A Peer-based Approach to Privacy-Preserving Context Management

Wolfgang Apolinarski[1], Marcus Handte[1], Danh Le Phuoc[2], and Pedro José Marrón[1]

[1] Networked Embedded Systems, University of Duisburg-Essen, Germany
wolfgang.apolinarski|marcus.handte|pjmarron@uni-due.de
[2] Digital Enterprise Research Institute, Galway, Ireland
danh.lephuoc@deri.org

Abstract. Providing adequate support for context acquisition and management is a non-trivial task that complicates application development. To mitigate this, existing middleware systems rely on a centralized context service that manages the context of all devices located within a certain area. However, in many cases, centralized context management impacts privacy as it requires users to share their context with a possibly untrusted device. In this paper, we show how this problem can be avoided by a peer-based approach to context management. To validate the approach, we have implemented it on top of the BASE middleware and used it extensively in the PECES European research project. The evaluation shows that given a customizable implementation, the approach provides high flexibility while being suitable for a broad spectrum of devices.

Keywords: Context Management, Peer-to-Peer, Privacy, Security

1 Introduction

To provide seamless and distraction-free task support, ubiquitous computing applications must be able to perceive their context. To avoid repetitive development effort for context management at the application layer, existing middleware systems for ubiquitous computing provide context management support as one of their main building blocks. Thereby, they usually rely on a central system service that manages the context of all devices that are located within a certain physically limited area [2]. Sensors and other devices that are located within the area report their information directly to this service which can then provide a comprehensive view to interested applications. Such a centralized approach can considerably reduce the application development effort. Yet, if the number of devices increases, the central service may become a bottleneck.

While scalability issues can be countered by adding more resources, hierarchical [3] or decentralized context management [5, 9], from a user's perspective, a more problematic issue resulting from centralization is privacy. Given that context information may reveal sensitive information, centralized context management is only viable in cases where users are willing to put a sufficiently high

degree of trust in the intentions and capabilities of the operator. While this might be appropriate for some scenarios such as smart class rooms or smart homes, it is clearly not the case when considering scenarios such as smart shopping malls in which shop owners might operate context management services for their customers. In this paper, we show how such privacy problems can be avoided by peer-based context management. To validate the approach, we have implemented it on top of the BASE [7] middleware and used it for application development in the PECES project. The evaluation shows that the approach provides high flexibility while being applicable to a broad spectrum of devices.

The remainder of this paper is structured as follows. In Section 2, we first outline our approach for context modelling. Then, in Section 3 we present the system architecture which we evaluate in Section 4. Finally, we conclude the paper in Section 5.

2 Context Model

As basis for a generic context model, we adopt OWL-based context modeling [3, 6] since it facilitates autonomicity and interoperability. This enables the usage of an ontology to establish a common understanding among application developers and processing engines. Furthermore, it is possible to reuse existing ontologies such as SOUPA [4] or CONON [10] and it is easy to create extensions to support different application domains. In the following, we outline the underlying technologies and describe how they can be used to define and execute queries.

As OWL is a vocabulary extension of RDF[3], the atomic data element for context information is an RDF triple. Every piece of context information is implicitly or explicitly represented as a set of triples. To avoid inconsistency of RDF blank nodes in distributed settings, we do not use blank nodes to model context information. Assume there is an infinite set I (Internationalized Resource Identifiers IRIs) and an infinite set L (RDF literals). A triple (s, p, o)$\in$(I)$\times$(I)$\times$(I$\cup$L) is called a non-blank node RDF triple, where s is the subject, p is the predicate and o is the object. The assertion a triple states that some relationship, indicated by the predicate, holds between the subject and object. As examples for such triples consider statements such as 'user' 'has' 'name' or 'sensor' 'reads' 'value'. Conceptually, a set of triples can then be connected along the subjects to create a graph structure. The assertion of such an RDF graph accounts to asserting all the triples in it, so the meaning of an RDF graph is the conjunction (logical AND) of the statements corresponding to all the triples it contains. This enables the formulation of arbitrary expressions such as 'device' 'has' 'sensor', 'sensor' 'reads' 'value' which can then be used by means of complex queries.

As context information is represented as RDF triples, we naturally chose SPARQL query fragments as basis for defining queries. In order to be suitable for resource-poor embedded devices, however, we only include basic graph matching patterns as well as built-in conditions. Furthermore, we add simple geo

[3] `http://www.w3.org/TR/rdf-syntax/`

spatial extensions to express nearest neighbor as well as range queries analog to *http://code.google.com/p/geospatialweb/*. Informally, basic graph matching patterns allow us to formulate sequences of triples that may contain variables. Upon execution, the variables are bound based on the triples contained in the model. As a simple example, we might query sensors by specifying a pattern consisting of 'x' 'reads' 'y' and 'x' 'is a' 'sensor'. Using built-in conditions, we can restrict the binding of variables to particular values, e.g. 'y' is greater than 5.

Given this model, the query evaluation process is a state machine which can be presented as a query graph [8]. The query graph consists of operators that perform operations on their input data to generate output data. After decomposing a query process into smaller operators, it can be shown that all queries are composed from triple pattern operations. Therefore the minimum operators for building a state machine to execute a query is the triple pattern matching operator. Other operators such as $\bowtie$, $\cup$ and $\vDash$ can be optionally built as relational JOIN, UNION and SELECTION operators, if needed.

3 System Architecture

To avoid the inherent privacy issues of centralized context management approaches, we take a peer-based approach to context management in which each device is managing its own context that can then be shared with other devices on demand by executing remote queries locally. For spontaneous interaction between devices in the vicinity, we are using BASE as underlying communication middleware. BASE is structured as a minimal extensible micro-broker that is deployed on each device and mediates all interactions. This results in the layered system structure depicted on the left side of Figure 1. At the highest layer, application objects and middleware services interact with each other, either locally or remotely, through their local micro-broker. Usually, this interaction is mediated through stub objects that can be generated automatically from interface definitions. Underneath, the micro-broker takes care of providing a uniform interface for device discovery and interaction. To realize this, BASE utilizes an extensible plug-in architecture to abstract from different communication abstractions, protocols and technologies. So, each middleware instance can be configured with different sets of plug-ins and, at runtime, BASE takes care of automatically composing suitable communication stacks. We refer the reader to [7] for details. As indicated on the left side of Figure 1, the extensions to support peer-based privacy-preserving context management affect all layers of BASE.

The key storage takes care of establishing the identity of a device or a domain in a secure manner. It associates each device with a unique asymmetric key pair that represents its identity as well as a certificate that is signed by a particular domain. Thereby, domains may either represent individual devices as well as individual users or even larger corporations, e.g. by means of hierarchically signed certificates. Avoiding a central point of trust, individual trust levels are associated to the certificates and stored in the key storage to model trust relationships between devices/domains. The certificate's trust level recursively covers

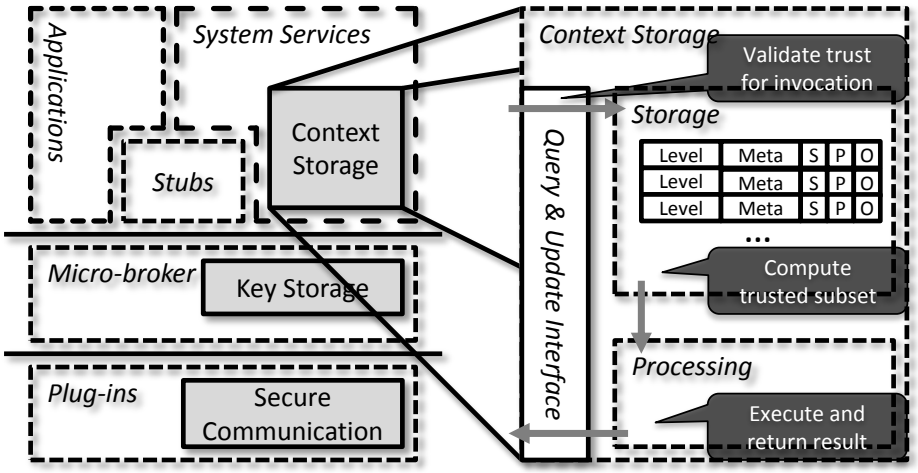

Fig. 1. Overall Architecture

the whole tree spanned by the certificate which reduces configuration effort. The trust levels are only valid for a particular device which enables users to model trust individually. At the same time, it enables the definition of uni-directional relationships. Regarding the trust levels, the key storage does not prescribe a particular semantic. To automate the evaluation of trust relationships, the key storage assumes transitivity such that a higher level of trust includes lower levels. Consequently, the two extremes are **full trust** and **no trust**.

The extensions for secure communication are verifying and enforcing the sharing according to the trust relationships. Due to the cost of asymmetric cryptography, the certificates contained in the key store are not suitable to secure interactions. Therefore, a key-exchange plug-in negotiates a secure symmetric key using the asymmetric keys. Thereby, the correct trust level is determined by exchanging certificate chains. Once a symmetric key is established and the trust level is evaluated, it is cached to secure all further interactions. This key can then be used by a second plug-in which ensures authenticity, integrity and optionally secrecy. For authenticity and integrity, an HMAC is attached to the transmitted messages. Encryption can be added additionally, e.g. by using AES.

With the extensions described previously, the task of the context storage service is reduced to ensure the generation of query results that adhere to the given trust level. As shown in Figure 1, a query and update interface is responsible for accepting incoming queries and update requests. Besides, it is responsible for determining the trust level of the incoming request. For this, it interacts with the key storage to determine the context of the call. The storage part is responsible for storing the context in a generic way that can be supported by resource-constrained devices. As discussed in Section 2, we rely on the triple representation defined by RDF, stored internally in the quoted string representation. To restrict the sharing of context on the basis of trust, each triple is

```
Q1: select ?device where{ ?device rdf:type device:Coordinator }

Q3: select ?device where{ ?device rdf:type device:Member.
    ?device device:hasAccessory ?accessory.  ?accessory rdf:type device:Screen.
     ?accessory device:hasResolution ?resolution.  ?resolution device:width.
     ?width ?resolution device:height ?height. FILTER (width>=240 && height>=320)}

Q5: select ?device where{ ?device smartspace:hasContext ?locCxt.
    ?locCxt smartspace:relatedLocation ?location.
    ?location spatial:within(53.27 -9.05 53.275 -9.055).
    {{?device service:provides data:printingService}
    UNION {?device  service:provides data:otherService} }}
```

Fig. 2. Excerpt for Classes of Queries

extended with a trust *level*. In addition, each triple may contain further *meta* information to allow the decentralized validation as described in [1].

Given this context representation, a caller may add/remove triples with the associated level and meta information to manipulate the device's context. In our implementation, the addition of level information as well as the triple generation for a particular context type is done statically as part of a sensor implementation. When executing a query, the update and query interface uses the attached level information to determine the relevant triples. It selects all triples whose level is lower or equal to the caller's level and forwards them together with the request to the processing engine. The processing engine then executes the query over the set of triples. Since the context that is used during query evaluation is restricted already, the result can be passed directly to the caller.

4 Evaluation

We implemented the described architecture and used it as basis for two application prototypes in the PECES European research project. Next, we first discuss our experiences before we present excerpts of an experimental evaluation.

To validate the applicability of the OWL-based model, we used it to model an ehealth and a traffic management application. Both scenarios shared a number of concepts such as different types of users, sensors, devices and smart spaces. To reduce the modeling effort, we developed a core ontology by combining and extending various established ontologies (FOAF, OWL-S, WGS-84, etc). On top of the core ontology we developed scenario specific extensions. While using the models, we found that even unexperienced application developers where quickly able to define static context characteristics and queries using Protege or SPARQL. Both scenarios also required the addition of intermediate trust levels which can be seen as an indicator for the need of a privacy-preserving approach. In the traffic management application, sharing the destination was necessary to automatically pay toll roads and parking lots. In the e-health application, the health-related sensor readings had to be shared with nurses and doctors. In both cases the definition of trust levels, certificates and relations was straight forward.

In contrast to centralized context management, peer-based management must be suitable for a broad range of heterogeneous devices. Thus, it is necessary that

Dataset	Implementation	Q1	Q2	Q3	Q4	Q5
S1	Embedded	49.06	42.01	3246.30	-	-
S1	Mobile	95.75	60.79	506.67	544.93	35.96
S1	Desktop	1.71	1.41	4.11	9.16	2.84
S2	Mobile	860.92	194.35	3589.34	2156.09	63.60
S2	Desktop	4.24	1.17	12.56	97.09	1.58

Table 1. Query Latency in Milliseconds

the concepts can be realized with little resources or that they can be tailored to the devices. To support a broad set of devices, we developed three interoperable context processors.

- **Desktop:** To support desktops that are for instance controlling patients homes, we developed a context storage using JENA and ARQ which supports processing and storage of thousands of triples and enables complex reasoning.
- **Mobile:** To provide a storage for less powerful devices such as the smart phones used by a driver or a patient, we created a stripped-down processor by removing unnecessary code from JENA and ARQ. Thereby, we chose a particular database backend which uses a B-Tree for indexing.
- **Embedded:** For embedded devices, such as the actuators in the traffic management scenario or on-body sensors of a patient, we created a minimal context storage without relying on existing libraries. The variant stores triples in-memory and performs an un-indexed query evaluation. It is suitable for 300-500 triples which turned out to be sufficient for our applications.

For the performance evaluation, several experiments were performed that measured the storage's response time with regard to different query types and data sets. As basis for this evaluation, we used a PC (AMD Opteron 250, 4GB RAM, Sun JVM 1.6) to test the desktop variant, a HTC Desire (CPU 1GHz, 576 RAM, Android 2.1) smart phone to test the mobile variant and a SunSPOT (CPU 180 MHz ARM920T, 512KB RAM, 4MB Flash, CLDC 1.1) to test the embedded variant. Based on the PECES applications, we generated 2 data sets and we identified 5 classes of queries with increasing complexity shown in Figure 2. The small set (S1) encompasses 350 triples with 50 devices and 2 smart spaces. The large set (S2) consists of 12000 triples with 2000 devices and 50 smart spaces. The first three queries, Q1, Q2 and Q3 are representative queries for all types of devices. The other two queries only appear on desktop type devices.

Table 1 shows the resulting latency. The implementation for embedded devices can easily handle Q1 and Q2. As a result of the non-indexed query processing, Q3's complexity increased the latency drastically. Given sufficient memory, it would be possible to decrease the latency significantly. The implementation for mobile devices can easily handle all queries in the small dataset (S1). In the medium dataset (S2), the latency rises which can be justified by the comparatively large size. The desktop machine can easily handle all queries on both set. Given these results as well as our experiences with application development in PECES, we are convinced that our peer-based approach to context management can provide a suitable privacy-preserving alternative to centralized approaches.

5 Conclusion

Providing adequate support for context acquisition and management is a non-trivial task that complicates the development of ubiquitous applications. In this paper, we have presented a peer-based alternative to centralized context management which overcomes privacy issues by supporting a fine-granular device- and thus, user-dependent definition of sharing policies. Although, our current implementation enables a fine-granular protection of the user's context, the actual policy specification must be done statically upfront. However, in some cases the user's privacy goals may not only depend on the type of context but also on the actual context itself. To handle that, we are currently working on languages and tools to automatically adapt the sharing policy at runtime.

Acknowledgments. This work has been partially supported by CONET (Cooperating Objects Network of Excellence) and PECES (PErvasive Computing in Embedded Systems), both funded by the European Commission under FP7 with contract numbers FP7-2007-2-224053 and FP7-224342-ICT-2007-2.

References

1. Apolinarski, W., Handte, M., Marron, P.: A secure context distribution framework for peer-based pervasive systems. In: 8th IEEE International Conference on Pervasive Computing and Communications Workshops. pp. 505–510 (April 2010)
2. Baldauf, M., Dustdar, S., Rosenberg, F.: A survey on context-aware systems. Int. J. Ad Hoc Ubiquitous Comput. 2, 263–277 (June 2007)
3. Chen, H., Finin, T., Joshi, A.: Semantic web in the context broker architecture. In: 2nd IEEE Intl. Conference on Pervasive Comp. and Comm. p. 277 (2004)
4. Chen, H., Perich, F., Finin, T., Joshi, A.: Soupa: Standard ontology for ubiquitous and pervasive applications. Mobile and Ubiquitous Systems, Annual International Conference on pp. 258–267 (2004)
5. Dey, A.K., Abowd, G.D., Salber, D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Hum.-Comput. Interact. 16, 97–166 (December 2001)
6. Gu, T., Wang, X.H., Pung, H.K., Zhang, D.Q.: An ontology-based context model in intelligent environments. In: Communication networks and distributed systems modeling and simulation conference. pp. 270–275 (2004)
7. Handte, M., Wagner, S., Schiele, G., Becker, C., Marron, P.J.: The base plug-in architecture - composable communication support for pervasive systems. In: 7th ACM International Conference on Pervasive Services (July 2010)
8. Hartig, O., Heese, R.: The sparql query graph model for query optimization. In: Proceedings of the 4th European conference on The Semantic Web: Research and Applications. pp. 564–578. ESWC '07, Springer-Verlag, Berlin, Heidelberg (2007)
9. Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G., Altmann, J., Retschitzegger, W.: Context-awareness on mobile devices - the hydrogen approach. In: 36th Hawaii International Conference on System Sciences. HICSS '03 (2003)
10. Zhang, D., Gu, T. and Wang, X.: Enabling context-aware smart home with semantic technology. International Journal of Human-friendly Welfare Robotic Systems 6(4), 12–20 (2005)