Profiling von Software-Energieverbrauch

Michael Fiedler
Friedrich-Alexander-Universität Erlangen-Nürnberg
michael.fiedler@informatik.stud.uni-erlangen.de

KURZZUSAMMENFASSUNG

Energieverbrauch ist ein wichtiger Aspekt beim Entwurf heutiger Hard- und Software. Gerade im Bereich von mobilen Geräten, bei denen die Laufzeit durch die Batteriekapazität stark beschränkt ist, gleichzeitig aber Ansprüche an die Rechenleistung und Datenübertragung zunehmen, kommt einer möglichst geringen Leistungsaufnahme eine wesentliche Rolle zu. Das Profiling des Energieverbrauchs von Software, also die Zuordnung von Energieverbrauch zu denjenigen Programmstrukturen, die auf seine Höhe Einfluss haben, kann Entwickler auf die energiehungrigen Problemstellen seiner Programme hinweisen, sodass diese gezielt verbessert werden können. Diese Arbeit gibt hierzu einen Einblick in Konzepte und Probleme aus dem Bereich des Profilings von Software-Energieverbrauch. Zusätzlich werden mehrere Umsetzungen von Profilern genauer betrachtet.

1. EINLEITUNG

Bei heutigen Rechnern spielt neben Geschwindigkeit auch der Energieverbrauch eine wichtige Rolle. Dies ist zum Einen durch eine im Alltagsleben zu beobachtende zunehmende Verbreitung von mobilen Geräten wie Notebooks, Smartphones und Tablet-Computern bedingt, bei denen die Gerätelaufzeit durch die verfügbare Batteriekapazität beschränkt ist. Auch bei batteriebetriebenen Sensorknoten in Sensornetzwerken [1] existiert diese Problematik. Darüber hinaus ist ein möglichst geringer Energieverbrauch auch bei größeren Rechnern oder Rechenanlagen ein Thema: Hier sind die Senkung von Stromkosten, der Umweltschutz (beispielsweise aus Marketing-, Image- oder rechtlichen Gründen) und das Abführen der durch den Energieverbrauch entstehenden Wärme anzuführen. Nicht nur die Hardware, sondern auch die Software trägt maßgeblich dazu bei, dem Ziel möglichst energieeffizienter Systeme näher zu kommen – gerade auf der Software-Ebene besteht ein besserer Überblick über das Gesamtsystem, der für Energieverbrauchsoptimierungen genutzt werden kann.

Um energieeffiziente Software zu bauen, benötigen Entwickler geeignete Werkzeuge, die sie bei ihrer Aufgabe unterstützen. Diese sollen diejenigen Stellen im System mit Verbesserungsmöglichkeiten bezüglich des Energieverbrauchs aufzeigen, gegebenenfalls vorhandene Fehler in der Energieverwaltung (zum Beispiel sogenannte *Wake-Lock-Fehler* in Smartphone-Anwendungen [2]) entdecken und nach Änderungen am Quelltext die Auswirkungen auf den Energieverbrauch beziffern. Dafür ist es notwendig, den Energieverbrauch anteilig den ihn verursachenden Hard- und Software-Komponenten auf verschiedenen Genauigkeitsebenen (Prozedur/Funktion, Prozess, einzelnes Gerät, Teilsystem des Betriebssystem, ...) zuordnen zu können – nicht nur, um ein bereits umgesetztes System bei seiner Ausführung zu beobachten, sondern auch, um noch nicht im Einsatz befindliche Systeme wie einzelne Sensorknoten in ihrem Energieverbrauchsverhalten und damit in ihrer Laufzeit abschätzen zu können [1].

Der beschriebenen Aufgabe widmet sich das Profiling von Software-Energieverbrauch mit der Leitfrage "An welcher Stelle meiner Software wird wie viel (qualitativ/quantitativ) Energie verbraucht?". Während im Bereich des Profilings für die Rechenleistung des Prozessors schon seit vielen Jahren Programme wie Gprof [3] etabliert sind, ist dies beim Energieverbrauchsverhalten noch nicht der Fall. Diese Ausarbeitung soll daher einen Einblick in Problemstellungen und Lösungsansätze aus Veröffentlichungen zum Thema "Profiling von Software-Energieverbrauch" bieten.

Die allgemeine Vorgehensweise beim Profiling von Software-Energieverbrauch besteht dabei aus folgende Schritten:

1. *Durchlaufen des Programms*: Dies kann durch Ausführen des untersuchten Programms in einer für Energiemessungen instrumentierten realen [4] oder simulierten [1] Umgebung durchgeführt werden. Auch die zusätzliche Verwendung symbolischer Ausführung [5] für eine maximale Programmpfadüberdeckung ist dabei möglich [6].

2. *Ermitteln des bei Ausführung benötigten Energieverbrauchs*: Dies kann beispielsweise durch eine tatsächliche Messung mit Hilfe eines Multimeters [4], durch Schätzung basierend auf einem Modell [1] oder unter Zuhilfenahme einer Ersatzmetrik und eines für diese ermittelten Hardware-spezifischen Energieverbrauchsprofils [6] geschehen.

3. *Herstellen der Verbindung zwischen dem aktuellen Energieverbrauch und den Programmstrukturen, die darauf Einfluss haben*: Eine Möglichkeit dafür ist die Analyse des Stapelspeichers während der Programmausführung [4, 7].

Als problematisch beim dritten Punkt erweist sich dabei der *asynchrone Energieverbrauch* [7]. Im Gegensatz zum synchronen Fall, bei dem der Energieverbrauch zeitgleich mit der Ausführung des ihn verursachenden Programmbestandteils auftritt (zum Beispiel beim Energieverbrauch durch das Ausführen von Maschinenbefehlen durch den Prozessor), tritt asynchroner Energieverbrauch verzögert zum

Verursachungszeitpunkt auf (zum Beispiel bei Ein- und Ausgabeoperationen mit Pufferspeichern, wo der eigentliche Gerätezugriff erst nach mehreren Ein- und Ausgabeoperationen stattfindet). Auch dieser zeitlich verzögerte Energieverbrauch muss also im Profiling berücksichtigt werden, um korrekte Ergebnisse zu erhalten. Insbesondere ist er bei der Betrachtung von Anwendungssoftware für mobile Geräte zu beachten, wo asynchroner Energieverbrauch eine wichtige Rolle spielt [2].

Diese Arbeit ist wie folgt aufgebaut: In Abschnitt 2 werden Entwurfsaspekte von Software-Energieverbrauch-Profilers diskutiert und eine Übersicht über Problemstellungen und mögliche Lösungsansätze gegeben. Danach wird in Abschnitt 3 auf mehrere Arbeiten zum Profiling genauer eingegangen. Abschließend folgt eine Zusammenfassung der wichtigsten Aspekte dieser Arbeit.

2. ENTWURFSASPEKTE FÜR PROFILER

In diesem Abschnitt werden diverse Konzepte betrachtet, mit deren Hilfe Profiler für Software-Energieverbrauch in der Forschung der letzten Jahre umgesetzt wurden. So unterscheiden sich die Arbeiten unter Anderem in folgender Hinsicht:

- *Messung des Energieverbrauchs*: Die Messung kann zum Beispiel per Hardware-Instrumentierung oder per Schätzung mit Hilfe von Ersatzmetriken erfolgen.
- *Granularität*: Die Zuordnung des Energieverbrauchs kann unter Anderem auf der Ebene von Prozessen oder von Funktionen stattfinden.
- *Grad der Programmpfadabdeckung*: Wird bei der Erstellung des Energieverbrauchsprofils nur ein konkret ausgeführter Programmpfad berücksichtigt oder wird angestrebt, die Zahl der überprüften möglichen Pfade zu maximieren?
- *Vorgehensweise zur Verknüpfung von Enerieverbrauch und Programmstrukturen*
- *asynchrones Energieverbrauchsverhalten*: Wird die Energie nur der jeweils in Ausführung befindlichen Programmstelle zugeordnet, oder werden verzögerte Geräteaufrufe (zum Beispiel bedingt durch Pufferspeichereffekte) berücksichtigt? Und wie wird die Zuordnung umgesetzt?

2.1 Messung des Energieverbrauchs

Um eine korrekte Zuordnung zwischen dem Energieverbrauch bei der Ausführung einer Software und den ihn verursachenden Programmstrukturen herstellen zu können, ist eine ständige Verfolgung des Energieverbrauchs notwendig. Da heutige Rechner normalerweise nicht in hinreichendem Maße mit Sensoren ausgestattet sind, über die per Software die aktuellen Energieverbrauchswerte der einzelnen Geräte ausgelesen werden können, sind alternative Methoden erforderlich.

Eine Messung kann daher entweder durch direkte Instrumentierung des Rechners geschehen, auf dem das Programm ausgeführt wird, oder aber man bedient sich eines Modells bzw. einer Ersatzmetrik, über die der reale Energieverbrauch indirekt abgeschätzt wird. Der Vorteil der letzteren Variante ist das Wegfallen des Aufwands dafür, die physische Messinfrastruktur bei der Ausführung der untersuchten Software anzubringen. So können Geräte auch emuliert werden oder das Profiling erfolgt während der Programmausführung gleich auf dem untersuchten Rechner.

2.1.1 Direkte Instrumentierung der Hardware mit Messgeräten

Flinn et al. verwenden für ihr Profiling-Werkzeug PowerScope [4] ein Digitalmultimeter, mit dem sie den Stromverbrauch des gesamten untersuchten Systems verfolgen. Die durch Abtastung erzeugten Messwerte werden auf einem separaten Rechner gesammelt und später mit den zeitgleich erhobenen Daten über die aktuell ausgeführte Programmstelle verbunden (siehe dazu auch Abbildung 4 unten). Ein externes Messgerät wurde hierbei einem in den untersuchten Rechner eingebauten vorgezogen, um eine Beeinflussung der Messergebnisse durch die Messung selbst zu verhindern.

2.1.2 Modellierung des Energieverbrauchs der einzelnen Rechnerbestandteile

Als Alternative zu einem Profiling mit direkter Instrumentierung des verwendeten Rechners kann der Energieverbrauch auch über Energieverbrauchsmodelle geschätzt werden [1, 8, 2, 7]. Dabei wird zum Beispiel die Leistungsaufnahme verschiedener Zustände eines Geräts ermittelt. Bei der Ausführung des untersuchten Programms wird dann von den aktuellen Zuständen aller Geräte des Systems auf den momentanen Energieverbrauch geschlossen. Vor ihrer Verwendung können diese Modelle auf ihre Anwendbarkeit, Korrektheit und Genauigkeit hin überprüft werden – hier wiederum mit Hilfe von realen Stromverbrauchsmessungen an einer Referenzhardware. Für das eigentliche Software-Profiling wird dann das validierte Energieverbrauchsmodell angewandt.

In der Literatur finden sich verschiedene Vorgehensweisen, darunter die folgenden:

- *Modell für die Emulation von Hardware und Ausführungsumgebung*: Bei AEON [1] wird für die Analyse eines Knotens in einem Sensornetzwerk die in den Sensorknoten verwendete Hardware emuliert. Zunächst wird ein Modell für den Energieverbrauch der einzelnen Geräte in ihren verschiedenen Gerätezuständen erstellt. Anschließend wird die untersuchte Software mit Hilfe eines Emulators der Hardware ausgeführt. Dabei kann über die Zustandswechsel der Geräte und die im Modell enthaltene Leistungsaufnahme der einzelnen Geräte in den jeweiligen Gerätezuständen der Energieverbrauch des Gesamtsystems ermittelt werden.
- *Modell für das Nachverfolgen von Systemaufrufen* (engl. *system call tracing*): Bei Eprof von Pathak et al. [8, 2] hingegen wird ein Modell verwendet, das die bei der Programmausführung auftretenden Systemaufrufe betrachtet. Da über die Systemaufrufe zum Beispiel Ein- und Ausgabeoperationen abgehandelt werden, werden die Systemaufrufe als Auslöser für Zustandsübergänge der Geräte betrachtet. Über die im Energieverbrauchsmodell enthaltenen Verbrauchswerte der Gerätezustände wird ein Wert für den Gesamtenergieverbrauch ermittelt.

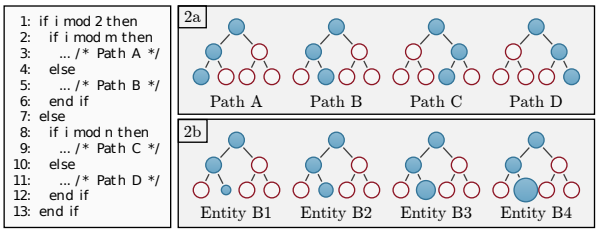

Abbildung 1: Programmpfade und Programmeinheiten bei SEEP. Die Größe der blauen Knoten der Graphen stellt den Energieverbrauch des jeweiligen Programmabschnitts schematisch dar. Quelle: [6]

2.2 Granularität der Zuordnung des Energieverbrauchs zu Programmstrukturen

Um Software-Programmierern einen möglichst guten Überblick über energieaufwändige Bereiche ihres Programms zu ermöglichen, betrachten die in dieser Arbeit vorgestellten Profiling-Konzepte die Programmstruktur-Energieverbrauch-Zuordnung auf der Ebene von Funktionen (und auch Prozessen, bei [2] werden zusätzlich Threads und Systemaufrufe genannt). Auch auf höheren Abstraktionsebenen wie bei der Betrachtung von ganzen Rechnern, Sensornetzwerken [9] oder im Cloud-Computing-Bereich [10] existieren Ansätze zum Profiling. Auf diese wird jedoch in dieser Arbeit nicht weiter eingegangen.

2.3 Programmpfadabdeckung

Die Ausführung von Software mit bestimmten Aufrufparametern und unter bestimmten Umgebungsbedingungen führt zu einem Programmdurchlauf, der nur einen von vielen möglichen darstellt (siehe dazu Abbildung 1). Der resultierende Ausführungspfad durch das Programm muss dann jedoch nicht repräsentativ für das Energieverbrauchsverhalten weiterer möglicher Ausführungspfade sein. Hinzu kommt, dass selbst innerhalb eines Ausführungspfads das Energieverbrauchsverhalten von der konkreten Eingabe und Umgebung abhängt. Das durch einen einzelnen Profiling-Durchlauf gelieferte Energieverbrauchsprofil kann also nur bedingt Aussagen über das Gesamtverhalten der untersuchten Software liefern. Idealerweise sollte das Profil also einen möglichst großen und repräsentativen Anteil der Wege durch ein Programm berücksichtigen.

Eine Möglichkeit, die Überdeckung der Ausführungspfade zu erhöhen, ist die Zuhilfenahme symbolischer Ausführung [5] von Programmen. Mit Hilfe dieser Technik werden bei SEEP [6] Eingabewerte für eine möglichst große Programmpfadüberdeckung gefunden. Anschließend werden für jeden Programmpfad für mehrere Eingabedaten Programmeinheiten erstellt. Diese werden dann mit dem Profiler untersucht und die Ergebnisse können für den Programmpfad zusammengefasst werden.

2.4 Verknüpfung von Energieverbrauch und Programmstrukturen

Um Energieverbrauchsprofile für Software zu erstellen, ist es erforderlich, den Zusammenhang zwischen dem Maschinencode des untersuchten Programms, der im Prozessor ausgeführt wird, und der zugehörigen Programmstruktur auf Programmiersprachenebene herzustellen. Erst dann kann der während der Ausführung der Maschinenbefehle festgestellte Energieverbrauch zu denjenigen Programmbestandteilen wie einzelnen Funktionen zugeordnet werden, die für den Programmierer interessant sind. Die Zuordnung von Programmzähler zur zugehörigen Funktion ist hierbei über die Symboltabelle der ausgeführten Programmdatei bzw. der verwendeten Programmbibliotheken möglich. Dazu finden sich folgende zwei Möglichkeiten, um vom Programmzähler über die Symboltabelle der ausgeführten Programmdateien bzw. der verwendeten Programmbibliotheken an die zugehörigen:

- *Abtasten der Werte des Programmzählers*: Bei PowerScope [4] wird regelmäßig der aktuelle Wert des Programmzählers des Prozessors erfasst, sodass er später über die Symboltabelle des ausgeführten Programms zu einer Funktion im Programmquelltext zugeordnet werden kann.
- *Nachverfolgen der Funktionsaufrufe*: Eprof von Schubert et al. [7] und Eprof von Pathak et al. [8, 2] betrachten die Programmaufruffolge auf dem Stapelspeicher, bei AEON [1] wird jeder Funktionsaufruf registriert.

2.5 Asynchrones Energieverbrauchsverhalten

Bei der Zuordnung von gemessenem Energieverbrauch zu Software-Programmstrukturen sind zwei verschiedene Fälle für den zeitlichen Zusammenhang zwischen der Ausführung eines Programmbefehls bzw. einer Programmstruktur und dem daraus resultierenden Energieverbrauch zu unterscheiden:

1. Der Energieverbrauch findet zeitgleich mit der Ausführung statt (*synchroner Energieverbrauch*).
2. Der Energieverbrauch findet zeitlich verzögert zur Ausführung oder auf einen längeren Zeitraum verteilt statt (*asynchroner Energieverbrauch*).

Im synchronen Fall kann der Energieverbrauch einfach mit den zum Messzeitpunkt ausgeführten Maschinenbefehl und damit der zugehörigen Programmeinheit verknüpft werden – es müssen also zum Beispiel nur der Energieverbrauch und der Programmzähler oder die Reihenfolge der Funktionsaufrufe auf dem Stapelspeicher gleichzeitig abgegriffen werden. Der asynchrone Fall ist komplizierter zu handhaben. Im Folgenden wird zunächst genauer betrachtet, in welchen Fällen Asynchronität auftritt [2]. Anschließend wird auf mögliche Vorgehensweisen eingegangen, mit deren Hilfe zeitverzögert auftretender Energieverbrauch zu Programmstrukturen zugewiesen werden kann [2].

2.5.1 *Ursachen für und Problemstellungen bei asynchronem Energieverbrauchsverhalten*

Asynchrones Energieverbrauchsverhalten kann verschiedene Ursachen haben, auf die in diesem Abschnitt eingegangen wird. Neben den verschiedenen Betriebszuständen von Geräten und das Beibehalten eines relativ viel Energie benötigenden Aktivitätsbereitschaftszustands nach einem Gerätezugriff sind gerade in heutigen Mobiltelefonen auch Wachzustandssperren (engl. *wake locks*) sowie Spezialgeräte wie GPS-Empfänger von Bedeutung.

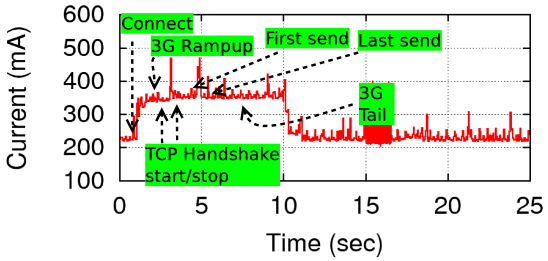

Abbildung 2: Stromverbrauch eines Mobiltelefons beim Senden von 5 Dateneinheiten zu je 10 kB über das Mobilfunknetz (3G) mit den *Sendebefehlen direkt nach dem Verbindungsaufbau.* Quelle: [2]

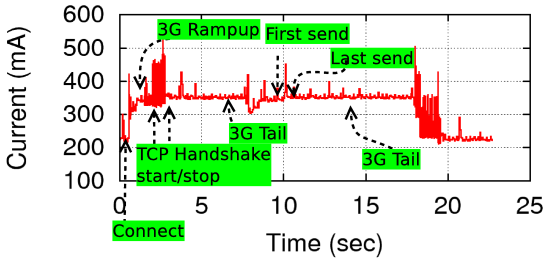

Abbildung 3: Stromverbrauch eines Mobiltelefons beim Senden von 5 Dateneinheiten zu je 10 kB über das Mobilfunknetz (3G) mit den *Sendebefehlen 5 Sekunden nach dem Verbindungsaufbau.* Quelle: [2]

Die in heutigen Rechnern verbauten Geräte haben verschiedene Betriebszustände, die dabei helfen, ihren Energieverbrauch gering zu halten. Neben einem Grundzustand, in dem keine bzw. möglichst wenig Energie verbraucht wird, kann es mehrere Zustände für Aktivitätsphasen geben [2]. Darunter kann zum Beispiel bei Netzwerkgeräten wie WLAN- und Mobilfunkgeräten ein "Schweifzustand" (engl. *tail state*) sein, von dem aus schnell in einen Sendezustand gewechselt werden kann, aber in dem im Vergleich zum Sendezustand weniger Energie verbraucht wird. In diesem Schweifzustand kann jedoch deutlich mehr Energie als im Grundzustand benötigt werden. Wird auf das Gerät zugegriffen, so muss vom aktuellen Gerätezustand in den Sendezustand gewechselt werden. Anschließend geht das Gerät in den Schweifzustand über und verweilt dort eine Zeit lang.

Der Schweifzustand erweist sich für die Zuordnung des aktuellen Energieverbrauchs zu denjenigen Programmeinheiten, die ihn verursachen, als Problem. Im Gegensatz Geräten, die ein synchrones Energieverbrauchsverhalten aufweisen, wie Prozessoren bei der Ausführung von Maschinenbefehlen, wirkt die Geräteaktivität durch das Verweilen im Energie verbrauchenden Schweifzustand noch nach: Dessen Ursache hat nichts mit der gerade ausgeführten Programmanweisung zu tun, sondern mit einem Gerätezugriff aus einer anderen Programmanweisung, die irgendwann in der Vergangenheit ausgeführt wurde.

Eine Illustration der Problematik findet sich in den Abbildungen 2 und 3. Sie zeigen den Verlauf des Stromverbrauchs eines Mobiltelefons über das Mobilfunknetzwerk (3G) beim Übertragen von fünf Dateneinheiten zu je 10 kB mittels fünf Sendebefehlen. Nach dem Aufbau einer TCP-Verbindung werden die Daten gesendet. Im ersten Fall (Abbildung 2) folgt der Sendebefehl für die Datenpakte zeitlich direkt auf den Verbindungsaufbau. Im Anschluss an den letzten Sendebefehl ist zu beobachten, dass über mehrere Sekunden hinweg der Stromverbrauch auf hohem Niveau verweilt (Schweifzustands des Mobilfunkgeräts). Erst dann kehrt er mit einem Sprung wieder auf sein Ausgangsniveau zurück. Gibt es nun zusätzlich eine Verzögerung zwischen dem Verbindungsaufbau und dem Senden der Datenpakete (Abbildung 3), so verlängert sich die Zeit im Schweifzustand, in dem ein im Vergleich zum Grundzustand erhöhter Stromverbrauch auftritt, zusätzlich. Um den Stromverbrauch auf einzelne Programmbestandteile aufzuteilen, ist daher eine passende Strategie erforderlich, in die die Zustandsübergänge der einzelnen Geräte einfließen und mit deren Hilfe die verbrauchte Energie auf die beteiligten Funktionsaufrufe aufgeteilt wird. Auf mögliche Vorgehensweisen wird in Abschnitt 2.5.2 weiter eingegangen.

Ähnliche Effekte wie der Schweifzustand von Geräten können auch in anderen Fällen auftreten, in denen Geräte nicht unmittelbar nach dem Ende ihrer Aktivität auf den energiesparsamsten Zustand wechseln [2]. In aktuellen Betriebssystemen für mobile Geräte gibt es etwa Programmierschnittstellen, die es Anwendungsentwicklern ermöglichen, eine Wachzustandssperre (engl. *wake locks*) zu setzen, bis sie von ihnen explizit wieder gelöst wird. Profiler, die sich dieser Wachzustandssperren bewusst sind, können daher auch Fehler in der Benutzung dieser Programmierschnittstellen (engl. *wake lock bugs*) aufdecken. Darüber hinaus gibt es Geräte wie GPS-Empfänger und Kameras, die nach ihrer Aktivierung über einen längeren Zeitraum in einem aktiven, energieverbrauchenden Zustand verweilen können und in ähnlicher Weise von Profilern berücksichtigt werden sollten.

2.5.2 *Zuordnungsstrategien bei asynchronem Energieverbrauchsverhalten*

Pathak et al. [2] gehen auf mehrere Möglichkeiten ein, asynchronen Energieverbrauch zu den ihn auslösenden Programmeinheiten zuzuordnen:

1. gleichmäßige oder gewichtete Aufteilung auf alle Programmeinheiten
2. Zuweisung zum ersten oder letzten Programmbestandteil, der das Verweilen auf einem energieverbrauchenden Gerätezustand zur Folge gehabt hätte

Problematisch bei ersten Punkt erweist sich demnach zunächst das Festlegen und die Umsetzung der Gewichtung, die das resultierende Energieverbrauchsprofil schwerer verständlich machen. Zusätzlich kommt hinzu, dass dem Benutzer des Profilers durch solche Zuordnungen suggeriert wird, dass das Entfernen einer Programmeinheit mit einer starken Gewichtung im ersten Fall bzw. im zweiten Fall der Programmeinheit, die die gesamten asynchronen Energieverbrauch zugewiesen bekommen hat, den Energieverbrauch seiner Anwendung entsprechend absenkt. Entgegen dessen wird sich der Energieverbrauch dann jedoch zum Großteil auf andere Programmeinheiten verteilen. Um eine intuitive Verwendung des Energieverbrauchsprofilers zu ermöglichen, kommen Pathak et al. [2] daher zum Schluss, dass in jedem Fall ein expliziter Vermerk im Energieverbrauchsprofil

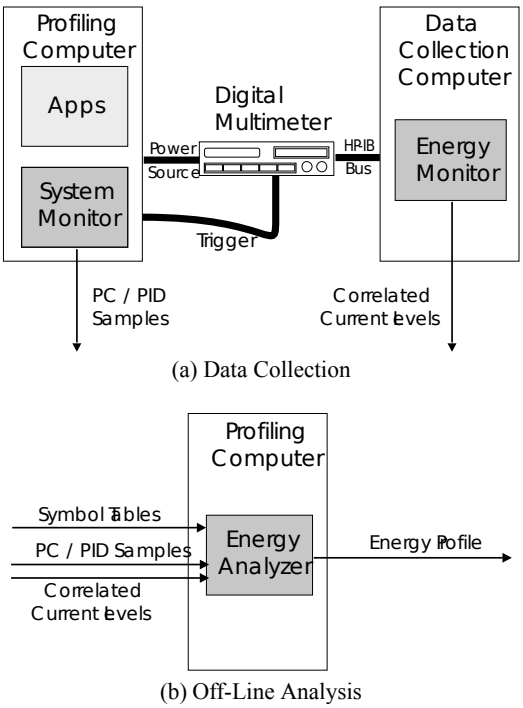

Abbildung 4: Überblick über PowerScope. (a) Abtastung von Werten für aktuellen Stromverbrauch, Programmzähler und Prozess-ID des untersuchten Rechners. (b) Zusammenführen der erhaltenen Werte mit der Symboltabelle des untersuchten Programms zum Energieverbrauchsprofil. Quelle: [4]

notwendig ist, der den Umgang mit asynchronem Energieverbrauch für den Benutzer ersichtlich macht, zum Beispiel eine Angabe des Energieverbrauchs eines Systemaufrufs mit einem Tupel (Energieverbrauch des eigentlichen Systemaufrufs, Energieverbrauch des nachfolgenden Schweifzustands).

3. BEISPIELE FÜR UMSETZUNGEN VON ENERGIEVERBRAUCH-PROFILERN

In den letzten anderthalb Jahrzehnten wurden Arbeiten aus verschiedenen Forschungsprojekten veröffentlicht, die Umsetzungen von Profilern für Software-Energieverbrauch zum Thema haben. In diesem Abschnitt werden drei der Projekte näher vorgestellt:

- *PowerScope* (Flinn et al., 1999) [4]: Profiling mittels Abtastung von Wertpaaren aus dem Programmzähler im Prozessor und einem Stromverbrauchsmesswert des untersuchten Rechners
- *AEON* (Landsiedel et al., 2005) [1]: Profiling mittels Energieverbrauchsmodellierung der Gerätezustände eines Sensorknotens und Ausführen auf simulierter Hardware
- *Eprof* (Pathak et al., 2011, 2012) [8, 2]: Profiling mittels Nachverfolgen von Funktions- und Systemaufrufen und Modellierung der Gerätezustände von Smartphones

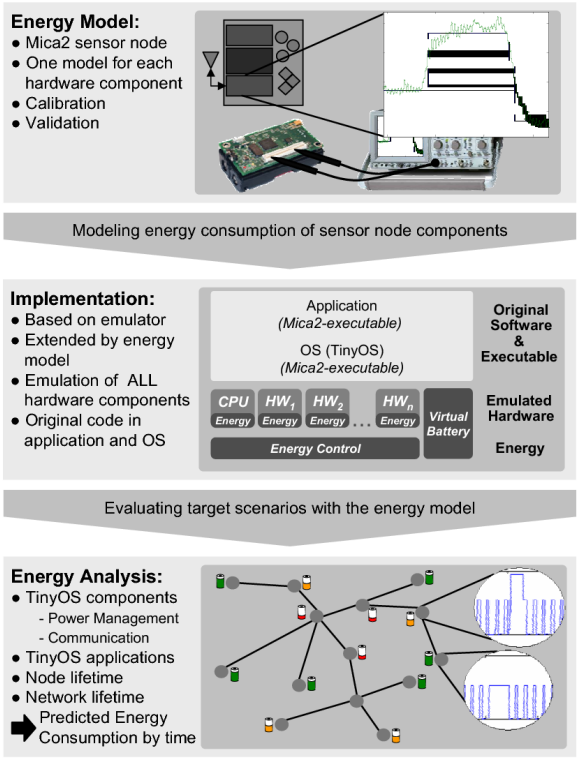

Abbildung 5: Überblick über AEON. Quelle: [1]

3.1 PowerScope

PowerScope [4] führt sein Profiling in zwei Schritten durch: In der ersten Phase (Abbildung 4a) wird eine Abtastung[1] von Wertpaaren bestehend aus dem aktuellen Programmzähler des Prozessors und der aktuellen Prozess-ID auf dem Rechner, auf dem das untersuchte Programm läuft (Profiling-Rechner), durchgeführt. Die Abtastung wird dabei durch ein digitales Multimeter ausgelöst, das gleichzeitig den aktuellen Stromverbrauch des gesamten Profiling-Rechners erfasst und an einen zweiten Rechner (Datensammlungsrechner) weiterleitet. Auf dem unteruchten Rechner läuft für das Erfassen der Werte ein angepasstes Betriebssystem, das zusätzlich für jeden Prozess den Pfad zur ausgeführten Programmdatei und bei jedem Laden von Programmbibliotheken auch deren Pfad erfasst.

In der zweiten Phase (Abbildung 4b) werden die erhaltenen Abtastwerte auf dem untersuchten Rechner mit den Symboltabellen der ausgeführten Programme und verwendeten Programmbibliotheken verknüpft und Energieverbrauchswerte aus den Strommesswerten und der Länge der Abtastintervalle berechnet. Als Ergebnis liefert PowerScope Energieverbrauchswerte für ganze Prozesse und Unterbrechungsroutinen, aufgeschlüsselt auf ihre verwendeten Funktionen.

[1]Die Abtastfrequenz kann vom Benutzer eingestellt werden, für Messungen in [4] werden Abtastintervalle von etwa 1,6 ms angegeben.

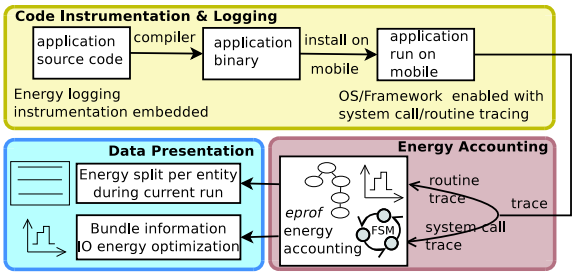

Abbildung 6: Überblick über die Architektur des Energieverbrauch-Profilers Eprof von Pathak et al. Quelle: [2]

3.2 AEON

Das AEON-Projekt [1] (siehe Abbildung 5) modelliert das Energieverbrauchsverhalten der Sensorknotenplattform Mica2 mit dem Betriebssystem TinyOS [11], um Aussagen über die Laufzeit von auf der Plattform ausgeführten Anwendungen treffen zu können. Zusätzlich implementiert das Projekt Profiling-Funktionalität. Bei AEON wird für jede Hardware-Komponente ein Modell für die verfügbaren Gerätezustände und den jeweiligen Energieverbrauch in diesen erstellt. Die Modelle werden mit Hilfe von Strommessungen kalibriert und validiert. Darauf basierend sind die Modelle im Sensorknotenemulator AVRORA [12] implementiert. Für das Profiling bzgl. des Energieverbrauchs der CPU werden die Funktionen im Programmquelltext auf die die Adressen im Objektcode abgebildet und während der Programmausführung alle Funktionsaufrufe mitgeschnitten.

3.3 Eprof (Pathak et al.)

Pathak et al. verfolgen für ihren Energieverbrauch-Profiler Eprof [8, 2] neben den verwendeten Funktionen zusätzlich nach, welche Systemaufrufe in den untersuchten Smartphone-Programmen aufgerufen werden (engl. *system call tracing*). Ihr Ziel ist es dabei, das Energieverbrauchsverhalten vollständiger abzubilden, als es durch eine alleinige direkte Verknüpfung von Gerätezuständen und der Gerätenutzung durch ein Programm möglich ist (sog. benutzungsbasierte Modellierung, engl. *utilization-based modelling*). So wird zum Beispiel das in Abschnitt 2.5 besprochene asynchrone Energieverbrauchsverhalten mit berücksichtigt.

Abbildung 6 zeigt das Vorgehen beim Profiling mit Eprof. Für das Android-Betriebssystem[2] werden die untersuchten Anwendungen, die verwendete Laufzeitumgebung und der Betriebssystemkern so präpariert, dass zur Laufzeit Funktions- und Systemaufrufe nachverfolgt werden können (Bereich *Code Instrumentation & Logging* in der Abbildung). Bei Anwendungen, deren Quelltext nicht verfügbar ist, werden dennoch die Aufrufe in der Laufzeitumgebung mit berücksichtigt. Basierend auf diesen erhaltenen Informationen wird in einem zweiten Schritt (Bereich *Energy Accounting* in der Abbildung) den ausgeführte Programmstrukturen, erhalten über die aufgerufenen Funktionen, ein Energieverbrauchswert zugeordnet. Der Energieverbrauch wird dabei unter Zuhilfenahme eines Modells für die verschiedenen Geräte in Form eines endlichen Automaten erhalten (die Zustandsübergänge des Automaten werden durch die verschiedenen Systemaufrufe ausgelöst). Diese erhaltenen Informationen werden dann dem Nutzer als Ergebnis dargestellt (Bereich *Data Presentation* in der Abbildung) – zum Einen als Energieverbrauch der einzelnen Programmbestandteile, zum Anderen als sog. *bundle*-Repräsentation für einen detaillierteren Einblick in die asynchron in den Geräten für Ein-/Ausgabe verbrauchte Energie.

[2]Eprof wurde neben Android auch für Windows Mobile umgesetzt.

4. ZUSAMMENFASSUNG

In dieser Arbeit wurde zunächst auf die grundlegenden Schritte beim Profiling von Software-Energieverbrauch eingegangen: Das untersuchte Programm wird durchlaufen und der dabei entstehende Energieverbrauch wird zu denjenigen Programmbestandteilen zugeordnet, die für ihn und seine Höhe ursächlich sind. Danach wurden verschiedene Aspekte, die es beim Entwurf eines Profilers für Software-Energieverbrauch zu beachten gilt, besprochen. Abschließend wurden mit PowerScope, AEON und Eprof drei Beispiele für Profiler-Umsetzungen genauer betrachtet.

Obwohl, wie dargelegt wurde, verschiedene Ansätze zum Profiling von Software-Energieverbrauch existieren, hat sich bisher keine Umsetzung für den Alltagsgebrauch in der Software-Entwicklung etabliert. Denkbare Aufgaben für weitere Arbeiten in diesem Bereich können daher, neben der zusätzlichen Betrachtung weiterer Hardware-Komponenten, in der Integration der verschiedenen Ansatzpunkte und verbesserter Anwenderfreundlichkeit liegen, um das sog. *energiegewahre Programmieren* (engl. *energ-aware programming*) [6] in der Praxis zu vereinfachen.

5. LITERATUR

[1] O. Landsiedel, K. Wehrle, and S. Gotz, "Accurate prediction of power consumption in sensor networks," in *Proceedings of the 2nd IEEE workshop on Embedded Networked Sensors*, EmNets '05, (Washington, DC, USA), pp. 37–44, IEEE Computer Society, 2005.

[2] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof," in *Proceedings of the 7th ACM european conference on Computer Systems*, EuroSys '12, (New York, NY, USA), pp. 29–42, ACM, 2012.

[3] S. L. Graham, P. B. Kessler, and M. K. Mckusick, "Gprof: A call graph execution profiler," in *Proceedings of the 1982 SIGPLAN symposium on Compiler construction*, SIGPLAN '82, (New York, NY, USA), pp. 120–126, ACM, 1982.

[4] J. Flinn and M. Satyanarayanan, "PowerScope: a tool for profiling the energy usage of mobile applications," in *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, pp. 2–10, 1999.

[5] C. Cadar and K. Sen, "Symbolic execution for software testing: three decades later," *Commun. ACM*, vol. 56, pp. 82–90, Feb. 2013.

[6] T. Hönig, C. Eibel, R. Kapitza, and W. Schröder-Preikschat, "SEEP: exploiting symbolic execution for energy-aware programming," *SIGOPS Oper. Syst. Rev.*, vol. 45, pp. 58–62, Jan. 2012.

[7] S. Schubert, D. Kostic, W. Zwaenepoel, and K. Shin, "Profiling software for energy consumption," in *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on*, pp. 515–522, 2012.

[8] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *Proceedings of the sixth conference on Computer systems*, EuroSys '11, (New York, NY, USA), pp. 153–168, ACM, 2011.

[9] J. Moreno, J. Wenninger, J. Haase, and C. Grimm, "Energy profiling technique for network-level energy optimization," in *AFRICON, 2011*, pp. 1–6, 2011.

[10] Z. Zhang and S. Fu, "Macropower: A coarse-grain power profiling framework for energy-efficient cloud computing," in *Performance Computing and Communications Conference (IPCCC), 2011 IEEE 30th International*, pp. 1–8, 2011.

[11] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. Culler, "The emergence of networking abstractions and techniques in tinyos," in *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, NSDI'04, (Berkeley, CA, USA), pp. 1–1, USENIX Association, 2004.

[12] B. L. Titzer, D. K. Lee, and J. Palsberg, "Avrora: scalable sensor network simulation with precise timing," in *Proceedings of the 4th international symposium on Information processing in sensor networks*, IPSN '05, (Piscataway, NJ, USA), IEEE Press, 2005.