P2PCP : a peer-to-peer middleware for on-demand context provisioning in spontaneous networks

Tuan Dung Nguyen, Siegfried Rouvrais, Antoine Beugnard, Guy Bernard

P2PCP: A Peer-to-Peer Middleware for On-Demand Context Provisioning in Spontaneous Networks

Tuan Dung Nguyen*†,Siegfried Rouvrais*, Antoine Beugnard* and Guy Bernard†
*GET / ENST Bretagne
Technopôle de Brest Iroise, 29238 Brest, France
Email: {td.nguyen,siegfried.rouvrais,antoine.beugnard}@enst-bretagne.fr
†GET / INT CNRS SAMOVAR
9 rue Charles Fourier, 91011 Evry, France
Email: guy.bernard@int-evry.fr

***Abstract*— Nowadays, context-awareness has been widely accepted as an important requirement to provide appropriate services to users in dynamic environments. In this paper, context can be considered as any information relevant to an entity's situation. In spontaneous networks, the large number of context sources and the heterogeneity of mobile devices have raised new research challenges for supporting context-aware collaborative applications. This paper presents an ongoing work towards P2PCP, a peer-to-peer middleware for on-demand context provisioning based on a component-oriented approach.**

***Index Terms*— middleware, context provisioning, overlay, component, peer-to-peer, epidemic**

I. Introduction

The increasing popularity of personal wireless-capable computing devices (e.g. PDAs, smart phones) has engendered a new networking paradigm, namely spontaneous networks. Network nodes communicate using autonomously created ad hoc networks, without the need for any predefined infrastructure. These networks are also characterized by their mobile and heterogeneous participating devices. Applications in such highly dynamic environments [1] should be context-aware in order to provide appropriate services to users based on their current context (e.g. time, location, social-related).

Let's consider the following application scenario. Bob, equipped with a PDA, has just arrived in a supermarket. He has some information to share with others in proximity such as the outside temperature, or the nearest parking availability. He may be also interested in other information such as new products or flashed discounts. This information can be obtained directly from other users without depending on any specific and costly infrastructure.

Dey [2] defined context as any information that can be used to characterize the situation of an entity. Most existing context-aware systems often limit to only contextual information acquired from integrated logical or physical sensors. Some works in the literature have recently proposed a broader definition of context and clearly motivated the distributed provisioning of contextual information to provide more useful contexts in dynamic environments [3], [4]. Context provisioning consists of context acquisition, sharing and interpretation. Internal contexts are those acquired using integrated physical and/or logical sensors while external contexts come from the surrounding environment in a centralized or decentralized manner. In the former approach, specific entities are used for context storage and lookup while in the latter case, context data is provided by any entities within the proximity of an ad hoc network [4]. The centralized approach is rarely relevant to spontaneous networks due to the lack of infrastructure and the frequent mobility. The principal design requirements for context provisioning systems can be summarized as follows.

- Generalized context representation. Contextual information coming from different sources needs a common abstract representation to facilitate their management regarding to application's requirements. Component-based approach can be an suitable solution due to its abstraction and separation of concern.
- Efficient context sharing. Contextual information should be spontaneously shared among mobile, resource-constrained and heterogeneous devices. Comparing to other different distributed architectures (e.g. client/server, publish/subscribe), peer-to-peer [5] is a promising one as it provides a scalable and decentralized data sharing mechanism.

Our main contribution is a middleware to facilitate the construction of context-aware applications. It answers to the aforementioned requirements by combining a component-based context representation with a peer-to-peer provisioning architecture.

The paper is organized as follows. We present our general design principles in Section II. Then the management of on-demand contexts and their distributed provisioning are discussed respectively in Section III and Section IV. Next, section V presents the early implementation of our framework. We review the related work in Section VI and finally, conclude the paper in Section VII.

II. Design principles

We introduce two new concepts of context peer and context overlay for context provisioning in spontaneous networks, then present the supporting middleware architecture.

A. Context peer and context overlay

In reality, a resource-constrained mobile device often supports only a limited number of physical or logical sensors for capturing the corresponding contextual information. However, as depicted in Figure 1, it may require some other contexts due to its hosted application requirements. To cope with this issue, we propose the concept of context peer, namely CXTPEER, as an abstract representation of a specific context [6]. These logical peers are dynamically created on-demand and can be associated or not with the hosting device's inherently integrated sensors. Due to the fact that an application normally needs only a subset of contexts, managing all available contexts at any time becomes a cumbersome task. By representing each context with a distinct logical peer, our approach respects the *separation of concern* design paradigm. The system keeps only necessary information thus avoids unnecessary processing overhead.

Context peers located on different mobile devices, but representing the same kind of context, organize themselves in a common overlay network. Figure 1 presents an example where there are two distinct context overlays, namely CXTOVERLAY_1 and CXTOVERLAY_2 (e.g. temperature, prices). A CXTPEER can query other peers in the same overlay to obtain the required information. The context value can be cached at a CXTPEER for later use during a predefined *freshness* timer. This allows a CXTPEER to acquire contextual information in an opportunistic fashion [7] in case of partitioned networks. When there is no end-to-end path between a CXTPEER and another peer owning the context, information can be acquired through other peers with the help of device's mobility.

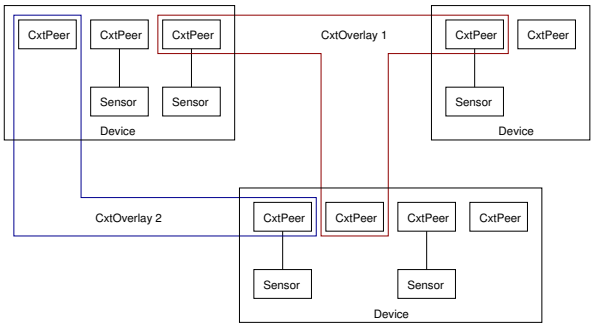

Fig. 1. Context peers and context overlays

B. Middleware architecture

To facilitate application's development with respect to the aforementioned design principles, a middleware layer between network operating systems and context-aware applications is necessary. Figure 2 shows our general middleware architecture which consists of the two following main components.

1) CXTMANAGER: responsible for managing locally located CXTPEER components (e.g. adding, removing, querying) and offering common programming interfaces to applications.

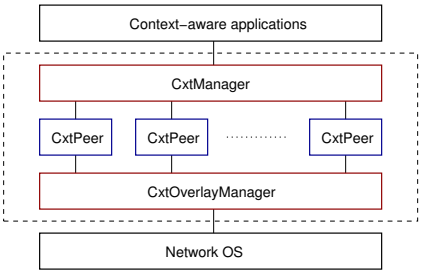

Fig. 2. Middleware architecture

2) CXTOVERLAYMANAGER: responsible for maintaining any available CXTPEEROVERLAY networks. This component exploits the co-existence of multiple overlays to optimize these operations.

The association between different software components of the middleware is illustrated using a UML class diagram in Figure 3. A CXTPEER is associated with a CONTEXT which can be associated or not with a SENSOR. A CXTPEER can be associated with several neighboring CXTPEER. These components are associated with their managing CXTMANAGER and CXTOVERLAYMANAGER components.

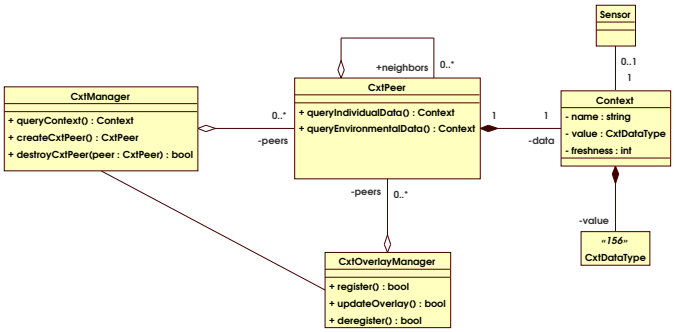

Fig. 3. Software class diagram

III. CONTEXT PEER MANAGEMENT

We followed a component-based approach for context peer modeling and implementation. There are several component models proposed in the literature, such as CCM [8], Fractal [9]. Fractal is a component model which has many interesting features (e.g. recursivity, reflectivity, component sharing) and supports different platforms. Its goal is to reduce the development, deployment and maintenance cost of software systems in general (e.g. applications, middleware platforms, operating systems).

A. From specification to deployment

In our system, a CXTPEER is modeled as a primitive component using Fractal. Multiple CXTPEER components can be managed in a CXTMANAGER composite component as depicted in Figure 6. Adding, removing a CXTPEER are realized using the container's control interfaces (e.g. ContentController

(CC), LifeCycleController (LC)). These operations are carried out on-demand based on current application's requirement. For example, when an application needs a context, it defines the corresponding logical CXTPEER. Then the middleware is in charge for managing this new component and returning the required contextual information. Applications can query a context in a synchronous or asynchronous fashion without caring about its sources.

Basically, a definition of CXTPEER using Fractal ADL [9] can be illustrated in a XML format as follows.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE definition PUBLIC
    "-//objectweb.org//DTD Fractal ADL 2.0//EN"
    "classpath://org/objectweb/fractal/adl/xml/basic.dtd">
<definition name="CxtPeer">
<interface name="out" role="server" signature="CxtService"/>
<interface name="in" role="client" signature="CxtService"/>
<content class="CxtPeerImpl"/>
</definition>
```

A CXTPEER has a provided interface, namely *out*, as well as a required interface, namely *in*, to respectively offer and acquire contextual information. Context acquisition is first locally done using the available corresponding sensor, otherwise basing on a query in the corresponding overlay.

We proposed a specification language to allow applications to define an on-demand CXTPEER. This XML-format specification is written by application's developers basing on their application's context needs. It provides the CXTMANAGER with all necessary information to autonomously manage the required CXTPEER component.

As depicted in Figure 4, an interpreter is responsible for converting the original specification into two parts: meta-information and Fractal ADL. The meta-information is used for high-level management of several CXTPEER (e.g. monitoring, caching) while the Fractal ADL is used for deploying the corresponding component.

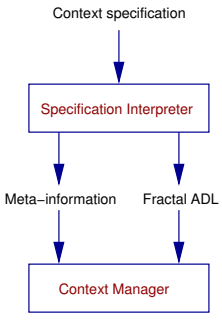

Fig. 4. Context specification processing

Later, the proposed language should also fulfill the following requirements:

- Take into account the potential relations between contexts and non-functional requirements (e.g. performance, reliability, security). The middleware will manage to create the related contexts based on application's non-functional requirements.
- Avoid potential ambiguities between different contexts. A specific context (e.g. temperature) should not appear in two distinct overlays. We plan to investigate an ontology-based approach to cope with this issue.

B. Context peer composition

The result of a query to a CXTPEER is a primitive contextual information. However, applications also need aggregated information from several contexts. This requirement leads to the dynamic composition of several CXTPEER components representing different contexts (cf. Figure 5). Recent works in the literature [10] have shown some benefits of component-based approach for processing of context information. We plan to investigate this issue in detail in our future work.

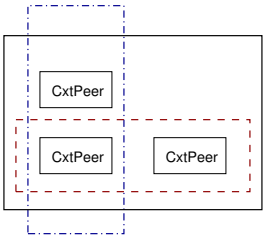

Fig. 5. Context peer composition

IV. CONTEXT OVERLAY MAINTENANCE

Context peers are connected in several peer-to-peer overlays. Generally speaking, a peer-to-peer network can be structured or unstructured [5]. We proposed using non-unstructured context overlay as it allows more flexible queries and less maintenance overhead in frequently changing environments. An overlay link is concretely represented by a distributed binding of interfaces between two CXTPEER components (cf. Figure 6). This kind of binding can be realized using Fractal RMI framework [11]. Due to node's physical mobility and its changing application's context demand, the overlay topologies change over time. In order to maintain these overlays in such dynamic environments, we have investigated several existing epidemic-style approaches [12], [13].

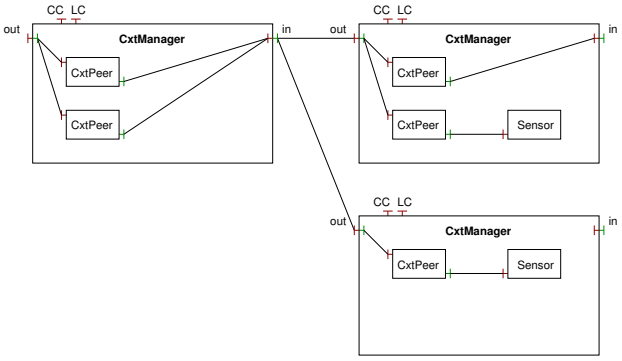

Fig. 6. Fractal-based component system modeling

A. Epidemic-style overlay maintenance

Epidemic-style algorithms have been widely studied in distributed systems (e.g. for failure detection, resource discovery, database replication, information dissemination). Inspired from the spreading mechanism of a contagious disease, information is disseminated from an entity to a set of randomly chosen neighbors. In turn, each of these entities buffers the received information for a defined period, does the same dissemination process, and so forth. These failure-resilient algorithms are efficient and reliable solutions for large-scale dynamic environments [14]. An epidemic-style overlay maintenance consists of two phases: peer sampling and peer clustering [13]. The former provides randomly selected peer candidates to the latter in order to update the corresponding overlay topology based on its semantic.

B. Coexistence of multiple overlays

The maintenance of multiple overlays in dynamic networks can become a costly operation. Therefore, optimization is necessary, especially in our resource-constrained environment. In the literature, the coexistence of multiple overlays has recently attracted several research efforts. Maniymaran et al. [15] have presented a mechanism to reduce the multiple overlays maintenance overhead without sacrificing the performance. The authors proposed that the maintenance of one overlay can be leveraged to partly maintain another overlay with no extra cost. However, the coexistence of multiple overlays in resource-constrained environments requires further consideration. We are currently investigating two approaches from horizontal and vertical points of view. The former exploits semantic proximity among distinct overlays while the latter bases on common phases of epidemic-style maintenance mechanism. As depicted in Figure 7, the maintenance of an overlay can be leveraged to update other overlays or eventually create a new one.

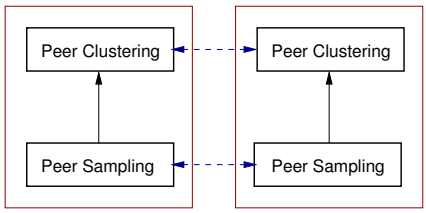

Fig. 7. Overlays maintenance principle

V. Implementation

The early prototype of our middleware has been developed using Julia [16], a standard Fractal Java-based implementation on desktop computers. It currently supports the management of on-demand context peers and context overlays. We plan to evaluate it with regard to some functional and non-functional criteria (e.g. performance, reliability) in a real application scenario. The system should fulfill the requirements already presented in Section I while remaining efficient in terms of computing and communication overhead due to its maintenance operations.

VI. Related work

The use of peer-to-peer overlays for context lookup was also presented in [17]. The authors proposed ContextBus, an architecture in which context producers are grouped based on the semantic of their data. An improved version, namely Semantic Context Space (SCS), was also proposed to reduce the maintenance cost when the number of semantic groups increases using a one-dimensional ring space [18]. Unlike this work, we propose a more abstract concept because any devices can have a context peer even if they do not directly produce context data. Moreover, we support dynamic peer management and effective mechanisms for overlay maintenance.

Tuple space was first presented in Linda [19] to provide a persistent and globally shared memory model to distributed processes. To support mobile ad hoc environment, LIME [20] proposed to break up the global tuple space into multiple spaces each located in different mobile components. The contents of these components are then transiently and transparently shared in temporary federated tuple space according to current connectivity. Its extended version, namely TinyLIME, was proposed to support limited power and computational resources requirement in sensor networks. EgoSpaces [3] is another tuple space variation where authors propose the view abstraction for asymmetric coordination in ad hoc mobile environments. The context peer concept in our work is close to tuple space approach but it allows more loosely couple between mobile devices. The combination of component-based and peer-to-peer sharing also offers comprehensive programming interfaces for application development.

The coexistence of multiple overlays and their application have recently gained much research interest. Component-based approaches facilitate overlay management. Gridkit [21] is a component framework for managing pluggable overlay networks at runtime. Behnel et al. [22] combined Gridkit with an overlay design modeling framework to enable the generation of specific code from platform-independent design of overlay networks. These works focus on a generic framework for overlay management and do not take into account the characteristics of spontaneous networks as we do.

VII. Conclusion

We have proposed P2PCP, a middleware for on-demand context provisioning in spontaneous networks. It offers an abstract representation and on-demand context management using a component-based approach. Context provisioning is carried out in a peer-to-peer fashion and supports frequently changing environments. Future work consists of proposing new lightweight algorithms for overlay maintenance and context composition. We plan to finish the early developed prototype and deploy it on real PDA devices in order to make further evaluations with more realistic environment's characteristics and application scenarios.

References

[1] L. M. Feeney, B. Ahlgren, and A. Westerlund, "Spontaneous Networking: An Application-Oriented Approach to Ad Hoc Networking," *IEEE Communications Magazine*, vol. 39, no. 6, pp. 176–181, June 2001.

[2] A. K. Dey, "Understanding and Using Context," *Personal Ubiquitous Computing*, vol. 5, no. 1, pp. 4–7, 2001.

[3] C. Julien and G.-C. Roman, "Egospaces: Facilitating rapid development of context-aware mobile applications," *IEEE Transaction on Software Engineering*, vol. 32, no. 5, pp. 281–298, May 2006.

[4] O. Riva, "Contory: A Middleware for the Provisioning of Context Information on Smart Phones," in *Proc. 7th ACM/IFIP/USENIX Int. Middleware Conference (Middleware'06)*. Melbourne, Australia: Springer-Verlag, Dec. 2006, pp. 219–239.

[5] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, "Peer-to-Peer Computing," HP Laboratories Palo Alto, Tech. Rep. HPL-2002-57R1, Mar. 2002.

[6] T. D. Nguyen and S. Rouvrais, "Towards a Peer-to-peer Middleware for Context Provisioning in Spontaneous Networks," in *Proc. 5th Workshop on Middleware for Network Eccentric and Mobile Applications (ESF/MiNEMA'07)*, Magdeburg, Germany, Sept. 2007, pp. 54–57.

[7] L. Pelusi, A. Passarella, and M. Conti, "Opportunistic Networking: Data Forwarding in Disconnected Mobile Ad Hoc Networks," *IEEE Communications Magazine*, vol. 44, no. 11, pp. 134–141, 2006.

[8] CCM. CORBA Component Model. OMG Document formal/06-04-01. Version 4.0, http://www.omg.org, 2006.

[9] Fractal. Fractal Component Model, http://fractal.objectweb.org, 2006.

[10] D. Conan, R. Rouvoy, and L. Seinturier, "Scalable Processing of Context Information with COSMOS," in *Proc. 7th IFIP Int. Conf. on Distributed Applications and Interoperable Systems (DAIS'07)*. Paphos, Cyprus: Springer Verlag, June 2007, pp. 210–224.

[11] Fractal RMI, http://fractal.objectweb.org/fractalrmi, 2006.

[12] M. Jelasity and O. Babaoglu, "T-Man: Fast Gossip-based Construction of Large-scale Overlay Topologies," University of Bologna, Department of Computer Science, Tech. Rep., 2004.

[13] S. Voulgaris and M. van Steen, "Epidemic-style Management of Semantic Overlays for Content-Based Searching," in *Proc. 11st European Conf. on Parallel and Distributed Computing (Euro-Par'05)*. Lisboa, Portugal: Springer-Verlag, Aug. 2005.

[14] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié, "Epidemic Information Dissemination in Distributed Systems," *IEEE Computer*, vol. 37, no. 5, pp. 60–67, May 2004.

[15] B. Maniymaran, M. Bertier, and A.-M. Kermarrec, "Build One, Get One Free: Leveraging the Coexistence of Multiple P2P Overlay Networks," in *Proc. 27th IEEE Int. Conf. on Distributed Computing Systems (ICDCS'07)*. Toronto, Canada: IEEE CS Press, June 2007.

[16] Julia, http://fractal.objectweb.org/julia, 2006.

[17] T. Gu, E. Tan, H. K. Pung, and D. Zhang, "A Peer-to-Peer Architecture for Context Lookup," in *Proc. 2nd Int. Conf. on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous'05)*. San Diego, CA, USA: IEEE CS Press, July 2005.

[18] T. Gu, H. K. Pung, and D. Zhang, "A Peer-to-Peer Overlay for Context Information Search," in *Proc. 14th IEEE Int. Conf. on Computer Communications and Networks (ICCCN'05)*. San Diego, CA, USA: IEEE CS Press, Oct. 2005.

[19] D. Gelernter, "Generative Communication in Linda," *ACM Computing Survey*, vol. 7, no. 1, pp. 80–112, Jan. 1985.

[20] A. L. Murphy, G. P. Picco, and G.-C. Roman, "LIME: A Middleware for Physical and Logical Mobility," in *Proc. 21st IEEE Int. Conf. on Distributed Computing Systems (ICDCS'01)*. Arizona, USA: IEEE CS Press, Apr. 2001, pp. 524–533.

[21] P. Grace, G. Coulson, G. Blair, L. Mathy, W. K. Yeung, W. Cai, D. Duce, and C. Cooper, "GRIDKIT: Pluggable Overlay Networks for Grid Computing," in *Proc. 6th OTM Int. Symp. on Distributed Objects and Applications (DOA'04)*. Agia Napa, Cyprus: Springer-Verlag, Oct. 2004, pp. 1463–1481.

[22] S. Behnel, A. Buchmann, P. Grace, B. Porter, and G. Coulson, "A Specification-to-Deployment Architecture for Overlay Networks," in *Proc. 8th OTM Int. Symp. on Distributed Objects and Applications (DOA'06)*. Montpellier, France: Springer-Verlag, Oct. 2006, pp. 1522–1540.