POET: Privacy on the Edge with Bidirectional Data Transformations

Nianyu Li
Key Laboratory of High Confidence Software Technology
Peking University, China

Christos Tsigkanos
Distributed Systems Group
TU Wien
Austria

Zhi Jin
Key Laboratory of High Confidence Software Technology
Peking University, China

Schahram Dustdar
Distributed Systems Group
TU Wien
Austria

Zhenjiang Hu
National Institute of Informatics
University of Tokyo
Japan

Carlo Ghezzi
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano, Italy

***Abstract*—Comprehensive privacy mechanisms are essential in the pervasive internet-of-things systems of today, which are comprised of multiple distributed devices and diverse software stacks, while located in different legal or administrative domains. In such systems, often consisting of resource-constrained devices, guarantees of correctness and conformance to privacy policies is required, while data need to be synchronized among different software components. Motivated by the "data protection by design and by default" principle, we propose a technical framework to support data synchronization among edge components tailored for pervasive IoT applications. Our privacy-driven synchronization approach is based on a generically applicable privacy model and able to capture roles and permissions, actions on data, conditions and obligations that arise in privacy requirements. For automated and correct reflection of synchronized data among components, we adopt bidirectional transformations, a mechanism where synchronization between models, consistency, and well-behavedness are formally guaranteed. Thus, automatically generated privacy-aware data transformations are correct by construction. We evaluate POET, our framework and accompanying tool with a case study on medical information privacy and demonstrate its performance in resource-constrained edge devices.**

***Index Terms*—Bidirectional Transformations, Privacy Models, Edge Computing, Requirements Assurance**

I. Introduction

The recent evolution towards an increasingly integrated world has at its basis novel types of pervasive systems achieved through new technologies and paradigms such as mobile and cloud computing and Internet of Things (IoT), inducing systems composed of heterogeneous devices, computing infrastructures and cloud services. The current state-of-the-art in systematically engineering such pervasive systems involves architecturally offloading data or control to the cloud. However, novel functional and non-functional requirements, including user requirements as well as new laws and regulations dictate data or computation to be situated locally near devices. Such requirements may capture diverse system concerns ranging from performance to security and privacy, whose satisfaction suggests computing entities architecturally located near the network *edge*, closer to the end-devices. Such edge entities may offer computational, communication and data resources to local devices [1].

Take privacy [2] as an example. Comprehensive privacy mechanisms are essential for widespread uptake and acceptance of the pervasive systems of today, as the ever-increasing number of devices collecting (possibly sensitive) data and interacting with the physical environment, combined with opaque data handling policies, contribute to a lack of trust. Hence, privacy emerges as a first-class design goal throughout the application development lifecycle. The challenge of data privacy in particular, is to enable utilization of data while protecting an individual's privacy preferences and their personally identifiable information.

Rigorous modeling and precise formalization of privacy requirements as defined by legislative acts, like the EU General Data Protection Regulation (GDPR [3]), enables the enforcement of privacy regulations in an information processing system. The unambiguous definition of policies allows reasoning upon them, ensuring compliance to information privacy laws, generally guaranteeing individuals' privacy. Recent developments in legal and regulatory frameworks have intensified the need for privacy compliance. Legislative acts such as GDPR [3] (about privacy, data handling and protection), HIPAA [4] (about medical records privacy) or CCPA [5] (consumer data protection), require enterprises to protect privacy of their users, by rigorously specifying privacy policies and ensuring their implementation in their activities.

In this paper, we propose a novel technical framework to engineer data privacy tailored for the pervasive edge computing systems of today – POET (Privacy On the Edge with Transformations). In such systems, often consisting of resource-constrained devices hosting software components i) formal assurances of correctness as well as ii) development support throughout the application lifecycle is required. We focus on applications' privacy needs, as they pertain to keeping data synchronized among software components; data should always flow between components in accordance to privacy policies capturing privacy requirements of the system. In our proposal, we ensure that in software components, changes applied to remote data are reflected back automatically while respecting defined privacy policies in the source data, and from there propagated to other remote data that may also be affected by the change. Such automated reflection is achieved thanks to the use of bidirectional transformations [6], a mechanism where synchronization between models, consistency, and well-behavedness are formally guaranteed. Model synchronization entails propagating changes back, while ensuring that changes made to models are always consistent. Well-behavedness pertains round-trip laws capturing the properties mentioned above. Essentially, given the support that POET concretely provides, transformations reflecting changes to remote data are generated automatically and correctly.

Our privacy reasoning approach is based on the generically adopted privacy model of P-RBAC [7], thus able to capture roles and permissions, actions on data, conditions and obligations that arise in privacy requirements. Privacy-wise, the cornerstone of our approach is that no data residing in a node may leave it unless it satisfies the defined policies. In our POET approach, automatically generated privacy-aware data transformations are correct by construction. We essentially implement "data protection by design and by default" (Art. 25 GDPR [3]), by providing facilities for assurances of correct design – our technical framework ensures that data synchronized always respect privacy policies defined.

The rest of the paper is structured as follows. Section II gives an overview of our approach, which is situated on the edge and providing data privacy and synchronization over a running example used throughout the paper. Section III describes the privacy model we have adopted to capture privacy requirements, and Section IV illustrates our privacy-aware data synchronization mechanism using bidirectional transformations. Section V presents the runtime deployment of our approach, while Section VI provides an assessment of the applicability and realizability of the proposed approach over a case study of medical data privacy. Related work is considered in Section VII, and Section VIII concludes the paper.

II. Privacy on the Edge

New challenges and opportunities arise as the rapidly growing cloud computing and pervasive mobile devices, sensors and networks meet [8]. A pervasive system is generally made up of different data-handling components, which may be operated by different users, located in diverse administrative domains and deployed in the cloud, on local devices or intermediate computational nodes.

From the *data* perspective, novel system requirements including timeliness, privacy or availability suggest that storage and computation upon data should be performed at or close to where sensory or raw data is generated, or where the end users of the data are situated. Keeping data locally and close to devices means that they can access it rapidly minimizing network latency, access it even in case of network disruption and very importantly, data is within the control of the device. The latter has implications about privacy, as data ownership is defined by the source of the data, and shared according to privacy policies capturing system privacy requirements.

Data obviously need to be synchronized among different data-handling components. In the modern pervasive environment data does not only flow from devices to the cloud – data flows are bidirectional and among different data consumers and producers [9]. As such, devices should handle privacy themselves. Another factor in favor of this is trust; although privacy requirements are known system-wide, a misconfiguration or data breach may allow possibly untrusted devices to access data that they should not. We advocate that since the edge is closer to data sources and users, there is an opportunity for stronger data privacy – something realized by empowering the edge to actively manage the privacy of data located within its scope, and automatically synchronize the right subset with the rest of the system.

Privacy requirements of the system – which can be considered to be global, as they permeate the whole system – dictate how and where data should be synchronized. For example, a privacy policy encoding some requirement conditionally restricts actions according to different data objects, data users, or data use purposes. The target is to synchronize data with other allowed software components, while synchronization usually means some create, read, update or delete [10] action.

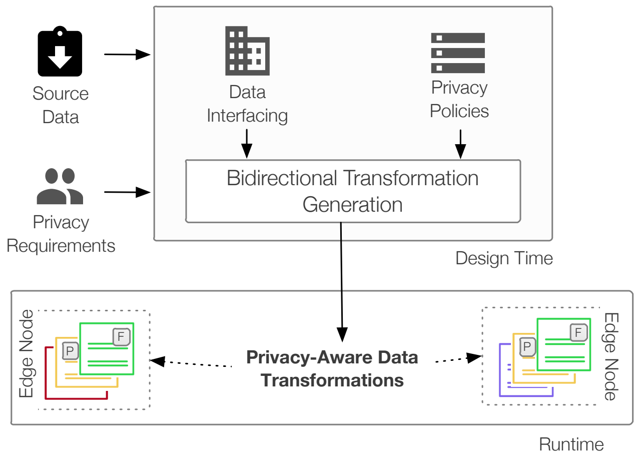

Fig. 1. Privacy on the Edge with Bidirectional Transformations.

Figure 1 provides a birds-eye view of our approach to privacy on the edge. At design time, two tasks are performed: i) privacy requirements are encoded in privacy policies, using the established privacy formalism of P-RBAC [7], and ii) source data are interfaced in order to be compatible with our privacy framework. Output artifacts of those two design time tasks constitute the input to a transformation engine [11].

At runtime, bidirectional data transformations are then automatically parameterized with the specified policies correctly by construction, and placed in edge components handling data within the system. During system operation, the generated privacy-aware transformations keep data on edge nodes synchronized with others (as well as any other devices configured, or the cloud). The essence of our approach is that this synchronization is formally defined, correct and always respects privacy policies that govern data located in the edge node. POET, a prototypical tool realizing our privacy-aware bidirectional data transformation approach, along with examples of use and privacy specifications is available [12].

Motivating Example. We consider the case of a distributed social network, where mobile users through their devices share privacy-sensitive data. Each user has a laptop computer as well as mobile devices. Users' laptop computers contain personal information, such as their photo albums, which are meant to be shared only among their friends. A list of friends is maintained at each laptop. Since users may roam, their IP address may change – a central cloud service actively maintains the list of friends as well as their IP addresses. We assume that users voluntarily share their IP address with the social network [13]. A mobile device may take a photo which is then uploaded on a user's laptop for storage – the photo may be explicitly allowed by the user as to be shared with friends through the social network the laptops are part of. For example purposes, we treat users' laptops as edge nodes – users' mobile phones (as IoT devices) may connect to them locally. Privacy requirements of the overall system dictate the following:

PR1: A mobile device can synchronize photos for the purposes of storage with a connecting laptop.

PR2: A friend's laptop can access photos that have been marked as to be shared by its owner.

PR3: A cloud service may update the IP addresses of friends on a user's laptop.

No other information should flow between the various constituents of the system – in other words, PR1-PR3 represent the only data operations that are allowed to occur in the system.

III. Modeling Data Privacy Requirements

As is common in information privacy, for all data collected and processed, there should be a stated purpose [14]; usage of data for another purpose than the one it was intended for must be prevented. This is reflected also in the GDPR Art. 6 – Lawfulness of processing [3]. Moreover, as is typical in data management, support of classifications identifying various kinds of data is desired. Support of roles allows access differentiation between different entities, in line with security access models. This is because in the context of an edge-intensive system, differentiation between architectural entities is desired – privacy-wise, a mobile device is a different architectural component than a cloud machine. Based on the above privacy objectives, in the following section we adopt a privacy model that can support them.

P-RBAC [7] is an extension to the well-known Role-Based Access Control (RBAC) model [15]. In the classic RBAC model, a user is assigned to one or multiple roles and each role has one or multiple permissions. A permission specifies what action can be performed on which data object. P-RBAC extends this notion of a permission by adding privacy-related attributes to it, such as purpose, condition and obligations. The purpose binds a permission to a range of duties; for example, sharing purposes may entail different permissions than storage. A condition specifies under which circumstances a permission can be granted, for instance, accessing a user's photo may require explicit consent from the user. Obligations denote a set of operations which need to be performed whenever a permission has been granted, such as adding a log entry.

Our instantiation of P-RBAC for the edge has the following constituents.

- **Privacy Role** ($\mathtt{R}$): A type identifier of the architectural component where data will be contained in, e.g. `Laptop` or `MobileDev`.
- **Data Object** ($\mathtt{D}$): The data object which a privacy requirement refers to, e.g. `Photo`.
- **Access Purpose** ($\mathtt{P_U}$): The purpose for which access to an object is to be allowed, e.g. `Storage`.
- **Access Action** ($\mathtt{A}$): The action with which data will be operated upon, i.e. CRUD actions.
- **Access Obligation** ($\mathtt{O}$): If permission is granted, operations that must be additionally performed.
- **Access Condition** ($\mathtt{C}$): A constraint encoding when a permission can be granted.

Assuming some architectural entities $\mathtt{U}$, the entity assignment $\mathtt{U_A} \subseteq \mathtt{U} \times \mathtt{R}$ links entities to one or multiple roles. Each role $\mathtt{r} \in \mathtt{R}$ has one or multiple permissions, denoted as the permission assignment $\mathtt{P_A} \subseteq \mathtt{R} \times \mathtt{P}$. A permission $\mathtt{P}$ extends the action-object privacy permission $(\mathtt{a}, \mathtt{d}) \in \mathtt{A} \times \mathtt{D}$ of RBAC [15] with an additional condition $\mathtt{c} \in \mathtt{C}$, purpose $\mathtt{p} \in \mathtt{P_U}$ and an optional set of obligations $\mathtt{ob} \subseteq 2^{\mathtt{O}}$. The set of all privacy permissions is denoted as $\mathtt{P} \subseteq (\mathtt{A} \times \mathtt{D}) \times \mathtt{P_U} \times \mathtt{C} \times \times 2^{\mathtt{O}}$. A privacy permission is then written as $(\mathtt{r}, ((\mathtt{a}, \mathtt{d}), \mathtt{p}, \mathtt{c}, \mathtt{ob}))$.

Recall our example system, where photos may be shared among friends for purposes of $\mathtt{P_U} = \{\mathtt{Sharing},\ \mathtt{Storage},\ \mathtt{Management}\}$. Roles $\mathtt{R} = \{\mathtt{MobileDev},\ \mathtt{Laptop},\ \mathtt{CloudSrv}\}$ operate upon data objects $\mathtt{D} = \{\mathtt{Photo},\ \mathtt{FriendIPList}\}$ with actions $\mathtt{A} = \{\mathtt{Write},\ \mathtt{Read},\ \mathtt{Update}\}$. Access condition $\mathtt{C} = \{\mathtt{photo\_shared}\}$ needs to be satisfied for a photo to enable sharing, and an access obligation $\mathtt{O} = \{\mathtt{notify()}\}$[1] must be performed whenever friend's IP addresses are updated. Privacy requirements PR1-PR3 of the motivating example can then be formally encoded as the following tuples:

(PR1) $(\mathtt{MobileDev}, (\mathtt{Write}, \mathtt{Photo}), \mathtt{Storage}, \mathtt{true}, \emptyset)$

(PR2) $(\mathtt{Laptop}, (\mathtt{Read}, \mathtt{Photo}), \mathtt{Sharing}, \mathtt{photo\_shared} \equiv \top, \emptyset)$

(PR3) $(\mathtt{CloudSrv}, (\mathtt{Write}, \mathtt{FriendIPList}), \mathtt{Management}, \mathtt{true}, \mathtt{notify()})$

Note that the other side of the data flow interaction such as the connecting laptop in the privacy requirements will be embeded in the architectural deployment presented in Section V.

IV. Privacy-Aware Data Transformation

We have seen how data privacy requirements can be precisely expressed between system nodes according to the policies in an P-RBAC model. However, a classic problem in this methodology is how to reflect modifications of one system component to the others connected to it. If an update is made to data residing on an edge node, it must be reflected to other connected edge nodes based on the semantics of the privacy policies defined. The change must not take place if the policy does not allow it.

[1]Obligation procedures operate upon data, are assumed to be application-dependent and already defined along with privacy policies.

Certainly, one could implement this by hand, by encoding the transformation algorithms for certain particular data. However, formal assurances on correctness and well-behavedness might not be guaranteed:

- Both directions, propagating data to and from the edge node must be implemented, so that data is synchronized. This is prone to errors, as the bidirectional transformation must correctly reflect changed data back and forth.
- Synchronization should occur for every data pair according to the privacy policy, while any data format should be supported.
- Should the need for changing the implemented transformation arises, development support should aid (correctly) updating the transformations.

In the following, we illustrate how the above challenges may be tackled by designing and implementing a well-behaved bidirectional transformation (BX) for data synchronization, which correctly by design propagates changes between data residing in edge nodes in accordance with policies defined in an P-RBAC specification.

A. Bidirectional Transformation

Bidirectional transformation (BX) [6] is a useful mechanism for data synchronization, which supports any format data pair from many different areas including software engineering, programming languages, databases, and document engineering. Asymmetric lenses, an influential framework in BX, are designed for synchronizing two pieces of data where one side, which is called the source, has more information than the other, which is called the view. A lens consists of a pair of transformations get and put [16], [17]. The *forward* transformation $get(s)$ is used to produce a target view v from a source s, while the *putback* transformation $put(s, v)$ is used to reflect updates on the view v to the source s. These two transformations should be *well-behaved* in the sense that they satisfy the following round-tripping laws:

$$\begin{array}{rcll} put(s, get(s)) & = & s & \textsc{GetPut} \\ get(put(s, v)) & = & v & \textsc{PutGet} \end{array}$$

The GETPUT property requires that no change of the view shall be reflected as no change of the source, while the PUTGET property requires all changes in the view should be completely reflected to the source so that the changed view can be computed again by applying the forward transformation to the updated source.

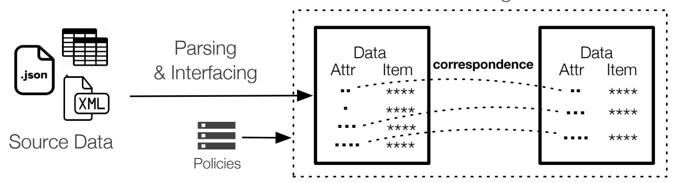

Fig. 2. Privacy-Aware BX overview: After interfacing of source data, the BX engine utilizes correspondences between items to synchronize structures.

B. Data Interfacing & Preparation

Since data in an edge-intensive application can be of various types, our aim is providing a privacy-aware mechanism that is generic – it is applicable to every underlying data that is sought to be bidirectionally transformed. Thus, the initial step required by our approach naturally consists of data preparation.

Data preparation, shown in the left part of Fig. 2, aims at interfacing source data with the privacy-aware data transformation engine. Since data may be of arbitrary source format, what is required by the developer is demonstrating a minimal set of data attributes or fields, that uniquely specify a data item. In the simplest case, this can be a single attribute. This is essential for the BX engine to establish correspondences between data items when performing privacy-aware transformations – essentially, establishing that a certain data field or data attribute is unique per data record. In practice, this entails showing how source data items can be represented in $(attribute, item)$ pairs within the BX engine, where $attribute$ is unique to the application. For example, this can be a primary key for a record in a table or a hash value for arbitrary data objects.

Recall our running example, where source data consist of the users' photos and the list of friend's IP addresses. A unique attribute for a photo –which is typically a binary data item– can be some unique number (e.g. a UUID) or a bit string fingerprint from a hash function (e.g. SHA). A list of friend's IP addresses would typically contain pairs of a user identifier (i.e. a nickname or a public key) and an IP address, so the identifier can be readily used.

C. Implementing Privacy-Aware BX using BiGUL

To realize the privacy-aware BX engine, a pair of transformations between the source and view models need to be developed. In our case, recall that the forward transformation get is the data item extraction from the source data which satisfy the privacy policy. Since *get* is essentially a projection of the data based on the policy, its corresponding *put* is an information embedding of the view data items into the source data items which have the same data attributes. Though the source update ways varies depending on actions denoted in a policy, the *put* should always be paired with a *get* to form a bidirectional transformation pair satisfying the GETPUT and PUTGET properties defined previously.

To use BX for our transformations, we need to develop a pair of transformations between the source and the view models as shown in the right part of Fig. 2. From a high level perspective, a transformation function producing a view V from a source S satisfying a policy $(\mathtt{r}, ((\mathtt{a}, \mathtt{d}), \mathtt{p}, \mathtt{c}, \mathtt{ob}))$ operates in the following manner:

```
For every component role r:
  For every purpose p:
    For every data object $o$ in $S$ which matches d:
      → Yield $o$ if c is satisfied for $o$.
```

Its corresponding put transformation should be an embedding of data view into the source model according to the semantics of this policy. Generally, privacy goals are achieved due to the following two points; first, the view always contains the information allowed “to go out” of a node. Second, the BX framework guarantees that one cannot get additional (i.e. private) information from the source by modifying the view.

The transformations *get* and *put* could be manually implemented. Although this solution provides the programmer with full control in two directions and can be realized using standard programming languages, we i) require formal assurances of correctness of the *get* and *put* transformations and ii) desire to minimize the maintenance effort required to keep the consistency between them when one is changed. Even a small modification to one of the transformations would require redefinition of the other as well as a new well-behavedness proof.

In this paper, we adopt BiGUL [18], a putback-based bidirectional programming language, where one is only required to implement the *put* transformation instead of both *get* and *put*, to implement the data synchronization above. This is based on the fact that *get* is uniquely determined by *put* based on well-behavedness [19]. In BiGUL, once a *put* transformation is given, the corresponding *get* transformation can be automatically derived for free. We will not dive into a detailed explanation of the implementation of *put* in BiGUL, but rather we give a flavor of it through the following fragment, which describes how updating a data record with the privacy-aware BX engine occurs.

```
datasyn :: BiGUL Record Record
datasyn = $(update
          [p| Rcd attr item |]
          [p| Rcd attr item |]
          [d| attr = Skip; item = Replace |])
```

In the above functional program fragment, `datasyn` uses a view of type *Record* to update a source of the same type, where a simplified data record here is internally represented by ($Rcd\ attr\ item$). The definition body of `datasyn` states that the source and view should be of the same form of ($Rcd\ attr\ item$), and that attribute in the source should be unchanged (via *Skip*), the item in the source should be replaced by that in the view (via *Replace*). For example, given the source (Rcd ”$photoName$” ”$family$”) and the view (Rcd ”$photoname$” ”$parents$”), the updated source via *get* shall be (Rcd ”$photoName$” ”$parents$”) with an unchanged attribute and an updated item value. Interestingly, from this *put*, BiGUL can automatically derive the *get*, which is exactly the semantics of generating view satisfying round-trip properties. The interested reader is referred to [20] for more details on the underlying mechanism of BiGUL.

The above mechanisms are encapsulated in two components of POET: a *Privacy Governor*, responsible for implementing P-RBAC, and a BX engine parameterized with the specified policies based on BiGUL responsible for generating the appropriate view and updating the source. Our prototype implementation is available in accompanying material [12].

D. Propagating Partial Changes

Naturally, for a pair of edge nodes it is not necessary that one node contains all the data of the other one. For our running example for instance, data in a user’s laptop consists of photos and IP addresses, while her mobile device may include additional personal information. Namely, data sources are not self-contained between each other, thus the BX asymmetric lenses framework is not suitable in this case. Intuitively, the data fragment corresponding to the photos can be treated as intermediate data for PR1, as the privacy policy explicitly specifies which data object can be operated upon. Such intermediate data with less information than the view component, constitute two pairs of asymmetric lenses with the data in the edge node.

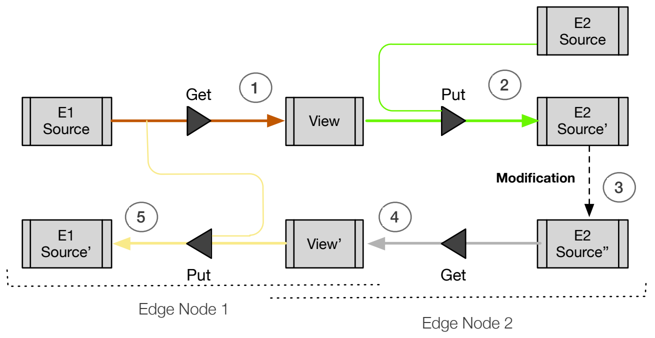

Fig. 3. Bidirectionally propagating partial changes between two edge nodes.

Concretely, the process of partial change propagation between two nodes consists of the following steps which are also illustrated in Fig. 3. We demonstrate the change propagation between two nodes, Edge Node 1 (E1) and Edge Node 2 (E2) with a privacy policy $(\mathtt{r}, ((\mathtt{a}, \mathtt{d}), \mathtt{p}, \mathtt{c}, \mathtt{ob}))$. To simplify presentation, we assume that E2 has the role of `r` and needs to comply with the policy. Nodes E1 and E2 are connected in deployment and are able to synchronize data. For `r`, if E1 contains the data object `d` of the policy:

1) An intermediate data view is produced via the *get* transformation from the source residing in E1 (Step 1 in Fig. 3).
2) The source in E2 is updated via the *put* transformation based on the generated view in Step 1 and its previous source (Step 2).
3) A change is made to the new source E2 (Step 3); it should be reflected back to E1. Obligation `o` is executed.
4) The projection of a new view is triggered (Step 4) by the change in Step 3.
5) The *put* transformation is executed to generate a new source on E1 (Step 5) – the new source is consistent with the new view while retaining other information of the original source.

Note that these *get* and *put* transformations are complying with action `a` and upon changes the privacy policies are always respected – the *put* transformation will be not allowed to reflect back changes if the action `a` has insufficient permissions, such as a *read* action having no permission to make changes in data d on E2.

V. Runtime Deployment on the Edge

IoT systems are characterized by device and software stack heterogeneity and deployment in possibly untrusted and different administrative domains [21]. Our approach exploits exactly the privacy preferences over data resources, facilitated by the deployment of *Privacy Governor* components encapsulating privacy policies and BX engine transformation mechanisms.

In edge computing architectures, IoT devices handling (possibly sensitive) data and interacting with the physical environment, the edge device is by definition located within the administrative domain of its local IoT devices – one can take that as the devices being in the same privacy scope. Our approach treats the edge as a first-class entity. Data flows between the edge and other external components (i.e. other edge nodes, the cloud etc) must always respect privacy policies. Architecturally, components (which may be devices or the cloud) are deployed in different environments, each containing local data. Edge and cloud components are connected through the network, but they do not necessarily trust each other. Each node sets its own data out- or in-flow privacy policies that govern data synchronizations. Data transformations mechanisms as well as privacy governors control how data leaves or enters the node. In case of a change in local data within some node, all others affected receive the updated data (through the generated views, Section V). Thus, nodes do not have to trust each other to exchange data, as this occurs based on privacy policies that are in their own control.

The diagram of Figure 4, illustrates a runtime deployment of our approach with respect to the motivating example of Section II, as combined architectural deployment and data flow diagrams. Edge nodes correspond to users' laptops – an additional device of User B is shown in the lower part, connected to her laptop. Local data on each architectural node, are managed by POET, consisting of a Privacy Governor, responsible for checking privacy policies and the Data BX, responsible for synchronizing data that are compliant to the defined access policies. Upon a change, POET generates views and synchronizes accordingly with other nodes. For our motivating example, photo data labeled "P" are synchronized between nodes of a user (User B in the lower part), and her friends (in this case, User A). FriendIPLists (labeled "F") are always synchronized between edge nodes and the cloud node.

To use our approach and its accompanying prototype tool in practice, a privacy engineer follows three distinct steps:

1) Privacy requirements of the system are encoded as privacy policies in P-RBAC (Section III).
2) Data correspondences are demonstrated, depending on the data formats used within the system (Section IV-B).
3) Automatically generated privacy-aware transformation components parametrized with the system's privacy policies are deployed in nodes (Section V).

Upon execution, POET produces data views which satisfy the privacy policies specified, and those are synchronized among the appropriate architectural components depending on the policies. If data are modified in one component, the changes are propagated automatically to other components affected by the change again based on privacy policies.

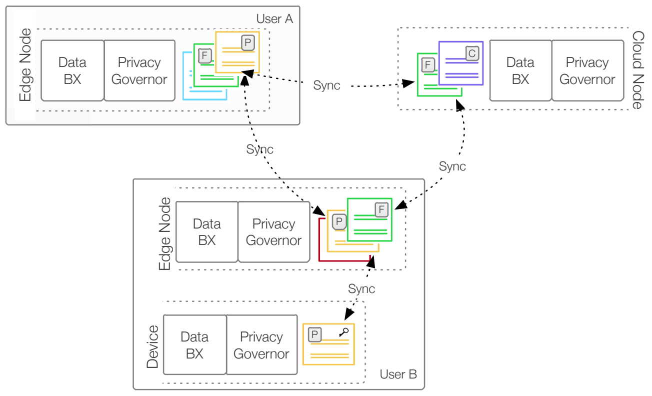

Fig. 4. Combined deployment and data flow diagrams at runtime.

VI. Evaluation

To provide concrete support for our data transformation for privacy on the edge framework, we realized POET, a prototypical tool based on BiGUL [11]. POET is freely available [12], is implemented in Haskell on top of BiGUL and contains a realization of a P-RBAC privacy governor. Thereupon, we evaluate our approach over a medical record privacy case study; experimental setup and results obtained are subsequently presented; we conclude with a discussion. We are interested in evaluation of both applicability and realizability aspects of the proposed approach.

Applicability entails the appropriateness of our privacy-aware bidirectional data transformation approach to capture typical privacy requirements, execute transformations on heterogeneous software stacks with no modifications while allowing vendor customization. As various data formats are used in different domains, a vendor situated in a domain should be easily able to customize data representations. This is also important to avoid technological lock-in. Realizability entails performance of the data transformations, especially regarding the execution of transformations on resource-constrained devices with different capabilities. We note that for our evaluation, we ignore networking and communication effects and we note that our available proof-of-concept implementation is on a prototypical stage [12].

A. *Case Study: Medical Information Privacy*

We replicate a representative data deployment scenario within the healthcare domain, where medical data reside in different logical and physical locations – medical record privacy is a significant issue often driving privacy developments [4]. In our case study, we consider the following involved entities.

- Doctor's Office (DO): a practitioner's office, as the key specialized treatment provider of a patient, makes use of available data for a patient –such as medical tests from a lab– for diagnostic reasons, as well as keeping patients' personal information. A doctor may issue diagnoses, and all this information may be synchronized with another doctor or hospital in case of a referral or joint treatment. In our case study we assume a DO to be associated with 0.3k patient records, including their medical tests and diagnoses.

- Medical Lab (ML): a clinical laboratory carries out diagnostic tests on visiting patients – in our case study we assume 2k records of medical tests to be in an ML node.
- Hospital (HL): database facilities in a hospital contain all data for 1k patients including their personal information from doctor's offices, 5k past admissions as well as 5k diagnoses. Such data are often used for patient management and hospital organization. Moreover, hospitals in a region may synchronize data with each other due to patient mobility and specialized care.
- University (UV): research in a university setting often makes use of certain medical data sourced from hospital databases (such as diagnoses of a disease) for scientific reasons. We consider 5k such diagnoses to be associated with a UV.
- Patient's residence (PR): monitoring of vital medical information may be performed on a patient's home, making use of recent IoT developments such as Body Area Networks [22]. Medical sensor data produced by small sensors are often used for live monitoring and live diagnosis reasons by medical practitioners. We assume 100 records of medical tests from sensors to be associated with a PR node.

We treat the above as different architectural entities, situated both at various edge nodes and the cloud. Specifically, the hospital's, and university's data stores are assumed to be on the cloud, with no computational or resource constraints. Medical labs, doctor's offices and patient residences are instead assumed to be edge nodes, in the scope of which IoT devices use and operate upon local data (e.g. a doctor's personal computer accesses freely data on the local edge node). Note that data used in each architectural entity often is of a different format – hospital database management systems often adopt XML-variant EHR records [23], tabular data or other proprietary formats, while edge nodes may utilize open formats such as JSON or XML. However, privacy requirements in such a medical setting govern how data flows between different architectural entities [4]. Privacy policies considered are the following:

(P1) $(\mathtt{DO}, (\mathtt{Update}, \mathtt{DiagnosisRecords}), \mathtt{MedicalCare}, \mathtt{true}, \mathtt{log}())$

(P2) $(\mathtt{ML}, (\mathtt{Write}, \mathtt{MedicalTest}), \mathtt{MedicalCare}, \mathtt{AdmEndDate} \neq \mathtt{null}, \emptyset)$

(P3) $(\mathtt{HL}, (\mathtt{Read}, \mathtt{PatientInfo}), \mathtt{Statistics}, \mathtt{true}, \emptyset)$

(P4) $(\mathtt{UV}, (\mathtt{Read}, \mathtt{DiagnosisRecords}), \mathtt{Research}, \mathtt{DiagnosisCore} \equiv \mathtt{Q21}, \emptyset)$

(P5) $(\mathtt{PR}, (\mathtt{Update}, \mathtt{MedicalTest}), \mathtt{Storage}, \mathtt{true}, \mathtt{notify}())$

P1 specifies that patient diagnoses maintained in a doctor's office can be updated for the purpose of medical care with the obligation of logging the information access. P2 specifies that a medical lab has the right to write medical test information as long as the admission end-date is not empty, which represents the duration of the patient's admission. The hospital, as specified in P3, can read patients' personal information for the purpose of statistics. P4 specifies the permission of reading diagnoses data for purposes scientific research by the university on the disease diagnosed as "Q21". In a patient's residence, sensors within a BAN can update lab readings elsewhere kept for storage purposes, which later can be used e.g. for diagnostic needs, while a notification to the doctor must be sent. Privacy policies P1-P5 must be always satisfied. We are not concerned with modeling here but with the technical infrastructure needed to support privacy-aware synchronization between resource-constrained edge devices. As such, P1-P5 may not be sufficient enough to assure privacy in a hospital setting but they can be considered representative of the case study. In essence, any P-RBAC policies that domain experts may model, POET would be able to handle.

B. Experiments Setup

We adopt a realistic synthesized medical dataset [24], to respect privacy of real patients – although the dataset is artificially generated [25] it contains similar characteristics with EHR data used in practice. Since we are concerned with data transformations, the semantic content of the EHR data is not relevant for our evaluation purposes. The experiments' setup, privacy policy specifications and datasets used are available in [12]. What we seek to evaluate in two experiments, are i) the applicability of our approach to deal with different devices and data formats that are used in the various entities in the case study and ii) the performance in which the privacy-aware data transformations are executed in each deployment type.

a) Applicability: In practical applications, data used in each deployment is different; we assume doctor's offices, BAN edge nodes and cloud-deployed nodes use EHR/JSON, while the university uses a tabular format. Moreover, we use different data according to the description of Section VI-A, along with the privacy policies defined (P1-P5). For executing experiments, we deploy our prototypical tools on appropriate settings – hospital and university nodes are virtual machines as typically deployed in the cloud, while edge nodes are limited devices physically deployed in their respective locations.

b) Performance: We measure performance of POET given an exemplar service level agreement (SLA) of 5 seconds in i) the cloud, ii) an edge node, and iii) a resource-constrained edge node, for comparison. In this case, we use identical data and identical privacy policies to ensure fairness. Our choice of a 5 seconds SLA is arbitrary; such an SLA reflects expected response time of this kind of applications that we target and enables comparison between devices.

Both experiments utilize computational nodes with the following specifications. Cloud nodes use a single-core Intel 2.70GHz processor with 4GB RAM. Edge nodes are ARMv8-based R-Pi3 devices featuring single-core 1.2GHz CPUs and 1GB RAM. Moreover, we consider a more resource-constrained edge device featuring a single-core ARMv6 1GHz processor with 512MB RAM, for the BAN within PR. The

TABLE I
EXPERIMENT RESULTS FOR THE MEDICAL INFORMATION USE CASE.

	Source size (datapoints)	Source size (bytes)	View size (datapoints)	BX Time
DO/Edge (ARMv8)	600	118k	300	0.26s
ML/Edge (ARMv8)	2000	413k	1900	2.33s
PR/Edge (ARMv6)	100	20k	100	0.19s
UV/Cloud (Intel2.7)	5000	954k	100	0.20s
HL/Cloud (Intel2.7)	11000	2180k	1000	0.38s

POET runtime is implemented in Haskell, applicable to a wide range of environments and software stacks.

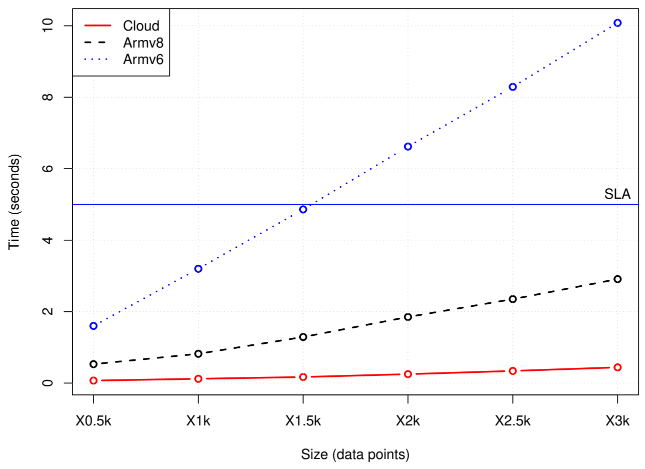

Fig. 5. Performance of POET deployed on different devices upon identical data and policies and across different data sizes, with a defined SLA of 5 sec.

C. Experiments Results

a) Medical Case Study: We consider two different data formats in the medical case study, tabular format and JSON. The prototypical data preparation stages for the two formats are implemented as per Section IV-B, consisting essentially of a parser and demonstration of unique data attributes. To give an indication of data interfacing, we note that prototypical implementations for the stages for the case study consist of just 38 lines (tabular data) and 85 lines (EHR/JSON) of code to transform other data formats into internal representation compatible within our framework.

b) Performance Comparison: Firstly, results of the the case study in Table I over data sizes that would realistically reside on the various deployment nodes and according policies (P1-P5) demonstrate that the computational overhead of the privacy-aware BX component of our approach is minimal. Figure 5 illustrates performance of the different computational devices upon identical data and identical privacy policies along different data sizes. A possible SLA of 5 seconds, which could be defined by an end-user application, allows cross-comparison and in line to expected response times of the applications that we target, giving an indication of the data sizes that can be adequately handled by deployments across resource constrained devices.

D. Discussion

Based on the results of our experiments we believe to have demonstrated that by using our privacy-aware transformation framework realistic applications can be managed and that performance on resource-constrained devices can be taken into account during application design. Specifically, the case study presented shows that for each data-device pair (Table I) reasonable performance is achieved for realistic data sizes used in practice. Moreover, from the perspective of practitioners aiming to use our approach the flexibility demonstrated in providing other source formats for input adds to the usability of POET (i.e., a few tens of lines of code for data interfacing). Performance results regarding the size of data (Fig. 5) provide an indication of performance on resource-constrained devices, which is in the range of about 5x (ARMv8) to 20x (ARMv6) with respect to a typical cloud virtual machine. Another factor that can affect performance is size and complexity of the privacy policies, something that we identify as an avenue for future investigation. Moreover, the feasibility of POET to run on diverse architectures has been demonstrated. We note that while our reference implementation is an unoptimized prototype, experimental results indicate feasibility for relevant models.

Our approach produces privacy-compliant bidirectional transformations of data. This consists the fundamental synchronization primitive. To realize a complete end-to-end application, operation and communication aspects must be treated, such as concurrency of updates, conflicting changes and incrementality. Such issues have been tackled extensively in scientific literature [26] [27] [28]. However, a basic assumption has been on data structures that fit in memory, as edge devices are resource-constrained and extensive data storage facilities are usually not available. Our paper focuses on privacy problems arising from data synchronization. Other privacy violations because of message exchanges may require a privacy model able to capture temporal sequences, such as Contextual Integrity [29]. In addition, recent developments in data within the IoT focus on streaming data. Our approach is based on demonstrating a minimal set of data attributes or fields on the source data preparation, that uniquely specify a data item. To support streaming data, instead of unique atttributes the position of a data item in the stream may be used, something which we identify as future work.

VII. Related Work

We presented a novel technical framework to engineering data privacy tailored for the pervasive edge computing systems of today, offering assurances on correct and well-behaved transformations. Consequently, we classify related work into three categories. First, we look into privacy models, positioning our work. Then, we review theoretical foundations on consistency and transformation mechanisms. Finally, we discuss related engineering approaches that aim in synchronizing structures applicable to the edge paradigm and thus to our approach.

Satisfaction of privacy requirements is highly relevant in edge computing. Such requirements include data confidentiality, access control within the IoT, trust among various devices and edge nodes, and the enforcement of privacy policies [30]. Role-based access control (RBAC) has been adopted as a pattern to goal-oriented models for detecting security problems by using model-driven transformation [31]. Inspired by this, our work focuses on privacy issues that the edge paradigm brings by applying model transformation. Many privacy models are available in existing research and industry; a typical example is the Usage Control model from the digital rights management domain, with some similarities with P-RBAC such as conditions and obligations [32]. The Privacy Preferences Project features a protocol allowing websites to express their privacy practices when they collect information about web browser users with the character of declaration, but not enforcement [33]. Other two popular languages that have been developed for expressing enforceable privacy policies are the Enterprise Privacy Authorization Language and the OASIS Standard eXtensible Access Control Markup Language [34]. The latter provides an independent policy language encoded in XML and enables different types of policies, not limited to privacy policies [35]. EPAL emphasizes the user categories that can access data but without data purposes and obligations [36]. Instead, our work utilizes the generically applicable P-RBAC –the privacy extension of RBAC.

Bidirectional Model Transformations are a popular mechanism for maintaining consistency of at least two related sources of information, and have been widely adopted. An approach that defines a consistency relation between two models is QVT Relations (QVT-R) language in the OMG QVT standard [37], supported by a QVT-R tool complying to checking semantics [38]. A Triple Graph Grammar [39] can also be used to conclude consistency, particularly between graph-like structures, as well as find a partial correspondence model combined with linear optimization techniques to detect maximum consistency portions [40]. However, it is time-consuming and non-trivial to manually maintain round-tripping laws. Other approaches such as the Atlas Transformation Language [41], graph querying [42] or security lenses [43] consider a standard forward-direction transformation with automatically derived backward transformations. With security lenses for instance, one would write a get (forward transformation) - the security lenses framework would automatically provide a secure put. However, the forward transformation may not be injective and its ambiguity of various corresponding put-directions is what makes bidirectional programming challenging and unpredictable in practice. Recently, putback-based approaches [19] have been proposed as an alternative, and allow to only write putback transformations. By contrast, a *put* transformation could uniquely determine *get* by well-behavedness, and the putback-based program guarantees that the *get* behaviours are unambiguously specified. BiYacc [44] and BiFlux [45] are typical examples where this is the case. BiGUL is a formally verified language which serves as a foundation for higher-level putback-based languages [11], [18]. In our approach, we adopt BiGUL to write a secure put and obtain a safe (privacy-aware) get with full control over the consistency restoration behaviors.

Edge computing is often referred as redefining users' interactions with IT services and integration with data synchronization services. Synchronization issues have already been emphasized in literature to cope with different data in the cloud [46]. A delta synchronization technique is available for web browsers – the most pervasive and OS-independent access – exhibiting fine granularity (i.e., only changed content need to be sent instead of the entire data item) [27]. A QoE-aware open synchronization framework using web technologies and adaptive synchronization model has been introduced to synchronize media streams and ensure the user experience of collective and interactive media [47]. In another approach, a file synchronization model presents a two-stage protocol along with a conflict resolution mechanism to manage file data which spans multiple devices [48]. Our approach provides a general mechanism to synchronize in-memory data in multiple formats through customized data interfacing. A middleware structure is proposed in [28] to facilitate efficient synchronization in unreliable mobile environments, involving bidirectional exchange of Electronic Health Record (EHR) data between patients and a care facility. However, it does not address privacy nor ensure bidirectional well-behavedness.

VIII. Conclusions

In this paper, motivated by the "data protection by design and by default" discipline [3], we proposed a technical framework to engineer data privacy tailored for the pervasive edge computing systems of today; it is based on a formal approach guaranteeing correct and well-behaved data transformations that respect privacy policies. In our approach, data leaving a component does so always in accordance to privacy policies, while changes applied to remote data are reflected back automatically. Such automated reflection is achieved through bidirectional model transformations featuring correct-by-construction guarantees. Our privacy reasoning approach is based on P-RBAC [7], a generically adopted model able to capture roles and permissions, actions on data, conditions and obligations that arise in privacy requirements. Our evaluation demonstrates the applicability of our approach on a medical information privacy use case and enables cross-comparison of its performance on resource-constrained devices.

Regarding future work, we aim to support streaming data, by investigating appropriate bidirectional transformations and privacy models based on communicating agents [29]. Moreover, to realize a complete end-to-end synchronization framework, techniques such as delta synchronization, conflict resolution and versioning must be integrated, besides our bidirectional transformations. Regarding the general edge computing setting, a basic assumption of our approach has been on structures that fit in memory – we plan to further investigate memory management within this resource-constrained environment as well as data storage and how they relate to data within a privacy scope.

Acknowledgment

Research partially supported by the National Basic Research Program of China (the 973 Program) under Grant No. 2015CB352201, the TU Wien Research Cluster Smart CT, and the National Natural Science Foundation of China under Grants No. 61620106007 and No. 61751210.

References

[1] Ju Ren, Hui Guo, Chugui Xu, and Yaoxue Zhang. Serving at the edge: A scalable iot architecture based on transparent computing. *IEEE Network*, 31(5):96–105, 2017.

[2] Rodrigo Roman, Jianying Zhou, and Javier Lopez. On the features and challenges of security and privacy in distributed internet of things. *Computer Networks*, 57(10):2266–2279, 2013.

[3] Regulation (EU) 2016/679 of the European Parliament of 27 April 2016 on the protection of natural persons with regard to the processing of personal data, on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). 2016.

[4] George J Annas et al. Hipaa regulations-a new era of medical-record privacy? *New England Journal of Medicine*, 348(15):1486–1490, 2003.

[5] California Senate Judiciary Committee et al. California consumer privacy act: Ab 375 legislative history. 2018.

[6] Krzysztof Czarnecki, J. Nathan Foster, Zhenjiang Hu, Ralf Lämmel, Andy Schürr, and James F. Terwilliger. Bidirectional transformations: A cross-discipline perspective. In *Theory and Practice of Model Transformations, Second International Conference, ICMT 2009, Zurich, Switzerland, June 29-30, 2009. Proceedings*, pages 260–283, 2009.

[7] Qun Ni, Elisa Bertino, Jorge Lobo, Carolyn Brodie, Clare-Marie Karat, John Karat, and Alberto Trombeta. Privacy-aware role-based access control. *ACM Trans. on Information and System Security*, 13(3), 2010.

[8] Amir Vahid Dastjerdi, Harshit Gupta, Rodrigo N Calheiros, Soumya K Ghosh, and Rajkumar Buyya. Fog computing: Principles, architectures, and applications. In *Internet of Things*, pages 61–75. Elsevier, 2016.

[9] Keiichi Yasumoto, Hirozumi Yamaguchi, and Hiroshi Shigeno. Survey of real-time processing technologies of iot data streams. *Journal of Information Processing*, 24(2):195–202, 2016.

[10] James Martin. Managing the data base environment. 1981.

[11] Hsiang-Shang Ko, Tao Zan, and Zhenjiang Hu. Bigul: a formally verified core language for putback-based bidirectional programming. In *Proceedings of theACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, 2016, St. Petersburg, USA*, pages 61–72, 2016.

[12] POET, evaluation datasets and privacy policy specifications. http://dsg.tuwien.ac.at/staff/ctsigkanos/poet, 2018.

[13] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*, pages 451–466. IEEE, 2017.

[14] H Teufel III. The fair information practice principles. *Department of Homeland Security: Department of Homeland Security*, 2008.

[15] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[16] J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Trans. Program. Lang. Syst.*, 29(3):17, 2007.

[17] Jeremy Gibbons, editor. *Generic and Indexed Programming - International Spring School, SSGIP 2010, Oxford, UK, Revised Lectures*, volume 7470 of *Lecture Notes in Computer Science*. Springer, 2012.

[18] Hsiang-Shang Ko and Zhenjiang Hu. An axiomatic basis for bidirectional programming. *PACMPL*, 2(POPL):41:1–41:29, 2018.

[19] Zhenjiang Hu, Hugo Pacheco, and Sebastian Fischer. Validity checking of putback transformations in bidirectional programming. In *FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings*, pages 1–15, 2014.

[20] Hsiang-Shang Ko. Project title, 2013.

[21] Ovidiu Vermesan, Peter Friess, Patrick Guillemin, Sergio Gusmeroli, Harald Sundmaeker, Alessandro Bassi, Ignacio Soler Jubert, Margaretha Mazura, Mark Harrison, Markus Eisenhauer, et al. Internet of things strategic research roadmap. *Internet of Things-Global Technological and Societal Trends*, 1(2011):9–52, 2011.

[22] E Monton, J Hernandez, JM Blasco, T Hervé, J Micallef, I Grech, A Brincat, and V Traver. Body area network for wireless patient monitoring. *IET communications*, 2(2):215–222, 2008.

[23] Tracy Gunter and Nicolas Terry. The emergence of national electronic health record architectures in the united states and australia: models, costs, and questions. *Journal of medical Internet research*, 7(1), 2005.

[24] Uri Kartoun. A methodology to generate virtual patient repositories. *arXiv preprint arXiv:1608.00570*, 2016.

[25] Uri Kartoun. Artificial emr data. *http://www.emrbots.org/*, 2016.

[26] Said Limam and Ghalem Belalem. A self-adaptive conflict resolution with flexible consistency guarantee in the cloud computing. *Multiagent and Grid Systems*, 12(3):217–238, 2016.

[27] Nitin Agrawal and Raju Rangaswami, editors. *16th USENIX Conference on File and Storage Technologies, FAST 2018, Oakland, CA, USA, February 12-15, 2018*. USENIX Association, 2018.

[28] Richard K. Lomotey, JoAnn Nilson, Kathy Mulder, Kristy Wittmeier, Candice Schachter, and Ralph Deters. Mobile medical data synchronization on cloud-powered middleware platform. *IEEE Trans. Services Computing*, 9(5):757–770, 2016.

[29] Adam Barth, Anupam Datta, John C Mitchell, and Helen Nissenbaum. Privacy and contextual integrity: Framework and applications. In *Security and Privacy, 2006 IEEE Symposium*. IEEE, 2006.

[30] Sabrina Sicari, Alessandra Rizzardi, Luigi Alfredo Grieco, and Alberto Coen-Porisini. Security, privacy and trust in internet of things: The road ahead. *Computer Networks*, 76:146–164, 2015.

[31] Andy Ozment and Ketil Stølen, editors. *Proceedings of the 4th ACM Workshop on Quality of Protection, QoP 2008, USA, 2008*. ACM, 2008.

[32] Jaehong Park and Ravi S. Sandhu. The ucon$_{abc}$ usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, 2004.

[33] MultiMedia LLC. MS Windows NT kernel description, 1999.

[34] Ari Juels, Ernesto Damiani, and Alban Gabillon, editors. *Proceedings of the 3rd ACM Workshop On Secure Web Services, SWS 2006, Alexandria, VA, USA, November 3, 2006*. ACM, 2006.

[35] T. Moses. Privacy policy profile of xacml v2.0; oasis standard, 2005.

[36] IBM. Enterprise privacy authorization language (ep al), v 1.2, 2003.

[37] Object Management Group (OMG). Meta-object facility (mof) specification, version 2.0. OMG Document Number formal/2006-01-01, 2006.

[38] Nuno Macedo and Alcino Cunha. Implementing QVT-R bidirectional model transformations using alloy. In *Fundamental Approaches to Softw. Engineering - 16th Intl. Conference, 2013*, pages 297–311, 2013.

[39] Andy Schürr. Specification of graph translators with triple graph grammars. In *Graph-Theoretic Concepts in Computer Science, 20th Intl. Workshop, Herrsching, Germany, 1994*, pages 151–163, 1994.

[40] Erhan Leblebici, Anthony Anjorin, and Andy Schürr. Inter-model consistency checking using triple graph grammars and linear optimization techniques. In *Fundamental Approaches to Software Engineering - 20th International Conference, 2017, Uppsala, Sweden*, pages 191–207, 2017.

[41] Yingfei Xiong, Dongxi Liu, Zhenjiang Hu, Haiyan Zhao, Masato Takeichi, and Hong Mei. Towards automatic model synchronization from model transformations. In *22nd IEEE/ACM Intl. Conference on Automated Software Engineering 2007, USA*, pages 164–173, 2007.

[42] Soichiro Hidaka, Zhenjiang Hu, Hiroyuki Kato, and Keisuke Nakano. A compositional approach to bidirectional model transformation. In *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada*, pages 235–238, 2009.

[43] J. Nathan Foster, Benjamin C. Pierce, and Steve Zdancewic. Updatable security views. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium, , New York, USA, 2009*, pages 60–74, 2009.

[44] Zirun Zhu, Hsiang-Shang Ko, Pedro Martins, João Saraiva, and Zhenjiang Hu. Biyacc: Roll your parser and reflective printer into one. In *Proceedings of the 4th Intl. Workshop on Bidirectional Transformations co-located with STAF 2015, L'Aquila, Italy, July 24, 2015.*, 2015.

[45] Hugo Pacheco, Tao Zan, and Zhenjiang Hu. Biflux: A bidirectional functional update language for XML. In *Proceedings of the 16th International Symposium on Principles and Practice of Declarative Programming, Canterbury UK, 2014*, pages 147–158, 2014.

[46] Kyle Chard, Steven Tuecke, and Ian T. Foster. Efficient and secure transfer, synchronization, and sharing of big data. *IEEE Cloud Computing*, 1(3):46–55, 2014.

[47] *13th IEEE Annual Consumer Communications & Networking Conference, CCNC 2016, Las Vegas USA, January 9-12, 2016*. IEEE, 2016.

[48] Chao Liang, Luokai Hu, Zhou Lei, and Jushu Wang. Synccs: A cloud storage based file synchronization approach. *JSW*, 9(7):1679–1686, 2014.