Towards Cloud-based Collaborative Software Development: A Developer-Centric Concept for Managing Privacy, Security, and Trust

Roy Oberhauser
Computer Science Dept.
Aalen University
Aalen, Germany
roy.oberhauser@htw-aalen.de

Abstract—**Cloud-centric collaboration in (global) software development is gaining traction, resulting in new development paradigms such as Tools-as-a-Service (TaaS). Yet both within and between clouds, there are associated security and privacy issues to both individuals and organizations that can potentially hamper collaboration. In this paper, an inter-cloud security and privacy concept for heterogeneous cloud developer collaboration environments is described that pragmatically addresses the distributed collection, storage, transmission, and access of events and data while giving individuals fine-granularity control over the privacy of their collected data. In a case study, the concept was implemented and evaluated by adapting an existing collaborative development and measurement infrastructure, the Context-aware Software Engineering Environment Event-driven framework (CoSEEEK). The results showed its practicality and technical feasibility while presenting performance tradeoffs for different cloud configurations. The concept enables infrastructural support for privacy, trust, and transparency within teams, and can support compliance with privacy regulations in such dynamic collaborative environments.**

Keywords-cloud-based software engineering environments; cloud-based software development collaboration; global software development; privacy; security; trust

I. Introduction

Global software development (GSD) [1] is increasingly taking advantage of cloud-based software applications and services [2] and realizing its collaboration potential. Data acquired and utilized during the software development and maintenance lifecycle is no longer necessarily locally controlled or even contained within an organization, but may be spread globally among various cloud providers with the acquired data retained indefinitely. Tools-as-a-Service (TaaS) [3] and cloud mashups will enable powerful new applications that utilize the acquired SE data [4]. And while the technical landscape is changing, the corporate landscape is also. A 2005 survey of American corporations conducted by the American Management Association showed that 76% monitored employee Internet connections, 50% stored and reviewed employee computer files, and 55% retained and reviewed email messages, with a rapidly increasing trend [5].

The ability to measure and minutely observe and track software developers during their work is becoming technically and economically viable to employers, managers, colleagues, virtual teams, and other entities. While metrics can be useful for personal improvement (cp. Personal Software Process), abuse is also possible (consider misuse of public profiling). While software services and apps for the public typically attend to user privacy due to their longevity, mass accessibility, and legal scrutiny, relatively little attention has been paid to the privacy needs of software developers, an estimated 17 million worldwide [6].

Consequently, privacy is becoming a looming concern for software developers that faces unique technical challenges that affect collaboration: it involves a highly dynamic technical environment typically at the forefront of software technology and paradigms (e.g., new languages, compilers, or platforms), uses diverse tools ([3] identifies 384) and heterogeneous project-specific tool chains (e.g., application lifecycle management, version control systems, build tools, integrated development environments, etc.), it is project-centric (unique, short-lived undertakings), it may involve multinational coordination (offshoring), etc.

Yet the trust climate plays a vital role in the success of virtual and distributed teams [7], and trust and transparency are considered vital values for effective teams and collaboration [8][9]. Where trust exists (cp. Theory Y [10]), collected data can be utilized collaboratively to enhance team performance [10], for instance by utilizing event data to coordinate and trigger actions and to provide insights, whereas where data is misused as an instrument of power, monitoring, or controlling (cp. Theory X [10]), individuals require mechanisms for protection. Since the technical development infrastructure cannot know a priori what trust situation exists between some spectrum of complete trust to complete distrust, infrastructural mechanisms should support collaboration within some spectrum, while allowing the individuals and organizations to adapt their level of data transparency to the changing trust situation.

Privacy is control over the extent, timing, and circumstances of sharing oneself. Cloud service users currently have few personal infrastructural mechanisms for retaining and controlling their own personal data. Diverse privacy regulations are applicable within various geographic realms of authority. Various (overlapping) (multi-)national laws and regulations may apply to such (global) collaborative cloud contexts. For instance, Germany has a Federal Data Protection Act, the European Union has a Data Protection Directive 95/46/EC, and within the United States, various states each have their own internet privacy laws.

Many privacy and security principles are typically involved including notice, consent, disclosure, security, earmarking, data avoidance, data economy, etc. Various challenges for security and privacy in cloud environments remain [11][13]. In the interim, pragmatic infrastructural approaches are needed to deal with the issues in some way.

The main contribution of this paper is to elucidate requirements for and describe a solution concept that pragmatically addresses various privacy and security concerns in cloud-based dynamic heterogeneous collaborative development environments (CDE). It is based on service layering, introduces distributed cloud-based datasteading for individuals, and mediates trust with brokers. Its technical feasibility and performance tradeoffs were investigated in a case study.

This paper is organized as follows: the next section details and provides some justification for the assumptions and requirements for a solution, and Section 3 describes related work. In Section 4, the solution concept is introduced, and the following section provides details of a technical implementation based on the concept. Evaluation results are presented in Section 6, which is then followed by a conclusion and description of future work.

II. Requirements

The following requirements, assumptions, or constraints (denoted by the prefix R: in italics) were elicited from the primary problems, goals, and challenges introduced in the preceding section, and considered to be generally applicable for any conceptual solution. They are summarized here to highlight key considerations in the solution concept.

Multi-cloud configurability (*R:MCC*): private cloud (*R:PrC*), public cloud (*R:PuC*), and community cloud (*R:CoC*) support for a wide array of deployment options. *Provider-specific cloud API independence* (*R:PAI*) to support wide applicability and avoid provider lock-in. *Cloud compatibility* (*R:CCO*) with current public cloud provider and private cloud APIs and services, meaning no exotic solutions requiring special configurations that would limit usage. *Single tenancy* (*R:ST*) in the personal (developer's) cloud to reduce risk (e.g., to avoid a misconfiguration compromising a much larger set of tenants simultaneously) and avoid access by an organizational administrator, which involves an additional trust issue.

Disclosure (*R:D*): three fundamental levels of shall be supported: non-disclosure, anonymized disclosure, and personally-identifiable disclosure to specific requestors. *Sensor Privacy* (*R:SP*): It is assumed that any client-side and server-side sensors, (e. g., version control system sensors) distribute personally-identifiable events according to a privacy concept. *Entity-level privacy control* (*R:EPC*): granularity of privacy is controllable by any and all entities involved (persons, organizations).

Restricted network access (*R:RNA*) to collaboration participants, e.g., via Virtual Private Networks (VPN), to reduce cloud accessibility to collaborators only. *Secure communication* (*R:SC*) to protect internal data transmission, even within a VPN for personal privacy. *Basic security mechanisms* (*R:BSM*): Reliance on widely-available off-the-shelf security mechanisms (e.g., HTTPS), without any dependence on specialized or exotic hardware or software security platforms (e. g., Trusted Platform Module) or research-stage mechanisms that would constrain its practicality. Beyond (*R:SC*), e*ncryption* (*R:ENC*) can protect data accessibility and storage.

Trusted code implementation (*R:TCI*): Open source and/or independent code audits together with secure distribution mechanisms (e.g., via digital signatures from a trusted website) provide assurance that the code implementation can be trusted. Additionally, remote runtime *code integrity verification* (*R:CIV*) shall be supported to allow agents (e.g., automated temporally random auditing requests or manually initiated user requests) to detect any tampering with the implementation, sensors, configuration, or the compromise of any privacy safeguards.

In summary, a primary tenet is that organizations and teams want to support privacy freedom for individuals, support and value self-organizing teams, and not hinder electronic collaboration and communication. While together these requirements are in no way sufficient or complete, they nevertheless provide a practical basis and can be useful for furthering discussion.

III. Related Work

In the area of global software development, [3] discusses support for TaaS and [14] Software-as-a-service in collaborative situations. Neither go into detail on various privacy issues, nor is support for various aforementioned requirements, e.g., individual (*R:EPC*). Examples industrial offers for cloud-based collaboration include Atlassian OnDemand and CollabNet CloudForge. Individual (*R:EPC*, *R:D*) do not appear to be supported.

Work on more general multicloud collaboration includes [4], which similarly supports opportunistic collaboration without relying on cloud standardization based on the use of proxies. However, aspects such as (*R:BSM*, *R:CI, R:EPC*) were not considered and a technical implementation was not investigated.

Work in the area of standardization and reference architecture includes [15], which mentions privacy but fails to prescribe a solution. [16] lists various security and interoperability standards and their status, but their maturity and market penetration considering (*R:MCC*) and (*R:CCO*) remain issues.

Various general cloud security mechanisms have been proposed. Privacy as a Service (PasS) [17] relies on secure cryptographic coprocessors to provide a trusted and isolated execution and data storage environment in the computing cloud. However, its dependency on hardware within cloud provider infrastructure hampers (*R:PAI*). Data protection as a service (DPaaS) [18] is intended to be a suite of security primitives that enforce data security and privacy and are offered by a cloud platform. Yet this would inhibit (*R:PAI*). Other work such as [19] describe privacy-preserving fine-grained access control and key distribution mechanisms, but are not readily available for a pragmatic approach that is usable today (*R:BSM*).

IV. SOLUTION CONCEPT

For a cloud-based context-aware collaboration system to have satisfactory utility, it will depend on some type of event and data collection and communication facilities. Thus this foundational infrastructure should be equipped with basic trust and security mechanisms, such that upper level services such as context-awareness and collaboration can ensue.

Thus, to provide a flexible solution for such environments, a primary principle in the solution is the application of the *Service Layer* design pattern to provide a decoupling and separation of concerns shown in Figure 1. The lower conceptual Event and Data Services Layer includes event and/or data services for an entity (person/team/organization) including acquisition, storage, retention, and dissemination, while the upper Collaboration and Tools Services Layer includes CDE and tool services that utilize lower layer data to provide collaboration, data sharing, analytics, and other value-added services over which an entity may have more limited privacy control mechanisms.

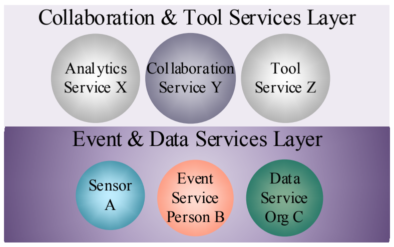

Figure 1: Services Layer Pattern

A second solution principle is the introduction of a *datastead*, shown in Figure 2, which is loosely analogous to the concept of homesteading or seasteading. In this case, an individual (or some unit) manages and controls clearly delineated data resources in the cloud for which they have or receive responsibility and ownership rights. The technical implementation of a datastead can be in the form of a personal cloud in the case of an individual, or within a private cloud for an organization. The third principle is the inclusion of a *Trust Broker* that mediates between service and data access, acting as both a cloud service broker (for interoperability with various tools) and cloud security broker (for security). Akin to the Trusted Proxy pattern [20] and Policy Enforcement Point [20], it constrains access to protected resources and allows custom, finely-tuned policies to be enforced (*R:EPC*). Rules can be used to configure and distinguish/filter access by event types, timeframes, projects, etc. It provides secure communication mechanisms (*R:SC*) to authenticate and authorize data acquisition and data dissemination in the datastead, as well as interoperability mechanisms for various collaboration and tool services. Only client requests from preconfigured known addresses are accepted. A management interface to the Trust Broker provides the datastead owner with policy management capabilities. It also supports data anonymization on a per request basis if so configured. For secure storage, the Trust Broker encrypts (*R:ENC*) acquired events and data (Encrypted Storage pattern [20]) to prevent unauthorized access by administrators or intruders, and protects access to the encrypted storage typically on a single port (Single Access Point pattern [20]). The Trusting Broker supports runtime code integrity (*R:CI*) via remote attestation, and a client, called the Trusting Tool, can be invoked periodically or event-based to ensure that the Trust Broker has not been tampered with.

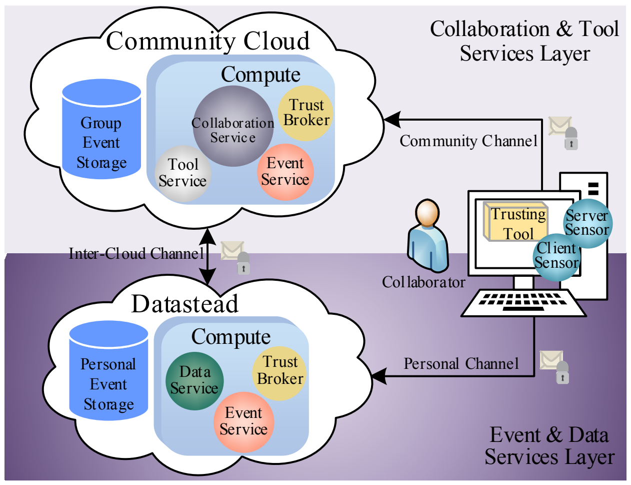

Figure 2: Generic Solution Concept

Secure Channels and Secure Sessions [20] are used to protect the transmission between the sensors and the datastead (the Personal Channel), between sensors and the Community Cloud (Community Channel), as well as between the datastead and any collaboration and tool services (Inter-cloud Channel). For a community cloud, a VPN is used to limit network access.

V. TECHNICAL IMPLEMENTATION

To determine the technical feasibility of the solution concept and provide a concrete case study, the solution concept was applied to an existing heterogeneous CDE called the Context-aware Software Engineering Environment Event-driven framework (CoSEEEK) [22], which had hitherto not incorporated privacy or security techniques. CoSEEEK's architecture and integrated technologies are shown in Figure 3. Its suitability is based on its portability (use of mainly Java and web-based languages), non-commercial access to the code and dependent technologies, its reliance on distributed communication mechanisms, and its heterogeneous tool support.

For event acquisition, CoSEEEK relies on the Hackystat framework [22] and its SE tool-based sensors (e.g., Ant, Eclipse, Visual Studio) for event extraction and event storage (shown in red in Figure 1). Hackystat does not currently provide extensive security and privacy mechanisms. For an insight, [24] briefly describes some of its security issues.

Service Layer Separation: the Hackystat-related elements (shown in red) were hereby separated into the Event and Data Services Layer and the remaining elements were placed in the Collaboration and Tools Services Layer.

Cloud configuration: To meet (*R:MCC*, *R:CCO*, *R:PAI*), two different cloud platforms were utilized in isolation. To represent a public IaaS cloud provider configuration (*R:PuC*), Amazon Web Services (AWS) was used, using Elastic Compute Cloud (EC2) for computing services, the Elastic Block Store (EBS) for storing configuration files and XML database, and the Relational Database Service (RDS) which holds the sensorbase.

To represent a private cloud (*R:PrC*) or community cloud (*R:CoC*) deployment, OpenStack was used with Compute used for computing and Object Storage used in place of EBS storage; and since nothing directly equivalent to AWS RDS was available, a Compute instance with Object Storage that contains a MySQL Server database was configured. Single tenancy (*R:ST*) with one Compute instance per developer with access restricted to the developer was configured.

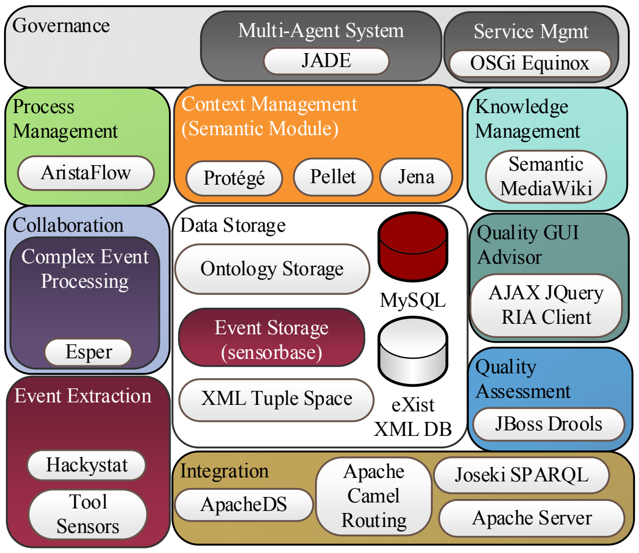

Figure 3. CoSEEEK Architecture (affected areas shown in red).

Trust Broker: the Trust Broker supports (*R:D*) was implemented in Java. The open source Restlet Framework for Java SE was used to provide the REST-based interface. An example of a query that can be sent is the following, specifying the project via the sensorbase_id, the timeframe, the sensor data type, the tool, and its uri source.

```
   GET
/trustbroker/sensordata/{sensorbase_id}?
startTime={startTime}&endTime={endTime}&
sdt_name={sdt_name}&tool={tool}
   &uriPatterns={uriPatterns}
```

Encryption of events (*R:ENC*) can be optionally configured. For encryption of arriving events and decryption of events on authenticated and authorized retrieval, Java's AES 128 and the SHA-256 hash algorithm were used (*R:BSM*). One reason for encrypting the storage is that it provides an additional form of protection, should, e.g., a provider's agent or intruder gain access.

The measurement database called sensorbase in Hackystat required a few minor adaptations. For (*R:D*), to support anonymization the HACKYUSER table was extended to include an anonymization flag, which is checked before responding, replacing a userid with anonymous. In order to support HTTPS connections, the sensorbase client (*R:SP*) was modified and rebuilt, requiring any sensors to utilize this modified jar file. HTTPS (*R:BSM*) was used to secure all three communication channels (personal, community, and inter-cloud) (*R:SC*). Additional properties were added to indicate the location of the keystore. SSH was used to configure and manage each cloud. Security groups were used in both AWS and OpenStack.

To implement remote attestation, on the client-side, a user configures the Trusting Tool with the expected checksum value (provided e.g., by the admin or a trusted website), version, and the interval for rechecking. On the service side, a REST interface sensorbase/checksum was added that loads the local adapted sensorbase.jar file, computes the SHA-256 hash value using `java.security.MessageDigest`, and returns this value and the sensorbase version to the Trusting Tool. While not foolproof, since any unauthorized access on the server or client could allow spoofing, it provides an additional level of confidence. Various stronger jar file tampering technologies could be employed if needed, such as componio JarCryp bytecode encryption.

VI. Evaluation

The case study evaluated the technical feasibility of the concept based on the technical implementation. However, security and privacy are highly contextually dependent on the expectations, requirements, environment, risks, policies, training, available attack mechanisms, implementation details (bugs), configuration settings, etc., making a comprehensive formal assessment in this area difficult. So the assumption is made that the prescribed privacy and security mechanisms suffice or are balanced for current developer needs in developer settings.

Since CoSEEEK is a reactive system, the ability to respond adequately to contextual changes via events is highly dependent on network latency; the evaluation focuses on this area for various cloud settings.

As to hardware, the Client PC (for use by a developer) has an i5-2410M (2.3-2.9 GHz) dual core CPU and 6GB RAM with 32-bit Windows XP SP3. The network consists of gigabit Ethernet and two 1 Gbit connections from the university campus in Germany to the Internet Provider.

Representative for a private (*R:PrC*) or community cloud where a datastead could also be placed, the OpenStack configuration (OSCfg) consisted of a local intranet server with an i5-650 (3.2-3.4GHz) dual core CPU, 8GB RAM, and 64-bit Ubuntu Server 12.04. The OpenStack Cloud Essex Release was installed on the Server via DevStack and the Compute instances also ran Ubuntu Server 12.04. MySQL v. 5.5.24 was used for Hackystat sensorbase storage in a Compute instance.

As a public cloud provider (*R:PuC*) representative, a free AWS configuration (AWSCfg) was chosen. It consisted of t1.micro EC2 instance types located in US-EAST-1d (Virginia) with 613 MiB memory, up to 2 EC2 units (for short periodic bursts) with low I/O performance running 64-bit Ubuntu Server 12.04. MySQL v. 5.5.27 was used for the Hackystat sensorbase storage in AWS RDS.

Common software included Hackystat 8.4 with the Noelios Restlet Engine 1.1.5 and JDK 1.6.

Typical network usage scenarios were considered, thus no optimizations were applied to any configurations nor was an artificially quiet network state created. All results are the average of 10 repeated measurements (with one exception noted below). A secure configuration denotes using the TrustBroker via HTTPS (*R:SC*) with encrypted storage (*R:ENC*), and an insecure configuration means no TrustBroker and using HTTP. VPN (*R:RNA*) overheads were not measured.

To determine delays from the client to the datastead in cloud variants, on the client PC Ant was invoked, causing the Hackystat Ant sensor to send one XML event to the Server (a write in the remote sensorbase) consisting of 235 bytes of event data and 73 bytes of overhead. As shown in Figure 4, the average network latency using an insecure OSCfg was 214 ms, for a secure OSCfg 389 ms, and for a secure AWSCfg 608 ms.

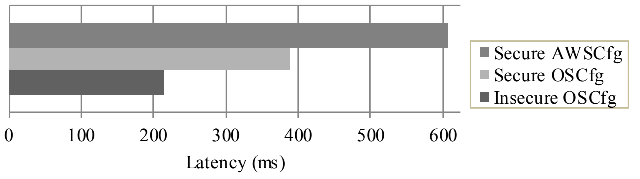

Figure 4: Latency (in ms) for sending an event (33 bytes) from the client PC to the server sensorbase.

Once events are in the datastead, then latencies between computing instances in a cloud are of interest, since the collaboration or tool services will be retrieving this data (shown in Figure 5 and Figure 6). For AWSCfg, a single query for 67 events (15818 bytes) between two EC2 instances took 78 ms on average via HTTP and 84 ms over HTTPS. In a secure configuration the retrieval took 347 ms. For OSCfg between two Compute instances, a single query took 38 ms to return 22 events (5243 bytes). Note that HTTP insecure reads in the private cloud had two anomaly values (178 and 210 ms) that would have changed the average from 38 to 69, and were also far larger than any secure value measurements. Thus, these 2 measurement values were removed, and the average created from the remaining 8 values. These large latencies could perhaps be attributed to a network, disk, operating system, or OpenStack related issue. Continuing with the measurements with 39 events (9238 bytes), HTTPS requests took 60 ms while in the secure configuration it averaged 61ms.The overhead of the privacy approach is the addition of SSL, brokering a second SSL connection, and encryption. For the OSCfg, the difference of TrustBroker and decryption showed on average only a 1ms difference to that with purely SSL. One explanation could be that the extra overhead is minimal compared to the data transfer delays between OpenStack instances, but further investigation of OpenStack internals and profiling would be required.

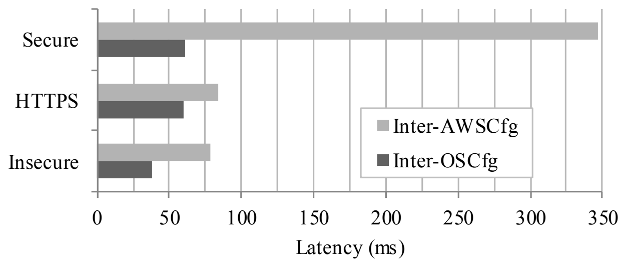

Figure 5: Private vs. public cloud inter-computing instance query latencies grouped by security (in ms)

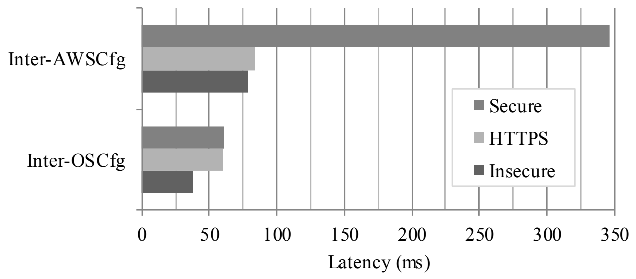

Figure 6: Inter-cloud query latency grouped by cloud type for different degrees of security (in ms).

Based on the results shown in the above figures, the use of the secure configuration of the OSCfg within a private or even a community cloud setting would appear to have acceptable performance overhead for cloud-centric collaborative development work, and distributed retrieval from datasteads is viable for responding to changes in the collaborative situation. On the other hand, the use of the secure configuration in the public cloud (AWSCfg), as shown in this perhaps worst case as a no cost offshore minimal public cloud setting, incurs substantially higher network latencies. Obviously choosing geographically close locations when possible is recommended. Also, provisioning sufficient computing and I/O resources support to deal with the additional inter-cloud and security mechanism overheads would also reduce such lags in public cloud configurations. Optimization in this area would also be promising.

To determine the remote attestation overhead, the Trusting Tool was measured on the PC using the AWSCfg over SSL. The average request-response latency was 702 ms. On the server, this involved loading and calculating the SHA-256 hash value for the 5.5 MB large sensorbase.jar file. Thus the attestation mechanism of the remote cloud instance could be configured to be automatically invoked periodically by client-side sensors at regular intervals in a separate thread or process to not interfere with other network communication.

In summary, the evaluation showed that network latencies incurred by the concept are most likely insignificant for collaboration in PrC settings, but that security overheads in global PuC settings may require optimization attention.

VII. Conclusion and Future Work

To address security and privacy in collaborative cloud development, this paper presented a practical concept with entity-level control of non-, anonymized-, and personally-identifiable disclosure for multiple cloud configurations. It can further both collaboration and trust by giving individuals transparency and control and allowing them to adjust disclosure to the changing trust situation. The paper contributes a practical basis for illustrating issues, eliciting awareness, community discussion, and and may increase self-regulation and infrastructural privacy offerings. Organizations adopting such a privacy infrastructure show that they value and trust their employees, enabling them to reap mutual trust rewards. Also, one could envision, for instance, that an audited "we don't spy here" seal might help attract and retain developers.

The evaluation showed its technical feasibility and practicality, requiring only minimal adaptation of the CoSEEEK CDE. The Trust Broker enables fine granularity access control to personal data. Performance was sufficient in private cloud configurations, while public cloud configurations using additional security and privacy mechanisms may require optimization to ensure fluid collaboration situational response.

Limitations and risks include: extending privacy/trust support within and across collaboration layer tools, non-detection/discovery of (un)intentionally unspecified/hidden sensors, data manipulation risk by datastead owners themselves, and provider-side access or manipulation risk. For service provider trust issues, building your own datastead cloud server site could be considered.

Future work can consider the inclusion of various data provenance and data integrity mechanisms to mitigate manipulation risk. In the face of shifting privacy norms, infrastructural support for data confidentiality is needed to limit disclosure of distribution data beyond its original intent, like lifetime constraints, transitivity bounds, and claims-based access [25]. Enhanced remote attestation mechanisms could be investigated. Since service privacy is also a broader issue, development and adoption of global industry service privacy standards combined with independent privacy audits involving all service layers would enhance trust of cloud-based data acquisition and usage offerings.

Acknowledgment

The author acknowledges Jürgen Drotleff for his conceptual, implementation, and measurement contributions.

References

[1] S. I. Hashmi et al. (2011, August). Using the Cloud to Facilitate Global Software Development Challenges. In Global Software Engineering Workshop (ICGSEW), 2011 Sixth IEEE International Conference on (pp. 70-77). IEEE.

[2] R. L. Grossman, "The case for cloud computing," IT professional, 11(2), pp. 23-27.

[3] M. A. Chauhan and M. A. Babar, "Cloud infrastructure for providing tools as a service: quality attributes and potential solutions," In Proceedings of the WICSA/ECSA 2012 Companion Volume, ACM, 2012, pp. 5-13.

[4] M. Singhal et al., "Collaboration in Multicloud Computing Environments: Framework and Security Issues, " Computer, 46(2), IEEE Computer Society, New York, 2013, pp. 76-84.

[5] G. D. Nord, T. F. McCubbins, and J. H. Nord, "E-monitoring in the workplace: privacy, legislation, and surveillance software," Communications of the ACM, 49(8), 2006, pp. 72-77.

[6] M. Parsons, "The challenge of multicore: a brief history of a brick wall," EPCC News, Issue 65, University of Edinburgh, 2009, p. 4.

[7] T. Brahm and F. Kunze, "The role of trust climate in virtual teams," Journal of Managerial Psychology, 27(6), 2012, pp. 595-614.

[8] A. C. Costa, R. A. Roe, and T. Taillieu, "Trust within teams: The relation with performance effectiveness," European journal of work and organizational psychology, 10(3), 2001, pp. 225-244.

[9] T. DeMarco and T. R. Lister, Peopleware. Dorset House, 1987.

[10] D. McGregor, The Human Side of Enterprise. McGrawHill, New York, 1960.

[11] B. Al-Ani and D. Redmiles, "Trust in distributed teams: Support through continuous coordination," IEEE Software, IEEE Computer Society, 26(6), 2009, pp. 35-40.

[12] P. Louridas, "Up in the air: Moving your applications to the cloud," IEEE Software, 27(4), IEEE Computer Society, New York, 2010, pp. 6-11.

[13] H. Takabi, J. B. Joshi, and G. J. Ahn, "Security and privacy challenges in cloud computing environments," IEEE Security & Privacy, IEEE Computer Society, 8(6), 2010, 24-31.

[14] R. Martignoni, "Global sourcing of software development-a review of tools and services," In Fourth IEEE International Conference on Global Software Engineering (ICGSE 2009), IEEE Computer Society, 2009, pp. 303-308.

[15] F. Liu et al., NIST cloud computing reference architecture. NIST Special Publication, 500, 292, 2011.

[16] M. Hogan, F. Liu, A. Sokol, and J. Tong, Nist cloud computing standards roadmap. NIST Special Publication, 35, 2011.

[17] W. Itani, A. Kayssi, and A. Chehab, "Privacy as a service: Privacy-aware data storage and processing in cloud computing architectures," In Eighth IEEE International Conf. on Dependable, Autonomic and Secure Computing (DASC'09), IEEE Computer Society, 2009, pp. 711-716.

[18] D. Song, E. Shi, I. Fischer, and U. Shankar, "Cloud data protection for the masses, " Computer, 45(1), 2012, pp. 39-45.

[19] M. Nabeel and E. Bertino, "Privacy-Preserving Fine-Grained Access Control in Public Clouds," Data Engineering, 21, 2012.

[20] D. M. Kienzle, M. C. Elder, D. Tyree, and J. Edwards-Hewitt, Security patterns repository version 1.0. DARPA, 2002.

[21] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, Security Patterns: Integrating security and systems engineering. Wiley, 2006.

[22] G. Grambow, R. Oberhauser, and M. Reichert, "Enabling Automatic Process-aware Collaboration Support in Software Engineering Projects," In Software and Data Technologies, Springer, Berlin Heidelberg, 2013, pp. 73-88.

[23] P. M. Johnson, "Requirement and Design Trade-offs in Hackystat: An In-Process Software Engineering Measurement and Analysis System," Proc. 1st Intl. Symposium on Empirical Software Engineering and Measurement, 2007, pp. 81-90.

[24] P. M. Johnson, C. A. Moore, J. Miglani, and S. Zhen, Hackystat design notes. 2001.

[25] D. Reed, D. Gannon, and J. Larus, "Imagining the future: Thoughts on computing," Computer, 45(1), 2012, pp. 25-30.