Carol is received, it is not possible to enforce the request if Alice's context information (i.e., location) is not available. In fact, Alice may consider her location as a "private" information or it could not be accessible at that specific moment.

If Bob decides to allow a Carol's request only enforcing his policy (regardless of Alice's decision), the situations listed in Table 1 may occur:

Table 1. Potential situation of conflict between Alice's policy and actual data sharing.

#	Alice's Context	Alice's Policy	Conflict?
1	A is in Catania	C is A's relative	NO
2	A is in Catania	C isn't A's relative	YES
3	A isn't in Catania	C is A's relative	YES
4	A isn't in Catania	C isn't A's relative	YES

According to Alice's and Bob's policies, case 1 is the only that does not cause any conflict between Alice's and Bob's authorization decisions. On the contrary, the conditions considered in cases 2, 3, and 4 lead to conflicting situations that Bob does not take into account Alice's policies and related context information at the enforcement time.

This is just one example of the issues that may arise when multiple UGSs are composed. In the rest of the paper, we will analyze these issues and propose different solutions to cope with them.

3. Security Policy Issues in UGS Composition

The behavior of IoT Services (either traditional or user-generated) is regulated by security access control policies that restrict the access to services to external subjects, represented by applications or other services. In the UGSs scenario, when a user assembles existing services, he/she has to guarantee that the policy of the composite service is coherent with policies of all the other involved services (the component services). For example, suppose that Bob wants to create a new service making use of another one provided by Alice. When Carol requires access to this new service, Bob cannot decide whether to allow or deny Carol's request only evaluating his policy: he also should know Alice's decision toward Carol's request.

In order to avoid ambiguity, it is necessary to define the notation we adopted to describe our case studies.

Let D be the set of service providers d_i, for example, in the considered scenario, the devices of an IoT ecosystem. Let S_i be the set services $s_{i,j}$, where $s_{i,j}$ is the j-th service provided by d_i. Furthermore, let P_i be the set of policies $p_{i,k}$, where $p_{i,k}$ denotes the set of policies for all the services provided by the device d_i. Finally, let $C_{i,j}$ be the set of the *component services* used to produce the *composite service* $s_{i,j}$.

For example, Figure 3 shows a graph representation for a user-generated service composition where:

- Each node is a service provider, e.g., a device owned by a user providing/consuming services;
- Each edge from d_x to d_y indicates a service provided by d_x that is requested by d_y.

Note that $s_{2,1}$ is a composite service having $s_{1,1}$ as component service, that is, $C_{2,1} = \{s_{1,1}\}$. In particular, three main aspects must be considered in the UGSs composition with regard to the access control policy. These aspects are:

- *Dynamicity*: service related policies may change frequently;
- *Context Awareness*: policies may depend on contextual information (e.g., user's context);
- *Degree of Privacy*: users could want to set restrictions on the disclosure degree of their policies towards other subjects (users/services).

These features that characterize UGSs composition may cause new issues to arise as compared with the traditional service composition.

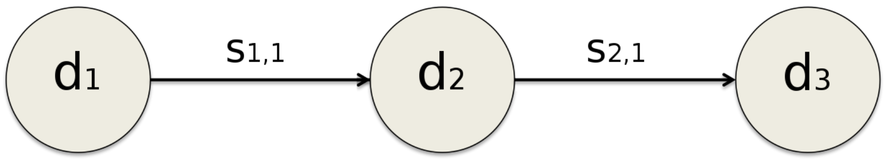

Figure 3. Graph formal representation of User-Generated Services (UGSs) composition.

3.1. Dynamicity

In fast changing environments, to promptly react to policies' changes, a security policy infrastructure with the following features is required:

- It should be able to update/synchronize composite services' policies when component services' policies change. For example, referring again to Figure 3, if the policies related to service $s_{1,1}$ change, it is necessary to synchronize the policies of all composite services that have $s_{1,1}$ as a component service (in this case only $s_{2,1}$);
- It should define efficient strategies to ensure that any change in component services' policies does not lead to conflicting authorization decisions taken by the security policy of the composite service.

In this perspective, already discussed techniques based on static composition of policies such [10,14] may not be suitable for highly dynamic environments.

3.2. Context Awareness

The use of policies referring to contextual information further complicates the authorization decision process to grant access to composite services. In the policy example related to Figure 2, Bob can take a conflict-free decision against a Carol's request only if he knows Alice's context. However it is not available to Bob since only Alice knows her context and it may be changed at the time of Carol's request. Contextual policies, therefore, add additional difficulties in user-generated service composition. In particular, the policy manager of the composite service provider needs to know remote contextual information from which the component services' policies depend on. We call this "context synchronization problem".

3.3. Degree of Privacy

UGSs composition can lead to policy disclosure. To avoid this, the service provider that assembles a composite service, referred in the following as *intermediate provider*, has to ensure that its policy is compliant with all the component services' policies. This is possible only if he knows all the policies that represent the behavior of each component service toward all the possible subjects. In many cases, especially in the UGS scenario, this could be a serious privacy problem since policies contain sensitive information that the owner of the component service might want to disclose partially (partial privacy) or not at all (full privacy) to other users. If all the component services adopted full privacy on their policies, the intermediate service provider would be unable to ensure any compliant authorization decision against a composite service's access request. Similar problems might arise also in the case of partial privacy. In addition, contextual user information could be in turn under partial or fully privacy restrictions, as it often refers to sensitive user's data.

4. Proposed Policy Models

By definition, UGSs are intended as composed services provided by different users. This implies that one of the main issues to address is to ensure the desired privacy criteria to each user involved in the composition process. Ensuring privacy does not only mean specifying a policy to decide who can see content (or use a service), but it also means to protect that policy. For example, in the case of UGC (e.g., in Facebook), the policy "I do not want John to see one of my photos", apart from preventing John to see that content, should imply that John will not be able to to know that he is not allowed to see one of my photos. This problem is even more significant in the case of context-aware policies, where the result of the enforcing may depend on sensitive information whose privacy must be protected. In the following, we introduce three approaches that, taking into account different degrees of privacy, can be used to cope with issues related to dynamicity and context-awareness. Depending on how the policy content is distributed and managed across the devices, we can distinguish the three different models described in the following subsections. These three policy enforcement techniques were primarily designed to preserve privacy (both policy content and user context). They also consider some approaches to cope with the dynamicity of the user context which directly impacts on the enforcement.

4.1. Distributed Policy Enforcement

Distributed Policy Enforcement (DPE) is the solution we propose in the case of full privacy degree for the component services' policies. The DPE approach is based on a distributed algorithm for policy enforcement that involves all the providers (partners) that take part in the service composition process. In this way, each partner is characterized by a "non disclosure" policy strategy that prevents the composite service provider from knowing partner's policies. Using DPE, an authorization decision to deny access to a service is taken only if the individual decisions of all partners have been collected and at least one of them is of the type Deny. The Policy Composition Algorithm (PCA) is therefore of type Deny-Override. When a provider receives an access request to its composite service, it firstly evaluates the request locally and if necessary forwards this request to all the providers involved in the service composition.

We can see in detail how this method works considering Figure 3. Let us suppose d_2 is Bob's Smart TV, which is composing a service $s_{1,1}$ offered by Alice camcorder d_1 to provide the *Screenshot* service $s_{2,1}$. When Carol requires through her smartphone d_3 access to $s_{2,1}$, the Policy Enforcement Point (PEP) of Bob's PZ firstly enforces local policy to decree whether Carol is allowed, and if not, it straight out rejects the request. On the contrary, if Bob's policy allows Carol, Bob's PEP forwards the incoming request to Alice's PEP (on behalf of Carol). At this stage, Alice's policy manager enforces the local policy and evaluates whether Carol can indirectly access $s_{1,1}$. Finally, Alice's PEP sends the decision to Bob's PEP, which issues (or not) the service $s_{2,1}$.

As direct consequence, this method is agnostic to dynamic policy changes of component services. Any change, in fact, is taken into account thanks to the distributed nature of the policy enforcement: each partner takes part in the final decision on the basis of its policy decision. This technique also works well in context based policy systems: the authorization decision of each service's partner will take into account also any other useful contextual information. From a practical point of view, there is a main drawback: this approach may lead to a composite service that could never work due to conflicts among component service's policies. In addition, from a performance point of view, an overhead is introduced by every enforcement. For example, when the intermediate service node forwards an access request to the partners, it has to transfer to them all the requestor's identity information plus the information about the required resources. On the other hand, there are no performance degradation issues due to policy or context synchronization.

4.2. Local Policy Enforcement

When providers' policies do not contain sensitive information, they could be stored and evaluated by intermediate providers. Local Policy Enforcement (LPE), unlike DPE, requires policy evaluation only for those service providers that supply a composite service. This means, for instance, that the intermediate provider d_2, in addition to its set of policies P_2, could store the set P_1 of the component service $s_{1,1}$ and evaluate it whenever a request is received (such as from d_3).

In general, an intermediate provider who stores all component's sets of policies could compose them and then, for each access request, evaluate the composite policy. As already mentioned, deriving from several component policies a composite policy in which there are neither conflicts nor redundancies is quite complex. For this reason, in a UGS scenario, where policies are dynamic and may change frequently, we can affirm that evaluating each single policy is preferable to extracting a composite policy. In this kind of evaluation, an authorization decision that denies service access to a subject is taken if at least one of the component policy returns a "deny" response. The PCA algorithm is therefore of type Deny-Override.

As shown in Section 2 (policy example of Figure 2), given that Bob knows that when Alice is in Catania she is willing to provide *Videostream* service to Carol, he cannot know at the enforcement time where Alice is and thus he cannot allow or deny Carol. Therefore, to evaluate the policy of a component service, an intermediate provider must know the remote context. Two different approaches enable policy enforcement by intermediate providers when context-dependent policies can be adopted. The first approach assumes that Bob stores Alice's policy set that contains n context-dependent policies. However, only a subset of these policies is currently active: those related to the current Alice's context. Thus, Alice sends Bob only information about her context and Bob uses this information to understand which of the policies contained in the policy set are active (applicable). This exchange could happen according to two strategies: (a) Alice tells Bob her context information whenever her context changes; (b) Bob asks Alice her context information whenever he receives a request (e.g., from Carol). If Alice's context changes frequently, it could be better to adopt approach (b). On the contrary, if Alice's context rarely changes, the proposed variant (a) may be preferable. The second approach considers that Bob knows only the current active subset of Alice's policies. In this case, Alice sends to Bob those policies enabled by her context.

Since LPE expects that composite service providers keep component services policies, it is necessary to update these every time they change. As shown in Figure 4, if n users (providers) require the $s_{1,1}$ service in their composition process, Alice's policy manager has to synchronize the policy in all the involved providers each time the policy for $s_{1,1}$ is changed.

Synchronization could be a burdensome task as a provider (e.g., Alice) should maintain information about all the "follower" providers, and it is even more problematic if the provider that supplies the service is a resource-constrained device.

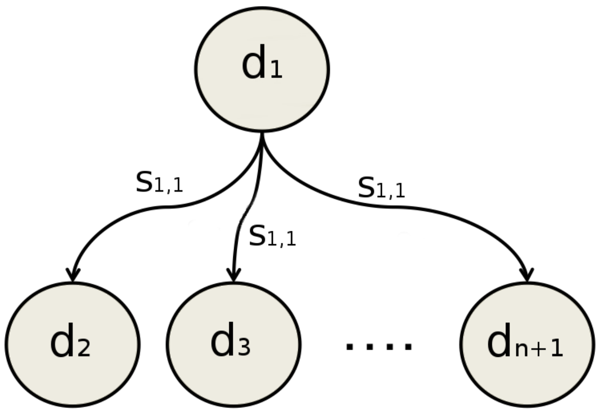

Figure 4. Service provisioning to multiple devices.

In those cases when not only the policy but also the context relative to a service provider may change, knowing these policies might be not enough to be able to evaluate them. As aforementioned, in both of the two approaches, Alice needs to realize when her context changes so that she can advise Bob about sending him her actual policy/context information. Therefore, there is the need to automatically spot every context change. For this purpose, we propose an extension of XACML (eXtensible Access Control Markup Language) architecture (Figure 5) that introduces the Context Listener component. The choice of using XACML as a basis for this work is mainly due to the flexibility and fine grained levels of customizability given by this standard. In detail, the Context Listener reads the policy obtained through the PAP (Policy Administration Point) module and stores both context-dependent parameters and relative boundaries. For example, if a policy rule was

"Permit if battery level is greater than 35%",

the Context Listener will store "*battery_level*" as a parameter and "*greater than 35%*" as related boundary. Using this information, the Context Listener can reorganize a user's policy on the basis of his/her context. In particular, the current values of context-dependent parameters affect the policy selection. For example, the pseudo code reported below may represent the logic followed by the Context Listener to choose the current policy:

```
if (context_params1 < boundary1)
    current_policy = P1
if (context_params2 > boundary2)
    current_policy = P2
```

Thus, the Context Listener periodically polls the PIP (Policy Information Point) module to obtain actual values for the contextual parameters (such as current battery level) and checks whether these values remain within the boundaries defined by the policy. When at least one of the parameters, included in the rule, exceeds its boundary, the Context Listener communicates to interested intermediate providers the current user's context or policy.

In summary, LPE can be adopted if there are no privacy concerns, but the synchronization process requires an overhead that, as we will show in Section 5, may be not acceptable in the case of devices with reduced performance as, for example, mobile phones.

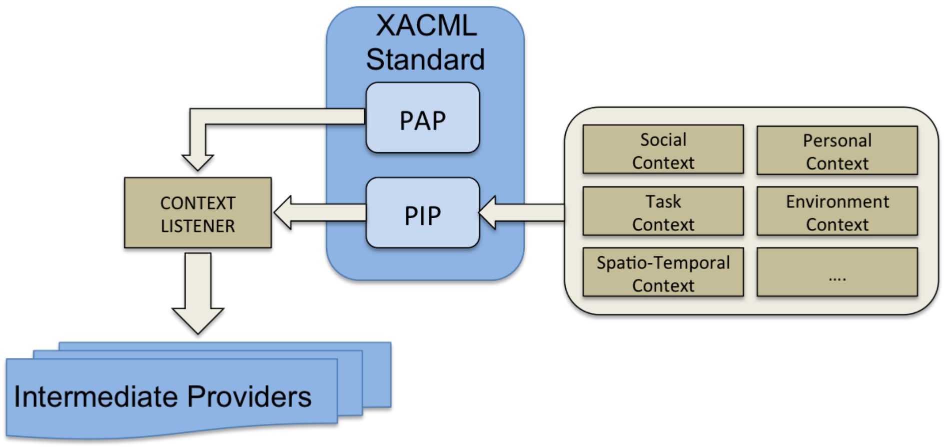

Figure 5. XACML (eXtensible Access Control Markup Language) extension with Context Listener.

4.3. Policy's Context Obfuscation

Local Policy Enforcement requires distribution of policies or parts of them. The process to exchange this kind of data requires some mechanisms to assure content authenticity and avoid tampering. Privacy is the other main aspect to cope with in situations such as this one, where personal information (i.e., contextual data) are exchanged. In fact, the policies themselves are privacy sensitive information because they can contain users' preferences (i.e., rules that are applied to specific subjects) and may depend on context parameters (e.g., users' location details). Policy Information Points are allowed to elicit, from remote providers (i.e., users' devices), personal contextual data in order to allow policy enforcement. This means that, knowing user's policy and the relative decision toward a request, it could be possible to retrieve information about user's context. This obviously leads to privacy issues.

In literature, the problem of policy's content protection is faced in different works as, for example, in [15] where the authors propose techniques based on encryption. Our proposal relies on obfuscation methods in order to mask sensitive information sending the policy structure in plain text.

Suppose that Alice chooses the following policy for the *Videostream* service:

```
<policy-set id="Alice's policyset">
   <policy algorithm="permit-overrides">
     <target>
       <subject param="cert" match="Carol.cert"/>
       <resource param="service" match="videostream"/>
     </target>
     <rule id="5ABE" effect="permit">
      <condition>
         <context param="location" match="home"/>
        </condition>
      </rule>
      <rule effect="deny"/>
   </policy>
    <policy>
     <target>
        <subject id="F3MK"/>
      </target>
      <rule effect="deny"/>
    </policy>
</policy-set>
```

Since this policy depends on Alice's context (specifically on Alice's location), giving it in plain-text to Bob could give rise to privacy issues: Bob, in fact, knowing the policy, could realize that Alice is at home at the instant when Carol is allowed to get *Videostream* service. This information does not concern Bob: he only should know that Carol will be able to get *Videostream* service (indirectly through *Screenshot* service) when a certain Alice's context will be active, but he should not know which one it is. The policy stored by Bob's devices would be the following:

```
<policy-set id="Obfuscated Alice's policyset">
   <policy algorithm="permit-overrides">
    <target>
       <subject param="cert" match="Carol.cert"/>
       <resource param="service" match="videostream"/>
     </target>
     <rule id="5ABE" effect="permit" obfuscated>
      <condition>
         <context param="E67QF1" match="AR7MG6"/>
        </condition>
      </rule>
      <rule effect="deny"/>
    </policy>
    <policy algorithm="deny-overrides">
      <target>
        <subject id="F3MK" obfuscated/>
      </target>
      <rule effect="deny"/>
    </policy>
</policy-set>
```

In this way, using LPE, Alice will transfer to Bob the pair $< E67QF1, AR7MG6 >$ representing her context, instead of the sensitive information $< location, home >$.

4.4. Partial Disclosure Policy Enforcement

Describing DPE and LPE, we pointed out some benefits and drawbacks of both solutions. DPE solves problems related to dynamic policy/context changes and privacy issues related to policy disclosure, but it could be very difficult to generate a new service without any knowledge about policies related to component services. This, in fact, may lead to having composite services that do not work properly. For instance, if Bob decides to provide $s_{2,1}$ only after *5:00 p.m.* without knowing that Alice is willing to provide $s_{1,1}$ only before *3:00 p.m.*, his service will never work. On the other hand, with LPE approach, intermediate providers know component policies: this could involve privacy problems (as discussed in previous sections), but it represents the best way, for an intermediate provider, to supply a working service. For these reasons, we want to introduce a new approach that can be considered as somewhere in between DPE and LPE: we call it Partial Disclosure Policy Enforcement (PDPE).

According to the idea behind PDPE, when an intermediate provider wants to supply a composite service to a set of known subjects, he can obtain from the component service provider not the whole policy but only the information regarding these subjects. PDPE thus consists of two phases:

1. the *initialization* phase, in which an intermediate provider (Bob) can "negotiate" the policy disclosure with the component provider (Alice) requiring only the policy's information toward a pre-determined set of subjects;
2. the *incremental update* phase, in which the intermediate provider requires additional information every time a new subject (which is not included in the original set of the considered subjects) requires access to the service.

The first phase can be optional as an intermediate provider could not negotiate the policy disclosure for a first set of subjects building this initial set when a subject requires the service the first time.

For example, suppose Bob does not know Alice's policy for $s_{1,1}$. When Carol requires $s_{2,1}$ Bob informs Alice who sends back to Bob the piece of her policy regarding Carol. From that moment onwards, Bob stores this partial policy, enforcing it every time Carol requires $s_{2,1}$ service. Whenever a new subject (e.g., David) requires $s_{2,1}$, Bob repeats the procedure, and so on. Since even partial policies may depend on user's contextual information, PDPE employs the same approaches shown in Figure 5 to overcome problems related to policy/context changes. In addition, it might apply the obfuscation technique already discussed previously.

4.5. Considerations

The proposed approaches mainly differ in the amount of policy's information revealed by component service providers to intermediate providers. However, all of the discussed approaches can be combined to meet different users' security requirements. Consider the scenario in which Bob wants to provide a service that shows a Google Map decorated with his and his friends' preferred restaurants. This scenario is represented in Figure 6. Alice and David provide two services that give information about their preferred restaurants. We can note that all the considered approaches are employed: Google, using PDPE, gives Alice only a piece of its policy (only the parts related to those subjects that Bob wants to consider), Alice, indeed, does not bother about privacy issues and gives Bob her whole policy (LPE). Finally, David does not want Bob to know his policy, so he adopts DPE.

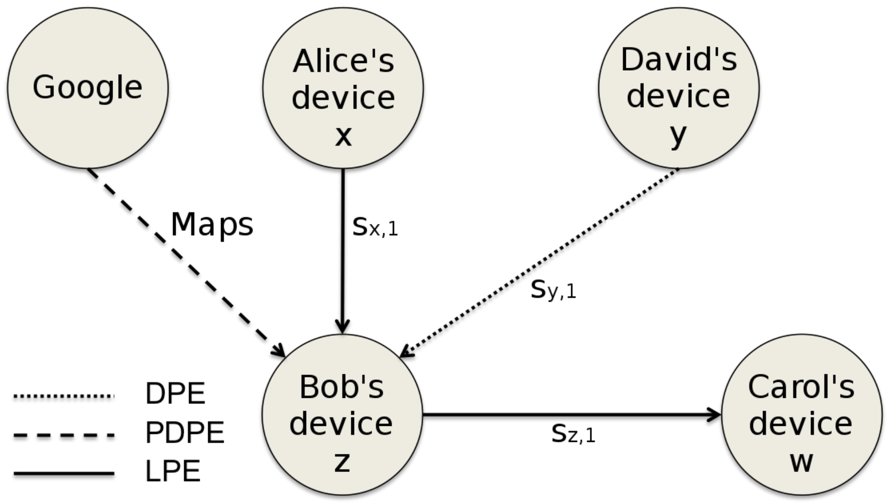

Figure 6. An example of hybrid service composition scenario in which $s_{z,1}$ composes all the other provided services, i.e., $C_{z,1} = \{Maps, s_{x,1}, s_{y,1}\}$.

5. A Cloud Approach

5.1. Mobile Constraints

As aforementioned, the broad diffusion of IoT devices, combined with the presence of platforms able to simplify interactions among them, represent the starting point for the spread of User-Generated Services. In some cases, these services could strictly depend on devices' features and local context. If we consider, for example, the scenario depicted in Section 2 we can find out that the service created by Alice is provided thanks to the presence and the features of her smart device. This scenario is just one example that points out that the next step is represented by those services that are not only generated but also provided by users through their devices (User Provided Services or UPSs). Limitations may be due to the utilization of kind of devices that are not supposed to work as providers. This is the case of smartphones and handsets in general. In fact, it is possible to recognize problems of at least three categories, which are performance, scalability and presence. For what concerns performance, it is clear that providing a service that is resource consuming (e.g., in terms of CPU time, memory and bandwidth) may reduce the usability of devices' main functionalities. Scalability issues are related to the number of users that can concurrently access our self-provided service.

The last point regards presence: some devices could be not reachable in any moment due to a temporary lack of connectivity (for example, a car or a smartphone inside a tunnel or in a place without coverage) or just because they are running out of battery. As is known, devices might provide services or contents and can regulate access to them via context-based policies. In the first case, the device presence is obviously needed to guarantee the service availability, and, in the second case, it is needed to gather the context information without which the policy to access or handle the remote content cannot be enforced. It is also important to notice that the problems described in previous sections still remain. In fact, it is difficult to properly handle policy disclosure and cope with the synchronization process of context and policy, especially in the case of many actors.

5.2. Architecture

In order to cope with the issues described in previous sections, we propose another structure that differs from the one depicted in Figure 3, introducing a new component, namely *service mediator*. The structure of this architecture is shown in Figure 7.

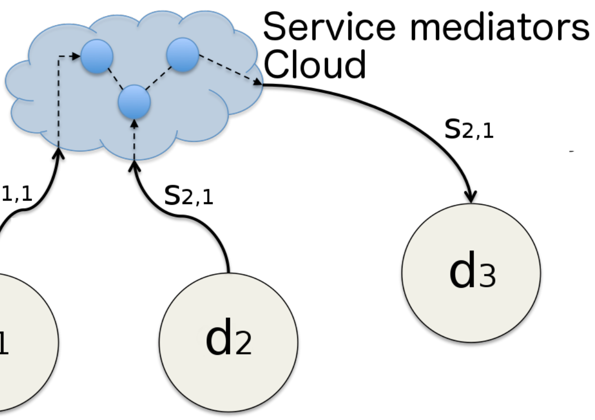

Figure 7. Cloud-based architecture for UGSs composition management.

This architecture still allows using the already described policy models and their combinations: however, it is important to point out that, in this case, the policy is enforced by components in the cloud and not by intermediate providers.

The role of the *service mediator*, which is intended to be a cloud component, is manifold; in fact, it is possible to recognize at least the following functionalities: (a) synchronizing user's preferences, context information and policies; (b) resolving problems related to "device presence"; (c) reducing the disclosure of sensitive data, context information and policies; (d) providing unified service discovery; (e) integrating logic to switch among different policy and context exchange approaches; and (f) enhancing scalability and performance.

One of the main advantages of a cloud approach regards synchronization (a). In fact, compared to the peer-to-peer model presented in the first part of this paper, a cloud based architecture simplifies the process of exchanging and keeping up to date the information needed. This might be done, for example, through a publish/subscribe notification system. Furthermore, such a publish/subscribe mechanism could be useful to resolve presence related issues (b), enabling users and devices to leave online records of taken decisions, context information and other data, also when they are disconnected. This problem, as mentioned before, is very relevant in the case of constrained devices. If we consider, for example, the scenario presented in Section 2 adding the clause that Alice allows Bob to record her videostream and reuse it with the aforementioned restrictions, the knowledge of Alice's location becomes very important. For obvious reasons, it is not possible to know Alice's location at any moment (e.g., all her devices ran out of battery), but, with the cloud approach here presented, it is possible to communicate and preserve context information, e.g., the last location available, for a more accurate evaluation.

The cloud approach also limits disclosure of personal data (c). In fact, the service mediator component could be the only point where sensitive information is stored. In addition, it enforces remote access requests on behalf of the remote service provider with the advantage of limiting exchange of policies, contextual information and sensitive data.

Devices can expose services registering them in the cloud. There the *service mediator* can: make them discoverable (d), manage their status and, as already mentioned, regulate remote access to them through policies. The *service mediator* could also contain some logic in order to switch between the different policy management models described. This smart switching system can allow users to change the desired degree of privacy at run time.

The gain in terms of scalability and performance (f) is due to the fact that the *service provider* will not be directly contacted by clients because exposed services will be accessed by the *service mediator*. It will be able to multiplex a service or serialize access to it in the case he cannot directly provide that service. Considering again the *Videostream* service, it is easy to imagine that a solution like the one proposed—with a server that acquires the videostream (*service provider*) and multiplexes it to other consumers—can reduce the provider's workload.

As a demonstrator prototype of the proposed approach, a policy management component has been implemented through the *webinos* foundation platform in order to expose devices' features to other devices registered in the virtual area called Personal Zone (PZ). The PZ has the main role of providing, in a simple manner, access to local and remote services, thus creating also an abstraction from underlying communication technologies. In particular, the approach proposed in *webinos*' makes use of the architecture depicted in Figure 8. In this architecture, the component called Personal Zone Hub (PZH) implements the behavior of the *service mediator*.

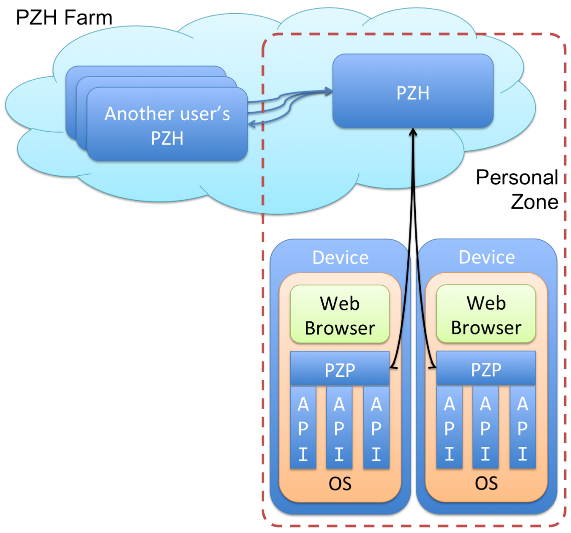

Figure 8. *Webinos* general architecture.

The PZH is the element in which the devices belonging to a PZ are registered. It enables the communication between devices of the same or a different PZ, allows the discovery of other PZHs and manages the synchronization process (e.g., of user's preferences, policy and context data). A PZH also controls security aspects. In fact, it is the certificate authority that issues certificates for each PZP registered in a PZ. These certificates allow mutually authentication and also encryption of the messages exchanged among devices in the zone.

The other main component of *webinos* architecture is the PZP that runs on each *webinos enabled device* and manages the communication between the device and the main hub (PZH). The PZP also stores the information that should be synchronized with the hub in the case it is not reachable in order to start the synchronization process when the PZH again becomes available. The PZP is responsible for the local discovery of features/services, accessible through a set of APIs (Application Programming Interfaces) that expose some capabilities of the underlying Operating System.

From this description, it is possible to notice that the structure proposed by *webinos* well suits the cloud approach where the *service mediator* is an extension of the PZH able to manage not only security and synchronization aspects but also service multiplexing. Choosing *webinos* as a basis for our work made it possible to move the focus on privacy/security issues, removing the necessity of implementing a new system from scratch to synchronize policies, context information and user's preferences. Furthermore, *webinos* provides full specifications for over 30 APIs to manage, for example, authentication, discovery of devices and services, context, contacts, filesystem, media content and streams, sensors and actuators, simplifying the creation of multi-user/multi-device/multi-domain applications in several Operating Systems, such as Android, Windows, MacOs, GNU/Linux, Chrome OS and Firefox OS [16].

The *webinos* platform provides privacy protection and access control features to meet the privacy and security requirements of web applications and users. This means protecting against malware and other users who may attempt to access more data than they should or simply avoid the unnecessary disclosure of personal data. To satisfy these requirements, *webinos* relies on the strong identification of web applications, users and devices, combined with a least-privilege access control, based on XACML. It is an OASIS standard that describes: (i) an XML (eXtensible Markup Language) representation for both a policy language and an access control decision request/response language, and (ii) a data-flow model for policy enforcement and relative involved actors. In *webinos*, the XACML architecture has been adapted using PrimeLife extensions [17], allowing for making access control decisions based on both the request context and user preferences.

The root element defined by XACML is a PolicySet that is made up by both a set of policies (containing at least one policy) and Policy Combination Algorithms (PCA) that take the authorization decision from each policy as input and apply some standard logic to come up with a final decision. Each XACML policy is in turn made up of a Target and a set of Rules.

A Target is basically a set of simplified conditions for the Subject, Resource and Action that must be met for a PolicySet, Policy or Rule to apply to a given request. Each Rule represents the core logic for a policy, and it is made up of a Condition (which is a boolean function and may contain nested sub-conditions) and an Effect (a value of Permit or Deny that is associated with successful evaluation of the Rule).

The Data-flow model introduced by XACML is shown in Figure 9. In short, a Policy Enforcement Point (PEP) is responsible for intercepting a native request for a resource and to forward this to the Context Handler (CH). The CH will form an XML request based on the requester's attributes, the resource in question, the action, and other information pertaining to the request. The CH will then send this request to a Policy Decision Point (PDP), which will look at the request and some policy that applies to the request, and come up with an answer about whether access should be granted. That answer (in XML format) is returned to the Context Handler, which translates this response to the native response format of the PEP. Finally, the PEP, based on the response, allows or denies access to the requester. A more detailed description about XACML is provided by [18] while a description of *webinos* is presented in [19].

The object model depicted in Figure 9 shows the main components of the *webinos* policy framework. Both the PZP and PZH enforce policies using standard XACML components, supplemented by:

- The *Decision Wrapper* creates the initial policy enforcement query based on incoming requests;
- The *Access Manager* makes the final decision by combining pieces of information from XACML access control and Data Handling Decision Function (DHDF);
- The *DHDF Engine* provides privacy and data handling functionalities;
- The *Request Context* manages all contextual information; it stores all the data and credentials released by a user in a given session;
- The *PDP Cache* (PDPC) stores PDP decisions that could be shared among personal devices.

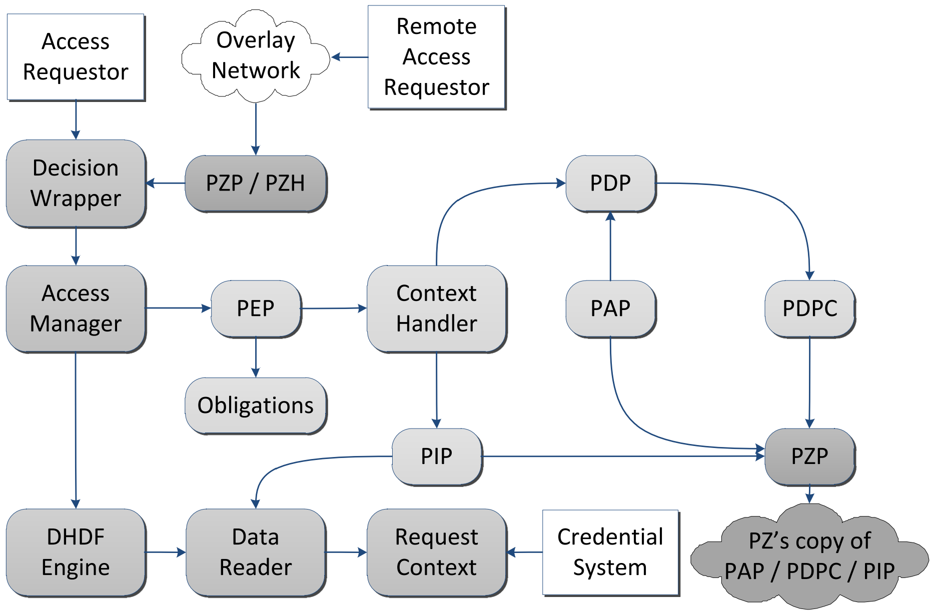

Figure 9. XACML-based Data-flow model of *webinos*.

5.3. Considerations on Design Choices

This work brings together two innovative aspects related to the world of next-generation services. The first is to imagine an extension of the concept of User Generated Content to services, thereby defining User Generated Services as services that are both directly generated and provided by users, for example through their devices (e.g., smartphone, TV, and car). The possibility of composing services to create new ones is certainly one of the key aspects of the SOA paradigm. In the case of UGSs, the composition of multiple services provided by different users introduces a number of privacy issues that have been the main focus of this work. In particular, the management of access control and privacy of a service is managed through policies that, in the case of composite services, are in turn composed, with the risk of creating inconsistencies or conflicts that may invalidate user preferences. The methods presented in previous sections are thought to mitigate the impact that fast changing user-centered environments, characterized by the presence of UGSs and the manipulation of sensitive data, may suffer from not as fast or approximate policies' updates. In facts, when existing services are put together, it is necessary to guarantee that the policy of the resulting composite service is coherent with policies of all the component services to avoid conflicting authorization decisions taken by the security policy of the composite service.

The second innovative aspect concerns the possibility to have service access governed by policies that depend on the user's context. It is important to consider that changes in user-centered environments are usually reflected in a physical (e.g., position and speed) and a logical (e.g., preferences and contacts in social networks) context, each of which contributes to increasing the dynamics to take into account while composing services. This introduces a further complexity in policy enforcement, as it is necessary to know the context of all the users involved in the composition of a service in order to decide whether or not to grant access to that service. In addition, due to the fact that the context of a user is characterized by a strong variability, applying caching strategies is not easy since policy constraints may change along with the context itself. For these reasons, we have considered implementing strategies for composing UGS that consider variations in the context of the users involved in the composition.

Then, it was important to consider the spread of constrained devices that are part of everyday life. To expose UGSs within these devices, it was necessary to take into account different factors, and in particular, the fact that resources needed to expose/exploit UGSs, as far as possible, should not compromise the normal functioning of the device. The cloud-based solution, proposed in this section, has been designed to overcome these limitations regarding performance, scalability, and presence (the latter, in terms of power consumption and network coverage) and to exploit the other policy models introduced in this work. Finally, although the proposed methods introduce an overhead due to the transfer of policy or context information, they provide greater security and privacy protection in scenarios likely to become common over the next few years.

6. Related Work

The scientific interest in UGSs and UGC is growing in these last years. In [5], the authors presented a comprehensive survey of the state-of-the-art UGSs and defined UGSs by comparison with the concept of UGC, describing advantages and limitations of each approach. Ref. [4] introduces some guidelines to support users in the services' creation and management. In [20], the requirements to let the vision of the "super prosumer" become true are investigated. In particular, they review the current technologies that enable an easy creation and discovery of mobile services and list the identified requirements for UGSs. The interest in this topic is further motivated by the result of diverse studies that evaluate, for example, the impact of UGSs and especially UGC on society [21,22], tourism [8], and advertising [23]. With regard to aspects explicitly focused on service composition, some works were proposed to prevent conflicting behaviors when policies are composed. In [14], authors propose a security policy composition mechanism which semi-automatically derives the policy consistency rules. In addition, Ref. [10] presents an approach that merges existing policies and restrictions into a new policy to remove redundancy and inconsistencies. It is important to point out how the process of policy composition might be very expensive from a computational perspective; as a consequence, these solutions can only be applied if we assume static policies (i.e., not changing over the time), so that the composition process can be performed only once off-line, when the composite service is made up. Another approach, proposed in [24], adopts a filtering phase and a differential evolutionary based algorithm to compose services taking into account service providers' mobility.

The composition and deployment of UGSs has also been investigated from the IoT perspective. In [11], the authors proposed an infrastructure to enable developers and designers to dynamically query, select, and use running instances of real-world services. More specifically, Ref. [25] analyzed the notion of ubiquitous services of IoT in connection with their usage in the context of social networks. Other works focused on particular domains of IoT services, such as [26], introducing an interconnection framework for mobile Health (mHealth), or [27,28] where the composition service is enabled by means of a semantic ontology. Some important contributions in this field have been directed through the definition of policy models suitable for context-aware scenarios, as in [28–30]. Access control systems should be able to support and understand existing and new context information types in order to address access control requirements. To make this possible, the authors of [31] presented an extensible access control solution based on XACML (eXtensible Access Control Markup Language) making it able to understand new attributes' data types and the functions that are used in the policy to evaluate the users' requests. Another interesting work is [32], which proposed an access control policy model based on context and the role that can be appropriate for web services. The model takes into account contextual information to define and perform access control policies. It works with contextual information from users, environments and resources to execute dynamic role assignment and constrain the authorization decision. Another approach to address conflicts in context-aware policies is presented in [33], where authors propose a framework in which the policy is chosen at run time based on context information. Recently, the authors of [34] presented a generic framework, OntCAAC (Ontology-based Context-Aware Access Control), modelling dynamic contexts and corresponding access control policies exploiting semantic technologies.

In conclusion, several works have addressed typical problems related to UGSs such as dynamic policy change and context-aware policies; however, to the best of our knowledge, none of these works has yet taken into account the kind of issues assuming the perspective of UGSs composition as the main challenge. The main contribution of this work is to introduce different models that are explicitly focused on a composite service scenario and present a concrete architecture that could address most of the critical issues described here.

7. Conclusions

The composition of User-Generated Services in an IoT environment introduces several security related challenges. In this work, three different approaches have been analyzed showing the available trade-off in policy management. Finally, a cloud based architecture that is flexible enough to embrace the different policy enforcement models has been introduced, together with a prototype based on the *webinos* foundation platform. Future work will focus on the support of those constrained devices, such as very simple sensors, that can be identified but cannot hold policies to regulate access to their features and data.

Acknowledgments: The research described in this paper was initially funded by the EU FP7 *webinos* project (FP7-ICT-2009-05 Objective 1.2). The extension of this work has been partially supported by the Italian Ministry of Education, University and Research on the "PAC" call funding the "SERVIFY (SERVIce FirstlY)" project, Ref. PON03PE_00132_1, CUP B72F14000300005.

Author Contributions: All authors contributed extensively to the work presented in this paper. Giuseppe La Torre, Salvatore Monteleone and Davide Patti wrote the code and performed experiments; Vincenzo Catania and Daniela Panno refined the initial study. All authors discussed enhancements, results and implications, wrote the manuscript and commented on it at all stages.

Conflicts of Interest: The authors declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, and in the decision to publish the results.

References

1. IFTTT. IF This Then That. 2011. Available online: http://www.ifttt.com (accessed on 5 July 2017).
2. OMA. Open Mashup Alliance. 2009. Available online: https://en.wikipedia.org/wiki/Open_Mashup_Alliance (accessed on 5 July 2017).
3. APIANT. The Cloud Integration and Automation Platform | APIANT. 2016. Available online: https://apiant.com (accessed on 5 July 2017).
4. Jensen, C.S.; Vicente, C.R.; Wind, R. User-Generated Content: The Case for Mobile Services. *Computer* **2008**, *41*, 116–118.
5. Zhao, Z.; Laga, N.; Crespi, N. A survey of user generated service. In Proceedings of the 2009 IC-NIDC IEEE International Conference on Network Infrastructure and Digital Content, Beijing, China, 6–8 November 2009; pp. 241–246.
6. Xia, F.; Yang, L.T.; Wang, L.; Vinel, A. Internet of Things. *Int. J. Commun. Syst.* **2012**, *25*, 1101–1102.
7. Fuhrhop, C.; Lyle, J.; Faily, S. The webinos Project. In Proceedings of the 21st International Conference Companion on World Wide Web, WWW '12 Companion, New York, NY, USA, 16–20 April 2012; pp. 259–262.
8. Marine-Roig, E.; Clavé, S.A. Tourism analytics with massive user-generated content: A case study of Barcelona. *J. Destin. Mark. Manag.* **2015**, *4*, 162–172.
9. Ventura, D.; Monteleone, S.; La Torre, G.; La Delfa, G.C.; Catania, V. Smart EDIFICE—Smart EveryDay interoperating future devICEs. In Proceedings of the 2015 International Conference on Collaboration Technologies and Systems (CTS), Atlanta, GA, USA, 1–5 June 2015; pp. 19–26.
10. Speiser, S. Policy of Composition $\neq$ Composition of Policies. In Proceedings of the 2011 IEEE International Symposium on Proceedings of the Policies for Distributed Systems and Networks (POLICY), Pisa, Italy, 6–8 June 2011; pp. 121–124.
11. Guinard, D.; Trifa, V.; Karnouskos, S.; Spiess, P.; Savio, D. Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services. *IEEE Trans. Serv. Comput.* **2010**, *3*, 223–235.

12. Catania, V.; La Torre, G.; Monteleone, S.; Panno, D.; Patti, D. User-Generated services: Policy Management and access control in a cross-domain environment. In Proceedings of the 2015 International Wireless Communications and Mobile Computing Conference (IWCMC), Dubrovnik, Croatia, 24–28 August 2015; pp. 668–673.
13. Catania, V.; La Torre, G.; Monteleone, S.; Patti, D.; Vercelli, S.; Ricciato, F. A novel approach to Web of Things: M2M and enhanced javascript technologies. In Proceedings of the 2012 IEEE International Conference on Green Computing and Communications (GreenCom), Besancon, France, 20–23 November 2012; pp. 726–730.
14. Satoh, F.; Tokuda, T. Security Policy Composition for Composite Web Services. *IEEE Trans. Serv. Comput.* **2011**, *4*, 314–327.
15. Dong, C.; Russello, G.; Dulay, N. Shared and Searchable Encrypted Data for Untrusted Servers. In *Data and Applications Security XXII*; Lecture Notes in Computer Science; Atluri, V., Ed.; Springer: Berlin/Heidelberg, Germany, 2008; Volume 5094, pp. 127–143.
16. Webinos. Webinos Developer Portal. Available online: https://developer.webinos.org (accessed on 5 July 2017).
17. Ardagna, C.A.; De Capitani di Vimercati, S.; Paraboschi, S.; Pedrini, E.; Samarati, P. An XACML-based privacy-centered access control system. In Proceedings of the first ACM Workshop on Information Security Governance, Chicago, IL, USA, 13 November 2009; pp. 49–58.
18. Moses, T. Extensible Access Control Markup Language 2.0 Specification Set. Available online: http://www.oasis-open.org (accessed on 5 July 2017).
19. Lyle, J.; Monteleone, S.; Faily, S.; Patti, D.; Ricciato, F. Cross-platform access control for mobile web applications. In Proceedings of the 2012 IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY), Chapel Hill, NC, USA, 16–18 July 2012; pp. 37–44.
20. Tacken, J.; Flake, S.; Golatowski, F.; Prüter, S.; Rust, C.; Chapko, A.; Emrich, A. Towards a Platform for User-Generated Mobile Services. In Proceedings of the 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA), Perth, Australia, 20–23 April 2010; pp. 532–538.
21. Shelton, T.; Poorthuis, A.; Zook, M. Social media and the city: Rethinking urban socio-spatial inequality using user-generated geographic information. *Landsc. Urban Plan.* **2015**, *142*, 198–211.
22. Venerandi, A.; Quattrone, G.; Capra, L.; Quercia, D.; Saez-Trumper, D. Measuring Urban Deprivation from User Generated Content. In Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW), Vancouver, BC, Canada, 14–18 March 2015; pp. 254–264.
23. Stavrianea, A.; Kavoura, A. Social media's and online user-generated content's role in services advertising. *AIP Conf. Proc.* **2015**, *1644*, 318–324.
24. Deng, S.; Huang, L.; Wu, H.; Wu, Z. Constraints-Driven Service Composition in Mobile Cloud Computing. In Proceedings of the 2016 IEEE International Conference on Web Services (ICWS), San Francisco, CA, USA, 27 June–2 July 2016; pp. 228–235.
25. Ortiz, A.; Hussein, D.; Park, S.; Han, S.; Crespi, N. The Cluster Between Internet of Things and Social Networks: Review and Research Challenges. *IEEE Int. Things J.* **2014**, *1*, 206–215.
26. Jara, A.J.; Zamora-Izquierdo, M.A.; Skarmeta, A.F. Interconnection Framework for mHealth and Remote Monitoring Based on the Internet of Things. *IEEE J. Sel. Areas Commun.* **2013**, *31*, 47–65.
27. Lee, S.; Chong, I. User-centric intelligence provisioning in web-of-objects based IoT service. In Proceedings of the 2013 International Conference on ICT Convergence (ICTC), Jeju, Korea, 14–16 October 2013; pp. 44–49.
28. Toninelli, A.; Montanari, R.; Kagal, L.; Lassila, O. A Semantic Context-aware Access Control Framework for Secure Collaborations in Pervasive Computing Environments. In *Proceedings of the 5th International Conference on The Semantic Web*; Springer-Verlag: Berlin, Germany, 2006; pp. 473–486.
29. Yahyaoui, H.; Almulla, M. Context-based specification of Web service policies using WSPL. In Proceedings of the 2010 Fifth International Conference on Digital Information Management (ICDIM), Thunder Bay, ON, Canada, 5–8 July 2010; pp. 496–501.
30. Kapsalis, V.; Hadellis, L.; Karelis, D.; Koubias, S. A Dynamic Context-aware Access Control Architecture for e-Services. *Comput. Secur.* **2006**, *25*, 507–521.
31. Cheaito, M.; Laborde, R.; Barrere, F.; Benzekri, A. An extensible XACML authorization decision engine for context aware applications. In Proceedings of the 2009 Joint Conferences on the Pervasive Computing (JCPC), Tamsui, Taipei, Taiwan, 3–5 December 2009; pp. 377–382.

32. Li, H.; Yang, Y.; He, Z.; Hu, G. Context-aware Access Control Policy Research for Web Service. In Proceedings of the 2011 First International Conference on Instrumentation, Measurement, Computer, Communication and Control (IMCCC), Beijing, China, 21–23 October 2011; pp. 529–532.
33. Mohan, A.; Blough, D.M. An attribute-based authorization policy framework with dynamic conflict resolution. In Proceedings of the 9th Symposium on Identity and Trust on the Internet, Gaithersburg, MD, USA, 13–15 April 2010; ACM: New York, NY, USA, 2010; pp. 37–50.
34. Kayes, A.; Han, J.; Colman, A. An ontology-based approach to context-aware access control for software services. In *International Conference on Web Information Systems Engineering*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 410–420.