Context-Aware Data-Flow in the Cloud

Mandy Weißbach and Wolf Zimmermann
Institute of Computer Science
University of Halle
Halle (Saale), Germany
Email: {weissbach, zimmermann}@informatik.uni-halle.de

Welf Löwe
Software Technology Group
Linnaeus University
Växjö, Sweden
Email: welf.lowe@lnu.se

***Abstract*—In the last few months, clients of services running in a cloud are getting more and more aware of storing and processing their data in the cloud. In this paper, we present a context-aware data-flow analysis approach to allow clients to negotiate services that store or process (directly or indirectly) their data in undesired locations. The approach is context-aware to satisfy the stateless character of services in a multi-tenant cloud. We show that the use of a dynamic context-aware data-flow analysis ensures that the clients' data does not reach undesired locations in the cloud.**

***Keywords*-context-aware; data-flow; service-oriented; data security;**

I. Introduction

Undoubtedly, Cloud Computing is one of the most growing internet technologies worldwide. The preparatory study undertaken for the European Commission estimates that the public cloud would generate EUR 250 billion in GDP (Gross domestic product) in 2020 [1].

Reasons for the popularity of Cloud Computing are obvious: IT-departments can be outsourced, investments in resources, e.g., hardware, software or space, become no longer necessary, energy costs can be reduced and cloud services are available from everywhere.

Cloud Computing also plays an important role in the private sector. About 56 % of the internet users store private data, e.g., pictures, music or documents, in the cloud.

Because of the private and commercial use of Cloud Computing sensitive data may be stored and processed by cloud services. Unfortunately, encryption of data is not an option to keep sensitive data secure. When data needs to be processed by the used cloud services, it needs to be available in decrypted form [4](research in the field of processing encrypted data is just at the beginning). The abstracted infrastructure of a cloud makes it impossible for the user to know the exact location their applications or data are running on [2], [3]. So, one major obstacle in using cloud services is that clients have no control where their data are being stored and processed [2].

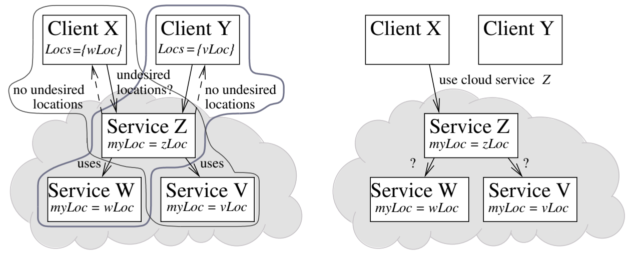
However, if cloud servers are located at different locations, they obey national laws on the server's location. These might be rather different than the location of the cloud user. Therefore there might be unauthorized access to clients' data that might be legal in the country of the server of the cloud service, e.g., through [5], but illegal in the client country [6]. Despite this fact, we focus on data-security in the cloud.

In our previous work [6], we described an approach that enables a client to control the data-flow in the cloud. Data-flow to undesired locations could be negotiated by the client. Cloud services were allowed to use other services in desired locations and so on. Even callbacks between cloud services installed at desired locations are allowed [6].

Our previous work assumes that there is one client, which has a list of undesired locations. This client uses the cloud services by its own. So, there exists only one view on the cloud services. In this work, we generalize to cloud services used by several clients where each client may have its own wish of undesired locations.

Suppose client X has country $wLoc$ as its undesired location and client Y has country $vLoc$ as its undesired location, cf. Figure I. Client X calls service Z which is installed on a server located in country $xLoc$. Service Z can use service W or service V. Service W is installed on a server located in country $wLoc$. Service V is installed on a server located in country $vLoc$. Since the Service Z is installed on a server in country $zLoc$, both clients are going to use service Z. While the static data-flow analysis for client X is done, service Z can use service V, because V is located in country $vLoc$. Service W would be negotiated by client X, because W is installed on a server located in country $wLoc$. The same data-flow analysis is done for client Y. Now, service Z can not use service V because service V is installed on a server in country $vLoc$, an undesired location of client Y, cf. 1(a). Service Z can use service W, because service W is a desired location of client Y. However, client X wants to use service Z, service Z can not derive whether client X or client Y is calling and therefore service Z does not know which service (service V or service W) to use (One-View-Problem described in Figure 1(b)). So, our approach is extended to support different views from different clients to support multi-tenant services.

We realized this approach with a context-aware data-flow analysis in the cloud. The used static data-flow analysis is an conservative approach [6], which can guarantee in the case of an positive answer that no sensitive data flow



(a) Static Data-Flow-Analysis. (b) One-View-Problem.

Figure 1. Conrolling Data-Flow in the Cloud.

```
/* @return : true−> data-flow to undesired location(s)
              false−> data-flow only to desired locations*/
/* UnDes_{X,S,L} : data-flow from service B over provided
              functions S to service X in countries L*/
BOOL undesired(SET(Provided_B) S, SET(Locations) Locs) {
    if myloc ∈ Locs return true;
    foreach service X used by B do
        if UnDes_{X,S,L} return true;
    return false;
}
```

Figure 2. Implementation of *undesired* [6].

direct or indirect to services installed on servers in undesired locations. In the case of a negative answer, there could be a direct or indirect data-flow to services installed on servers in undesired locations. Instantiation of one service on several servers in different countries are not considered. This work follows the service-level-agreement principle (SLA). So, based on the result of the context-aware data-flow analysis, the client can negotiate a service that is installed on a server at an undesired location. In order to increase trust in the given answer, we assume the use of the proposed cryptographic approach in [6].

The paper is organized as following: In Section II, we introduce a service model example. The context-aware data-flow analysis with respect to the presented example is given in Section III. Section IV discusses related work and Section V concludes this work.

II. Service Model Example

This section gives a short overview of our service model and states the problems that could occur if we are not aware of the context.

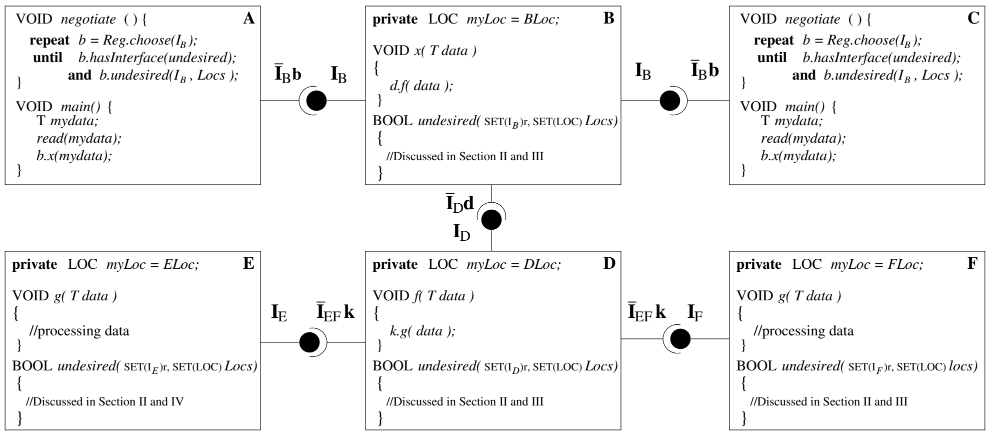
We assume that each service A provides a set of functions, denoted by $Provided_A$. This might be given as a WSDL-Description (Web Service Description Language). Furthermore, each service A must use another service. We assume that this is not hard-coded in the implementation of A, but there is a pair of variables I x where I contains the set of functions that is called on x, and x can be bound (dynamically) to a service X that provides at least I, i.e., $I \subseteq Provided_X$. Functions in I are called *required functions* of A w.r.t. x. The set of candidate services must be published and we assume that a registry Reg maintains all published services. We also assume an acyclic use structure of the services. Section III shows how this assumption can be dropped.

Example 1: Multi-Tenant Clients
Consider services A, B and C in Figure 3. $A.b$ can be bound to service B and also $C.b$ can be bound to service B. The provided interface of B is $Provided_B = \{x, undesired\}$. The required functions of A w.r.t. b are $\{x, undesired\}$. The required functions of C w.r.t. b are also $\{x, undesired\}$. The required functions of B w.r.t. d are $\{f, undesired\}$. So service B can simultaneously be used by service A and by service C. ■
A client would like to negotiate an agreement that a selected service guarantees to avoid data-flow from the clients' data to a set $Locs$ of undesired locations. For the purpose of negotiation, service B may offer a function $undesired \in Provided_B$ that returns $true$ iff data flows via some operations o from the provided interface of B to services at undesired locations, cf. with Figure 2.

Remark: It is sufficient to take into account only the set $S \subseteq Provided_B$ of operations used by the client. □

If service A uses service B, it needs to ask B (via B's function $undesired$) whether it can guarantee that its data do not flow to a location in $l \in Locs$ (undesired locations). Obviously, this needs only to be guaranteed for those operations of B where B passes (possibly processed) data of A. For simplicity, we assume that each service X knows its location and this location is stored in a constant $X.myLoc$.

Example 2: Negotiation of Undesired Locations
Consider services A, B, C, D, E and F in Figure 3. Service A would like to use service B. Service B is located in *BLoc*. B can also be used by service C. B itself uses service D located in $DLoc$. Service D uses service E or F. E is located in $ELoc$. F is located in $FLoc$. We assume that all services (except possibly client A and client C) are published.
Suppose that client (service) A wants to avoid storing its data (neither in original nor in processed form) at servers in *FLoc*. Before client A actually uses service B it would like to know whether data passed to B are never stored at a server in $FLoc$. Let $Locs$ be the set of servers in $FLoc$. The procedure $negotiate$ searches for a published service B offering at least the operations specified in I_B (I_B is the set of functions of the required service that are called from client A). For the purpose of negotiation, client A calls $undesired(I_B, Locs)$ because client A calls $b.x(mydata)$, if b is bound to service B. Service B calls function $f \in Provided_D$, if d is bound to service D. So, a call of $b.x(mydata)$, if b is bound to service B, implies that data of client A flows to service D by the call $d.f(data)$ of service B, if d is bound to service D. Thus, the call $undesired(I_B, Locs)$ must return $false$ only if $B.myLoc \notin Locs$ and $undesired(I_D, Locs)$ returns $false$. The functions $f \in Provided_D$ calls g and $g \in Provided_E$ or $g \in Provided_F$, it depends on whether



```
VOID negotiate ( ) {                                      A
  repeat b = Reg.choose(I_B);
   until b.hasInterface(undesired);
        and b.undesired(I_B , Locs );
}
VOID main() {
   T mydata;
   read(mydata);
   b.x(mydata);
}
```

```
private LOC myLoc = BLoc;                                 B

VOID x( T data )
{
   d.f( data );
}
BOOL undesired( SET(I_B)r, SET(LOC) Locs)
{
  //Discussed in Section II and III
}
```

```
VOID negotiate ( ) {                                      C
  repeat b = Reg.choose(I_B);
   until b.hasInterface(undesired);
        and b.undesired(I_B , Locs );
}
VOID main() {
   T mydata;
   read(mydata);
   b.x(mydata);
}
```

```
private LOC myLoc = ELoc;                                 E

VOID g( T data )
{
   //processing data
}
BOOL undesired( SET(I_E)r, SET(LOC) Locs)
{
  //Discussed in Section II and IV
}
```

```
private LOC myLoc = DLoc;                                 D

VOID f( T data )
{
   k.g( data );
}
BOOL undesired( SET(I_D)r, SET(LOC) Locs)
{
  //Discussed in Section II and III
}
```

```
private LOC myLoc = FLoc;                                 F

VOID g( T data )
{
   //processing data
}
BOOL undesired( SET(I_F)r, SET(LOC) locs)
{
  //Discussed in Section II and III
}
```

Figure 3. Storing Data at undesired locations: Two-View-Example

k is bound to service E or F. The argument of the call $d.f(data)$ flows to the call $k.g(data)$ of service E if k is bound to service E or $k.g(data)$ of service F if k is bound to service F, respectively. So, if k is bound to service E, there is a data-flow from client A over services B and D to service E. Service E is located in $ELoc$, which is not a undesired country. Therefore $B.undesired(I_B, Locs)$ returns *false*, because $D.undesired(\{f\}, Locs)$ returns *false* i.e., client A can use service B. But if there is a data-flow from client A over service B and D to service F, located in $FLoc$, $B.undesired(I_B, Locs)$ returns $true$. Because client A does not want to store or process data in *FLoc*. $D.undesired(\{f\}, Locs)$ returns $true$ and therefore $B.undesired(I_B, Locs)$ returns $true$, i.e., client A cannot use service B. ■

Because the service model architecture is multi-tenant, service B can simultaneously be used by client A and by client C. The set $Locs_A$ of undesired locations of client A might be different from the set $Locs_C$ of undesired locations of client C. If B uses service D, it needs to ask D (via D's function $undesired$) whether it can guarantee that A's data do not flow to a location in $Locs_A$. Obviously, the data-flow needs to be guaranteed in context of the clients. If the context is not considered, service D can not distinguish between the calling services A and C. So if A calls B and B calls D, it is possible that a later call of D by B which was called by C is not detected as a call from C. So the undesired countries $Locs_A$ may be applied for client C.

Example 3: Context-Aware Data-Flow
Consider the services A, B, C, D, E and F in Figure 3. Service A would like to use service B. Service B is located in *BLoc*. B itself uses service C located in $CLoc$ while C uses service D or E. D is located in $DLoc$. E is located in $ELoc$. F is located in $FLoc$. Client A wants to avoid storing its data (neither in original nor in processed form) at servers in *FLoc*. C wants to avoid storing its data (neither in original nor in processed form) at servers in *ELoc*.

Suppose the negotiation process starts. Service B, D and E will be accepted by A because $undesired(I_B, Locs)$ returns $false$. Before client A starts to use service B, client C tells service B it also wants to use service B. A starts the negotiation process and for service B, D and F, the negotiation process will succeed. However, service A starts to use service B. Service B calls function f of service D. But service D can not distinguish between clients A and C. So it is possible, that service D binds to service F. But the undesired countries $Locs_A$ include the location $FLoc$ of service F.

To distinguish between client A and client C, we need to introduce a context-aware data-flow analysis mechanism to know or compute the chain of used services by a client in the service model.

III. Context-Aware Mechanism

We introduce the principle of context-aware attributes. Lists of context-dependent attributes are created. If the function $undesired$ of a service B, called by client A, returns $false$, the $call\,id$ of the caller and the called service is added to the attribute list, e.g., cl of service B. However, service B can distinguish with the help of the $call\,id$, whether client A or client C was the original caller.

Example 4: Context-Aware Multi-Tenant Clients
Consider the services A, B, C, D, E and F in Figure 3. Client A would like to use service B and specifies a set $Loc_A = \{FLoc, XLoc\}$ of undesired locations. Also, client C would like to use service B and specifies a set $Loc_B = \{ELoc, XLoc\}$ of undesired locations. Service B is located in *BLoc*. B calls service D located in $DLoc$ while D uses service E or F. Both, service E and service F offer the same functionality. The only difference is, that service E is located in $ELoc$ and F is located in $FLoc$.

Suppose, the following scenario: client A wants to bind to service B. The negotiation process starts. Service B uses service D and service D finds through a Registry Reg

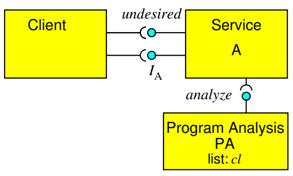


Figure 4. Architecture of the Context-Aware Mechanism.

service E and F. Since service F is located in $FLoc$, an undesired location, the call $undesired(I_F, Locs)$ will return $true$. Client A will negotiate the use of B and client A will start to find via the Reg new services. A is going to bind to service B again. This time B is going to ask D and D is calling service E. The call $undesired(I_E, Locs)$ by service D will return $false$ because $ELoc$ is not in $Locs_A$. However, service D wants to bind to service E in context of service A (service A called B, B called D). So, the $call\,id$, computed with the service use chain, can be stored in a context attribute list cl. Before service D calls function $g \in Provided_E$, service D checks if the $call\,id$ of E is registered for A. If there is a $call\,id$ of E registered, service D knows that service E can be used. Now, if service C wants to use service B, a new negotiation process starts. Now, the call of $undesired(I_B, Locs_C)$ and the call of $undesired(I_D, Locs_C)$ will return $false$. The call of $undesired(I_E, Locs_C)$ will return $true$ and no $call\,id$ is set and the list with the used service chain will be discarded. Client C will negotiate service B and the negotiation process starts as described before. This time, service D calls F. The function $undesired(I_F, Locs_C)$ will return $false$. So, the $call\,id$ of B, D and F will be added to the context attribute list cl of client C. However, if client C calls service B and service B calls service D, service D can choose with respect to the context attribute service F.

To implement the context-aware data-flow analysis, we need a trusted third party which

- can compute the resource information (location) of the used cloud service,
- ensures that the used cloud services act according to promised behavior of $undesired$ and
- maintains the information of the chain of used services by a certain client.

Example 5: Context-Aware Mechanism
The first requirement is satisfied by every service itself, cf. section II. Every service stores its location information. The second and third requirement can be given by an independent certified program analysis service PA. PA performs the program analysis, computes the result of $undesired$ and will be extended to maintain the information of the chain of used services by a certain client. For more details of the work of the unextended PA, we refer to [6].

We propose a context-aware mechanism described by the following algorithms in pseudo code and a sequence diagram in Figure 5. To start the negotiation the client calls a registry to ask for a service with the required Interfaces by providing the set of undesired locations $Locs$, the $callID$ of the client A and the required Interface I_{req_A}:

```
**Algorithm 1:** negotiate
**INPUT:** $callID,\ Locs,\ I_{req_A}$
**OUTPUT:** $true,\ service\ can\ be\ used$
            $false,\ service\ can\ not\ be\ used$
    **repeat** $Service\ b = Registry.choose(I_{req_A})$
        **until** $\neg\ b.undesired(I_{req_A},\ Locs,\ callID)$
    **end**
    **return** $true$
```

However, the used service B selects a Program Analyzer pa and starts the data-flow analysis, by calling $analyze$, cf. 5.

```
**Algorithm 2:** undesired
**INPUT:** $I_{req_A},\ Locs,\ callID$
**OUTPUT:** $true,\ data-flow\ to\ undesired\ locations$
            $false,\ \neg data-flow\ to\ undesired\ locations$
$pa \leftarrow choose();$
**return** $pa.analyze(\ Locs,\ callID,\ I_{req_A},\ sourceText_B)$
```

Besides the data-flow analysis the Program Analyzer pa also stores the context-aware attribute $callID$ of the client to keep track of the used services by client A.

```
**Algorithm 3:** analyze
**INPUT:** $Locs,\ callID,\ I_{req_A},\ sourceText_B$
**OUTPUT:** $true,\ data-flow\ to\ undesired\ locations$
            $false,\ \neg data-flow\ to\ undesired\ locations$
$callID \leftarrow computeCallID(callID)$
**for each** $location$ **in** $Locs$ **do**
    **if** $(location == sourceText_B.myLoc)$ **then return** $true;$
    **end**
**end**
$cl \leftarrow cl.add(callID))$
$I_{req_B} \leftarrow doDataFlowAnalysis(I_{req_A}, sourceText_B)$
**return** $negotiate(callID,\ Locs,\ I_{req_B})$
```

Remark:
As the PA is able to keep track of the analysis requests of client A, it can check for cycles before processing the analysis request. In particular, it checks whether a query $undesired(callID,\ Locs,\ I_{req_B})$ for client A is currently being analyzed, i.e., whether there is an open analysis request $undesired(S', Locs)$ with $S' \subseteq S$. If yes, it can return immediately $false$. This is valid because if there is a data-flow from S' to an undesired location loc, then there must be another call of a provided function to service B with a data-flow to an undesired location. □

Now, with the help of the computed list containing the chain of used services of client A, this information can be used to guarantee, that the data of client A flows only to undesired locations. Before every service connects to another service it can be checked asking the used PA if in the context of the client this connection is allowed.

Remark: We assume a IAAS in a trusted cloud environment [7]. This approach depends on trust in a trusted cloud federation and we are using the encryption mechanism

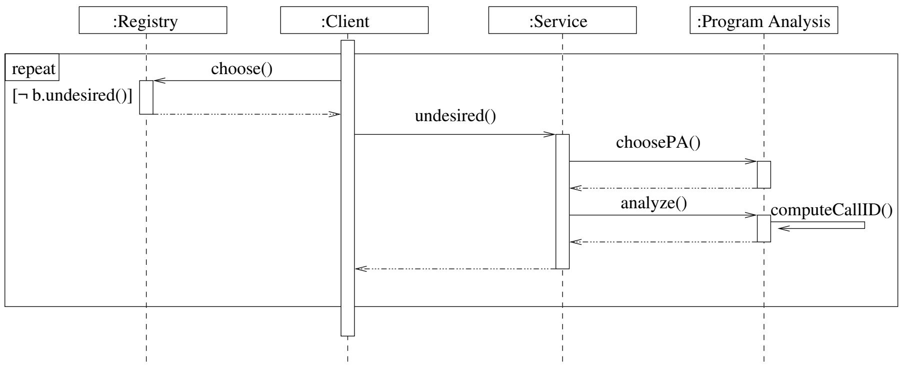


Figure 5. Sequence Diagram of the Context-Aware Mechanism.

described in [6]. □

IV. Related Work

[6] considers data security in the cloud. In contrast to our work, this approach is not context-aware. [8] monitors data-flow between services in order to detect malicious services. They do not do a static data-flow analysis but they assume a multi-tenant cloud infrastructure. Also, context-awareness with respect to the client is not assumed. [9] investigates data-flow analysis in the context of service computing. Compared to our work, they focus on static process adaptation to investigate if a service gets all the data it needs. [10], [11] focus on data security within smart-phone applications. While we allow sensitive data to leave the client, they forbid sensitive data to leave the smart-phone.

There are also many works on context-aware service-oriented systems. Truong and Dustdar [12] present a couple of projects, e.g., CA-SOA [13], CoWSAMI [14], WASP [15] and inContext [16], [17], to make service-oriented systems context-aware. CoWSAMI [14] is an interface-aware context-gathering-environment. CA-SOA [13] formalizes an ontology-based context model. Different views of different clients using a chain of web services were not considered. [18] proposes a multi-tenant service-oriented architecture middleware for Cloud Computing. They focus on multiple users sharing an instance and native multi-tenancy. In contrast to our work, using certain services in context of the location is not considered.

Baldauf et al. also states some requirements that need to be supported by a context-aware system. In contrast to our work, [13], [14], [15], [16], [17] assume, that the context information of the user has to be collected by some mechanism, e.g., polling [16], [13]. In our work, the client itself supports the system with context information, the list of undesired locations, mechanisms like polling are not needed. [19] also presents techniques to compute, with the help of context information, the right service to get coupled to. In our work, the client itself can decide whether a context is given or not. A computation of contextual information [20] to find the best fitting service does not need to be done.

Focusing data security in the cloud is done by [3]. Brandic et al. guarantee data security by data fragmentation. A data analyst or the domain expert decide where data can be stored and which data need to be fragmented and stored in different geographical regions. The client itself can not decide where its data is stored or processed.

To the best of our knowledge, there exist no paper that is using the context information of a client to control the data-flow in the cloud and enables the client to negotiate services.

V. Conclusion

In this work, [6] was extended to allow different views on a service-oriented system in the cloud. The extended work allows multiple clients to decide where their data is stored, processed and transferred within the cloud. Our approach supports different views to fit into multi-tenant service-oriented architectures.

We have two context information: the client information of used services and a list of undesired countries specified by the client. With the extended static data-flow analysis and the contextual information, the coupling of services in context of the user can be computed at runtime. We obtain a multi view or multi-tenant environment with loosely coupled services, which will be coupled on demand in context of the client.

Techniques to collect contextual information, e.g., polling, are not an issue. Every service is supported with the list of undesired countries by the client itself (direct or indirect). Information of the used services are stored by a program analysis service.

To evaluate the proposed approach, the implementation of a tool is in process and subject for future work.

In this work, we considered data-flow analysis on the SaaS (Software as a Service) level. Subject of further work will be the generalization of the data-flow analysis to IaaS (Infrastructure as a Service) and PaaS (Platform as a Service).

Due to the complexity of the IaaS and PaaS, we expect on this level that data-flow analysis becomes more complex and maybe some new abstraction mechanisms are needed for feasibility. Another opportunity for program analysis is to analyze the conformance to compliance rules as they have similar characteristics as data-flow: the client cannot always check the conformance or may even not observe violations of compliance.

Acknowledgment

The authors would like to thank the anonymous reviewers for their helpful comments.

References

[1] *Quantitative Estimates of the Demand for Cloud Computing in Europe and the Likely Barriers to Up-take*, Final Report, IDC Std. D4, July 2012. [Online, retrieved: 03.2013]. Available: http://ec.europa.eu/information_society/activities/cloudcomputing/docs/quantitative_estimates.pdf

[2] D. Durkee, "Why cloud computing will never be free," *Queue*, vol. 8, no. 4, 2010, p. 20.

[3] I. Brandic, S. Dustdar, T. Anstett, D. Schumm, F. Leymann, and R. Konrad, "Compliant cloud computing (c3): Architecture and language support for user-driven compliance management in clouds," in *IEEE CLOUD*, 2010, pp. 244–251.

[4] L. Wei, H. Zhu, Z. Cao, W. Jia, and A. Vasilakos, "Seccloud: Bridging secure storage and computation in cloud," in *Distributed Computing Systems Workshops (ICDCSW), 2010 IEEE 30th International Conference on*, Jun 2010, pp. 52 –61.

[5] "Uniting and strengthening america by providing appropriate tools required to intercept and obstruct terrorism act of 2001 (usa patriot act)," Oct 2001, effective February 1, 2002. [Online, retrieved: 03.2013]. Available: http://thomas.loc.gov/cgi-bin/query/z?c107:H.R.3162.ENR:

[6] M. Weissbach and W. Zimmermann, "Controlling data-flow in the cloud," in *The Third International Conference on Cloud Computing, GRIDs, and Virtualization*, W. Zimmermann, Y. W. Lee, and Y. Demchenko, Eds. ThinkMind, 2012, pp. 24–29.

[7] N. Santos, K. P. Gummadi, and R. Rodrigues, "Towards trusted cloud computing," in *HOTCLOUD*. USENIX, 2009.

[8] J. Du, W. Wei, X. Gu, and T. Yu, "Runtest: assuring integrity of dataflow processing in cloud computing infrastructures," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '10. New York, NY, USA: ACM, 2010, pp. 293–304. [Online, retrieved: 03.2013]. Available: http://doi.acm.org/10.1145/1755688.1755724

[9] W. Song, X. Ma, S. Cheung, H. Hu, and J. Lu, "Preserving data flow correctness in process adaptation," *Services Computing, IEEE International Conference on*, vol. 0, 2010, pp. 9–16.

[10] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6. [Online, retrieved: 03.2013]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924971

[11] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, "PiOS: Detecting privacy leaks in iOS applications," in *Proceedings of the 18th Annual Network & Distributed System Security Symposium (NDSS)*, Feb. 2011. [Online, retrieved: 03.2013]. Available: http://www.isoc.org/isoc/conferences/ndss/11/pdf/9_2.pdf

[12] H. Truong and S. Dustdar, "A survey on context-aware web service systems," *International Journal of Web Information Systems*, vol. 5, no. 1, 2009, pp. 5–31.

[13] I. Y. L. Chen, S. J. H. Yang, and J. Zhang, "Ubiquitous provision of context aware web services," in *Proceedings of the IEEE International Conference on Services Computing*, ser. SCC '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 60–68. [Online, retrieved: 03.2013]. Available: http://dx.doi.org/10.1109/SCC.2006.110

[14] D. Athanasopoulos, A. V. Zarras, V. Issarny, E. Pitoura, and P. Vassiliadis, "Cowsami: Interface-aware context gathering in ambient intelligence environments," *Pervasive Mob. Comput.*, vol. 4, no. 3, Jun. 2008, pp. 360–389. [Online, retrieved: 03.2013]. Available: http://dx.doi.org/10.1016/j.pmcj.2007.12.004

[15] M. Zuidweg, J. Goncalves Filho, and M. van Sinderen, "Using p3p in a web services-based context-aware application platform," in *Proceedings of EUNICE 2003 9th Open European Summer School and IFIP WG6.3 Workshop on Next Generation Networks*, E. Halasz, C. Lukovszki, and T. Marosits, Eds. Budapest: Budapest University of Technology and Economics, Sep. 2003, pp. 238–243. [Online, retrieved: 03.2013]. Available: http://doc.utwente.nl/66531/

[16] H.-L. Truong, L. Juszczyk, S. Bashir, A. Manzoor, and S. Dustdar, "Vimoware - a toolkit for mobile web services and collaborative computing," in *Proceedings of the 2008 34th Euromicro Conference Software Engineering and Advanced Applications*, ser. SEAA '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 366–373. [Online, retrieved: 03.2013]. Available: http://dx.doi.org/10.1109/SEAA.2008.42

[17] H.-L. Truong, S. Dustdar, D. Baggio, S. Corlosquet, C. Dorn, G. Giuliani, R. Gombotz, Y. Hong, P. Kendal, C. Melchiorre, S. Moretzky, S. Peray, A. Polleres, S. Reiff-Marganiec, D. Schall, S. Stringa, M. Tilly, and H. Yu, "incontext: A pervasive and collaborative working environment for emerging team forms," in *Proceedings of the 2008 International Symposium on Applications and the Internet*, ser. SAINT '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 118–125. [Online, retrieved: 03.2013]. Available: http://dx.doi.org/10.1109/SAINT.2008.70

[18] A. Azeez, S. Perera, D. Gamage, R. Linton, P. Siriwardana, D. Leelaratne, S. Weerawarana, and P. Fremantle, "Multi-tenant soa middleware for cloud computing," in *IEEE CLOUD*, 2010, pp. 458–465.

[19] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context aware systems," *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 2, no. 4, Jun. 2007, pp. 263–277. [Online, retrieved: 03.2013]. Available: http://dx.doi.org/10.1504/IJAHUC.2007.014070

[20] A. Danylenko, C. Kessler, and W. Löwe, "Comparing machine learning approaches for context-aware composition," in *Proceedings of the 10th international conference on Software composition*, ser. SC'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 18–33. [Online, retrieved: 03.2013]. Available: http://dl.acm.org/citation.cfm?id=2025951.2025954