PROVIDENCE: A Framework for Private Data Propagation Control in Service-Oriented Systems*

Roman Khazankin and Schahram Dustdar

Distributed Systems Group, Vienna University of Technology
Argentinierstr. 8/184-1, A-1040 Vienna, Austria
lastname@infosys.tuwien.ac.at

Abstract. As data integration proliferates, private information can be spread across the system extensively. Striving for protection of possessed sensitive information, enterprises thus need comprehensive means to control such propagation. As shown in the paper, approaches to date do not address this need sufficiently. We, therefore, propose a private data propagation control framework (Providence), which aims to give a comprehensive view on private data usage throughout the system and to facilitate privacy-related decision making.

1 Introduction

The proliferation of data integration intensifies the propagation of data throughout the enterprise, since the result of one activity can serve as a source for another. The more sources are involved, the easier it is to overlook the inappropriate use of data, as it comes to be maintained in various locations by different parties. Indeed, proliferation of resource virtualization and cloud computing requires even more delicate consideration of privacy concerns [11,18]. Thus, striving for protection of possessed sensitive information, enterprises need comprehensive means of control over the propagation of private data.

We address the problem in the general case of Service Oriented Architectures (SOAs) where actual implementation of services is inaccessible and their functionality is unrestricted. This implies, for example, that data might be stored by one service and retrieved or transferred later by another service, so this fact can't be established by workflow analysis. To our best knowledge, there are no solutions to date that address this problem.

In this paper we present the *Pri*vate *D*ata Propagatio*n* *C*ontrol Fram*e*work (Providence). Monitoring the message exchange, it employs content inspection that is successfully applied in Data Loss Prevention (DLP) solutions[1]. The framework detects and logs private data disclosures that happen in the SOA environment. The log is then used to give a comprehensive view on private data usage

* This work was supported by the Vienna Science and Technology Fund (WWTF), project ICT08-032.

[1] http://www.cio.com/article/498166/Best_Data_Loss_Prevention_Tools

E. Di Nitto and R. Yahyapour (Eds.): ServiceWave 2010, LNCS 6481, pp. 76–87, 2010.

throughout the SOA and to facilitate privacy-related decision making. The rationale behind the framework is to have a practically-applicable solution that requires as few integration efforts as possible.

The paper is organized as follows: Section 2 outlines the motivating scenario for our work, Section 3 discusses the Providence framework; related work is discussed in Section 4, evaluation of the approach and the framework is given in Section 5. Section 6 concludes the paper.

2 Motivating Scenario

[16] outlines the scenario with harmful inappropriate use of private data, where Alice, who has a child with a severe chronic illness, buys a book about the disease from online book store. Later she applies for a job and gets rejected because the employer somehow received the information about her purchase and flagged her as high risk for expensive family health costs.

Below we show how the aforementioned scenario can take place due to inappropriate control over the private data in manifold data integration activities in SOAs.

Consider an enterprise Things'n'Books, which runs several businesses including online book trading. It has complex infrastructure with various departments, each responsible for a particular business. Things'n'Books also has a delivery department handling all delivery needs of a company and a joint human resources department. To achieve interoperability between departments, Things'n'Books employs SOA based on Web services.

When the client orders books on-line, the order details are put into orders database exposed as a data service (OrderService). Once the payment is received, a delivery workflow, implemented in BPEL, is initiated. It takes care of order preparation and eventually calls the Web service exposed by the delivery department (DeliveryService) which takes delivery items and the recipient contact data on input.

The delivery department, in time, besides using the received data to perform delivery, stores this data for both accounting reasons and further usage in delivery routes analysis and optimization through the instrumentality of an enterprise mashup *M*. The human resources department also uses an enterprise mashup platform and benefits from partial re-use of mashup *M*. Also, it outsources some duties to an independent HR company, and a representative of that company, *Eve*, works in Things'n'Books's office.

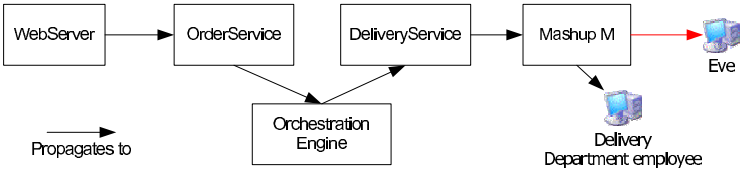

Fig. 1. Private data propagation scenario

Thus, if Alice purchases a book from Things'n'Books's, Eve has an opportunity to access this information via the mashup M having no access to its origin source, the online orders database. The propagation chain is illustrated on Fig. 1.

3 Providence Framework

In this section we introduce the Providence framework. We show the architecture of the framework, give formal specifications for control paradigms and replaceable components, specify implementation remarks, and outline limitations of the framework.

3.1 Architecture

We start from specifying required adoptions in SOA, then we describe each component of the framework. The design of the framework is illustrated in Fig.2. The framework demands two adoptions to be performed in SOA infrastructure, requiring it to:

1. Submit private data entries to the registrator service of the framework, whenever such entries appear in the system (e.g., when private data is entered manually, received from a partner organization, or submitted to a service in SOA).
2. Intercept messages exchanged in the integration and submit them together with the context information to the monitor service of the framework for inspection (e.g., SOAP messages that travel on Enterprise Service Bus). Context information enables the framework to distinguish between different integration activities. Examples of context elements are shown in Table 1.

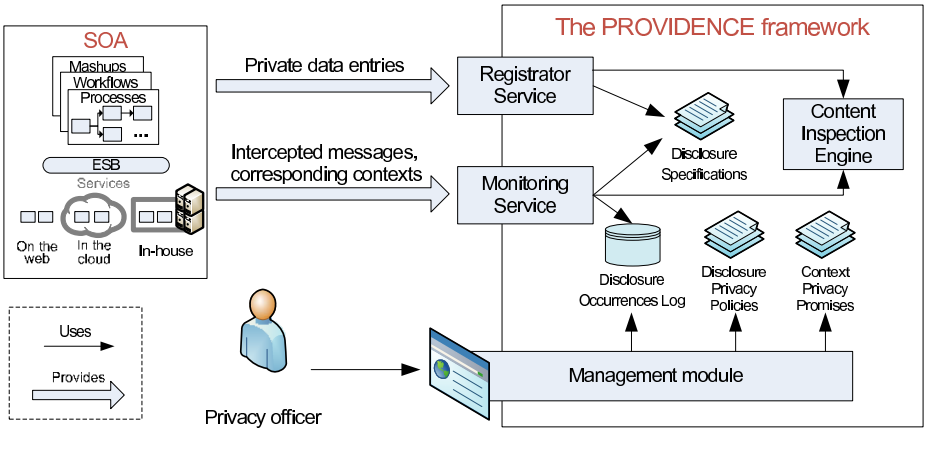

Fig. 2. The Providence framework

Table 1. Examples of context elements

Level	Context element
Network	Requestor host Responding endpoint
Application	Consuming application's identifier Requestor's credentials from responder's perspective Requestor's credentials in requestor application, e.g. in mashup platform
Process	Corresponding process identifier, e.g. in business process engine

Registrator Service. Private data entries are submitted to the Registrator Service in form of *disclosures*. Each disclosure contains typified pieces of private data, *primitives*, and a logical rule, whose variables correspond to those primitives. Also, each disclosure has a type. For example, a disclosure of private information about a 50000$ bank loan issued to John Johnson who lives at 1040 Paulanergasse 2 can be specified as primitives p_1 of type $Name$ which has value *"John Johnson"*, p_2 of type *Address* which has value *"1040 Paulanergasse 2"*, and p_3 of type *Amount* which has value *50000*, with rule "$(p_1$ *or* $p_2)$ *and* p_3" and type *"PersonalLoan"*. It means that if the message exchange contains the amount together with either address or name, possibly in the different form (e.g., J. Johnson), then the disclosure occurs. The type of primitive indicates how its form can vary. When a disclosure is submitted, primitives are separated from rule and group identifier, so only the mapping is kept. The rule and group identifier are stored to Disclosure Specification Repository, the primitives are registered at the Content Inspection Engine.

Table 2. Examples of primitive types

Type	Value	Possible detectable forms
Name	John Johnson	John Johnson, J. Johnson, JOHNSON J
Date	02.01.2010	02/01/10, 1 Feb 10
Amount	50000	50 000, 50.000

Monitoring Service. The monitoring service receives messages exchanged in data integration processes and corresponding context information. The messages are forwarded to the Content Inspection Engine that detects previously registered primitives in messages' content. If any primitives are detected, the corresponding disclosures are retrieved from Disclosure Specifications Repository and their rules are checked against detected primitives. Occurred disclosures are logged together with the context.

Content Inspection Engine. The Content Inspection Engine is responsible for detecting primitives in the messages content. It receives primitives from

Registration Service and content for inspection from the Monitoring Service. The type of a primitive defines the possible transformations of its data value. Examples of such types are shown in Table 2. There are various content inspection techniques and algorithms [15,8] whose explanation and evaluation is beyond the scope of this paper.

Management Module. This is the core component of the framework. Using the information from disclosure occurrences log, this module provides means to control private data propagation to the privacy officer. She can (i) assign privacy promises to contexts according to real data usage practices in those contexts, (ii) assign privacy policies to disclosure types, and, based on actual disclosure occurrences log and assigned policies, (iii) get answers to following types of questions:

1. Which privacy policy violations happened?
2. Which disclosures happen in specified context?
3. In which contexts disclosure of specified type happens?
4. What promise is enough for specified context to keep compliant with current private data usage practices?
5. How is the private data of specified type actually used?
6. How was the particular piece of private information used?
7. What if we want to set another policy for private data or context, what violations will it produce for the current environment?

The explanation of corresponding answers is given in the next section.

3.2 Formal Specifications

The framework is not coupled to a particular policy and context specification. Nevertheless, the framework's logic heavily relies on these components. Therefore, we specify formal requirements for these components and formally explain the logic of the management module as follows:

Policy. We assume that both private data policies and context promises are expressed in the same form. The policy specification should reflect both requirements (policies) and promises for data usage. It can encapsulate allowed usage purposes, receiving parties, and required obligations[2]. The implementation must provide a means to check whether data treatment under one policy satisfies another policy (promise) and to calculate the intersection and union of policies.

Formally, let P - set of all possible policies. There must be defined relation $satisfies$ on P which we mark as $\prec$, so that $p_1 \prec p_2; p_1, p_2 \in P$ indicates that data treatment under policy p_1 satisfies policy p_2.

Further, there must be defined union operator $\cup : P \times P \to P$ that associates two policies with a policy that is satisfied by either of them: $\forall p_1, p_2 \in P \; p_1 \prec (p_1 \cup p_2), p_2 \prec (p_1 \cup p_2)$.

[2] Currently the framework does not support temporal policy elements such as Retention.

Finally, there must be defined intersection operator $\cap : P \times P \rightarrow P$ that associates two policies with a policy that satisfies both of them: $\forall p_1, p_2 \in P\ (p_1 \cap p_2) \leftharpoondown p_1, (p_1 \cap p_2) \leftharpoondown p_2$.

Context. Context specification should distinguish activities that happen in SOAs in a way that it is possible to assign feasible privacy promises to those activities. The implementation must provide a means to check whether one context is a subcontext of another. Occurrence of a disclosure in some context implies its occurrence in all contexts, of which the given context is a subcontext.

Formally, let C - set of all possible contexts. There must be defined relation $\subseteq$ on C, so that $c_1 \subseteq c_2; c_1, c_2 \in C$ indicates that c_1 is a subcontext of c_2.

Configuration. The privacy officer manages context promises and private data policies. Formally, we can consider that for a given point in time there is *configuration*, which contains these mappings. As policy or promise might be unassigned, we introduce unassigned policy Ω, and extend P to $P' = P \cup \{\Omega\}$. The rationale of unassigned policy is passivity in computations, thus we refine $\leftharpoondown, \cap, \cup$ for P' as follows: $\Omega \leftharpoondown \Omega, \Omega \cap \Omega = \Omega, \Omega \cup \Omega = \Omega$; $\forall p \in P \Rightarrow \Omega \leftharpoondown p, p \leftharpoondown \Omega$; $\forall p \in P \Rightarrow \Omega \cap p = p, p \cap \Omega = p$; $\forall p \in P \Rightarrow \Omega \cup p = p, p \cup \Omega = p$.

Configuration is a tuple ($Promise$, $Policy$), where $Promise : C \rightarrow P'$, $Policy : T \rightarrow P'$, where T is a set of all disclosure types. $Promise$ maps contexts to privacy promises assigned, $Policy$ maps disclosure types to privacy polices assigned. For any context, its promise must always satisfy a promise of any its subcontext: $\forall c, c' \in C : c' \subseteq c \Rightarrow Promise(c) \leftharpoondown Promise(c')$.

Management Actions Logic. Based on the current configuration and a part of the log (e.g., log for last month), management module enables the privacy officer to get answers on question templates.

Formally, let $L = \{\langle c_i, d_i \rangle\} : c_i \in C, d_i \in D, 1 \leq i \leq N$ - disclosure occurrences log, where D is a set of all registered disclosures, N is number of log records, d_i is a disclosure and c_i is a context that corresponds to log entry i. Let $Type(d), d \in D$ indicate the type of disclosure d. Now, given current configuration $\Lambda = (Promise, Policy)$, for any part of log $L' \subset L$ answers can be given as specified in Table 3:

3.3 Implementation

The prototype of Providence framework was designed and implemented considering separation of concerns. The core components implement the principal logic using unified interfaces to replaceable components that are responsible for content inspection and privacy policies storing. Replaceable components were implemented in an elementary way to demonstrate functionality of the framework. The prototype's work is demonstrated in the screencast[3]. The evaluation of results is given in Section 5.

[3] `http://www.infosys.tuwien.ac.at/prototype/Providence/`

Table 3. Privacy-related questions and answers

Question	Answer
1. Which privacy policy violations happened?	$\langle c, d\rangle \in L' : Promise(c) \not\vdash Policy(Type(d))$
2. Which disclosures happen in context c	$d : c' \subseteq c, \langle c', d\rangle \in L'$
3. In which contexts disclosure of type T happens?	$c : Type(d) = T, \langle c, d\rangle \in L'$
4. What promise p is enough for context c to keep compliant with current private data uses?	$p = \bigcap_{c' \subseteq c, \langle c', d\rangle \in L'} Policy(Type(d))$
5. What policy p reflects the actual usage of private data in disclosures of type T?	$p = \bigcup_{Type(d)=T, \langle c, d\rangle \in L'} Promise(c)$
6. What policy p reflects the actual usage of private data in disclosure d?	$p = \bigcup_{\langle c, d\rangle \in L'} Promise(c)$
7. If we want to change the configuration, what violations will the new configuration Λ' produce?	(1.) for configuration Λ'.

The success of the framework will depend on the selection of particular adoption points and replaceable components. The context elements should be chosen in a way that (i) disclosures that occur during the same data integration activity share as much contexts as possible, and (ii) disclosures that occur during different activities share as few contexts as possible.

3.4 Limitations

Our framework has several limitations:

- Context elements must be accessible by message submitting adoption point. However, it is a technical matter.
- Content inspection might give false negatives for complex data integration patterns. False positives might occur as well. The balance between false positives, false negatives, and the detection rate in general will always depend on particular information systems, business specifics, IT architecture and inspection engine settings. Therefore, to estimate these rates, it is necessary to test the approach in real business environments. However, fine-tuned DLP tools provide 80-90% accuracy and almost no false positives in commercial product tests[4].
- It is impossible to obtain the data for inspection from encrypted messages. However, the adoption points can be selected in such a way, that data will be sent to inspection before encoding. As the monitoring service does not forward the data any further and is supposed to be deployed in-house, this should not raise any security issues.
- Content inspection of internally exchanged messages brings computational overhead in the system. However, submitted messages can be analyzed asynchronously on the dedicated resource, neither interrupting nor slowing down

[4] `http://www.cio.com/article/498166/Best_Data_Loss_Prevention_Tools`

business processes. Thus, we regard the framework's performance as a secondary concern.
– The privacy officer will need to obtain actual context privacy promises. However, it is inevitable due to assumption of inaccessibility of service implementations. Moreover, given the solution that is able to conclude about actual treatment of data by a service (e.g., using source code analysis), it can be coupled with the framework, thus automating context promises assignment.

4 Related Work

To the best of our knowledge, none of current research fully addresses the issue of private data propagation control within the organization SOAs.

Works like [6,9,1] eventually come to rule-based access restriction to private data. However, none of these solutions can control the usage of private data *after* it gets released from the guarded resource.

Approaches like [7,4,3] address the issue only within bounds of processes or workflows, whereas integration techniques go beyond this paradigm. For instance, services can call each other independently of the workflow engine, or they can store private data, so it can be later picked up by other processes.

[17] proposes a framework for building privacy-conscious composite Web services which confronts service consumer's privacy preferences with component services' policies. This framework does not address scenarios where private data is firstly stored by a component service and then retrieved outside of the composite service's scope, whereas such scenarios are inevitable for data integration.

[14] proposes to record and store logs of private data usage together with this data throughout its lifespan. It requires services, which consume private data, to record all actions performed with this data to the log, and return this log together with the response. Such logs can then be analyzed to conclude about appropriate use of data. Unlike our framework, this approach interferes with the business logic of services, requiring them to report performed actions. Therewith, it does not consider cases, when private data is stored by one service and then retrieved and used by another.

[10] proposes to encrypt the messages that are exchanged between enterprises, so that a request to the trusted authority must be performed to decrypt them. Using this technique, trusted authority can check privacy promises of enterprises and control the access to the private data. This technique neither applies to private data control within the enterprise, nor addresses the usage of data once access to it was granted.

[13] provides tool support for solving a number of problems in data usage control. However, the authors rely on their own model of a distributed system [12] which assumes that each concerned peer has a secure data store that possesses all the private data available to this peer, and that the peer routes this data through the usage control mechanisms whenever the data leaves this store. Such assumption makes the work inapplicable for SOAs in the general case.

[2] addresses similar problems of private data misuses. The authors build their work upon the Private Data Use Flow expressed as a state machine. However,

they give neither instructions nor any explanations of how to build this state machine in an automated fashion having a live system. Unlike it, our approach was designed to work in real SOAs.

Data Loss Prevention solutions are different to our approach in a way that they protect the system from casual leakages of private data from end users, whereas our approach aimed to detect systematic inappropriate uses of data which ensue from SOA complexity.

5 Evaluation

To evaluate our work, we emulated (created and deployed) the environment and business logic of scenario from Sec. 2 using the Genesis 2 testbed [5] and adopted the implemented Providence prototype using service interceptors for monitoring. Besides elements and logic outlined in Sec. 2, the testing environment included printery department which executes private orders and employs the delivery department for shipping. After performing the business logics simulation, we tested the management module's functionality to proof the concept. The screencast of the performed emulation is available online[5].

The business logic was executed as enumerated below. Figure 3 depicts the testing environment and log records inserted during the emulation.

1. Two online book orders are made, thus two *BooksPurchase* disclosures are registered in the framework.
2. *PlaceOrder* operation of *OrderService* is called for each purchase, disclosures are detected, and log records 0,1 are generated.
3. For each purchase orchestration engine retrieves details via *GetOrder* operation of *OrderService* (log records 2,4) and later submits it to *Deliver* operation of *DeliveryService* (log records 3,5).
4. Delivery department uses *MashupService* to run the mashup, which features *BooksPurchase* disclosures (log records 6,7).
5. HR department accesses private information via *OrderService* (log record 8), *DeliveryService* (log records 9,10), and *MashupService* (log records 11,12)
6. Printery department composes private printery order, thus a *PrinteryOrder* disclosure is registered in the framework.
7. The order details are submitted to the *Deliver* operation of *DeliveryService* thus generating log record 13.
8. Delivery department runs the mashup which now involves printery order (log record 14).

Disclosure type policies and context promises of the testing environment are shown in Fig. 4. Data from *BooksPurchase* disclosures is allowed to be used for system administration and research and development; *PrinteryOrder*'s policy is not assigned. Data submitted to *OrchestrationHost* within bounds of *OrderProcess* is known to be used for system administration and individual analysis. Data

[5] `http://www.infosys.tuwien.ac.at/prototype/Providence/`

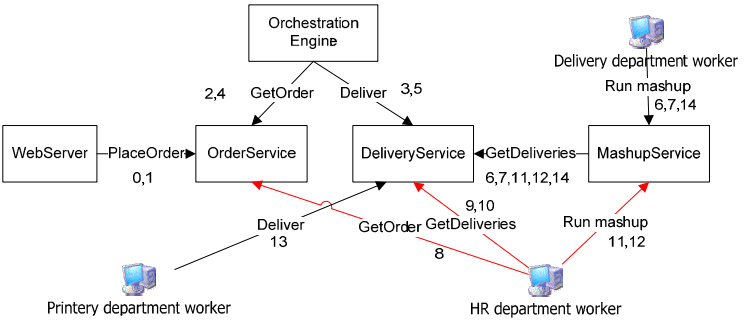

Fig. 3. Testing environment

Disclosure type	Policy
BooksPurchase	Self=[admin, develop]
PrinteryOrder	Unassigned

Context	Promise
process = OrderProcess destinationHost = OrchestrationHost	Self=[admin, individual-decision]
destinationHost = DeliveryServiceHost	Self=[historical]
destinationLogin = DeliveryWorker destinationHost = MashupHost	Self=[develop]
destinationHost = HRDepartmentHost	ThirdParty=[other]
destinationLogin = HRWorker destinationHost = MashupHost	ThirdParty=[other]

Fig. 4. Privacy policies and promises in testing environment

Table 4. Detected violations

Record N	Context	Policies
2,4	process = OrderProcess host = OrderServiceHost destinationHost = OrchestrationHost	Private data policy:{Self=[admin, develop]} Context promise:{Self=[admin, individual-decision]}
3,5	process = OrderProcess host = OrchestrationHost destinationHost = DeliveryServiceHost	Private data policy:{Self=[admin, develop]} Context promise:{Self=[historical]}
8	host = OrderServiceHost destinationHost = HRDepartmentHost	Private data policy:{Self=[admin, develop]} Context promise:{ThirdParty=[other]}
9,10	host = DeliveryServiceHost destinationHost = HRDepartmentHost	Private data policy:{Self=[admin, develop]} Context promise:{ThirdParty=[other]}
11,12	hostLogin = MashupService destinationLogin = HRWorker host = DeliveryServiceHost destinationHost = MashupHost	Private data policy:{Self=[admin, develop]} Context promise:{ThirdParty=[other]}

propagated to *DeliveryServiceHost* is known to be used for historical preservation. If data is sent to *MashupHost* and remote login is *DeliveryWorker*, then it is used for research and development. Data propagated to *HRDepartmentHost* or retrieved by *MashupHost* with remote login *HRWorker* is known to be exchanged with third party for undefined purposes.

Table 4 shows the output of the framework's management module for violation detection. Having discovered such violations, privacy officer can decide to strengthen context promise (e.g., as in case of orchestration engine, log records 2,4), loose private data policy (e.g., as in case of delivery service, log records 3,5) or assume administrative measures to prevent similar access to private data (e.g., in case of HR worker, log records 8,9,10,11,12).

The management module is able to give answers on question templates from Sec. 3.2, such as itemizing disclosure types that take place in a context (e.g., `BooksPurchase`, `PrinteryOrder` for the context `destinationHost = DeliveryServiceHost`), itemizing the contexts, in which disclosure of particular type happens (e.g., `PrinteryOrder` takes place in two contexts, see Fig. 5), or inferring the actual policy for particular disclosure type (e.g., `Self=[historical, develop]` for `PrinteryOrder`).

host	= PrinteryDepartmentHost
destinationHost	= DeliveryServiceHost

hostLogin	= MashupService
destinationLogin	= DeliveryWorker
host	= DeliveryServiceHost
destinationHost	= MashupHost

Fig. 5. Contexts where PrinteryOrder disclosure takes place

Thus, the framework explicitly detects any disclosures that happen during the data integration and helps to prevent the inappropriate access to the data even after it is propagated throughout the system.

6 Conclusion

In this paper we discussed the challenges of private data propagation control in SOAs. We introduced the Providence framework which is able to give a comprehensive view of private data usage throughout the enterprise and to facilitate privacy-related decision making. We demonstrated our framework's functions and discussed limitations regarding its adoption in real-world SOAs. Future investigations include (i) extension for temporal policy elements, (ii) framework integration with enterprise policy brokers, (iii) automation of decisions to prevent violations, (iv) coupling the framework with data integration tools to facilitate design-time privacy concerns consideration.

References

1. Ashley, P., Hada, S., Karjoth, G., Schunter, M.: E-p3p privacy policies and privacy authorization. In: Jajodia, S., Samarati, P. (eds.) WPES, pp. 103–109. ACM, New York (2002)
2. Benbernou, S., Meziane, H., Hacid, M.: Run-time monitoring for privacy-agreement compliance. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 353–364. Springer, Heidelberg (2007)

3. Cheung, W., Gil, Y.: Towards privacy aware data analysis workflows for e-science. In: AAAI Workshop on Semantic e-Science 2007, pp. 22–26. AAAI Press, Menlo Park (2007)
4. Gil, Y., Fritz, C.: Reasoning about the appropriate use of private data through computational workflows. In: Spring Symposium on Intelligent Privacy Management. AAAI Press, Menlo Park (2010)
5. Juszczyk, L., Dustdar, S.: Script-based generation of dynamic testbeds for soa. In: ICWS, pp. 195–202. IEEE Computer Society, Los Alamitos (2010)
6. Karjoth, G., Schunter, M., Waidner, M.: Privacy-enabled services for enterprises. In: DEXA Workshops, pp. 483–487. IEEE Computer Society, Los Alamitos (2002)
7. Li, Y., Paik, H., Chen, J.: Privacy inspection and monitoring framework for automated business processes. In: Benatallah, B., Casati, F., Georgakopoulos, D., Bartolini, C., Sadiq, W., Godart, C. (eds.) WISE 2007. LNCS, vol. 4831, pp. 603–612. Springer, Heidelberg (2007)
8. Lin, P.C., Lin, Y.D., Lai, Y.C., Lee, T.H.: Using string matching for deep packet inspection. IEEE Computer 41(4), 23–28 (2008)
9. Casassa Mont, M., Pearson, S., Thyne, R.: A systematic approach to privacy enforcement and policy compliance checking in enterprises. In: Fischer-Hübner, S., Furnell, S., Lambrinoudakis, C. (eds.) TrustBus 2006. LNCS, vol. 4083, pp. 91–102. Springer, Heidelberg (2006)
10. Mont, M.C., Pearson, S., Bramhall, P.: Towards accountable management of identity and privacy: Sticky policies and enforceable tracing services. In: DEXA Workshops, pp. 377–382. IEEE Computer Society, Los Alamitos (2003)
11. Pearson, S.: Taking account of privacy when designing cloud computing services. In: Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing, pp. 44–52. IEEE Computer Society, Los Alamitos (2009)
12. Pretschner, A., Hilty, M., Basin, D., Schaefer, C., Walter, T.: Mechanisms for usage control. In: Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, pp. 240–244. ACM, New York (2008)
13. Pretschner, A., Ruesch, J., Schaefer, C., Walter, T.: Formal analyses of usage control policies. In: ARES, pp. 98–105. IEEE Computer Society, Los Alamitos (2009)
14. Ringelstein, C., Staab, S.: Dialog: A distributed model for capturing provenance and auditing information. International Journal of Web Services Research 7(2), 1–20 (2010)
15. Sourdis., I.: Designs & Algorithms for Packet and Content Inspection. Ph.D. thesis, Delft University of Technology (2007)
16. Weitzner, D.J., Abelson, H., Berners-Lee, T., Feigenbaum, J., Hendler, J., Sussman, G.J.: Information accountability. Commun. ACM 51(6), 82–87 (2008)
17. Xu, W., Venkatakrishnan, V.N., Sekar, R., Ramakrishnan, I.V.: A framework for building privacy-conscious composite web services. In: 4th IEEE International Conference on Web Services (Application Services and Industry Track) (ICWS) (2006)
18. Yuhanna, N., Gilpin, M., Hogan, L., Sahalie, A.: Information fabric: Enterprise data virtualization. White Paper. Forrester Research Inc. (2006)