PYP: Design and Implementation of a Context-Aware Configuration Manager for Smartphones *

Sin-seok Seo, Arum Kwon, and Joon-Myung Kang
Dept. of Computer Science and Engineering
Pohang University of Science and Technology (POSTECH)
Pohang, Korea
{sesise, arumk, eliot}@postech.ac.kr

John Strassner and James Won-Ki Hong
Div. of IT Convergence Engineering
Pohang University of Science and Technology (POSTECH)
Pohang, Korea
{johns, jwkhong}@postech.ac.kr

ABSTRACT

Since the introduction of *iPhone* and *Android*-based smartphones, the number of smartphone users has increased dramatically. Smartphones provide various application features, such as web browsing, movies, music, and games, in addition to a voice call function. They also support several communication interfaces (e.g., 3G, WiFi, and Bluetooth) and sensors (e.g., accelerometer and proximity). As smartphones are equipped with many features, users need to manage a growing number of complex configuration settings. To meet these needs, we propose a context-aware configuration manager for smartphones named *Personalize Your Phone* (*PYP*). *PYP* changes the configuration settings of a smartphone in response to changes in context, based on user defined policy rules. We present the design and implementation of PYP on an *Android*-based smartphone. A qualitative comparison between *PYP* and similar applications shows that *PYP* has a number of advantages.

Categories and Subject Descriptors

D.m [**Software**]: Miscellaneous—*Smartphone applications*

General Terms

Design, Management

Keywords

Context-aware, personalization, policy, smartphone application

1. INTRODUCTION

Smartphones are mobile phones that offer more advanced connectivity and computing capabilities than other types of mobile phones. Smartphones run more advanced operating systems, and hence can provide support for more advanced applications (as well as custom applications) that traditional mobile phones cannot accommodate. For examples, smartphones are increasingly used as entertainment platforms. Users take them everywhere for web surfing, gaming, listening to music, watching movies, and communicating with other people.

According to *Gartner*'s report, worldwide smartphone sales grew 95% in the third quarter of 2010 compared to the third quarter of 2009, while mobile phone sales increased by only 35% [3]. Another report says that the number of smartphone users is expected to exceed one billion worldwide by 2014 [10]. Hence, we are motivated to address ease of use issues with smartphones.

Smartphones have many changeable configuration settings including volume settings, WiFi turn on/off, and application management. Users desire different groups of settings to be applied for different contexts. However, this is currently difficult to do, because it is typically a manually-intensive process that cannot adapt to changing context.

For example, let us assume a smartphone user, Peter, is a graduate student. If he is discussing his thesis topic with his senior laboratory colleagues, then he might mute his smartphone ringtone to avoid being disturbed by the calls from his friends. However, if the caller is an important person, such as Peter's advisor, then he might want to make his phone vibrate so as not to miss the call. After finishing the meeting, Peter would probably want to change the smartphone's volume setting to notify him of all calls except those from unknown people. As another example, Peter might want to change the phone's WiFi settings by location. If he is at his home, then he might want to turn off WiFi to reduce battery consumption because he does not have a WiFi access point at his home. On the other hand, if he is working in his office, then he might want to turn on WiFi because WiFi is available at the office and provides better throughput than 3G.

In this paper, we propose a context-aware configuration manager for smartphones named *Personalize Your Phone* (*PYP*). *PYP* collects context information from the smartphone's operating system (OS) and automatically changes the smartphone configuration settings according to user defined policy rules. *PYP* policy rules specify when the smartphone needs to be changed and what configuration settings it needs to have for a given context. Examples of context information include date, time, location, on a call or not, and which applications are currently running.

The remainder of this paper is organized as follows. Sec-

*This research was supported by the MKE(The Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency) (NIPA-2011-C1090-1131-0009) and by the WCU (World Class University) program through National Research Foundation of Korea funded by the Ministry of Education, Science and Technology (R31-2010-000-10100-0).

tion 2 surveys and analyzes the applications that help smartphone users manage their smartphones easily and efficiently. This section also deals with existing work on context-aware systems. Section 3 derives the requirements for *PYP*, and section 4 specifies the design details of *PYP*. In section 5, we describe the implementation details and an example application of *PYP*. A qualitative comparison of *PYP* with other similar applications is given in section 6. Finally, section 7 concludes this paper.

2. RELATED WORK

Many applications help users manage their complex smartphone configuration settings. We searched for these applications in the *Android Market*[1] with keywords like 'profile', 'configuration', and 'setting'. We found 88 applications, and analyzed their features and characteristics. As a result, we divided the applications into five categories: 1) *Manual Profile Managers*, 2) *Automatic Profile Managers*, 3) *User Interface (UI) Settings Providers*, 4) *Task Managers*, and 5) *Information Providers*. Some popular applications corresponding to each category are shown in Table 1.

- **Manual Profile Managers**: These applications allow smartphone users to make profiles that are combinations of predefined smartphone configuration settings. Then, smartphone users manually select the profile that is the most appropriate for a given context. Thus, smartphone users can change many configuration settings by making a single selection.

- **Automatic Profile Managers**: These applications are similar to the *Manual Profile Manager* applications, except that the selection is done automatically based on predefined rules that are triggered when contextual changes are recognized, such as date, time, and location changes.

- **UI Setting Providers**: Changing various smartphone configuration settings is a very complex and often difficult task. For example, if we want to turn on the WiFi feature of an *Android*-based smartphone, we have to press the menu button to launch the settings screen. Then, we have to sequentially choose the 'Wireless and network', 'Wi-Fi settings', and 'Turn on Wi-Fi' items. The applications of the *UI Setting Provider* category provide an easy and direct UI to users to simplify these tasks.

- **Task Managers**: Smartphones can execute multiple applications at the same time; this is called multitasking. If some unused applications are left running in the background, then they can degrade the phone's performance and drain battery power. To solve this problem, *Task Manager* applications kill unused running applications automatically.

- **Information Providers**: Smartphones have various types of system information, such as battery status, CPU usage, memory consumption, and running applications, but it is hard to get detailed system information by using the smartphone OS itself. Thus, *Information Provider* applications show detailed system information to smartphone users through a fancy UI.

[1] *Android Market* is a common place for *Android* developers to register their applications and let *Android* smartphone users download and install them.

Table 1: Application Category for Helping Users Configure Their Smartphones

Category	Applications
Manual Profile Manager	Quick Profiles, IP Manager, Phone Profile Manager, ...
Automatic Profile Manager	Profile Call Blocker, Setting Profiles, Tasker, PhoneWeaver, Locale, ...
UI Settings Provider	Quick Settings, Battery Improver, Brighteriffic, Battery Saver, ...
Task Manager	Advanced Task Killer, Task Manager Advanced Task Manager, ...
Information Provider	Android System Info, Batteryminder System Info Widget, Mini Info, ...

PYP can be classified as an *Automatic Profile Manager* because it changes smartphone configuration settings automatically based on user defined policy rules. A comparison of PYP and other applications in the same category is given in section 6.

The most important characteristic of *PYP* is context-awareness, which makes the smartphone change its configuration settings according to context changes. A large number of studies about context-awareness have been reported. In [9], the authors propose a general context-aware layered architecture and model a relationship between an entity and its context. In addition, they suggest the concept of Quality of Context (QoC), which is meta-information that describes the quality of context information. In [6, 8], the authors suggest information models that can represent context data. [6] proposes the concept of Context-Aware Management Domains (CAMDs), which are management domains defined based on context situations and situation events. The applicability of CAMDs is shown through a case study. [8] suggests a context-aware policy information model that supports machine learning techniques for reasoning by extending the DEN-ng information model [7]. [5] models the context data of mobile devices using an ontology. It uses a *Naive Bayes Classifier* to classify low-level context data into high-level data.

As an application for context-aware systems, [4] proposes a context-aware autonomic handover decision scheme in heterogeneous wireless networks. It exploits user defined preference information and context data to choose the best user satisfying wireless network and access point. The other application of context-awareness is an interruption management system [11]. The authors propose an *Android*-based application that controls incoming phone calls depending on the user's context. This is similar to *PYP*, but *PYP* has a broader management and context domain than [11]. An extensive survey of other context-aware systems can be found in [1].

3. REQUIREMENTS

This section derives requirements for *PYP*. *PYP* is a context-aware policy-based smartphone configuration manager. So, requirements for *PYP* can be divided into three main groups:

1) Context-Awareness, 2) Policy Management, and 3) UI. Detailed specifications about these three requirements follow.

3.1 Context-Awareness

- **Context Data Collection**: *PYP* has to collect data related to the user's current context; this can be easily obtained by using the APIs of the smartphone's OS.
- **Context Inference**: Sometimes, the raw context data, which are obtained directly from the OS, are meaningless. Thus, we need an inference function to derive meaningful information from raw data. For example, a user could decide to accept or reject a call from the same person based on the particular context. A smartphone should be able to *infer* which action to take based on the current context.
- **Context Data Management**: *PYP* should support saving the collected and inferred context information in a repository. Also, the stored context information needs to be available whenever it is required.

3.2 Policy Management

- **Policy Rule Definition**: A policy rule grammar is required to define and process policy rules. That is, a formal representation is needed to describe context events and conditions and the corresponding smartphone configuration setting changes.
- **Policy Decision**: *PYP* examines the current context, which supplies data that match the events and/or conditions of the policy rule.
- **Policy Enforcement**: Once the policy rule conditions have been satisfied, *PYP* has to execute the actions that are defined in the satisfied policy rules. This policy enforcement function is essentially tightly-coupled with the smartphone's OS.
- **Policy Conflict Handling**: As the number of policy rules increases, policy conflicts may occur. A policy conflict occurs if two policy rules specify two opposite actions like turning the WiFi on and off in the same context. Thus, *PYP* has to detect and resolve policy conflicts.

3.3 User Interface

- **Ease of Use**: The purpose of *PYP* is to help users manage their smartphones conveniently and easily. Accordingly, the UI of *PYP* should be easy to use.
- **Policy Administration**: *PYP* has to provide a way to Create, Read, Update, and Delete (CRUD) policy rules by interacting with the user. In addition, we need a way to enable or disable each policy rule or all policy rules. A testing method that verifies whether the defined policy rules work well or not is also desirable.
- **Settings**: A UI for backing up and restoring user defined policy rules is needed. A UI for changing the intervals of context collection and policy decision is also needed to manage battery consumption.

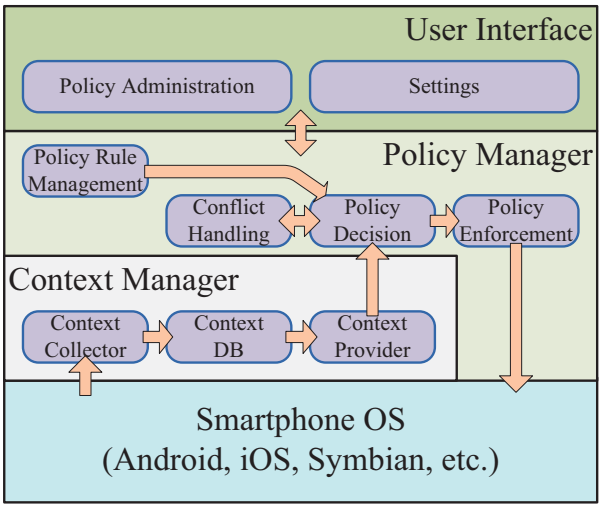


Figure 1: *PYP* Architecture

4. DESIGN

This section describes the design details of *PYP* to satisfy the requirements discussed in section 3. We first explain the high-level architecture of *PYP*, and then describe the conflict handling scheme and the policy rule definition.

4.1 Architecture

Figure 1 shows the high-level architecture of *PYP*. This architecture is based on the three main requirements of *PYP*, which are *Context-Awareness*, *Policy Management*, and *UI*. Thus, the architecture has three corresponding modules: the *Context Manager*, the *Policy Manager*, and the *UI*. The *Context Manager* obtains context data by using the APIs provided by the OS. It stores the collected context data in the context DB and provides the data to the *Policy Manager* upon request. We do not have an inference component in our current version of the *Context Manager* module. While we did mention it as a requirement for *PYP*, the proper inference of high-level user context from raw data is a very complex task, and is one of the major topics in the current context-awareness research. Moreover, integrating the inference function into smartphones, which have limited resources (e.g., CPU, memory, and battery), may cause severe performance degradation; this is assuming that there are enough resources to support context inference, which is likely *not* the case. Therefore, finding an efficient way to integrate an inference function into *PYP* is left as part of our future work.

The main task of the *Policy Manager* module is controlling the execution of policy rules that have been activated by comparing the current contextual data values with the definitions of the policy rules (*Policy Decision*). The other major task is executing the actions of the selected policy rules (*Policy Enforcement*). For example, suppose that a set of policy rules are activated. One execution approach could be to execute the first policy rule (based on either priority or first one available) and, if successful, stop execution. A second approach is to execute all rules until one fails. Other execution strategies can also be defined. Hence, this module provides *Policy Rule Management*, which saves or loads policy rules by using a formal rule grammar, and is

Table 2: Event, Condition, and Action Examples of Smartphone Policy Rules

ECA	Examples
Event	Battery Status Change, Headset Activated, Receiving Call, Receiving SMS, ...
Condition	WiFi State, Bluetooth Sate, Battery Level, Date, Time, Schedule, Location, Caller, ...
Action	Call Block, Send SMS, Set Volume, Turn on/off WiFi, Bluetooth, GPS, ...

responsible for policy *Conflict Handling* as well. The *Policy Manager* module obtains context information from the *Context Provider* component of the *Context Manager* module, and modifies policy rules by interacting with users through the *Policy Administration* component of the *UI* module.

The *UI* module interacts with the smartphone user to administer policy rules (Create, Read, Update, Delete, Enable, Disable, and Test). In addition, it provides a way to back up and restore policy rules, and to manage the intervals of context collection and policy rule decisions.

4.2 Policy Representation

4.2.1 A Policy Rule

A rule can be described using an *Event*, *Condition*, and *Action* (ECA) triplet. For example, a user may want to block (*Action*) incoming calls (*Event*) from his/her friends when he/she is having an important meeting (*Condition*). We adopt this principle to represent policy rules for *PYP*. Several examples of ECA components in smartphones are given in Table 2.

4.2.2 Policy Conflict Handling

As the number of policy rules increases, policy conflicts may occur. Detecting and resolving policy conflicts requires sophisticated algorithms and consumes computational resources. We decided to keep the conflict handling of *PYP* as simple as possible, because smartphone resources are limited in terms of adopting advanced conflict handling methods, and the conflicts that occur in smartphones usually do not result in critical problems.

PYP regards a policy conflict as having occurred when more than two policy rules are satisfied at the same time that apply different actions to the same configuration setting. *PYP* has three policy conflict handling methods: *No Action*, *Highest-Priority*, and *All*, and lets the smartphone user select one of them for each policy rule. The *No Action* method does not execute any actions when policy conflicts occur, but informs the user that a policy conflict exists. The *Highest-Priority* method executes only the actions of policy rules that have the highest priority. Each policy rule has a priority that is an integer number ranging from 1 to 5. A lower number means a higher priority. If more than two policy rules have the highest priority, then *PYP* executes all actions of them. The *All* method sequentially executes all the actions that are satisfied in order of priority for all conflict-free policy rules. If a policy conflict exists, then the user is notified.

If the conflicting policy rules specify different conflict handling methods, then *PYP* chooses a policy conflict handling method in the following order of priority:

No Action > *Highest-Priority* > *All*.

For example, if the two policy rules conflict and their conflict handling methods are *Highest-priority* and *All* respectively, then *PYP* executes only the actions of the policy rule with the highest priority.

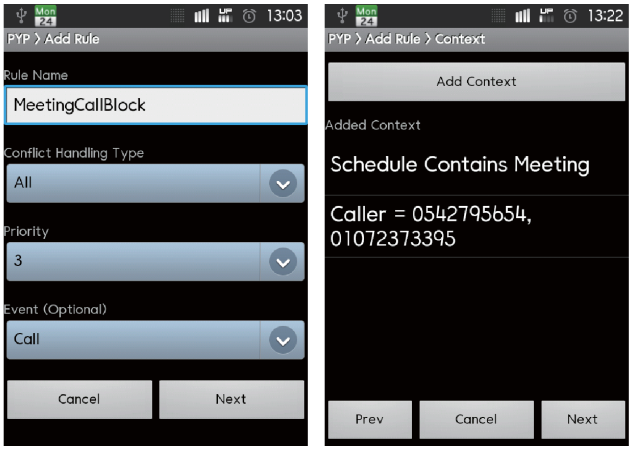
```
1  <!ELEMENT rulebase (rule*)>
2
3  <!ELEMENT rule (event?, condition*, action+)>
4    <!ATTLIST name CDATA #REQUIRED
5      conflict_handling (ALL|NO_ACTION|PRIORITY) #REQUIRED
6      priority (ONE|TWO|THREE|FOUR|FIVE) #REQUIRED
7      enabled (true|false) #REQUIRED>
8
9    <!ELEMENT event (#PCDATA)>
10
11   <!ELEMENT condition (context_data, value+)>
12     <!ATTLIST operator (LT|GT|EQ|LE|GE|NE|CONTAIN)
13                        #REQUIRED>
14     <!ELEMENT context_data (#PCDATA)>
15     <!ELEMENT value (#PCDATA)>
16
17   <!ELEMENT action (job, parameter*)>
18     <!ATTLIST order CDATA #REQUIRED>
19     <!ELEMENT job (#PCDATA)>
20     <!ELEMENT parameter (value+)>
21       <!ATTLIST name CDATA #REQUIRED>
```

Figure 2: DTD for PYP Policy Rule Representation

4.2.3 Policy Rule Representation using an XML

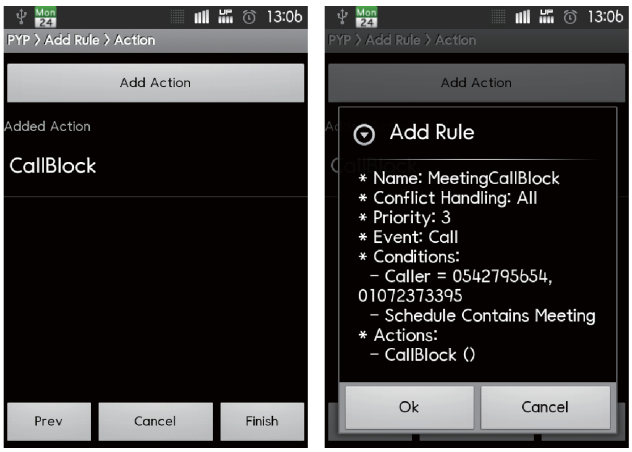
Internally, an ECA-based policy rule is represented using an eXtensible Markup Language (XML) in *PYP*. We chose XML because it provides a basis for describing grammatical restrictions, many open source programs are available to parse XML texts, and most importantly, the XML is simple and easy to develop. A Document Type Definition (DTD) is a set of markup declarations that defines the document type of an XML. It is a set of formal syntax that declares precisely which elements and references may appear where in the document, and what the elements' contents and attributes are.

The DTD that defines *PYP* policy rules is shown in Figure 2. The *rulebase* element contains *rule* elements. The *rule* element can have zero or one *event*, zero or more *condition*, and one or more *action* elements. If the *event* and *condition* elements are missing, then *PYP* executes actions that are described in the *action* elements every time it checks the policy rules. The *rule* element also has *name*, *conflict_handling*, *priority*, and *enabled* attributes. The *conflict_handling* and *priority* attributes specify the conflict handling type and priority of the policy rule respectively. The *enabled* attribute value depends on a user's choice and *PYP* only checks the event and the conditions of enabled rules. The *event* element has a character sequence datum that specifies the event type. The *condition* element consists of one *context_data*, one or more *value* elements, and one *operator* attribute. *PYP* compares *context_data* and *value* with the *operator* to find satisfied policy rules. Some actions require only the action type, whereas others require both action type and parameters. For example, an action that turn on WiFi does not require additional parameters, but one that changes the ringtone volume requires a parameter that designates the volume value.



(a) Setting rule name, conflict handling type, priority, and event type

(b) Adding context conditions



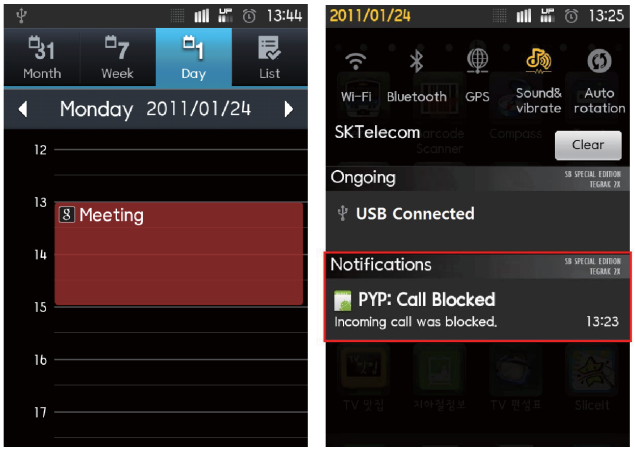
(c) Adding actions

(d) Summary of a new rule

Figure 3: Screen shots of Adding a Policy Rule

```
<rule name="MeetingCallBlock"
  conflict_handling="ALL"
  priority="THREE"
  enabled="true">

  <event>Call</event>

  <condition operator="EQ">
    <context_data>Caller</context_data>
    <value>0542795654</value>
    <value>01072373395</value>
  </condition>

  <condition operator="CONTAIN">
    <context_data>Schedule</context_data>
    <value>Meeting</value>
  </condition>

  <action order="1">
    <job>CallBlock</job>
  </action>
</rule>
```

Figure 4: An example of policy rule representation



(a) User's schedule

(b) A notification of a call block action

Figure 5: Screen shots of a User's Schedule and a Call Block Notification

Thus, the *action* element consists of one *job* that describes work to do and zero or more *parameter* elements. It also has one *order* attribute that specifies the execution sequence of actions when the policy rule requires multiple actions to be executed. An example of a *PYP* policy rule based on the above DTD is given in section 5.

5. IMPLEMENTATION

We implemented *PYP* on an *Android*-based smartphone. The reasons why we chose the *Android* platform are: 1) it supports powerful multi-tasking, which is essential for the background rule decision process of *PYP*, 2) it provides good documentation, 3) its source code is publicly available, and 4) it allows accessing and changing various system level resources. We installed the implemented *PYP* on a *Samsung Galaxy S* smartphone for testing. *PYP* worked well, and an example of the installed PYP follows.

In this example of *PYP*, we assume that a user wants to block incoming calls from the phone numbers '054-279-5654' or '010-7237-3395' when the user is having a meeting. The screen shots for setting a policy rule for the example are shown in Figure 3. A user launches PYP, then presses the menu button to add a new policy rule. Then the user can set the policy rule name, conflict handling type, priority, and event type for the rule (Figure 3a). Here we set the event type to 'Call'. If we press the 'Next' button, we can set conditions for this rule (Figure 3b). We add two conditions: one is a 'Schedule', which contains the text 'Meeting', and the other is a 'Caller' that equals either '0542795654' or '01072373395'. Finally, we add an action to block the incoming call (Figure 3c). If we click the 'Finish' button, PYP shows a summary of the newly defined policy rule (Figure 3d). The newly defined policy rule is stored in XML format and validated using the DTD defined in section 4.2.3.

The user was supposed to have a meeting from 13:00 to 15:00 on Jan. 24th, 2011, so he recorded it in his schedule (Figure 5a). At 13:23, in the middle of the meeting, the user got a phone call from the number '054-279-5654'. At that moment, PYP checked the policy rules, which had the

Table 3: Comparison of *PYP* with *Tasker*, *PhoneWeaver*, and *Setting Profiles*

Criteria	Tasker	Phone Weaver	Setting Profiles	PYP
UI	Hard	**Easy**	**Easy**	**Easy**
Supporting Context	**Rich**	Limited	Normal	**Rich**
Rule Type	CA	CA	CA	**ECA**
Conflict Handling	◯	×	×	◯
Price	$6.37	$3.95	$3.99	**Free**

'Call' event, and decided to block the call without bothering the user. The user was notified of the action through the *Notifications* feature of the *Android* OS (Figure 5b).

6. DISCUSSION

PYP can be classified as an *Automatic Profile Manager* (Table 1). We qualitatively compared *PYP* with other similar applications in the same category, and the results are shown in Table 3. The comparison target applications are *Tasker*, *PhoneWeaver*, and *Setting Profiles*. All applications except *Tasker* provide an intuitive and easy to follow UI. *Tasker* provides too many features, and some of them are not very useful for normal smartphone users. This makes *Tasker*'s UI hard to use. *PYP* and *Tasker* support rich context information compared to *PhoneWeaver* and *Setting Profiles*. The three applications use Condition and Action (CA) types to represent policy rules, while *PYP* uses the ECA type. *PYP* is more efficient than the others, because it checks only the policy rules that correspond to current events. Only *PYP* and *Tasker* provide conflict handling. *Tasker* takes a simple approach similar to *PYP* to handle policy conflicts. All applications except *PYP* are fee-based, and *Tasker* in particular is relatively more expensive. We are planning to distribute PYP through the *Android Market* for free.

In summary, we conclude that *PYP* and *Tasker* provide the most powerful features. However, *Tasker* is very difficult to use and expensive for normal smartphone users, whereas *PYP* has an intuitive UI and is free. Therefore, we can say that *PYP* is the most attractive application to automatically manage smartphone configuration settings.

7. CONCLUDING REMARKS

As smartphones are equipped with many features, smartphone users need to manage features according to varying contexts. We have proposed *PYP* as a context-aware smartphone configuration management application. *PYP* collects context data from the OS, and decides which policy rules have been satisfied by comparing the current context and the policy rule definitions. After that, *PYP* automatically changes smartphone configuration settings without user intervention. *PYP* was implemented and tested on an *Android*-based smartphone, and a qualitative comparison with other similar applications demonstrates that *PYP* is a very attractive application.

As future work, we will improve *PYP* by adding an advanced conflict handling method and context inference function. The current policy conflict handling method of *PYP* is trivial and simple, although the importance of policy conflict handling in *PYP* increases as the number of policy rules grows. We are considering several policy conflict handling methods, including information model-based policy conflict analysis [2], for *PYP*. In addition, we did not implement a context inference function in the current version of *PYP* because it would require a lot of computational resources. As a solution, we will deploy a dedicated context server that collects various context data from different user devices, including smartphones and laptops, and infers high-level context information from the collected raw data.

In addition, we are planning to implement *PYP* on other smartphone OSs including *iOS*, *Symbian*, and *Windows Mobile*. *PYP* will be offered for free to smartphone users, while maintaining a convenient UI and competitive features. We hope that *PYP* can help smartphone users manage their smartphones easily and efficiently.

8. REFERENCES

[1] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *Int. J. of Ad Hoc and Ubiquitous Comput.*, 2(4):263–277, 2007.

[2] S. Davy, B. Jennings, and J. Strassner. Application domain independent policy conflict analysis using information models. In *Proc. NOMS '08*, pages 17–24, Salvador, Brazil, Apr. 7–11, 2008.

[3] Gartner. Gartner says worldwide mobile phone sales grew 35 percent in third quarter 2010; smartphone sales increased 96 percent. http://www.gartner.com/it/page.jsp?id=1466313, Nov. 10, 2010.

[4] J.-M. Kang, J. Strassner, S. Seo, and J. W.-K. Hong. Autonomic personalized handover decisions for mobile services in heterogeneous wireless networks. *Computer Networks*, 2011. In Press.

[5] P. Korpipää, J. Mäntyjärvi, J. Kela, H. Keränen, and E.-J. Malm. Managing context information in mobile devices. *IEEE Pervasive Comput.*, 2(3):42–51, 2003.

[6] R. Neisse, P. D. Costa, M. Wegdam, and M. van Sinderen. An information model and architecture for context-aware management domains. In *Proc. POLICY '08*, pages 162–169, Palisades, NY, USA, June 2–4, 2008.

[7] J. Strassner. "Introduction to DEN-ng", tutorial for FP7 PanLab II project. http://www.autonomic-communication.org/teaching/ais/slides0809/Introduction_to_DEN-ng_for_PII.pdf, Jan. 21, 2009.

[8] J. Strassner, J. N. de Souza, D. Raymer, S. Samudrala, S. Davy, and K. Barrett. The design of a novel context-aware policy model to support machine-based learning and reasoning. *J. of Cluster Comput.*, 12(1):17–43, 2009.

[9] M. J. van Sinderen, A. T. van Halteren, M. Wegdam, H. B. Meeuwissen, and E. H. Eertink. Supporting context-aware mobile applications: An infrastructure approach. *IEEE Commun. Mag.*, 44(9):96–104, Sept. 2006.

[10] H. Wang. Smartphone: King of convergence. Technical report, Parks Associates, Mar. 10, 2010.

[11] S. Zulkernain, P. Madiraju, S. I. Ahamed, and K. Stamm. A mobile intelligent interruption management system. *J. Univ. Comput. Sci.*, 16(15):2060–2080, Oct. 2010.