Policy-Based Personalized Context Dissemination for Location-Aware Services

Yousif Al Ridhawi, Ismaeel Al Ridhawi, Loubet Bruno, and Ahmed Karmouch

School of Information Technology and Engineering (SITE).
University of Ottawa, PO Box 450,
Ottawa, Ontario, K1N 6N5, Canada
{yalri098,ialri083}@uottawa.ca, karmouch@site.uottawa.ca

Abstract. This paper presents a policy-based context-level negotiation protocol and context-aware system architecture to personalize consumer-received context information through negotiations. Location tracking and prediction empower the system to shape contextual information delivered to users according to their expected needs.

Keywords: Context, negotiation,location prediction, context-aware architecture.

1 Introduction

This paper provides an enhanced context-aware system architecture expanding our earlier work [1],[2],[3] to evaluate and enforce CLAs through policies, and enhance *Client Views* generation, modification, and adaptation by tracking and predicting client location. The remainder of this paper is organized as follows. In section 2 we provide our modified an enhanced context-aware system architecture. In section 3 we present our CLA policy management. In section 4 we discuss some related work. Finally in section 5 we discuss our conclusion and future work.

2 Context-Aware System Architecture

Readers are advised to refer to [2] for our complete ontology-based context model used in our system architecture. A tri-layered system architecture was developed by our group and presented in [1]. The three levels were a context acquisition layer, a management layer, and a consumer layer. Since then, the system has been improved, and the updated architecture is provided in Figure 1. The management layer design has been expanded to include a new multi-sourced location tracking and prediction component for initiating negotiations with clients, predicting future movement, and adapting CVs to predicted paths within system's coverage area. The CIC is the *CLA Policy Management* unit responsible for attaining CLAs and generating respective internal policies. The policies are in turn enforced to supply clients with needed contextual knowledge. Further negotiations affecting the original CLA must be reflected within their respective policies; hence the need for a *Policy Update* unit.

P. Sénac, M. Ott, and A. Seneviratne (Eds.): MobiQuitous 2010, LNICST 73, pp. 366–371, 2012.

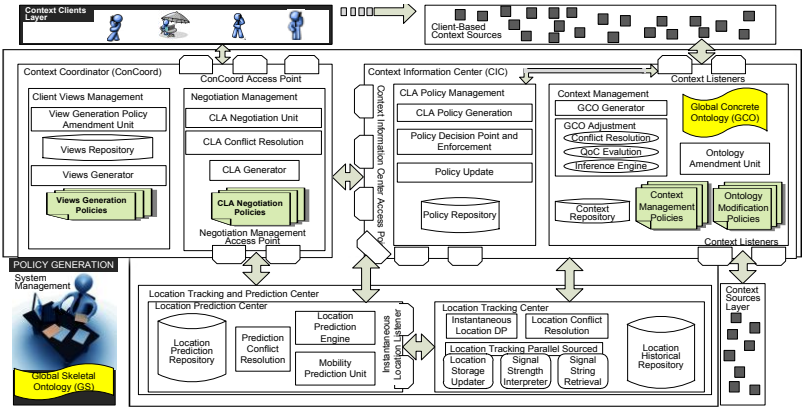


Fig. 1. Three-layered context-aware system architecture with location tracking enhancements

All established CLAs are utilized by the management system to modify existing CVs reflecting provider's changing abilities to supply context information. As CVs are modified, the provider must renegotiate some CLAs to reflect the changes. Modifications affixed to existing CLAs must be promoted into the respective CLA policies generated by the *CLA Policy Management* unit in the CIC. Our context-level negotiation protocol has been described in our earlier works [2], [3].

3 CLA Policy Management

All CLAs are translated into policies for enforcement and conflict resolution. The Policy Decision Point (PDP) examines applicable policies and determines actions to take, and the Policy Enforcement Point (PEP) enforces the outcome of policy decisions. ConCoord's CLA Policy Generator converts established CLAs into a set of policies that are enforced by the PDP in the CIC. In our system, the generator had as input a structure representing a CLA (from ConCoord Access Point) and a single external function call (from Policy Repository) to convert the CLA into policies.

Our choice of policy language must consider that all rules generated from our CLAs have a form of *Event-Condition-Action*. Additionally, users must be able to define their own operations added to existing ones, hence our decision to use Ponder[4]. Ponder policies act on *Managed Objects* representing a part of the system to control. The *Ponder toolkit* with its Java-based APIs allows writing new managed objects and creating operations for these objects. Two types of managed objects exist:

1. *Managed objects shared between all policies in the system*. It has two shared managed objects with one instance per CLA Policy Generator; GCO with context access operations, and CICAP with methods to send context information to clients.
2. *Managed objects created specifically for a CLA*. Two managed objects are created for each CLA. First a CLA object containing information about CLA clients, and methods to activate/deactivate the CLA. Second is a CV object representing a CV.

Every type of event carries information used in the condition and action parts of policies. Additionally, every event type must represent an element that triggers behaviour of the system thus executing a set of policies. Four elements are defined:

1. *Time*: contains information on the current date and time.
2. *GCO_Change*: generated every time a change occurs in the GCO property.
3. *CV_Change*: generated every time a change occurs to a Client View.
4. *CV_Request*: generated every time a client requests its CV from the provider.

The CLA policies we have implemented thus far are aimed at describing all the CLA applicability conditions which involve two steps.

1. Creating elementary policies to activate/deactivate the CLAs.
2. Grouping the policies to obtain complete CLA applicability conditions.

Some policies can take the following form: *at the occurrence of a TIME event, if the time matches START/END date and time, then ACTIVATE/DEACTIVATE the CLA*. Continuous CLAs, as seen in [3], require regular activations and deactivations and can be achieved through *time patterns*. For example, activating a CLA every Monday from the year 2010 at 8:30, the time pattern will be: "*year: 2010; month: -1; day: -1; day of the week: Monday; hour: 8; minute: 30; second: 0*", -1 marks irrelevant data.

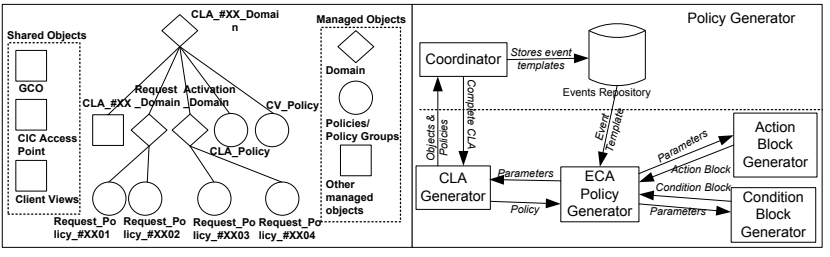
Logical Applicability Conditions; for both WHILE and START/END, the activation and deactivation of the CLA is triggered by logical conditions on the GCO. In both cases the following policy is used: *At any change in the GCO, if the condition C is true, then activate/deactivate the CLA*. Generating policies for the Hybrid CLA Validity Conditions, two policies are needed to activate/deactivate the CLA, and other policies are needed to activate/deactivate these policies. These can be logical conditions (WHILE or Start/END) embedded within a Periodic condition (Continuous or Non_Continuous). Other policies have been established for Continuous Periodic Hybrid Conditions, and Conditional-Periodic Hybrid Validity Conditions.

Context Request policies have also been implemented and divided into 3 types: *Notification_Request*, *Property_Value Request*, and *Class_Value_Request*. Refer to [3] for further details. Each context request can have an activation condition guarding delivery of its context. These conditions are translated into Activation policies similar to those of CLA Activation Policies and hence will not require further description. Delivery of CVs follows three types of policies; *No_Updates*, *Periodic_Delivery*, and *On_Change*. Periodic policies are similar to those of periodic CLA activation policies while the On_Change is similar to the START Trigger policies.

Managed Objects and Policies are stored into different domains and sub-domains to avoid mix-ups. For each CLA, a domain *CLA_#XX_Domain* is created, where *XX* is the ID of the CLA. This domain contains the *CLA_#XX* managed object from the CLA type to activate/deactivate the CLA, the CLA policy or policy group, the CV Policy or Policy Group, the Request and activation sub-domains containing all request policies and activation policies respectively. Figure 2a illustrates the CLA Domain.

The policy generator consists of several modules, figure 2b. During system initialization the Coordinator creates event templates and stores them in the Event Repository. At runtime, and for every established CLA, the Coordinator sends a description of the CLA to the Generator that creates domains and managed objects related to the CLA. The Coordinator also sends a description of the policy to the

ECA Policy Generator which creates an empty policy. The Generator then picks the right Event Template and adds it to the policy and sends the information to the Condition Block Generator which returns the condition block back to the Generator. The ECA Condition Block Generator adds the condition and action block to the policy and returns it to the CLA Generator which sends to the Coordinator a set with all the managed objects and policies. The Coordinator interacts with the clients and has the responsibility of executing the necessary policies to enforce established CLAs.



(a) (b)

Fig. 2. a) CLA Domain with all sub-domains b) Policy Generator archiecture

We can adapt CVs according to current clients' locations and their predicted motion paths. Responsible components are located within the Location Tracking and Prediction Center (fig. 1) and a complete description of this component was presented in [5]. Accordingly, the steps of adapting CVs are:

1. Create an initial set of CVs based on policies entered by system management.
2. Update a set of existing policies to fit the context access rights presented within the Client's Profile (CPs are received during client-provider negotiations).
3. Track/Predict the user's location/path and update existing CV policies to reveal/hide context according to the client's needs and rights, as well as the availability of contextual sources.

Our system also involves prediction methods for service pre-configuration and enhancement. Multiple position sensing technologies were incorporated to enhance current user location estimation. Location is identified through signal strengths emitted from WLAN access points and RFID readers. The first step consists of discovering surrounding access points and measuring WLAN signal strengths sent to mobile devices. The second step improves accuracy by active RFID tagging, where signal strengths from the RFID reader is measured and transferred to a 2-D map.

WLAN location tracking unit uses NetStumbler to gather information about available access points and emitted signal strengths. The values are compared with those stored in the repository to provide a signal strength map. Once the offline process of information gathering is complete, live user tracking is initiated. Signal strengths from available access points are compared to those in the repository. Once a location is estimated, it is stored in the location history repository.

The RFID location tracking unit uses RFID readers and active tags operating at a frequency of 433.92MHz, giving a range of up to 150 feet. Each reader is responsible

for its zone, where it signs the user's RFID tag in and out as the client moves into and out of its coverage area. When a client enters a coverage area, the profile is obtained from the repository and tracking begins. Signal strengths emitted from the RFID reader are gathered and plotted onto an RFID signal strength map. Those values are stored in the repository, where they are compared to find the user's estimated location. The locations are also stored in the location history repository for future use. Location Conflict Resolver Center aims at resolving conflicts between the two sets of location estimates. If the two methods estimate conflicting locations, conflict detection is triggered to choose the candidate closest to previous location.

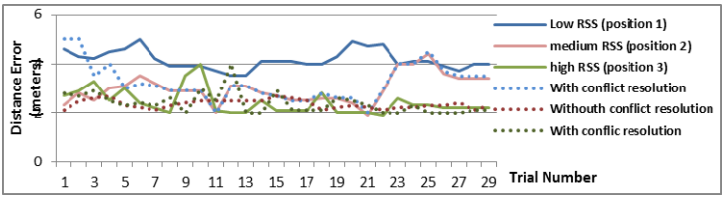


Fig. 3. System results with/without RFID and conflict resolution

4 Experimental Results

A prototype of our system was built and results were presented in [2]. Since then, a testing scenario was conducted at the University of Ottawa to simulate benefits of the dual location-tracking method. The system was implemented using Java built on the same server onto which the provider was activated. This was a 2.00 GB, 3.6 GHz Intel Pentium D station. Clients were installed on 2.0 GHz Intel Pentium Centrino laptops with an RF Code M100 active RFID tag.Information regarding nearby access points and received signal strength values were gathered using NetStumbler by all clients. Once user's location was determined, this information was displayed on a UI containing a map of the client's surrounding area. Since RSS decreases with distance and presence of obstacles, we enhanced our system through RFIDs.

Location tracking experiments were conducted over three different positions with low, medium and high RSS values respectively. Tests in the three locations were done first without utilizing conflict resolution or RFID tags, and then were repeated with the use of our conflict resolution center and RFID tags. As illustrated in figure 3, reductions in the overall distance errors between actual user location and the deduced user location were seen when the dual mode of RSS and RFID were used. Similar results were experienced in tests at positions 2 and 3.

5 Related Work

CASS [6] is a server-based middleware separating reasoning from context-aware applications, and provides event-based supply of context to mobile devices. CASS lacked a description on how context knowledgebase was structured and modeled, and there was no indication to the syntax and semantics of received context notifications.

Gaia's [7] distributed middleware permits development of context-aware applications and acts as a coordinator between software entities and network devices.

Context knowledge within Gaia is presented to applications through a Context File System (CFS) in the form of directories whose path components represent context types and values. Gaia is unsuitable for an environment characterized by mobility. Querying the CFS for context information requires context-aware applications to be aware of the directory structure and path to the needed context.

Hydrogen [8] provides a distributed solution with a firmware located on each device to share context knowledge with other devices. However, Hydrogen lacks two important components: an ontology on which context information is modeled, and a protocol by which context is shared between context servers located on different devices. The missing protocol for sharing context between devices is of particular interest since we provided a negotiation protocol for establishing CLAs. that could greatly improve Hydrogen architecture, by establishing CLAs between servers located on each mobile device giving it ability to acquire context outside its limited abilities.

6 Conclusion

The goal of our work was to develop a context-level negotiation protocol to establish CLAs in context-aware systems. We proposed a tri-layer architecture composed of context sources, providers, and consumers. The design allowed context-level negotiations with Clients and CLA enforcement through policy generation. The system also provided location tracking and prediction components to enhance CLA negotiation. We believe that the protocol can sufficiently meet location tracking and prediction requirements. However, we continue to improve on this system and protocol by adding new options and capabilities as our research progresses.

References

[1] Al Ridhawi, Y., Karmouch, A.: Ontology-Based Context-Level Agreements and Negotiations Protocol. In: International Conference on Intelligent Environments, USA (2008)

[2] Al Ridhawi, Y., Karmouch, A.: Ontology-Based Negotiation Protocol and Context-Level Agreements. In: 4th International Conference on IEs, pp. 1–8 (2008)

[3] Al Ridhawi, Y., Harroud, H., Karmouch, A., Agoulmine, N.: Policy Driven Context-Aware Services in Mobile Environments. In: IIT 2008 International Conference on Innovations in Information Technology, pp. 558–562 (December 2008)

[4] Damianu, N., Dulay, N., Lupu, E., Sloman, M.: The Ponder Policy Specification Language. In: Workshop on Policies for Distributed Systems and Networks, pp. 18–39 (2001)

[5] Al Ridhawi, I., Aloqaily, M., Karmouch, A., Agoulmine, N.: A location-aware user tracking and prediction system. In: Global Information Infrastructure Symposium, GIIS 2009, June 23-26, pp. 1–8 (2009)

[6] Fahy, P., Clarke, S.: CASS-Middleware for Mobile Context-Aware Applications. In: Workshop on Context Awareness, MobiSys 2004 (2004)

[7] Roman, M., Hess, C., Cerqueira, R., Anand, R., Campbell, R.H., Nahrstedt, K.: A Middleware Infrastructure for Active Spaces. IEEE Pervasive Computing, 74–83

[8] Devlic, A., Reichle, R., Wagner, M., Pinheiro, M.K., Vanrompay, Y., Berbers, Y., Valla, M.: Context Inference of User's Social Relationships and Distributed Policy Management. In: Proceedings of IEEE International Conference on Pervasive Computing and Communications, pp. 1–8 (2009)