International Journal of Computer and Communication Technology

Volume 8 | Issue 2 Article 2

April 2017

WORK DONE ON MIDDLEWARE INFRASTRUCTURE FOR PREDICATE DETECTION

NAGMA FATIMA
Department of Computer Science and Engineering, Institute of Technology and Management, GIDA, Gorakhpur, India, nagmafatima8@gmail.com

SONAL SHARMA
GBTU, Institute of Technology and Management, Gorakhpur, India, rosy.sonal@gmail.com

SHIPRA SRIVASTAVA
GBTU, Institute of Technology and Management, Gorakhpur, India, shrivshipu77@gmail.com

WORK DONE ON MIDDLEWARE INFRASTRUCTURE FOR PREDICATE DETECTION

NAGMA FATIMA[1], SONAL SHARMA[2], SHIPRA SRIVASTAVA[3]

[1,2,3]GBTU, Institute of Technology and Management, Gorakhpur, India
E-mail: nagmafatima8@gmail.com, rosy.sonal@gmail.com, shrivshipu77@gmail.com

Abstract- Context-awareness is an essential feature of pervasive applications, and runtime detection of contextual properties is one of the primary approaches to enabling context awareness. However, existing context-aware middleware does not provide sufficient support for detection of contextual properties in asynchronous environments. The contextual activities usually involve multiple context collecting devices, which are fully-decentralized and interact in an asynchronous manner. However, existing context consistency checking schemes do not work in asynchronous environments, since they implicitly assume the availability of a global clock or relay on synchronized interactions. To this end, we present the Middleware Infrastructure for Predicate detection in Asynchronous environments (MIPA), which supports context awareness based on logical time. Design and Structure of MIPA are explained in detail.

Keywords- *Middleware, pervasive computing, predicate, asynchronous, context awareness*

1. INTRODUCTION

Middleware: Middleware is defined as a set of services that facilitate the development and the deployment of distributed systems in a heterogeneous networking environment. In the context of this paper, we further narrow this definition to be the software substrate which enables transparent remote invocations of services. The design requirements for middleware are still very complex because they cover two orthogonal middleware characteristics, identity coupling and temporal coupling, among applications requesting services (clients) and ones providing computing services (servers). Identity coupling characterizes how much clients and servers know about each other, and temporal coupling characterizes the degree of synchrony of the message exchange between them. Traditional RPC-based middleware, such as DCOM, CORBA, and Java RMI, exhibits strong identity and temporal coupling. Clients and servers of publish/ subscribe middleware, on the other hand, do not know about each others' identities and do not synchronize when Communicating either[16].
Pervasive applications are typically context-aware, using various kinds of contexts, such as location and time, to provide smart services [1] [2]. Context-aware applications need to monitor whether contexts bear specified property, thus being able to adapt to the computing environment accordingly. This brings the primary issue of contextual property detection. Though detection of contextual properties has been widely studied in pervasive computing and software engineering communities, it still remains a challenging issue, mainly due to the following two observations [1]. Contextual properties of concern to the context-aware applications bear great variety and dynamism [1][2]. Specifically, users are not only interested in local contextual properties, which can be easily obtained by singular context collecting devices, but also interested in global ones, which involve multiple decentralized devices for context collection[1]. Meanwhile, users are not only interested in static properties of contexts. Though static properties capture interesting aspects of contexts, they inherently lack the delineation of temporal and relative order. In many cases, users are also interested in behavioral properties, delineating temporal evolution of the environment [1].

In contrast to the variety and dynamism of contextual properties, the detection is greatly complicated by the intrinsic asynchrony in pervasive computing environments. Specifically, context collecting devices do not necessarily have a global clock. They heavily rely on wireless communications, which suffer from finite but arbitrary delay. Moreover, due to resource constraints, context collecting devices (usually resource-constrained sensors) often schedule the dissemination of context data. The different context update rates also result in asynchrony. In order to achieve context-awareness in asynchronous environments, the concept of time needs to be reexamined. Instead of assuming the availability of global time or synchronous interaction, we should rely on logical time. The basic rationale behind is to utilize the happen-before relation resulting from message causality and it's "on the fly" coding given by logical vector clocks. However, existing context-aware middleware does not provide sufficient support for detection of contextual properties in asynchronous environments [1].

We develop the Middleware Infrastructure for Predicate detection in Asynchronous environments (MIPA) [1] [2]. MIPA is the first open-source context-aware middleware, which provides systematic support for coping with the asynchrony while achieving context-awareness in pervasive computing environments, as far as we know. Based on MIPA, users can flexibly specify contextual properties by different types of predicates defined over asynchronous pervasive computing environments. MIPA accepts such predicates and supports context-awareness by online predicate detection [1].

2. THREE LAYER ARCHITECTURE

In a pervasive computing scenario, the modeling and processing of context is often handled by a Context-aware Middleware (CA-Middleware in short). From the middleware's perspective, a Context-Aware computing Environment (CA-Env in short) consists of three layers, as shown in Fig. 2.1:

2.1 Context-aware Applications (CA-Apps in short)

- Delineate their concerns about the computing environment via formal specification of contextual properties, and send the properties to the CA-Middleware;
- Conduct context-aware adaptation based on the result of contextual property detection from the CA-Middleware;

2.2 CA-Middleware

- Receive contextual properties from the CA-Apps;
- Obtain context data involved in the contextual properties from the sensors;
- Perform detection of contextual properties;
- Inform the CA-Apps of the detection results;

2.3 Sensors

- Collect raw context data;
- Disseminate the data to the CA-Middleware via push or pull

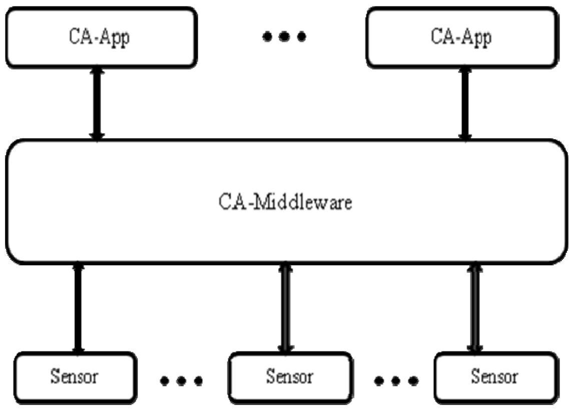

Figure 2.1 the Three Layer Architecture

3. EXISTING SYSTEM OF MIPA

MIPA aims at supporting the development and deployment of various predicate detection-based contextual property detection schemes for different pervasive computing scenarios [2].

Some of the work based on MIPA is as follows:

- Runtime Detection of the Concurrency Property in Asynchronous Pervasive Computing Environments.
- Detection of Behavioral Contextual Properties in Asynchronous Pervasive Computing Environments.
- A Lattice-theoretic Approach to Runtime Property Detection for Pervasive Context.

3.1 Context-aware Application: A Running Example
We envision a context-aware scenario (CA-Scenario) in a smart office, where a CA-App on each attendee's mobile phone automatically turns the phone to silent mode when the attendee attends a lecture. The location context is detected by the attendee's mobile phone. When the phone connects to the access point in the meeting room, we assume that the attendee is in this room. We assume that a presentation is going on if the projector is working. A CA-Middleware supports operations of the CA-Apps. The mobile phone reports the attendee's location and the projector reports its status to the CA-Middleware. Thus, the middleware can detect whether the attendees attend a lecture and send the results to the CA-Apps. When the attendee attends a lecture, the CA-app then turns the phone to silent mode. Meanwhile, administrator of the CA- Middleware checks the detailed trace about the operation of the CA-Middleware.

3.2 Propose Project
MIPA is java application base software. The basic idea behind MIPA is that it is open source software. In open system, system take input from the environment continuously and give the output depending upon the input. We have implemented the MIPA methodology in a different way in our propose project. We have implemented a MIPA in client-server architecture using RMI (Remote method invocation). In which, server has in infinite mode; client (processes) has continuously invoke to the server. There is a constructor which can work as a sensor. The processes come from the client passes to the server using constructor. In server we can see the output that how many process invoke the server and at which temperature. Server interface is work as a middleware.

4. CONTEXT PROCESSING FLOW

The context processing flow includes five steps. We now discuss how to support the context processing flow in detail.

4.1 Obtainment of specification

- Get the specification of the contextual property from the CA-App. The contextual property denotes the environment behavior of concern to the CA-App.

- Parse the property into an understandable form (according to the modeling of context) and determine the context involved in the property.

4.2 Acquisition of context

- Get connection to the Sensors that provide the context data required in the contextual property.
- Determine how to get context data from the Sensors (i.e., proactively query the Sensors or passively be notified by the Sensors).
- Process the raw context data and obtain low-level context.

4.3 Dissemination of context

- Enable transparent delivery of context data to some (usually remote) process deployed on the CA-Middleware.

4.4 Reasoning of context

- Aggregate low-level contexts (usually from multiple distributed sources).
- Reason over the low-level contexts (according to the PDPC framework) and obtain high-level contexts.

4.5 Interaction with the CA-App

- If the specified contextual property holds, inform the corresponding CA-App of the result. The CA-App then conducts the context-aware adaptation.
- If the CA-App is not concerned with the contextual property any more, dispose the contextual property.

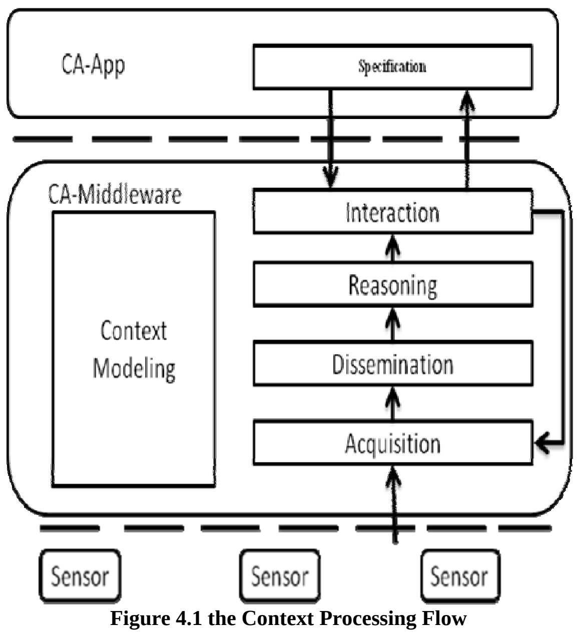

Figure 4.1 the Context Processing Flow

MIPA aims at supporting the development and deployment of various predicate detection-based contextual property detection schemes for different pervasive computing scenarios [2].

Some of our work based on MIPA is as follows:

- Runtime Detection of the Concurrency Property in Asynchronous Pervasive Computing Environments.
- Detection of Behavioral Contextual Properties in Asynchronous Pervasive Computing Environments.
- A Lattice-theoretic Approach to Runtime Property Detection for Pervasive Context.

A comprehensive case study is conducted to evaluate MIPA. In the case study, They implement over MIPA a smart lock scenario. The evaluation results show the cost-effectiveness and scalability of MIPA in pervasive computing scenarios [1].

Testing and monitoring distributed programs running on a distributed system involve the basic task of detecting whether a predicate holds during the execution. For example, a software engineer might want to detect the predicate “variable x has changed to value 2” to find out at what point in the execution x takes on a bad value. Another example arises in the area of safety-critical systems, where engineers need to maintain a certain invariant state of the system or guarantee a certain order in which events happen. In all these cases, predicate detection can provide information on whether certain conditions have held during the execution of the program on the system [9]

5. STRUCTURE OF MIPA

The middleware layer is the kernel part of MIPA. Its fundamental functionalities include:

- Checker process: The checker process collects vector clock timestamps of local contextual activities. It executes the predicate detection algorithm to decide whether the application-specified consistency constraint is satisfied. The checking result is sent back to the application via the predicate broker [2].
- Non-checker process: The non-checker process monitors the local predicate value based on the Event-Condition- Action (ECA) mechanism. It corresponding sensor agents. accepts source contextual events from the The local predicate serves as the event condition. When value of the local predicate changes, the consistency checking algorithm on the checker process side is triggered .The non-checker process sends messages to build the requisite happen-before relationship. It also sends checking message to the checker process, which finally decides whether the consistency constraint is satisfied.
- Predicate broker: The predicate broker accepts consistency constraints specified by the context-aware application. It first parses

the consistency constraint, and then initiates the non-checker processes and the checker process accordingly.

6. MAPPING CHARACTERISTICS OF PROPERTY DETECTION TO MIDDLEWARE STRATEGIES

In this section, we first discuss the characteristics of contextual property detection in asynchronous environments. Then we discuss how such characteristics motivate design of the MIPA. Abbreviations and Acronyms

6.1 Characteristics of Predicate Detection

The detection of contextual properties is transformed to the detection of logic predicates specified over the contexts. Predicate detection in asynchronous environments has the following salient characteristics:
C1. Dynamic composition of predicate checkers. Predicate detection can be viewed in a top-down manner based on the hierarchy of predicates. In the highest level, detection of a GSE predicate requires detection of the constituent CGS predicates. It also requires that all the CGSs involved form a GSE. Similarly, detection of a CGS predicate requires detection of the constituent local predicates. It also requires all the local states involved form a CGS. Due to the hierarchical structure of predicates, the checker of an upper level predicate can be constructed from checkers for lower level predicates. Moreover, checkers for the constituent predicates may lie on multiple distributed devices. The upper level checker may need to coordinate multiple distributed checkers.
C2. Monitoring the dynamic environment. Checking a local predicate is simple in the sense that it does not require interaction among distributed devices. However, the critical issue is that checking of local predicates must capture dynamic changes in the computing environment.
C3. Integration of new sensors. Users' requirements on context-awareness is open-ended. When new types of context collecting devices are available, users need to integrate such new devices, obtain new types of contexts, and specify predicates over such new contexts.

6.2 Overview of MIPA Design

Design of MIPA is motivated by the characteristics of predicate detection. From MIPA's point of view, a pervasive computing environment is composed of an application layer, a property detection layer and a context source layer, as shown in Fig. 2. The context source layer persistently collects contexts of concern. The property detection layer receives contexts from multiple decentralized and asynchronous context sources, and detects specified contextual properties at runtime. The key components of MIPA are outlined below.Predicate detection in groups . Motivated by C1 discussed above, i) MIPA supports composition of lower level predicate checkers to obtain higher level checkers; ii) MIPA supports distributed deployment of the checkers. This is achieved by grouping of different checkers. Specifically, the grouping can be decided by the predicate structure, i.e., lower level checkers consisting the same upper level checker belong to the same group. We can also further group the checkers according to the network condition Contextual event notification. In order to achieve accurate and timely monitoring of the environment as required in C2, we adopt the Event-Condition-Action (ECA) mechanism. Changes in the environment are modeled as contextual "events". Local predicates are interpreted as the "condition" to filter raw contextual events. Non-checker process serves as the event listener ("action").Pluggable sensor agents. To ease the inevitable process of integrating new types of context collecting devices, as required in C3, MIPA treats the sensor agents (in charge of manipulating the hardware sensors) as plug-ins[1].

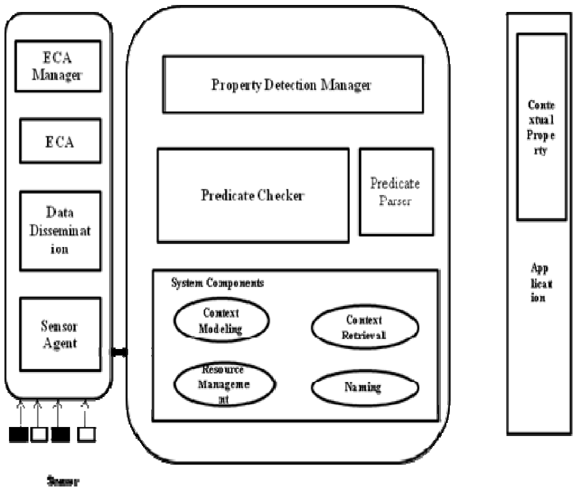

Figure3. System architecture of MIPA

7. CONCLUSION

In this work, we study how to provide middleware support for achieving context-awareness in asynchronous pervasive computing environments. Toward this objective, our contributions are: i) we introduce runtime detection of contextual properties based on logical time, to cope with the asynchrony in pervasive computing environments; ii) we design and implement MIPA to provide middleware support for runtime detection of contextual properties in asynchronous environments; iii) a comprehensive case study is conducted to evaluate MIPA. Currently, the MIPA middleware still suffers from several limitations. In our future work, we will investigate how to design a general algorithmic skeleton, to enhance the design and implementation of various predicate detection algorithms. We also need to study how to reduce the message complexity induced by our property detection schemes. Moreover, we will deploy MIPA to multiple distributed devices and conduct more realistic experimental evaluations.

8. ACKNOWLEDGMENTS

Our thanks to the experts who have contributed towards development of the template.

REFERENCES

[1] Jianping Yu, Yu Huang, Jiannong Cao and , Xianping Tao. 2010 Middleware Support for Context-awareness in Asynchronous Pervasive Computing Environments “

[2] Yu Huang, Jianping Yu, Jiannong Cao, Xiaoxing Ma, Xianping Tao and Jian Lu .Checking Behavioral Consistency Constraints for Pervasive Context in Asynchronous Environments.

[3] C. Xu, S. C. Cheung, and W. K. Chan.2006 Incremental consistency checking for pervasive context. in Proc. International Conference on Software Engineering (ICSE’06),

[4] Y. Huang, Y. Yang, J. Cao, X. Ma, X. Tao, and J. Lu. 2012 Runtime detection of the concurrency property in asynchronous pervasive computing environments.

[5] Anand Ranganathan and Roy H. Campbell, A Middleware for Context-Aware Agents in Ubiquitous Computing Environments.

[6] Hong, J. I., et al. An Infrastructure Approach to Context-Aware Computing.

[7] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, 2002 A middleware infrastructure for active spaces.

[8] C. Xu and S. C. Cheung.2005 Inconsistency detection and resolution for context-aware middleware support.

[9] FELIX C. GA¨RTNER and STEFAN PLEISCH Predicate Detection in Asynchronous Systems With Crash Failure.

[10] . Pierre-Charles David and Thomas Ledoux, An Infrastructure for Adaptable Middleware.

[11] Lukasz Juszczyk. A Middleware for Service-oriented Communication in Mobile Disaster Response Environments.

[12] Hengfeng Wei1,2, et al. 2012 Formal Specification and Runtime Detection of Temporal Properties for Asynchronous Context.

[13] Y. Huang, X. Ma, J. Cao, X. Tao, and J. Lu. 2009 Concurrent event detection for asynchronous consistency checking of pervasive context.

[14] Yiling Yang .2012 Formal Specification and Runtime Detection of Dynamic Properties in Asynchronous Pervasive Computing Environments.

◈◈◈