Jugend Forscht 2016

The eco-friendly next Generation.

Ein Projekt zur Verbesserung des Standbys

Florian Baader, Stephan Le, Matthias Weirich

Projektnummer: 139362

Inhaltsverzeichnis

Abbildungverzeichnis

1. PROBLEMBESCHREIBUNG

1.1. Das Problem

Das Energiesparen und der ressourcenschonende Umgang werden heutzutage immer wichtiger. Überall wird geforscht, wie man den Energieverbrauch eindämmen oder zumindest erneuerbare Energiequellen einsetzen kann.

Da Elektronik eine immer größere Rolle in unserer Gesellschaft spielt, haben wir uns das Ziel gesetzt, in diesem Bereich Verbesserungen zu finden.

Unsere Idee ist die Verbesserung des von Windows integrierten Standby-Modus‘.

Auf den Gedanken sind wir gekommen, nachdem wir festgestellt haben, dass unsere PCs zum Beispiel während des Mittagessens immer laufen, obwohl sie zu der Zeit nicht genutzt werden und keine Aufgaben ausführen.

Eigentlich müsste nun der Standby von Windows greifen, allerdings haben diesen alle von uns deaktiviert! – Warum? – Weil schon oft ein gestarteter Download oder ein Kopiervorgang von diesem unterbrochen wurde.

In solchen Situationen ist der Standby-Modus nicht richtig angewandt. Klar - sein Ziel ist, den Stromverbrauch des Computers, wenn er gerade nicht gebraucht wird, so weit wie möglich zu reduzieren.

Aber der Computer wird ja gebraucht – es sitzt nur gerade kein Nutzer davor!

Nachdem wir weiter über das Problem nachgedacht haben, konnten wir feststellen, dass in vielen anderen Situationen der Standby-Modus des Computers nicht richtig angewandt wird und seine eigentliche Funktion nicht erfüllt:

Es ist häufig der Fall, dass nach dem Download von großen Dateien der Computer für längere Zeit eingeschaltet bleibt. Das liegt daran, dass der Benutzer während dieses Vorgangs meistens den Computer verlässt und oft erst lange nach dem Abschluss des Downloads zum Computer zurückkommt, um ihn herunterzufahren. Während dieser Zeit wird jedoch der Computer nicht gebraucht – und dies führt letztendlich zum unnötigen Verbrauch von Energie.

Diese Leerlaufphasen treten auch in der Berufswelt auf: Während eines Praktikums bei Infineon konnten zwei von uns beobachten, dass die Computer oft eingeschaltet bleiben, obwohl sie nicht benötigt werden. Dies ist beispielsweise während der täglichen Mittagspause, bei regelmäßig stattfindenden Meetings oder manchmal sogar auch über die Nacht der Fall. Durch die Eindämmung dieser Energieverschwendung könnte man – besonders angesichts der großen Anzahl an Computern in Unternehmen – hohe Energiekosten einsparen.

Weitere Fälle von häufig vorkommenden Leerlaufphasen konnten wir in den Computerräumen unserer Schule beobachten: Die Computer werden von den Schülern meistens nicht zum Ende der Schulstunde heruntergefahren. Dadurch bleiben sie auch während den Pausen sowie während der Zeit, in denen in diesen Räumen kein Unterricht stattfindet, eingeschaltet. Dies verursacht auch einen erhöhten Energieverbrauch, der mit höheren Energiekosten verbunden ist.

1.2. Der Konflikt zwischen Umweltschutz und Komfort

Wir haben festgestellt, dass oft **Umweltschutz und Komfort nicht gleichzeitig möglich** sind. So wurde z.B. bei uns an der Schule beschlossen, dass der Windows-interne Standby deaktiviert werden soll, damit möglichst wenig Unterrichtszeit verloren geht, bis die Computer an sind. Diesen Konflikt versuchen wir mit unserem Programm zu lösen.

Dieses Schaubild veranschaulicht noch einmal diesen Konflikt:

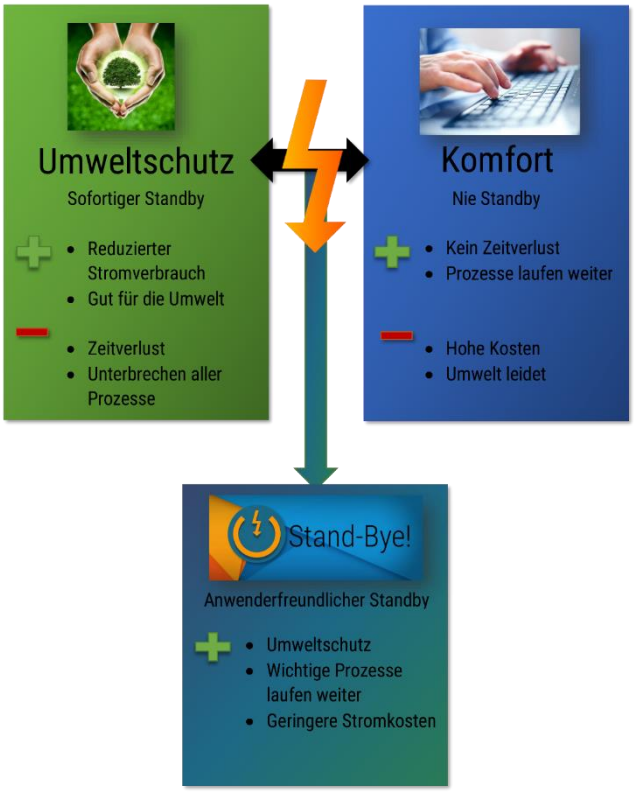

Abbildung 1: Der Konflikt zwischen Umweltschutz und Komfort

1.3. Unsere Lösung des Problems

Hier setzt unsere Idee an: **Das intelligente Stromsparen!** Das von uns entwickelte Programm „Stand-Bye!“ soll der Verschwendung von Energie während Leerlaufphasen vorbeugen.

Dazu ermittelt es zunächst anhand der getätigten **Maus- und Tastatureingaben** des Nutzers, ob dieser zurzeit aktiv am Computer arbeitet. Falls dies nicht der Fall ist, wird nach einer im Programm einstellbaren Zeit überprüft, wie hoch die Auslastung des Computers ist. Dafür werden Leistungsdaten wie die CPU-, RAM-, Internet- und Festplattenauslastung herangezogen. Die jeweiligen Obergrenzen lassen sich im Programm konfigurieren.

Wenn diese Analyse ergibt, dass der Computer nicht ausgelastet ist, wird überprüft, ob aktuell eines der Programme aktiv ist, die als Ausnahmen eingestellt wurden. Fall das nicht der Fall ist, wird der Energiesparmodus (engl. Standby / Sleep) des Computers aktiviert. Dabei gehen keinerlei Daten wie angefangene Word-Dokumente oder E-Mail-Entwürfe verloren, sodass Benutzer das Programm bedenkenlos den PC überwachen lassen können.

Dieser Modus reduziert den Energieverbrauch des Computers, indem es den Stromverbrauch der Hardware so weit wie möglich reduziert. Der Arbeitsspeicher bleibt weiterhin aktiv und wird mit Strom versorgt (suspend to RAM), damit der Benutzer seine Arbeit innerhalb weniger Sekunden wieder fortsetzen kann.

Im Gegensatz dazu wird beim Ruhezustand (engl. Hibernate) der gesamte Inhalt des Arbeitsspeichers auf der Festplatte gesichert (suspend to disk), weswegen der Arbeitsspeicher nicht mehr mit Strom versorgt werden muss. Deshalb lässt sich der Computer ohne Datenverlust vom Stromnetz trennen, dagegen dauert das erneute Hochfahren etwas länger, da der Inhalt des Arbeitsspeichers erst wieder eingelesen werden muss.

Computern mit einem Betriebssystem ab Windows Vista bietet sich eine weitere Möglichkeit: der hybride Standby-Modus. Hierbei handelt es sich um eine Kombination der beiden oben angesprochenen Modi. Der Arbeitsspeicher wird weiterhin wie im Energiesparmodus mit Strom versorgt, zusätzlich aber auch auf der Festplatte gesichert (suspend to RAM and disk). Dadurch kann einerseits der Computer – wie nach dem Energiesparmodus – schnell wieder aufwachen, andererseits kann der Benutzer auch – wie beim Ruhezustand – nach unterbrochener Stromzufuhr den Computer hochfahren und seine Arbeit fortsetzen.

Wenn bei aktiviertem hybriden Standbymodus der Computer in den Energiesparmodus versetzt wird, verwendet er automatisch in den hybriden Standbymodus.

Diese Tabelle fasst die wichtigen Aspekte der drei Arten Energiesparmodus, Ruhezustand und hybrider Energiesparmodus zusammen:

Bezeichnungen			**RAM weiterhin aktiv**	**Sicherung des RAM-Inhalts auf Festplatte**
TECHNISCH	ENGLISCH	DEUTSCH		
suspend to RAM	Standby / Sleep	Energiesparmodus	Ja	Nein
suspend to disk	Hibernate	Ruhezustand	Nein	Ja
suspend to RAM and disk	Hybrid Sleep	Hybrider Standby-Modus	Ja	Ja

Abbildung 2: Überblick über die energiesparenden Maßnahmen unter Windows

1.4. „Stand-Bye!“ als ein Open Source-Projekt

Weil uns der Gedanke hinter Open Source gefällt und wir unser Projekt anderen Entwicklern zur Verbesserung, Anpassung oder Mitarbeit zur Verfügung zu stellen, ist Stand-Bye ein Open Source Projekt.

Außerdem handelt es sich bei dem Projekt nicht um Eigennutz, sondern wir wollen der Umwelt damit etwas Gutes tun – und deswegen ist es wichtig, dass so eine Software frei verfügbar ist.

Der Quelltext ist deshalb auf Plattform GitHub unter https://github.com/flobaader/Stand-Bye zu finden. Außerdem ist im Moment eine Webseite zum Projekt in Arbeit, die helfen soll, das Programm schneller an Anwender zu verteilen, diese ist unter http://stand-bye.de zu erreichen. Dort finden sie auch noch weitere Informationen!

2. FUNKTIONSWEISE DES PROGRAMMS

1.5. Der Zugriff auf das System – die Klasse SystemAccess

Um Zugriffe auf das System in einer Klasse zu sammeln und so sowohl die Übersichtlichkeit als auch die Lesbarkeit des Codes zu verbessern, wurde die Klasse `SystemAccess` erstellt. Einen großen Teil in unserem Programm spielen `PerformanceCounter`, die es uns ermöglichen, verschiedene Systemwerte auszulesen. Wir haben uns für vier Werte zum Beobachten entschieden:

1. Die **prozentuale CPU-Auslastung** gibt Aufschluss darüber, ob gerade Prozesse auf dem Computer laufen, die nicht beendet werden sollten. Als Beispiel kann man hier eine Konvertierung eines Films nennen oder einen Rendervorgang, den man vor dem Verlassen des Computers gestartet hat, in der Hoffnung, dass er bei der Rückkehr erledigt ist.

2. Die **RAM-Belegung** wird ebenso in Prozent angegeben, da dies die deutlich komfortabelste und einfachste Weise ist. Da der Arbeitsspeicher zur temporären Sicherung von Daten da ist, die vom CPU verarbeitet werden sollen, ist auch dieser ein Hauptindikator für die Auslastung des PCs.
3. Für die HDD-Auslastung haben wir uns für einen absoluten Wert in Megabit pro Sekunde entschieden, da der prozentuale Wert oft sehr gering ist, denn die maximale Nenngeschwindigkeit ist oft sehr hoch angesetzt. Sie dient als **Indiz für IO-Vorgänge** wie z.B. Backups, Kopiervorgänge oder Downloads.
4. Die **Netzwerkauslastung** hat sich als etwas schwieriger herausgestellt. Um die Auslastung der Netzwerkverbindung herauszufinden, muss der Name der Instanz (also in diesem Fall der Netzwerkadaptername) mit angegeben werden. Wir haben uns dafür entschieden, dass einfach alle Netzwerkadapter überwacht und die Auslastungen **addiert** werden, da z.B. Laptop-Benutzer sowohl den Drahtlosadapter als auch den LAN-Adapter verwenden. Es wurde ein weiteres Mal der absolute Wert gewählt, da bei Netzwerkkarten die reale Geschwindigkeit oft sehr stark von der Nenngeschwindigkeit abweicht.

Die `PerformanceCounter` werden im Konstruktor der Klasse `SystemAccess` initialisiert:

```
//Initializes the PerformanceCounters
//                                Category        ,Counter_name,  Instance
perfCPU = gcnew PerformanceCounter("Processor", "% Processor Time", "_Total");
perfNET = gcnew PerformanceCounter("Network Interface", "Bytes Total/sec",
    gcnew String(settings_provider->getRawSetting(SettingName::NET_ADAPT).c_str()));
perfHDD = gcnew PerformanceCounter("PhysicalDisk", "Disk Bytes/sec", "_Total");
```

Abbildung 3: Die Initialisierung der PerformanceCounter

Und so sieht beispielsweise eine typische Abfrage eines Counters aus:

```
float SystemAccess::getNetworkUsage() {
    try {
        float mbytes_per_sec = perfNET->NextValue() / MILLION; //->NextValue() returns usage in Byte/s
        return mbytes_per_sec;
    }
    catch (System::InvalidOperationException^ ex) {
        //Selected adapter name is empty or not valid
        DEBUG("No valid Adapter selected!");
        DEBUG(ex->Data);
        return 0.0f;
    }
}
```

Abbildung 4: Eine typische Abfrage eines PerformanceCounters

1.6. Der SystemMetricWatcher – Messung der Auslastung

Die Aufgabe von `SystemMetricWatcher` ist es, konstant die Systemauslastungswerte zu messen und in einen `Buffer` zu speichern. Die Klasse bedient sich hierbei der Helferklasse `AverageBuffer`, deren einzige Aufgabe es ist, die Werte für einen Aspekt der Systemauslastung in einen Ringspeicher zu schreiben und auf Anfrage einen Durchschnittswert über alle gespeicherten Werte zu berechnen. Frequenz der Messung und Größe der Ringspeicher können im Konstruktor als Parameter angegeben werden.

Zum einen besteht der Vorteil dieser Implementierung darin, dass Messwerte **sofort** abgerufen werden können, andererseits wird der Messwert **präziser**, da der Messwert immer stark schwankt und der Durchschnitt diesen stabilisiert. Nachteile dieses Verfahrens sind jedoch eine insgesamt **höhere Rechenintensivität** und eine gewisse „Aufwärmzeit“ am Anfang, bis alle Plätze im Ringspeicher gefüllt sind und ein Durchschnitt erstmals berechnet werden kann. Als Standartwert haben wir eine Zeit von 15 Sekunden bei zwei Messungen pro Sekunde angesetzt, da dies einen guten Kompromiss zwischen Performance, Genauigkeit und Aufwärmzeit vor dem ersten Durchschnittswert zu sein scheint.

Die CPU-Auslastung, welche durch das Programm selbst verursacht wird, liegt bedingt durch die Art der Messung bei **unter 0.2%.**

1.7. Die Überwachung der Eingaben – die Klasse InputMonitor

Die Klasse `InputMonitor` ist, wie der Name bereits vermuten lässt, dafür zuständig, alle Eingaben über Maus und Tastatur zu überwachen. Die Schleife, die regelmäßig die Zeit seit der letzten Eingabe misst, läuft auf einem eigenen **Hintergrundthread**, um die Benutzeroberfläche und andere Operationen nicht zu blockieren. Im Kern benutzt sie dazu die `GetLastInputInfo()`, die die **Zeit seit der letzten Benutzereingabe** (Maus oder Tastatur) zurückgibt. Das Aktivitätsdiagramm von `InputMonitor` sieht, etwas vereinfacht, so aus:

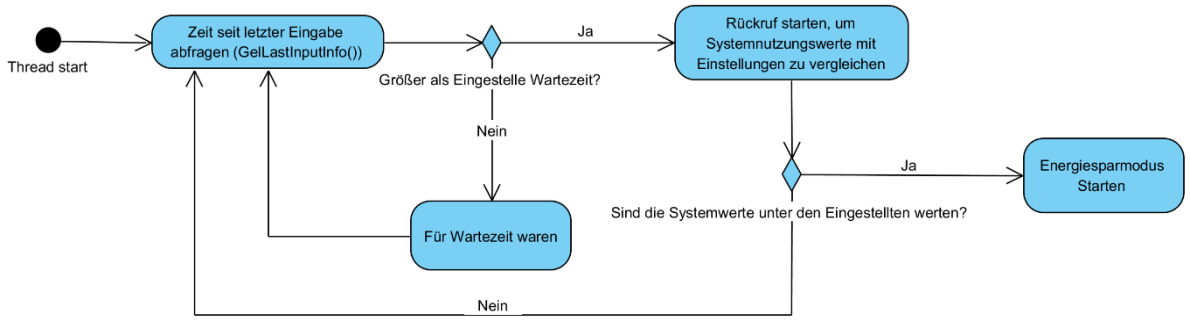

Abbildung 6: Aktionsdiagramm von InputMonitor

Nachdem dem Start des Threads wird zuerst die eingestellte Wartezeit vom `SettingsProvider` geladen. Danach wird die Methode `Monitor()` aufgerufen, die die Zeit seit der letzten Benutzereingabe überwacht und diese mit der eingestellten Wartezeit vergleicht.

```
void InputMonitor::Monitor() {
    while (aborted == false) {
        if (SystemAccess::GetLastInputTime() == 0) {
            DEBUG("LastInputTime == 0 milliseconds");
        }

        if (SystemAccess::GetLastInputTime() > wait_time) {
            DEBUG("Wait Time is over!");
            OnFinished();
        }
        //Sleep only 10 milliseconds at once to handle to Close() Event
        for (int x = 0; x < wait_time; x = x + 10) {
            watcher->Sleep(10);
            if (aborted) {
                return;
            }
        }
    }
};
```

Abbildung 7: Die Methode Monitor()

Hier wird zuerst die Zeit seit der letzten Eingabe abgefragt. Liegt diese **über** der festgelegten Wartezeit, wird die Schleife direkt beendet. Andernfalls „schläft“ bzw. **wartet der Thread** für diese Wartezeit, danach startet die Schleife erneut.

Werden die Einstellungen vom Benutzer geändert, wird der Hintergrundthread kurzerhand **neu gestartet**, so dass keine Synchronisation zwischen Haupt- und Hintergrundthread nötig wird.

1.8. Der Präsentationsmodus

Eine zusätzliche Funktion der Software ist der **Präsentationsmodus**. Er kann direkt aus dem Kontextmenü des Symbols in der Taskleiste aktiviert werden und bewirkt, dass der **Bildschirm nicht ausgeschaltet wird**, solange der Präsentationsmodus aktiviert ist. Dieses Feature wurde uns von verschiedensten Personen vorgeschlagen, denn obwohl sie einfach ist und oft gebraucht wird, gibt es in Windows keine eingebaute Funktion. Die einzige Möglichkeit besteht darin, den Energiesparmodus sowie den Bildschirm-Timeout **ganz zu deaktivieren**.

1.9. Das Überprüfen der Systemwerte nach der Wartezeit

Sobald die Wartezeit beendet wurde, der Nutzer also schon `x` Minuten keine Eingabe mehr gemacht hat, würde nun Windows in den Energiesparmodus wechseln, sofern dieser noch nicht von einem genervten Benutzer deaktiviert wurde.

Bevor unser Programm den Computer in einen der Energiesparmodi versetzt, werden nun **alle eingestellten Werte verglichen**. Durch den `InputMonitor` wird die Methode `CheckUsage()` aufgerufen, sobald die eingestellte Zeit vorbei ist.

Dabei werden nacheinander **folgende Bedingungen** geprüft:

1. **Ist das Programm im Präsentationsmodus?**

```
if (inPresentationMode) {
    DEBUG("Application in presentation mode! \n Canceled Sleep mode!");
    return;
}
```

Abbildung 8: Überprüfen des Präsentationsmodus'

2. **Ist ein Prozess gestartet, bei dem der Standby nicht aktiviert werden soll?**

```
//Check if an exception process is running
boolean exception_process_running = false;
for each(std::string process in settings_provider->getProcessList()) {
    if (BasicFunc::VectorContains(SystemAccess::GetRunningProccesses(), process)) {
        exception_process_running = true;
        break;
    }
}
if (exception_process_running) {
    DEBUG("An exception process is running! \n Canceled Sleep mode!");
    return;
}
```

Abbildung 9: Prüfen, ob ein Ausnahme-Prozess aktiv ist

3. **Ist eine Systemauslastung über dem eingestellten Grenzwert?**

```
//Check Thresholds

if (((bool)active(USE_HDD) ? threshold(MAX_HDD) < sysMetric(HDD) : false))return;
if (((bool)active(USE_CPU) ? threshold(MAX_CPU) < sysMetric(CPU) : false))return;
if (((bool)active(USE_RAM) ? threshold(MAX_RAM) < sysMetric(RAM) : false))return;
if (((bool)active(USE_NET) ? threshold(MAX_NET) < sysMetric(NETWORK) : false)) return;
```

Abbildung 10: Prüfen der Systemauslastung

Wenn alle Prüfungen das logische „falsch“ ergeben, bedeutet das, dass der Computer gerade nicht gebraucht wird (weder für eine Präsentation noch für eine andere Aufgabe) und somit nur unnötige Energie im Leerlauf verschwendet wird.

4. **Ist der Benutzer wirklich nicht mehr vor dem PC?**

Um trotzdem sicher zu sein, dass der Nutzer nicht mehr am PC ist, wird ein `TimeoutWindow` angezeigt, dass den Nutzer darauf hinweist, dass der Computer in 15 Sekunden in dem Energiesparmodus versetzt wird. Das kann durch Anklicken von „Cancel“ oder „OK“ abgebrochen oder sofort ausgeführt werden.

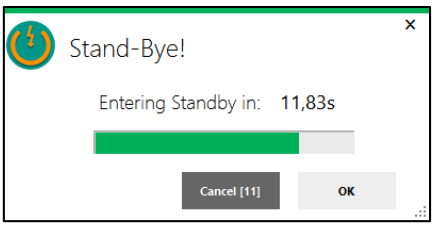

Abbildung 11: Das TimeOutWindow

5. **Nach Ablauf der 15 Sek. oder dem Drücken von OK aktiviert unsere Applikation den Energiesparmodus.**

```
TimeoutWindow^ msgWnd = gcnew TimeoutWindow(15);

if ((msgWnd->ShowDialog()) == (::DialogResult::OK)) {
    DEBUG("Going to Sleep mode!");
    SystemAccess::StartESM();
}
```

Abbildung 12: Sequenz zum Aufruf des TimeoutWindows und anschließendes Starten des ESM

Um von der Installation an zu funktionieren, braucht das Programm natürlich voreingestellte Werte für RAM-, CPU-, Festplatten- und Netzwerkauslastung. Die Wahl dieser Werte gestaltete sich nicht ganz einfach, da typische Nutzungswerte stark zwischen Systemen und Anwendern variieren. Die Werte betragen **30%** für die CPU, **70%** für die RAM und **1MBit/s** für das Netzwerk. Lediglich der Wert für die Festplatte, festgelegt auf **2 Mbit/s** dürfte für die meisten Systeme funktionieren, da fast alle auf dem Markt erhältlichen Festplatten dieselben Schreib- und Lesegeschwindigkeiten haben.

1.10. Die Verwaltung der Einstellungen – die Klasse SettingsProvider

`SettingsProvider` hat die Aufgabe, die Einstellungen des Programmes zu **verwalten und zu speichern**, sodass sie zwischen Systemneustarts erhalten bleiben. Dabei ist die Klasse nicht nur eine reine Schnittstelle zur Einstellungsdatei, sondern erledigt auch alle nötigen Konvertierungen von Variablentypen sowie Fehlerüberprüfungen.

Von dieser Klasse wird nur eine **einzige Instanz** erstellt, um die Zahl der Lese- und Schreiboperationen von und auf die Festplatte zu minimieren. Diese wird in der `Main`-Methode erzeugt und an alle Klassen, die sie benötigen, **als Parameter im Konstruktor** übergeben.

Wir haben uns dafür entschieden, eine Einstellungsklasse selber zu schreiben, da C++ und Visual Studio von sich aus keine geeigneten Lösungen mitbringen. Außerdem wird so eine große Übersichtlichkeit im Code durch die eigene Benennung der Methoden garantiert.

Nach einer Besprechung haben wir uns für eine Speicherung der Einstellungen in einer Datei statt der Windows Registry entschieden, da dies die einfachste Möglichkeit war.

Der `SettingsProvider` liest und schreibt in den folgenden Pfad:

```
C:\Users\[User]\AppData\Roaming\SmartLogout\Settings.ini
```

Die Klasse lädt die Einstellungen im Konstruktor, um unnötige IO-Operationen zu vermeiden. Allerdings haben wir uns entschieden, dass die Datei nach jeder Änderung neu geschrieben werden soll, was hilft, um verlorene oder korrupte Daten zu vermeiden.

So wird z.B. auch eine fehlerhafte Datei erkannt und mit Standardwerten neu erstellt:

```
SettingsProvider::SettingsProvider() {
    //Load settings
    if (loadSettings() == false || ((int)SettingsList.size() != SETTINGS_COUNT)) {
        //File does not exists or has not excepted number of settings
        DEBUG("Settings file could not be loaded or configuration is corrupt!");
        repairFile();
        if (loadSettings() == false) {
            DEBUG("Settings could not be repaired!");
        }
    };
};
```

Abbildung 13: Überprüfung und ggf. Wiederherstellung einer fehlerhaften Datei

Zu `SettingsProvider` gehört auch die Containerklasse `Setting`, in der ein einzelner Einstellungswert gespeichert werden kann.

1.11. Die Graphische Oberfläche (GUI)

Damit das Programm besonders **benutzerfreundlich** ist, wurde viel Arbeit in das Design der Benutzeroberfläche investiert. Nach einer gemeinsamen Diskussion haben wir uns für ein **Metro-Design** entschieden, da wir eine intuitive Bedienung und ein modernes Design als Anforderungen an das User-Interface gestellt haben.

Die GUI dient dazu, dem Benutzer ein einfaches Setzen der Einstellungen zu ermöglichen und ihn in die Funktionsweise des Programms einzuführen.

Da die `.NET Framework Forms` keine Metro-Funktionalität bieten, haben wir auf eine Open Source Lösung namens `MetroFramework` zurückgegriffen. Weitere Infos dazu befinden sich in dem Abschnitt Danksagung.

Die Oberfläche zeigt als Orientierungshilfe für die nötigen Einstellungen auch die aktuell im `Buffer` gespeicherte Auslastung der gemessenen Systemwerte. Ob die aktuellen Werte über oder unten den eingestellten liegen, wird außerdem durch die Hintergrundfarben Rot und Grün deutlich.

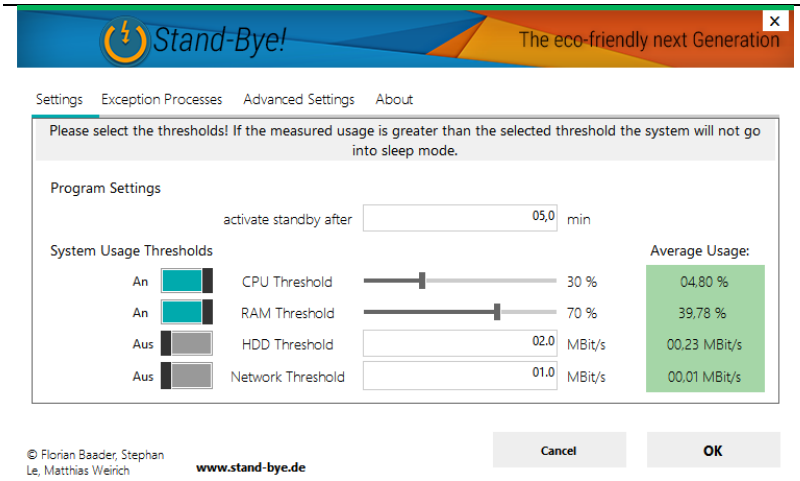

Abbildung 14: Startseite von "Stand-Bye!"

Hier werden alle Schwellenwerte eingegeben und die Wartezeit festgelegt.

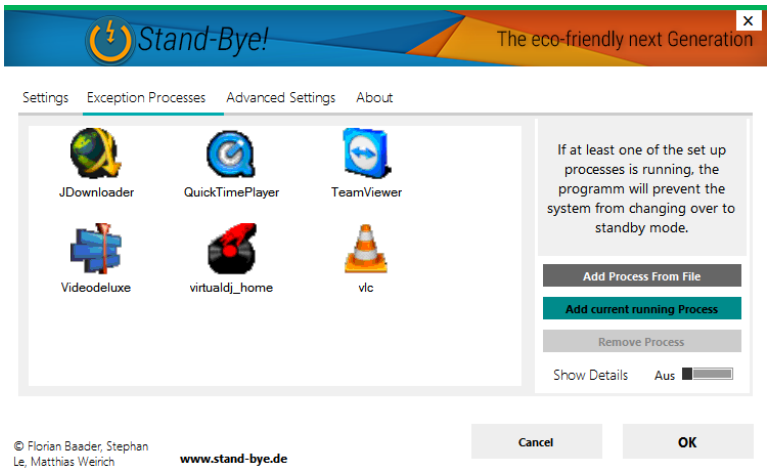

Abbildung 15: Seite zur Einstellung der Ausnahme-Prozesse

Wenn einer dieser Prozesse läuft, wird der Standby nicht gestartet.

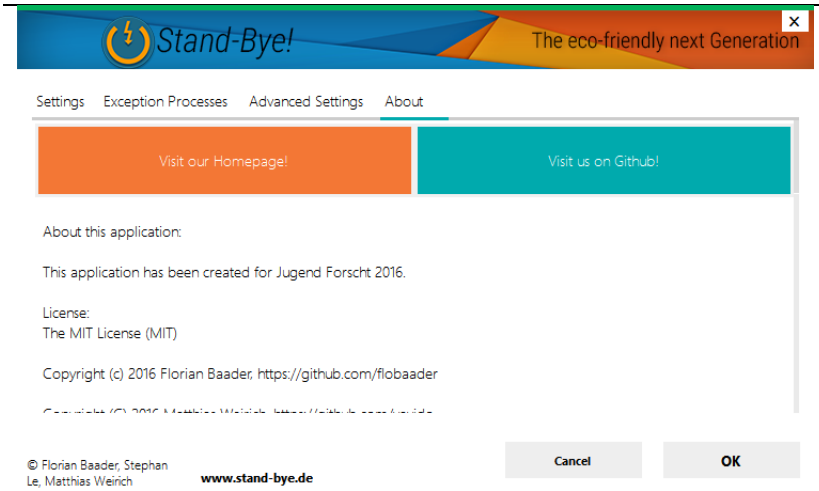

Abbildung 16: Die About-Seite

Hier findet man den Link zu unserer Github-Page und unserer Homepage.

3. ERGEBNISSE

In den vorhergehenden Abschnitten wurden die **Idee** sowie die **Funktionsweise** des Programms genauer erläutert. Im Anschluss sollen nun Ergebnisse folgen – es werden anhand zweier Anwendungssituationen die möglichen Einsparpotentiale berechnet.

1.12. Anwendungsfall 1: Angestellter eines Unternehmens

In diesem Fall soll die mögliche Stromeinsparung durch die Verwendung von „Stand-Bye!“ für den Anwendungsfall eines durchschnittlichen Angestellten hypothetisch berechnet werden.

1.1.1. Die Ausgangssituation: Mittagspause und Besprechungen als inaktive Phasen

Ein durchschnittlicher Vollzeit-Angestellter arbeitet im Rahmen einer 40-Stunden-Woche von Montag bis Freitag täglich etwa acht Stunden im Unternehmen. Für diese Hochrechnung haben wir angenommen, dass der Angestellte mit einem **Mittelklasse-PC** arbeitet. Als Grundlage unserer beispielhaften Berechnungen haben wir die Leistungsdaten einer unserer Rechner mit den folgenden Komponenten bestimmt:

- Intel i5-4670K CPU
- 8 GB RAM
- NVIDIA GeForce 9600GT
- Netzteil: 630W mit 80PLUSBronze (durchschnittlich 86 Prozent Effizienz)

Der PC wird morgens **zu Beginn der Arbeit** von dem Angestellten hochgefahren und abends nach der Arbeit wieder heruntergefahren. Auch während der einstündigen Mittagspause bleibt der Computer eingeschaltet.

Die Energiespareinstellungen wurden deaktiviert, da sich viele Unternehmen in dem oben beschriebenen Konflikt zwischen Komfort und Energiesparen für die Komfort-Seite entscheiden. Dies haben wir unter anderem während verschiedener Praktika festgestellt.

Somit kann davon ausgegangen werden, dass der Computer während des **gesamten** täglichen Aufenthalts im Unternehmen durchgehend eingeschaltet bleibt. Für diese Zeit gilt:

$$\Delta t_{ges} = Arbeitszeit + Mittagspause \ = 8{,}0\ h + 1{,}0\ h = \ 9{,}0\ h$$

In dieser Zeit benutzt der Angestellte aber nicht ständig seinen Computer. Neben der einstündigen Mittagspause wird auch der Computer **während den Meetings mit Kollegen**, die täglich etwa 1,5 Stunden in Anspruch nehmen, nicht benutzt. Für diese inaktive Zeit kann angenommen werden, dass der Computer keine Aufgaben zu erledigen hat. Daher ergibt sich für die Zeit, in der der Computer nicht benötigt wird und trotzdem eingeschaltet bleibt:

$$\Delta t_{inakt} = 1{,}0\ h + 1{,}5\ h = 2{,}5\ h$$

1.1.2. Berechnung der Energieersparnis durch „Stand-Bye!“

Bei einem **geöffneten Browserfenster und einem offenen Word-Dokument** haben wir bei unserem Rechner folgende Leistungsaufnahme gemessen: $P_{online} = 109{,}0\ W$

Daher verbraucht der Computer während der inaktiven Zeit täglich folgende Energie:

$$E_{inakt1} = \ P_{online} * \ \Delta t_{inakt} = 109{,}0\ W * 2{,}5\ h = 272{,}5\ Wh$$

Nun wird unser Programm „Stand-Bye!“ auf diesem Rechner installiert und ausgeführt. Da dieses Programm überprüft, ob der Computer durch einen Prozess ausgelastet ist, kann genau bestimmt werden, ob dieser momentan benötigt wird. Dadurch kann die Wartezeit **wesentlich kürzer** eingestellt werden, ohne den Abbruch dieses Vorgangs zu befürchten. In den Einstellungen wird daher festgelegt, dass das Programm bei Inaktivität den Rechner nach **zehn Minuten** in den Energiesparmodus versetzen soll. Da diese Zeit im Vergleich zur inaktiven Zeit sehr gering ist, kann diese Zeit bei der Berechnung der eingesparten Energie vernachlässigt werden.

Die Unterschiede bei der Leistungsaufnahme des Computers bei der Verwendung mit und ohne „Stand-Bye!“ lassen sich im folgenden Diagramm gut erkennen:

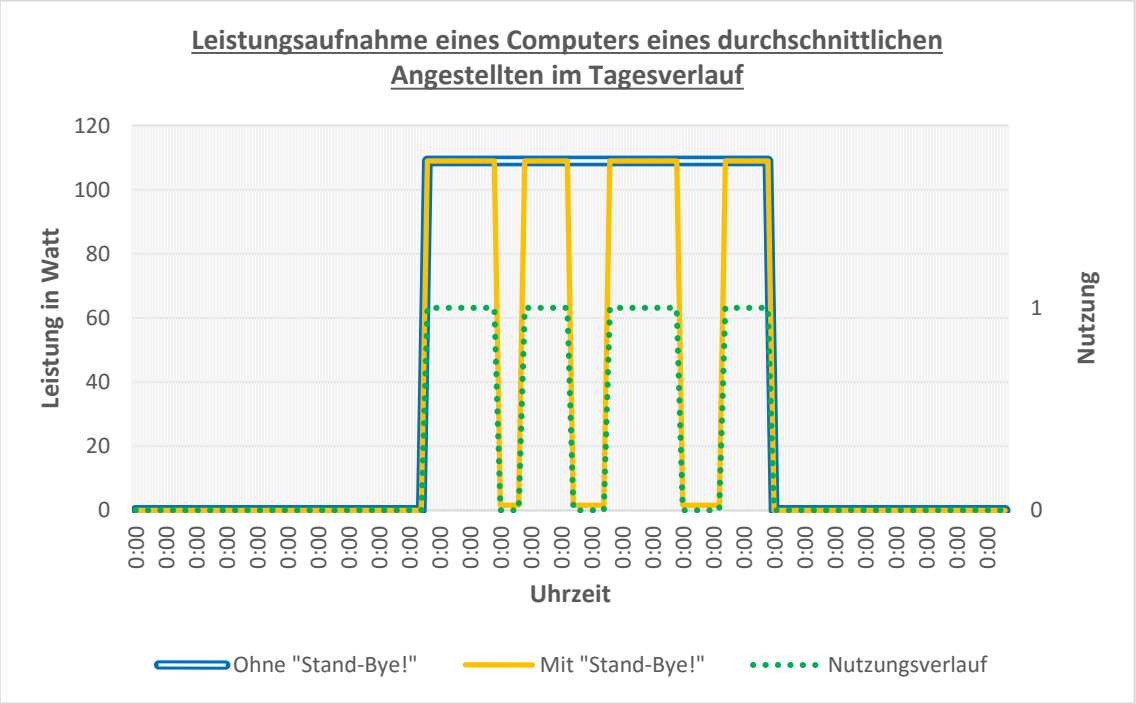

Abbildung 17: Erwartete Leistungsaufnahme eines Computers eines Angestellten im Tagesverlauf

In diesem Beispiel findet die Mittagspause zwischen **12.00 Uhr** und **13.00 Uhr** statt. Die Besprechungen mit den Kollegen werden zwischen **10.00 Uhr** und **10.30 Uhr** sowie zwischen **15.00 Uhr** und **16.00 Uhr** angesetzt.

Somit kann davon ausgegangen werden, dass der Rechner die inaktive Zeit **vollständig** im Energiesparmodus verbringt. Mit einer Leistungsaufnahme in diesem Zustand von
$P_{ESM} = 1{,}6\ W$ gilt für die **verbrauchte Energie** während der inaktiven Zeit folglich:

$$E_{inakt2} = P_{ESM} * \Delta t_{inakt} = 1{,}6\ W * 2{,}5\ h = 4{,}0\ Wh$$

Die daraus resultierende **tägliche Energieeinsparung** durch die Verwendung von „Stand-Bye!“ beträgt:

$$\Delta E_{spar/d} = E_{inakt1} - E_{inakt2} = 272{,}5\ Wh - 4{,}0\ Wh = 268{,}5\ Wh \approx 0{,}27\ kWh$$

Dementsprechend beträgt die **jährliche Energieeinsparung** bei $N_{ATage} = 250$ Arbeitstagen (Bayern 2016) auf:

$$\Delta E_{spar1/a} = \Delta E_{spar/d} * N_{ATage} = 0{,}27\ kWh * 250 = 67{,}5\ kWh$$

Bei einem Strompreis von $p = 28{,}72\ \frac{ct}{kWh}$ (durchschnittlicher Strompreis für deutsche Haushalte im Jahr 2015) beträgt die jährliche **Ersparnis an Energiekosten** für diesen Angestellten:

$$p_{ges1/a} = \Delta E_{spar1/a} * p = 67{,}5\ kWh * 28{,}72\ \frac{ct}{kWh} = 1.938{,}6\ ct \approx 19{,}39\ €$$

1.1.3. Fazit und weiterführende Überlegungen

Diese Berechnungen haben gezeigt, dass sich jährlich allein durch die Installation von „Stand-Bye!“ für einen Angestellten etwa 20€ an Stromkosten einsparen lassen. Bei Unternehmen mit mehr als 10.000 Mitarbeiter ließen sich so Ersparnisse von über 200.000€ erzielen.

Wenn **alle Büroarbeitskräfte in Deutschland** ($N_{BüArb} = 3{,}9 * 10^6$) „Stand-Bye!“ auf ihrem Rechner verwenden würden, würde sich hochgerechnet die folgende **jährliche Energieersparnis** ergeben:

$$\Delta E_{spar2/a} = \Delta E_{spar1/a} * N_{BüArb} = 67{,}5\ kWh * 3{,}9 * 10^6 = 263{,}25 * 10^6\ kWh$$

Somit könnte man dadurch **folgende Energiekosten** einsparen:

$$p_{ges2/a} = \Delta E_{spar2/a} * p = 263{,}25 * 10^6\ kWh * \ 28{,}72\ \frac{ct}{kWh} = 7.560{,}54 * 10^6\ ct = 756.054\ €$$

Diese enorme Energie- und Kosteneinsparung könnte durch die Verwendung der oben angesprochenen **Master-Slave-Steckdose** deutlich erhöht werden, da dadurch andere Geräte auf dem Schreibtisch wie Monitor, Schreibtischlampen oder Drucker auch abgeschaltet werden können, wenn der Computer in den Energiesparmodus versetzt wird. Dadurch wird die Effektivität dieser energiesparenden Maßnahme vergrößert.

1.13. Anwendungsfall 2: Das Projekt an unserer Schule

Wir planen, unser Projekt an unserer Schule einzuführen. Da noch keine Testergebnisse vorliegen, berechnen wir im Folgenden die mögliche Stromeinsparung mit folgenden Annahmen:

1.1.4. Die Ausgangssituation: Unterrichtsbeginn ohne Verzögerung vs. Energiesparen

Energiesparendes Handeln ist auch an unserer Schule sehr wichtig – es wird daher auch versucht, dies umzusetzen. Jedoch gibt es auch hier einen Konflikt zwischen Komfort und Energiesparen.

Die früher gängige Praxis, die Computer in den Computerräumen **nach jeder Unterrichtsstunde** herunterzufahren, um Energie zu sparen, stellte sich als umständlich und zeitintensiv heraus, da die Schüler der darauffolgenden Stunde diesen wieder hochfahren mussten. Dadurch ging – besonders bei jüngeren Schülern – wertvolle **Unterrichtszeit verloren**. Deshalb wurde versucht, diesem Problem entgegenzuwirken – die Computer sollen **nicht mehr** nach jeder Unterrichtsstunde heruntergefahren werden. Dies sollte dazu führen, dass der Stundenwechsel so wenig Zeit wie möglich beansprucht und der Unterricht ohne Verzögerungen begonnen werden kann.

Nun geschieht das tägliche Starten der Rechner eines Computerraumes **durch den Lehrer**, der die erste Unterrichtsstunde in diesem Raum hält und seinen Lehrer-PC mitsamt dem Überwachungsprogramm „Vision“ startet. Dadurch werden alle Computer im Raum **automatisch hochgefahren**. Da sie nach jeder Stunde nicht mehr ausgeschaltet werden sollen, bleiben sie – aufgrund der deaktivierten Energiespareinstellungen – auch **während Freistunden und Pausen** sowie nach der letzten Unterrichtsstunde eingeschaltet. Als Maßnahme gegen den erhöhten Energieverbrauch werden alle Computer um **17.30 Uhr** per Skript automatisch heruntergefahren, damit sie nicht während der Nacht laufen.

Es ist offensichtlich, dass eine solche Regelung die Produktivität der Unterrichtsstunden erhöht, jedoch lassen sich auch die Nachteile leicht erkennen: Die Computer verbrauchen nun deutlich **mehr Energie**, was sich auch auf die Energiekosten auswirkt.

Unser Programm „Stand-Bye!“ findet einen Kompromiss zwischen diesen beiden schwer vereinbaren Zielen. Durch den Einsatz von „Stand-Bye!“ lässt sich der Stromverbrauch während der inaktiven Phasen so weit wie möglich **reduzieren**, **ohne den Unterrichtsfluss zu stören**. Die Arbeit kann schnell wiederaufgenommen werden, da sich der Rechner innerhalb weniger Sekunden aus dem Energiesparmodus wieder reaktivieren lässt.

An unserer Schule gibt es zwei Computerräume mit jeweils einem Computer für die Lehrkraft und 31 Computern für die Schüler. Diese Desktop-PCs sind neben einem Arbeitsspeicher von **4 GB**, einem **i5-Prozessor** und einer **Intel HD Graphics 4600**-Grafikkarte ausgestattet. Die Anzahl der Rechner in jedem Raum ist:

$$\boldsymbol{N_{PC} = N_{LehrPC} + N_{SchlPC} = \ 1 + 31 = 32}$$

Der Unterricht in den Computerräumen endet täglich spätestens zur **9. Schulstunde (15.30 Uhr)**. Danach bleiben die Computer bis zum Herunterfahren per Skript um **17.30 Uhr** eingeschaltet, da eventuell Fortbildungen etc. stattfinden können. Für unsere Berechnungen haben wir jedoch nur die Unterrichtsstunden, die auch in den Raumbelegungsplänen vermerkt sind, berücksichtigt.

Die Auswertung dieser Pläne liefert für eine Schulwoche folgendes Ergebnis:

	Raum A	Raum B	Summe
Anzahl der Unterrichtsstunden (à 45 min)	25	8	33
Anzahl der Freistunden (à 45 min)	18	21	39
Anzahl der Pausen (à 20 min)	9	4	13
Zeit nach Unterrichtsende bis zum Herunterfahren	5 * 2,0h	5 * 2,0h	10 * 2,0h

Abbildung 18: Auswertung der Raumbelegungspläne der Computerräume unserer Schule

Somit ergibt sich für die Zeit, in der die Rechner in den Computerräumen eingeschaltet sind und nicht benötigt werden:

$$\Delta t_{inakt} = \Delta t_{Freistd} + \Delta t_{Pausen} + \Delta t_{Ende} = 39 * 45\ min + 13 * 20\ min + 10 * 120\ min = 3.215\ min$$

1.1.5. Berechnung der Energieersparnis durch „Stand-Bye!“

Die Leistungsaufnahme der Rechner im eingeschalteten Zustand beträgt: $P_{online} = 15{,}0\ W$

Daher beträgt der **wöchentliche Energieverbrauch** aller Computer während der inaktiven Zeit:

$$E_{inakt1} = N_{PC} * P_{online} * \Delta t_{inakt} = 32 * 15{,}0\ W * \frac{3.215}{60}\ h = 25.720{,}0\ Wh$$

Beim Einsatz von „Stand-Bye!“ wird der Computer **nach 10 Minuten** Inaktivität in den Energiesparmodus versetzt. Da der Einfluss dieser kurzen Zeitspanne sehr gering ist, kann er bei der Berechnung der Energieersparnis vernachlässigt werden. Demzufolge kann man davon ausgehen, dass während der inaktiven Zeit der Energiesparmodus durchgehend aktiv ist und die Leistungsaufnahme der Rechner jeweils auf $P_{ESM} = 0{,}1\ W$ reduziert. Somit gilt für die im Energiesparmodus wöchentlich verbrauchte Energie während der inaktiven Zeit:

$$E_{inakt2} = N_{PC} * P_{ESM} * \Delta t_{inakt1} = 32 * 0{,}1\ W * \frac{3.215}{60}\ h \approx 171{,}47\ Wh$$

Die **wöchentliche Energieersparnis** durch die Verwendung von „Stand-Bye!“ beträgt:

$$\Delta E_{spar/7d} = \Delta E_{inakt1} - \Delta E_{inakt2} = 25.720{,}0\ Wh - 171{,}47\ Wh = 25.548\ Wh \approx 25{,}55\ kWh$$

Die in einem **Schuljahr eingesparte Energie** lässt sich wie folgt berechnen:

$$\Delta E_{spar/a} = \Delta E_{spar/7d} * N_{Schulwoche}$$

Dabei errechnet sich $N_{Schulwoche}$ aus der Differenz der Summe der Ferienwochen von den Kalenderwochen eines Jahres. In Bayern gibt es pro Jahr 14 Ferienwochen, außerdem noch einige gesetzliche Feiertage wie den Tag der Deutschen Einheit (03. Oktober), die berücksichtigt werden sollen. Insgesamt machen diese Feiertage etwa eine Schulwoche aus. Somit ergibt sich für die Anzahl der Schulwochen:

$$N_{Schulwoche} = N_{Kalwochen} - N_{FerWochen} = 52 - (14 + 1) = 37$$

Eingesetzt in die oben genannte Formel errechnet sich folgende **jährliche Energieeinsparung** durch „Stand-Bye“:

$$\Delta E_{spar/a} = \Delta E_{spar/7d} * N_{Schulwoche} = 25{,}55\ kWh * 37 = 945{,}35\ kWh$$

Die von unserer Schule **jährlich eingesparten Energiekosten** betragen bei einem Strompreis von $p = 28{,}72\ \frac{ct}{kWh}$ (durchschnittlicher Strompreis für deutsche Haushalte im Jahr 2015):

$$p_{ges/a} = \Delta E_{spar/a} * p = 945{,}35\ kWh * 28{,}72\ \frac{ct}{kWh} \approx 27.150{,}45\ ct \approx 271{,}50\ €$$

1.1.6. Fazit und weiterführende Überlegungen

Anhand dieser Berechnungen kann man erkennen, dass durch die Verwendung von „Stand-Bye!“ enorme Energiemengen und somit auch hohe Geldbeträge für die Stromkosten eingespart werden können.

Das nachfolgende Diagramm vergleicht den Stromverbrauch bei der **herkömmlichen Nutzung** mit dem bei der **Verwendung von „Stand-Bye!“:**

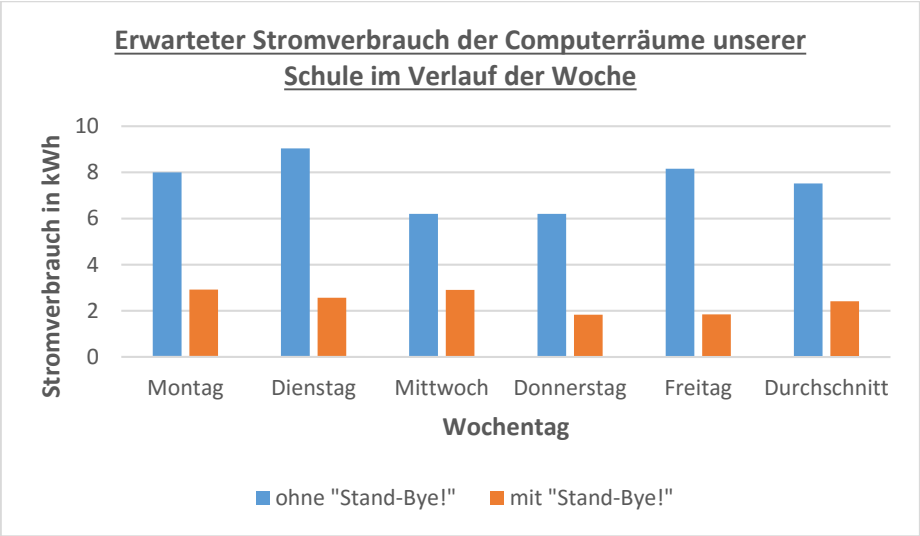

Abbildung 19: Erwarteter Stromverbrauch der Computerräume unserer Schule im Verlauf der Woche

Diesen Kontrast wird auch anhand einer kurzen Rechnung ersichtlich:

Den oben ausgeführten Berechnungen kann die wöchentlich eingesparte Energie entnommen werden:

$$\Delta E_{spar/7d} \approx 25{,}55\ kWh$$

Demgegenüber soll nun die **wöchentlich durch die Rechner verbrauchte Energie** für den normalen Unterrichtsbetrieb errechnet werden:

$$E_{UStd} = N_{PC} * \Delta t_{UStd} = 32 * 15{,}0\ W * \frac{33 * 45}{60}\ h = 11.880\ Wh = 11{,}88\ kWh$$

$$\frac{\Delta E_{spar}}{E_{UStd}} = \frac{25{,}55\ kWh}{11{,}88\ kWh} \approx 2{,}15 > 2$$

Diese Rechnung führt noch einmal die Größenordnung der eingesparten Energie vor Augen: Sie würde ausreichen, um den normalen Unterrichtsbetrieb für **zwei weitere Wochen** aufrechtzuerhalten.

Ferner muss beachtet werden, dass während den Unterrichtsstunden **nicht immer Plätze im** Computerraum von Schülern besetzt sind und daher nicht alle Rechner verwendet werden. In diesem Fall würden dann auch die nicht benötigten Computer in den Energiesparmodus versetzt werden.

Zudem werden die Computer – auch wenn ein Schüler an diesem Platz sitzt – nicht immer von diesem auch bedient. Theoriestunden, Referate und Leistungserhebungen stellen Beispiele für diese Situation dar. Auch in diesem Fall wird die energiesparende Wirkung von „Stand-Bye!“ vergrößert.

4. Erweiterbarkeit mit Hardware

Unsere Software sorgt dafür, dass der PC möglichst oft Energiesparen soll, wenn er gerade nicht gebraucht wird. Ein dabei nicht beachteter Aspekt, sind die **weiteren Geräte**, die mit der Nutzung des PCs unmittelbar etwas zu tun haben, jedoch trotzdem weiterhin Strom verbrauchen, obwohl sie nur bei der Nutzung des PCs gebraucht werden.

Die Rede ist hier von Geräten wie **Monitoren, Druckern, externe Festplatten, Lautsprecher und Schreibtischlampen.**

Hier möchten wir eine Lösung vorstellen, die nicht von uns entwickelt wurde, aber den Effekt unserer Software noch weiter unterstützt und ergänzt: **Eine Master-Slave-Steckdose.**

Die Funktionsweise ist recht einfach: Es handelt sich dabei um eine Steckdosenleiste, die eine Steckdose besitzt, die mit „Master“ gekennzeichnet ist. Die anderen Steckdosen sind vom Stromfluss durch die „Master“-Steckdose abhängig. Deswegen werden sie „Slaves“ genannt.

Wenn durch die „Master-Steckdose“ mehr als eine eingestellte Menge an Strom fließt – der PC also gerade eingeschaltet ist – werden die anderen Steckdosen auch aktiviert. Wenn die Strommenge geringer ist – der PC also gerade im Standby oder aus ist -, sind die anderen Steckdosen vom Strom getrennt.

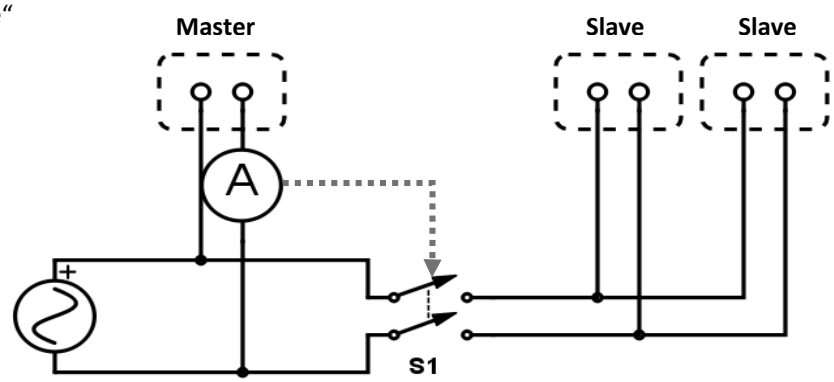

Abbildung 20: Funktionaler Schaltplan der Master-Slave-Steckdose

So kann in vielen Situationen die eingesparte Energie **mehr als verdoppelt** werden.

Zur besseren Verständlichkeit wird hier ein vereinfachter Schaltplan dargestellt, der die Funktionsweise nochmal verdeutlichen soll.

Diese Lösung verbindet genauso wie unsere Software Komfort und Energiesparen. Der Nachteil gegenüber einer konsequent genutzten Steckdosenleiste mit Schalter liegt darin, dass sowohl die Steuerelektronik, als auch ein angeschlossenes „Master“-Gerät im Standby dauerhaft Strom verbraucht. Allerdings ist dieser Stromverbrauch recht gering (< 1 Watt) und in den meisten Fällen geringer, als wenn alle Slaves an bleiben würden.

5. Diskussion

1.14. Schwierigkeiten bei der Umsetzung

Wir haben uns für dieses Projekt eine zusätzliche Herausforderung gestellt: Wir haben die komplette Anwendung **in C++** geschrieben, da dies eine sehr **systemnahe und performante** Programmiersprache ist. Da wir davor noch nie C++programmiert haben, haben wir extra für dieses Projekt C++ gelernt.

Für Anfänger bietet die Sprache einige Tücken, wie z.B. verschiedenen Arten von Strings (`std::string` / `System::String`) oder auch die schwer verständliche `winAPI`.

Ein Problem ist außerdem mit den `PerformanceCountern` aufgetreten, da diese vorher erst beim Aufrufen eines Wertes initialisiert und danach wieder gelöscht worden sind. Bei der Methode haben sie als Auslastung aber immer 0 zurückgegeben. Nach einer Recherche haben wir dann, dass ein `PerformanceCounter` die aktuelle Auslastung mit der letzten berechnet und somit der erste Wert herausgefunden, immer 0 ist. Deswegen werden die `PerformanceCounter` im Konstruktor der Klasse `SystemAccess` initialisiert.

1.15. Weiterentwicklungsmöglichkeiten

Als Weiterentwicklung könnte man als einen weiteren Indikator verwenden, ob gerade Sound ausgegeben wird.

Auch fällt es **unerfahrenen** Nutzern schwer, die Einstellungen richtig zu setzten. Deswegen könnte man über mitlernende Einstellungen nachdenken. So könnte das Programm die Nutzungsweise des Benutzers analysieren und so die Schwellenwerte anpassen.

Auch könnte man nach ca. 30 min – 1 h Im Energiesparmodus den Ruhezustand aktivieren, um so den Stromverbrauch bei einer längerfristigen Pause **noch weiter** zu reduzieren.

6. DANKSAGUNG

An dieser Stelle möchten wir all jenen danken, die durch ihre Unterstützung zum Gelingen des Projektes beigetragen haben.

Unser Dank gilt vor allem unseren betreuenden Lehrerinnen Frau Frisch und Frau Rottmann, die uns stets mit Rat und Tat zur Seite standen.

Außerdem danken wir unserer stellvertretenden Direktorin, Frau Trinder, sowie dem Systemadministrator Herr Hahn, die es uns ermöglicht haben, das Programm an der Schule zu installieren und zu testen.

Weiterhin möchten wir allen danken, die mit ihren Ideen oder durch Korrekturlesen dieser Arbeit zum Projekt beigetragen haben, insbesondere unserer Informatiklehrerin Frau Hansky.

Besonderer Dank geht außerdem an Jens Thiel, den Autor des Open Source-Projektes `MetroFramework`, das die moderne und ansprechende Gestaltung der Benutzeroberfläche ermöglicht hat. Das Projekt ist auf GitHub zu finden unter https://github.com/thielj/MetroFramework

QUELLEN- UND LITERATURVERZEICHNIS

http://windows.microsoft.com/de-de/windows7/sleep-and-hibernation-frequently-asked-questions, letzter Aufruf: 21.01.2016.

http://www.pcwelt.de/tipps/Hybrider_Standbymodus_-_Was_ist_das_eigentlich_-Energie_sparen-7853948.html, letzter Aufruf: 21.01.2016.

https://www.bdew.de/internet.nsf/id/8CFC3276B7FF3A9CC1257DDA0049A5D0/$file/150831_BDEW_Strompreisanalyse_August2015.pdf, letzter Aufruf: 21.01.2016.

http://www.schnelle-online.info/Arbeitstage-pro-Jahr.html, letzter Aufruf: 21.01.2016.

http://de.statista.com/statistik/daten/studie/243300/umfrage/anzahl-der-beschaeftigten-buerofachkraefte-in-deutschland/, letzter Aufruf: 21.01.2016.

Grundkurs C++: C++-Programmierung verständlich erklärt (Galileo Computing)

https://github.com/cecon/winforms-modernui, letzter Aufruf: 21.01.2016.

http://www.codeproject.com/Articles/9104/How-to-check-for-user-inactivity-with-and-without , letzter Aufruf: 21.01.2016.

http://pinvoke.net/default.aspx/user32.GetLastInputInfo , letzter Aufruf: 21.01.2016.