Configurable integrated monitoring system for mobile devices

Christopher Miller\textsuperscript{a,∗}, Christian Poellabauer\textsuperscript{b}

\textsuperscript{a}Electrical and Computer Engineering Department, Rose-Hulman Institute of Technology, Terre Haute, IN 47803
\textsuperscript{b}Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, Indiana 46556

Abstract

Monitoring on smartphones and mobile devices presents several challenges, such as limited battery power and computing resources, but also provides some unique opportunities, particularly the ability to capture contextual data. Incorporating the context of the activities being monitored could improve performance modeling. Real-time monitoring could also be used to facilitate context-aware applications. In this paper, we introduce a configurable integrated monitoring solution for mobile devices. The monitoring service efficiently integrates multiple monitoring requests, and allows clients to monitor simple metrics or receive notification of custom defined events.

© 2014 The Authors. Published by Elsevier B.V.

Keywords:
monitoring, context-awareness, mobile, smartphone, middleware

1. Introduction

As highly capable mobile devices become more prevalent, more activities are being pushed to these mobile devices, rather than using more robust systems. For many, mobile devices have become the primary computing interface, which has a significant impact on usage. Much work has been done to study usage on workstations and laptops\textsuperscript{1,2,3,4}, but typical usage on smartphones can differ significantly\textsuperscript{5,6}. Smartphones are natural candidates for more context-aware applications, since they typically contain multiple sensors, such as GPS and light sensors, which can provide information about the user’s environment. Understanding how users interact with these devices can help developers optimize applications and system level components to improve performance and make devices more efficient. Research is needed to understand these new usage patterns, as well as the impact of the integration of these new usage patterns with mobile device hardware and software design. Tools are needed to support such research, to enable a rapid growth of understanding of improved design principles.

Beyond off-line modeling, a monitoring tool can have real time benefits to mobile devices as well, particularly with the increasing move towards context-aware applications and context-aware system activities. Monitoring in real time...
can allow the development of applications which dynamically adjust to the available resources of the system, or which adjust based on dynamically built models of user or application activity to improve performance or efficiency. Real-time monitoring can also support the development of context-aware applications. Developers may define a complex set of conditions which are being monitored which may be thought of as a context event. Providing easier methods of obtaining useful context information will allow more developers to incorporate context-aware capabilities in their applications.

Allowing each application developer to implement and integrate their own needed monitoring services can lead to duplication of activities, and inefficiencies in devices which have limited resources. A centralized monitoring solution for mobile devices can abstract the monitoring services. This would allow applications and services to request the information they need, and the central monitoring service can combine and integrate these requests in the most efficient manner possible. This would also improve the ease of development on the platform, as the abstracted services would not be device or version dependent.

This paper presents a framework for a centralized and configurable monitoring service for mobile devices called CIMON (Configurable Integrated Monitoring Service). Rather than focus on an individual aspect of performance or activity, such as system performance or user activity, CIMON integrates all observable activity on mobile devices. The configurable design supports development of applications which can measure any desired metric, or monitor for complex events. This framework is intended to support researchers designing experiments to model the complex interactions between different dimensions of activity on mobile devices, and developers seeking to incorporate real-time monitoring or context-aware functionality in systems and applications.

1.1. Related Work

Mobile applications are being developed which use sensor data from smartphones and user interaction with applications to model human behavior. Centralizing sensor management to support contextual applications in smartphones has been done in ContextDroid. Other works have looked at methods to improve sensing efficiency for a single application. CIMON attempts to provide an efficient, centralized abstraction of all measurable data on a device, rather than focus on a single source of context such as sensors.

With the obvious benefits of context-aware mobile designs, there have been many recent proposals for context-aware models, frameworks and middleware to support the development of context-aware applications. Some of these define frameworks for application design to utilize context information. Other models provide frameworks for determining context based on the data available. Henricksen and Indulska presented a process for modeling context as well as a software infrastructure for building context-aware applications with this model. The event notification features of CIMON may be used to facilitate these models.

2. Architecture

CIMON is designed as a centralized service which will serve as a layer of abstraction between the operating system and applications. It provides an easy-to-use API, which supports a comprehensive and configurable monitoring service. Details of the API follow this section. CIMON uses an efficient scheduling mechanism to minimize the impact of monitoring while handling requests from multiple applications. The system architecture is shown in Figure 1. Applications may register to receive periodic updates of a metric value, or an event notification, through the CIMON API. CIMON will integrate the new monitoring request with existing monitors in the Efficiency Engine to satisfy the request with the minimum additional resources necessary. The monitoring service will provide updates or notifications to the client applications through the callback handler that was provided by the client when registering the new monitor. Further detail of each component of the architecture is provided in the remainder of this section.

2.1. CIMON API

CIMON provides a simple and easy to interface API, which will allow developers and researchers to easily access the desired metrics from the system and still provide a comprehensive tool which allows them to customize the monitoring metrics to their specific needs. The CIMON API layer supports five primary types of requests: instantaneous,
periodic monitor, opportunistic monitor, event monitor, and conditional monitor. The syntax for each supported call is shown in Table 1. Though not shown in the table, each register command has a corresponding unregister command.

Table 1. API for CIMON

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>object synchronousRequest(int metric, long timeout)</td>
</tr>
<tr>
<td>int registerPeriodicMonitor(int metric, void *callback, long period, long duration)</td>
</tr>
<tr>
<td>int registerOpportunisticMonitor(int metric, void *callback, long period)</td>
</tr>
<tr>
<td>int registerEventMonitor(string condition, void *callback, long period)</td>
</tr>
<tr>
<td>int registerConditionalMonitor(int metric, string condition, void *callback, long period)</td>
</tr>
</tbody>
</table>

**Instantaneous monitor request** The instantaneous request, provided by `synchronousRequest`, is simply a request for the current value of a particular feature that can be monitored. For instance, this can be a request for the current load on the CPU.

**Periodic monitor request** For metrics that need to be periodically monitored, the developer may register an asynchronous monitor using `registerPeriodicMonitor`. CIMON merges new requests with existing requests to meet the monitoring needs of all applications with the minimum frequency possible. This returns an integer to the registering application representing the unique identification number of the monitor that was registered.

**Opportunistic monitor request** An opportunistic monitor is similar to a periodic monitor, except that it receives updates as frequently as they are available, even if they were generated for another monitor. This allows applications to potentially receive more frequent updates, without imposing an additional load on the system. Ideal for when frequent updates are preferred but not required.

**Event monitor request** The CIMON monitoring system can also be used to receive notification when a specific condition is met. These monitors will be referred to as event notifiers. These conditions may be absolute qualifiers, such as when the feature value rises above a certain threshold, or relative qualifiers, such as when the feature value changes by a certain threshold. The event notification service of CIMON can be a powerful tool. In addition to simple conditions, the `condition` argument may be a compound expression, using AND’s and OR’s. This allows the CIMON monitoring service to be a context notification system. This greatly simplifies the development of context-aware applications by allowing developers to register an event notifier with a compound condition which represents the contextual event of interest.

**Conditional monitor request** In some scenarios, developers may wish to monitor certain metrics under specific conditions. Conditional monitoring requests are a hybrid of periodic update requests and event notification requests, allowing developers to specify a periodic monitor to register, and the condition that the monitor should be active under.
This condition may be a compound expression, just as in an event notification request. A periodic monitor for the desired metric will be activated when, and only when, the condition specified is true. An additional benefit of this functionality is that it makes it easier for developers to incorporate resource-aware monitoring in their applications. For instance, developers may link periodic update requests with conditions to ensure that battery levels are above a minimum threshold, or that the load on the CPU is not too high.

2.2. Expression Module and Interpreter

The expression module provides a class which defines the format of condition expressions for event notification requests, as well as helper methods for construction of the expression string. Condition expressions are boolean expressions incorporating one or more conditions. Conditions may be combined using AND or OR statements to generate a more complex condition. The condition expressions are strings with a strict syntax to support efficient parsing by the expression interpreter. The syntax definition is defined in Table 2.

The expression interpreter parses condition expressions and generates a boolean expression tree consisting of three node types: AND, OR, and condition. All the nodes of the expression tree are generated using a recursive descent parser utilizing two stacks, one for terminals and another for nodes. At the completion of parsing, a handler for the root node of the expression tree representing the condition expression is passed to the event monitoring agent.

Cost information is also populated during the expression tree construction. This information is used by the Event Monitoring Agent to prioritize selection of which conditions are actively monitored to reduce costs. Cost here refers to the cost to the system, in particular, energy and processing resources. Each condition node that is constructed is populated with the cost for the metric of this particular node. During the construction of an OR node, the cost data is populated with the sum of the cost of the left and right subtrees of the node. For AND nodes, the cost data is populated with the lesser of the cost for the left and right subtrees. The reasons for this are due to the method of short-circuit analysis of the expression tree, which will be explained further in Section 2.3.

2.3. Event Monitoring Agent

The event monitoring agent manages the collection of event expression trees representing the active event notification and conditional monitoring requests. The agent manages the activation and deactivation of nodes within the trees, registering and unregistering the conditions with the efficiency engine as needed. Algorithm 1 shows the recursive

<table>
<thead>
<tr>
<th>Syntax / Terminals</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>METRIC</td>
<td>A metric supported by the CIMON monitoring system</td>
</tr>
<tr>
<td>CONDITION</td>
<td>A supported condition, such as maximum threshold</td>
</tr>
<tr>
<td>THRESHOLD</td>
<td>A threshold value for the metric specified. This may be an absolute value, or</td>
</tr>
<tr>
<td></td>
<td>a relative value, depending on the condition</td>
</tr>
<tr>
<td>EXPRESSION</td>
<td>Any valid expression: condition expression, AND/OR statement</td>
</tr>
<tr>
<td>(OR:EXPRESSION:EXPRESSION)</td>
<td>An OR statement, with two expressions, representing the left and right</td>
</tr>
<tr>
<td></td>
<td>parameters of the OR expression</td>
</tr>
<tr>
<td>(AND:EXPRESSION:EXPRESSION)</td>
<td>An AND statement, with two expressions, representing the left and right</td>
</tr>
<tr>
<td></td>
<td>parameters of the AND expression</td>
</tr>
<tr>
<td>[METRIC:CONDITION:THRESHOLD]</td>
<td>A condition expression</td>
</tr>
</tbody>
</table>
process that takes place when a new event notification requests is received, beginning with the root node. It also handles notifications from the efficiency engine signifying the trigger of a condition. Upon each trigger, it updates the state of the condition, and re-evaluates the overall expression state.

The expression trees are evaluated using short-circuit analysis in order to reduce the impact of monitoring on the system. Short-circuit analysis can be leveraged to reduce the number of conditions which must be monitored simultaneously. In any AND expression, only one side of the expression must be monitored at a time. This allows the system to save the resources necessary for monitoring the other half of the expression until it is necessary to do so. For OR expressions, both conditions must be monitored initially, but once one condition has triggered, the other may be removed to free those resources. Due to the expression tree construction process described in Section 2.2, cost data is populated in each node. For interior nodes, this cost represents the cost of the subtree rooted at that node. The event monitoring agent can use this information to determine which condition or subtree to activate first. Using the short-circuit analysis scheme, the event monitoring agent manages which conditions within the expression tree should be active at any given time.

When the event monitoring agent receives a notification from the efficiency engine, signifying the trigger of a condition, it sets the state of the condition node to \textit{true}. In order to ensure that the condition remains true while the other conditions of the tree are being monitored, the anti-condition is registered with the efficiency engine. It then calls the trigger method of its parent so that the parent can update its state as needed. This process propagates up the tree until a node that does not change its state to true is reached. If the root is reached, and its state is set to true, the agent will perform the necessary action. For an event notification request, this will issue an event notification to the client that registered the request. For a conditional monitoring request, this will register a periodic monitor of the specified metric with the efficiency engine.

2.4. Efficiency Engine

One of the primary goals of CIMON is to centralize all monitoring services to provide a more globally efficient system, and reduce inefficiencies presented by redundancies. To do this, the monitoring service must be capable of seamlessly integrating the monitoring requests of all applications in order to meet all of their needs with the minimum total resources used. It is also important that the monitoring system itself is lightweight and computationally efficient. The architecture of the Efficiency Engine is shown in Figure 2, and a description of each component is provided below.

2.4.1. Metric Manager

The Metric Manager is the entry point for all requests to the Efficiency Engine. It manages the list of all Metric Monitors, and routes all periodic update and condition monitor requests to the appropriate Metric Monitor. It organizes the Metric Monitors into groups where appropriate, and manages the update frequency for these consolidated groups, as well as the allocation of resources. Some metrics may share the resources needed for updates. For instance,
there are multiple metrics which may be obtained for the state of memory, such as free memory, cached memory, dirty pages, etc. The values for these metrics are all obtained through the same system call, so it is more efficient to consolidate these metrics to update them all with a single system call. The Metric Manager handles this consolidation, and coordinates the updates between the metrics. It also frees the resources of a Metric Monitor when it has no active requests.

2.4.2. Metric Monitor

All monitoring requests fall into one of three categories: notification of metric falling below threshold, notification of metric rising above threshold, and value of metric at specified time. Each feature supported by CIMON should only be monitored when there is an active request for that feature, and only at the minimum rate necessary to meet the needs of all requests. For each monitoring feature, there will be a central node which defines the current state of that metric, as shown in Figure 3. It will contain the current cached value of this metric, and the timestamp when the value was acquired. Each central node will branch to four sorted linked-lists: a minimum-threshold list, a maximum-threshold list, a periodic update list, and an opportunistic update list. The minimum and maximum threshold linked-lists will have the same data structure, their only difference is that one will be sorted in ascending order, according to threshold value, and the other in descending order, with the value nearest to the central node value first in each. The key-value for the periodic update list is a system time, when the next notification should occur. This list is sorted with the earliest deadline first. The key-value for the opportunistic update list is the maximum allowable period for the monitor. This list is sorted with the smallest maximum allowable period listed first.

Threshold lists

The minimum and maximum threshold lists maintain the ongoing requests for notification of any conditions, representing requests registered using registerEventMonitor(). The key value for each node will be the threshold value for the condition. When an update to a monitored metric occurs, the service will compare the new value to the head of each list. For simplicity of explanation, we will consider the case when the new value is greater than the previous value. Consider a monitor of the battery percentage level. The current level is 55, so this is the value of our central node. A new value of 60 arrives, the value and timestamp of the central node are updated to reflect the new value. The monitoring service will begin with the first node of the maximum threshold list. If its key value is greater than 60, no further action will be taken until the next update. If the key value of the first node is less than or equal to 60, it will pop this node from the list and send it to the Notification Agent. The monitoring service will repeat this step until it reaches a node with a key value greater than 60, or until the list has been emptied. The process is the same for the opposite list if the new value is less than the current value. Structurally, the two threshold lists can be thought of as one linked-list, where the central node acts as a divider that moves up and down the list as its value changes. A visualization of this can be seen in Figure 3.

Periodic update list

The periodic update list works similarly, except instead of threshold values, it uses deadline times, which represent the time for the next notification. This list provides notifications for monitoring requests registered using registerPeriodicMonitor(). When a new value for the metric arrives, it goes sequentially through the
periodic update list, popping off any nodes which have a deadline less than the current timestamp and sending them to the Notification Agent. If these nodes have not reached the end of their duration, their key will be revised to the next update time, and they will be reinserted into the list based on the time for the next update.

**Opportunistic update list** The opportunistic update list is very similar to the periodic update list, except all monitors in this list are updated each time an update is available. The key value for each node in this list is the maximum allowable interval between updates for that monitor. This list provides notifications for monitoring requests registered using `registerOpportunisticMonitor()`. When a new value for the metric arrives, it goes sequentially through the opportunistic update list, sending each to the Notification Agent. Monitors which have reached the end of their duration will be removed during this sweep of the list.

This structure is used for all features supported by CIMON, so each feature will have its own central node which checks its four branches on each update. This structure allows CIMON to maintain all required monitoring requests with a minimal amount of memory space, and limited computational overhead. For each update of a metric, it only needs to evaluate the first node in two linked-lists along with any nodes present in the opportunistic update list. This structure also allows the system to easily determine what monitoring services should be active, and at what rate. If all four branches of a central node are empty, then there are no active requests for this feature and monitoring for this feature can be shut off, freeing any related resources. Otherwise, a new update can be scheduled at a time which is equal to the minimum of the next periodic request deadline, the smallest allowable opportunistic update interval, and the minimum threshold request period. This provides a simple algorithm for determining the most efficient rate which meets all request deadlines. This update interval is provided to the Metric Manager, which consolidates the update request with other metrics, and coordinates with the Update Engine to schedule the update.

### 2.4.3. Update Engine and Notification Agent

The Update Engine coordinates with the Metric Manager to schedule updates for all active metrics. The method of update is metric-dependent. Some updates may not be strictly scheduled. For instance, most sensor readings on an Android device can only be requested asynchronously, and therefore can not guarantee an update at a specific time. In such cases, the Update Engine will initiate the update request at a time which provides the closest approximation of the desired update time, and will issue the update once available. The Update Engine provides the updated values directly to the Metric Monitors. The Notification Agent receives nodes from the Metric Monitor when they have triggered, or received a periodic update. Periodic update nodes contain a handler for the callback function of the client that issued the request. The agent will issue a callback, passing the metric and its updated value as parameters. For threshold list nodes, the agent will notify the Event Monitoring Agent of the condition trigger.

### 3. Evaluation

For evaluation of our monitoring framework, we implemented CIMON on the Android mobile operating system with a range of initial metrics. Due to space constraints, we do not include details of the implementation here, but more details along with source code, executables, measurements, analysis and initial results can be found on the project website at sourceforge.net/p/cimon. An example of initial results is shown in Figure 4. To evaluate the effectiveness of the Efficiency Engine, a periodic monitor of CPU utilization is performed by an increasing number

![CPU utilization for increasing number of periodic monitors](image1)

![Energy usage for increasing number of periodic monitors](image2)

**Fig. 4.** CPU utilization and energy usage measured during tests using a periodic monitor of CPU utilization at 25 Hz. Error bars represent 95% confidence interval.
of applications on the device. Two sets of tests are performed. In one, the CIMON implementation handles the monitoring requests. In the other, each application performs the monitoring individually. For each test, the CPU utilization and energy usage of the Android device is measured. As can be seen in these figures, CIMON efficiently integrates the monitoring requests, experiencing minimal additional costs as the number of requests increases from 1 to 10. For the standalone implementation, the resource usage increases linearly with the number of applications performing monitoring.

4. Future Work

This monitoring service integrates three dimensions of metrics observable on mobile devices: physical, system, and user activity. This integration could lead to a better understanding of the complicated interdependencies of multiple aspects of system design and user-device interaction. It can also provide improved context-awareness, by integrating multiple features which provide a more complete picture of the current context. CIMON is designed to be a lightweight monitoring service, suitable for mobile devices. Due to the efficient integration of monitors, CIMON may be used as a centralized monitoring service which abstracts monitoring activities from user-level applications, allowing multiple applications to actively monitor on the device without excessive resource usage due to redundancies. Future work will include the implementation of additional metrics in our prototype, and further measurement and evaluation of the efficiency and overhead of CIMON. We feel that this work will help support research which improves system and application design, and support developers in producing novel and context-aware applications.

References