

Praktikum zum Modul Software Engineering

Sommersemester 2019

Louis Seubert, 3246700
Lukas von Loefen, 3232060
Moritz Wein, 3232286
Jens Wöhler, 3255305

Kurzbeschreibung des Projekts

Mit unserem Projekt wollen wir dem Benutzer die immer wiederkehrenden Aufgaben abnehmen und automatisieren. Das Ganze beruht auf der Idee, dass der Benutzer einmal eine Routine eines bestimmten Aufgabentyps anlegt, welche ab dann immer wieder wiederholt werden kann. Da es eine Vielzahl von möglichen Routinen gibt, ist das gesamte System modular aufgebaut und kann nach Belieben erweitert werden. Dabei soll die Benutzerschnittstelle unabhängig sein von der Hintergrundanwendung, welche die Aufgaben dann ausführt.

Das gesamte Projekt ist mehr als eine Art Framework zu sehen, welches aber durch eine Beispielimplementierung einer einfachen Datei Synchronisation Aufgabe zumindest einen konzeptionellen Beweis für eine Funktion liefert. Die beispielhafte Implementierung soll dabei alle Features der Anwendung benutzen. Funktionell macht diese Implementierung lediglich einen exakt gespiegelten Ordner an eine andere Stelle im Dateisystem.

Um nun dem Benutzer eine bestmögliche Erfahrung zu geben, soll die Konfiguration der Aufgaben über eine Benutzeroberfläche erfolgen. Diese muss dabei möglichst einfach zu bedienen sein sowie alle Features der Hintergrundanwendung berücksichtigen. Hierbei ist zu beachten, dass die Benutzerschnittstelle auch später hinzugefügte Aufgabentypen darstellen können muss.

Für die Implementierung des Projektes ist das .NET-Framework vorgesehen in Verbindung mit der objektorientierten Programmiersprache C#.

Anforderungsanalyse

User Stories

- Als **Benutzer** muss es möglich sein, nach dem Öffnen der Benutzeroberfläche eine Übersicht von allen bestehenden Aufgaben zu sehen, um diese gegebenenfalls bearbeiten zu können.
- Dem **Benutzer** muss es ermöglicht werden, Einstellungen an den bestehenden Aufgaben über die Benutzeroberfläche vorzunehmen; diese müssen dabei auf ihre Richtigkeit überprüft werden.
- Falls der **Benutzer** die Benutzeroberfläche beendet, erscheint eine Anfrage, ob die Änderungen gespeichert werden sollen, damit keine verloren gehen.
- Das **System** kann die Einstellungen aus einer Datei laden und ausführen.

Anforderungen

1. Die Benutzeroberfläche muss dem Benutzer die bereits erstellten Aufgaben anzeigen können.
2. Die Benutzeroberfläche muss die Möglichkeit bereitstellen die existierenden Aufgaben zu bearbeiten bzw. neue Aufgaben anzulegen.
3. Die Applikation sollte bei der Eingabe der Werte in die Benutzeroberfläche überprüfen ob diese korrekt sind.
4. Die Applikation wird die Möglichkeit bieten weitere Aufgaben Typen aus unterschiedlichen Quellen zu beziehen.
5. Die Applikation muss in der Lage sein die Daten in einer Konfigurationsdatei zu speichern und auch wieder zu laden.

Modellierung

Use Cases (Diagramme und tabellarische Form)

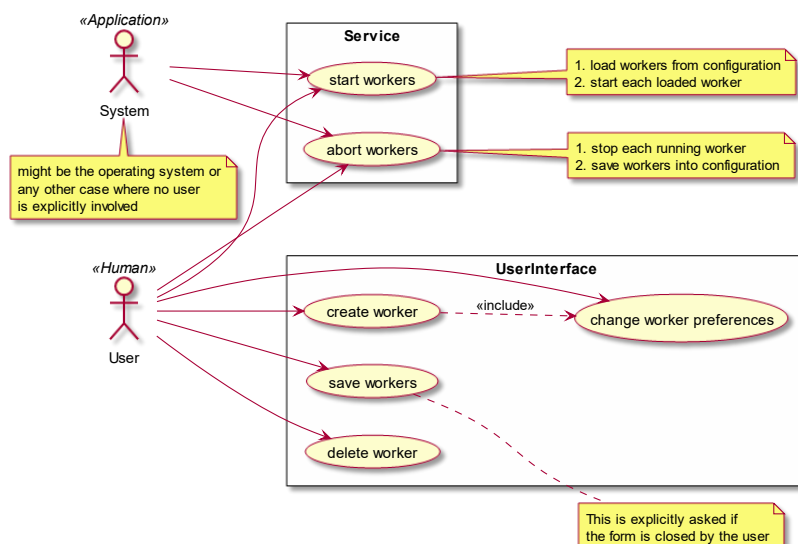


Diagramm ist auch auf GitHub zu finden, für eine genauere Betrachtung:
<https://github.com/Louis9902/Sosse19-SE/blob/master/Diagramme/Anwendungsfall/TinyTasks.svg>

Name des Anwendungsfalls:	Change Worker Preference	
Nummer:	Anwendungsfall 1	
Ziel:	Eine Änderung der Parameter für einen bestimmten „Worker“ über das Benutzerinterface	
Geltungsbereich:	Die Benutzeroberfläche	
Level:	Hauptfunktionalität	
Kategorie:	Primär	
Beteiligte Klassen/Objekte:	Benutzerinterface, Benutzer	
Vorbedingung:	Der Benutzer hat bereits das Benutzerinterface geöffnet und einen „Worker“ angelegt	
Nachbedingung (Erfolg):	Die Eingaben werden übernommen und beim Schließen der Oberfläche gespeichert	
Nachbedingung (Fehlschlag):	Der entstandene Fehler wird über eine Message Box dem Benutzer angezeigt sollte dies Jedoch ein Vorheergesehener Fehler sein wird eine erneute Eingabe verlangt	
Akteure:	Benutzer	
Auslösendes Ereignis:	Der Benutzer wählt einen im Dashboard angezeigten „Worker“ mit Doppelklick aus	
Auszuführende Aktionen (Erfolg):	1.	Die bereits vordefinierten Parameter lesen und gegebenenfalls eine Fehlerkorrektur bei falscher Definition durchführen
	2.	Das Anzeigen der Parameter über einen neuen Dialog mit der Möglichkeit diese zu bearbeiten
	3.	Beim Schließen des Parameter Dialogs eine Überprüfung der eingegebenen Daten sowie gegebenenfalls eine Reaktion, wenn benötigte Daten fehlen
Auszuführende Aktionen (Fehlschlag):	1.	Das Öffnen einer Message Box mit dem abgefangenen Fehler <i>(Dies ist immer der Fall für unvorhergesehene Fehler)</i>
	1.	Das erneute öffnen des Dialoges und eine akustische Mitteilung an den Benutzer das eine der Eingaben nicht korrekt gewesen ist <i>(Dies ist der Fall falls eine der getroffenen Eingaben nicht dem vordefinierten Muster bzw. Typen entsprechen)</i>
Erweiterungen:	--	
Verweise:	Anforderung 2	

Name des Anwendungsfalls:	Create Worker	
Nummer:	Anwendungsfall 2	
Ziel:	Einen neuen „Worker“ über das Benutzerinterface erstellen	
Geltungsbereich:	Die Benutzeroberfläche	
Level:	Hauptfunktionalität	
Kategorie:	Primär	
Beteiligte Klassen/Objekte:	Benutzerinterface, Benutzer	
Vorbedingung:	Der Benutzer hat bereits das Benutzerinterface geöffnet	
Nachbedingung (Erfolg):	Der neu erstellte „Worker“ wird nach dem Erstellen in den aktuellen geladenen Kontext mit aufgenommen und beim Schließen der Oberfläche gespeichert. Sollte der „Worker“ weitere Eingabewerte verlangen wird der Benutzer dazu aufgerufen diese einzugeben, sollte das nicht der Fall sein wird der Prozess abgebrochen	
Nachbedingung (Fehlenschlag):	Der entstandene Fehler wird über eine Message Box dem Benutzer angezeigt. Sollte dies jedoch ein vorhergesehener Fehler sein wird eine erneute Eingabe verlangt	
Akteure:	Benutzer	
Auslösendes Ereignis:	Der Benutzer klickt im Dashboard doppelt auf eine leere Zeile	
Auszuführende Aktionen (Erfolg):	1.	Anzeigen eines Auswahl Dialogs zum wählen des „Worker“ Typen
	2.	Das Instanzieren eines neuen „Worker“ Objektes
	3.	Bei weiteren benötigten Parametern soll das Parameter Fenster geöffnet werden, sollten diese Eingaben fehlerhaft sein wird der Vorgang abgebrochen
Auszuführende Aktionen (Fehlenschlag):	1.	Das Öffnen einer Message Box mit dem abgefangenen Fehler <i>(Dies ist immer der Fall für unvorhergesehene Fehler)</i>
	1.	Das Abbrechen des Vorgangs bei einer fehlerhaften Konfiguration <i>(Dies ist der Fall falls eine der getroffenen Eingaben nicht dem vordefinierten Muster bzw. Typen entspricht)</i>
Erweiterungen:	--	
Verweise:	Konfigurationsvorgang siehe Anwendungsfall 1 Anforderung 2	

Name des Anwendungsfalls:	Start Workers
Nummer:	Anwendungsfall 3
Ziel:	Das Laden einer Konfigurationsdatei, welche die von Benutzer erstellten „Worker“ beinhaltet
Geltungsbereich:	Der Service
Level:	Hauptfunktionalität
Kategorie:	Primär
Beteiligte Klassen/Objekte:	Betriebssystem, Benutzer
Vorbedingung:	Der Benutzer hat bereits über Benutzerinterface „Worker“ erstellt und diese gespeichert
Nachbedingung (Erfolg):	Die geladenen „Worker“ werden vom Service verwaltet und gegebenenfalls gestoppt
Nachbedingung (Fehl Schlag):	Der entstandene Fehler wird in der Konsole ausgegeben, wenn diese sichtbar ist, ansonsten wird der Fehler ignoriert und gegebenenfalls das Programm beendet
Akteure:	Betriebssystem
Auslösendes Ereignis:	Die Applikation wird gestartet mit entsprechenden Parametern
Auszuführende Aktionen (Erfolg):	1. Überprüfen ob Datei existiert
	2. Lesen der Datei
	3. Deserialisierung des Inhaltes und Wiederherstellung der Objekte mit entsprechenden Daten
	4. Sicheres starten der geladenen Aufgaben
Auszuführende Aktionen (Fehl Schlag):	1. Das Ausgeben des Fehlers auf der Konsole (<i>Dies ist immer der Fall für unvorhergesehene Fehler</i>)
	1. Die angemessene Reaktion auf den Fehler, sollte diese die gesamte Funktionalität beeinträchtigen, wird das Programm frühzeitig beendet (<i>Dies ist der Fall falls eine der benötigten Eingabeparameter fehlt</i>)
Erweiterungen:	--
Verweise:	Anforderung 5

Klassendiagramme

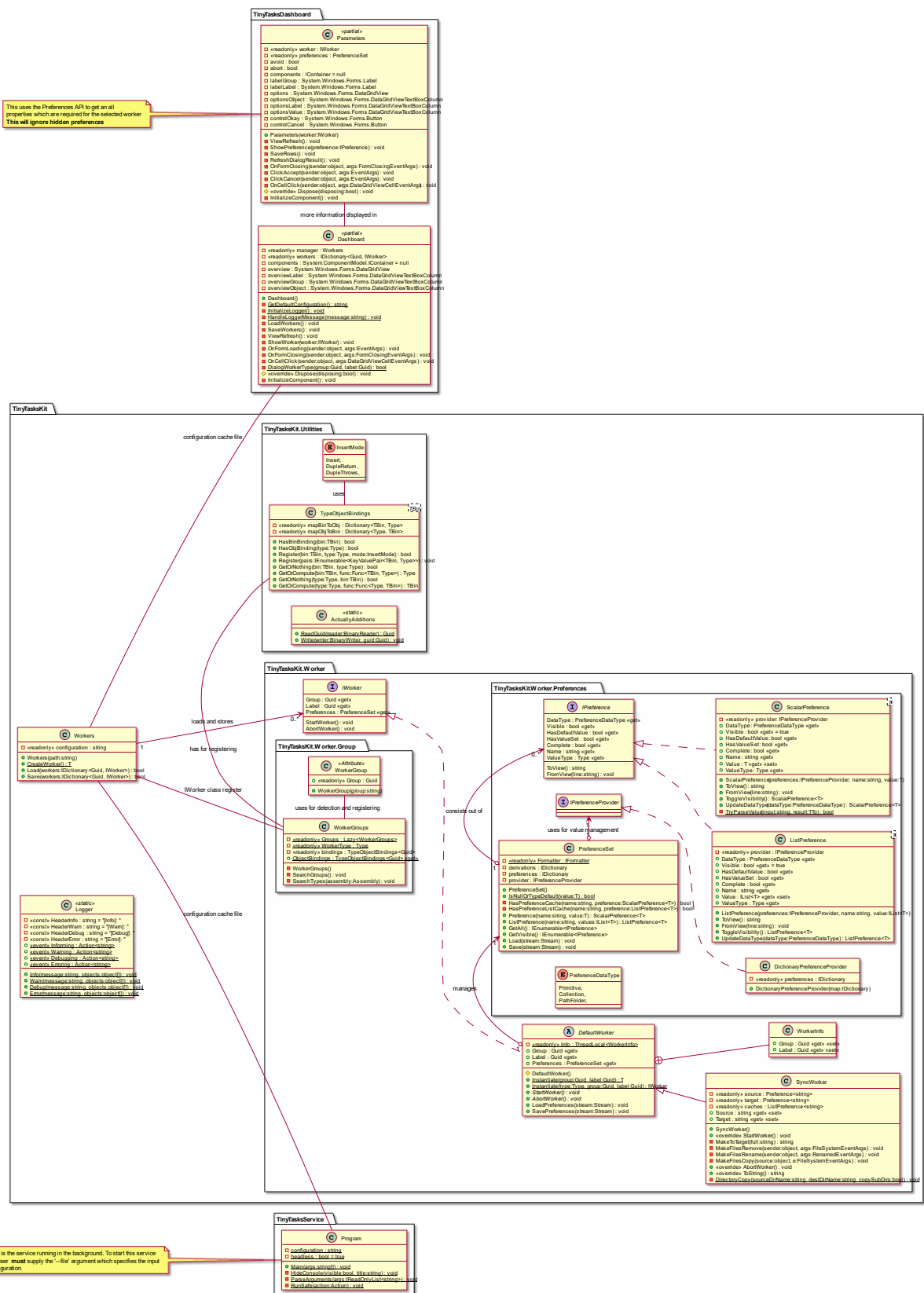


Diagramm ist auch auf GitHub zu finden, für eine genauere Betrachtung: <https://github.com/Louis9902/Sosse19-SE/blob/master/Diagramme/Klassen/TinyTasks.svg>

Objektdiagramme

Darstellung: Nach erfolgreichem einlesen der Konfigurationsdatei

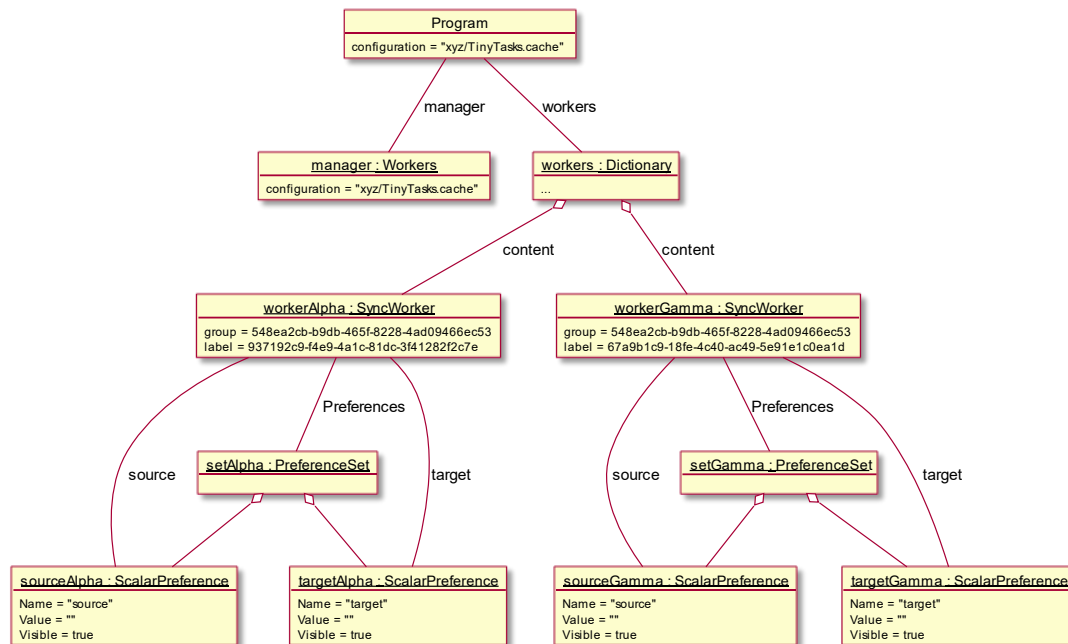


Diagramm ist auch auf GitHub zu finden, für eine genauere Betrachtung:

<https://github.com/Louis9902/Sosse19-SE/blob/master/Diagramme/Objekt/TinyTasks.svg>

Sequenzdiagramme

Darstellung: Einlesen der Konfigurationsdatei

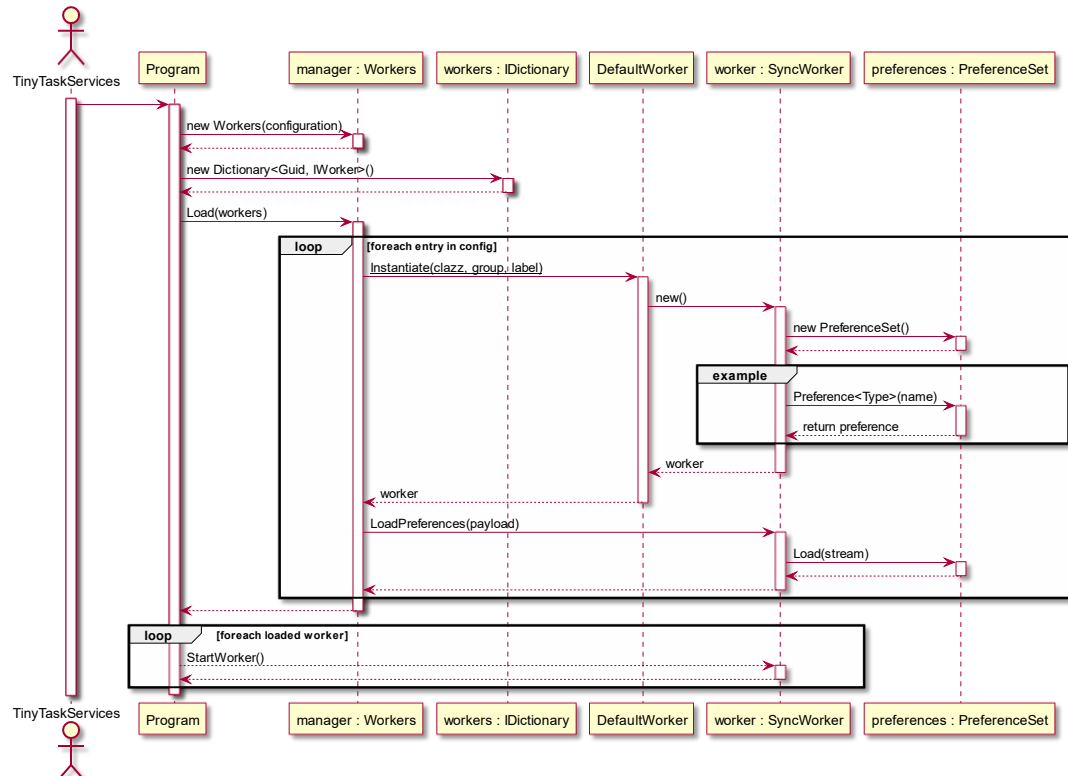


Diagramm ist auch auf GitHub zu finden, für eine genauere Betrachtung:

<https://github.com/Louis9902/Sosse19-SE/blob/master/Diagramme/Sequenz/TinyTasks.svg>

Zustandsdiagramme

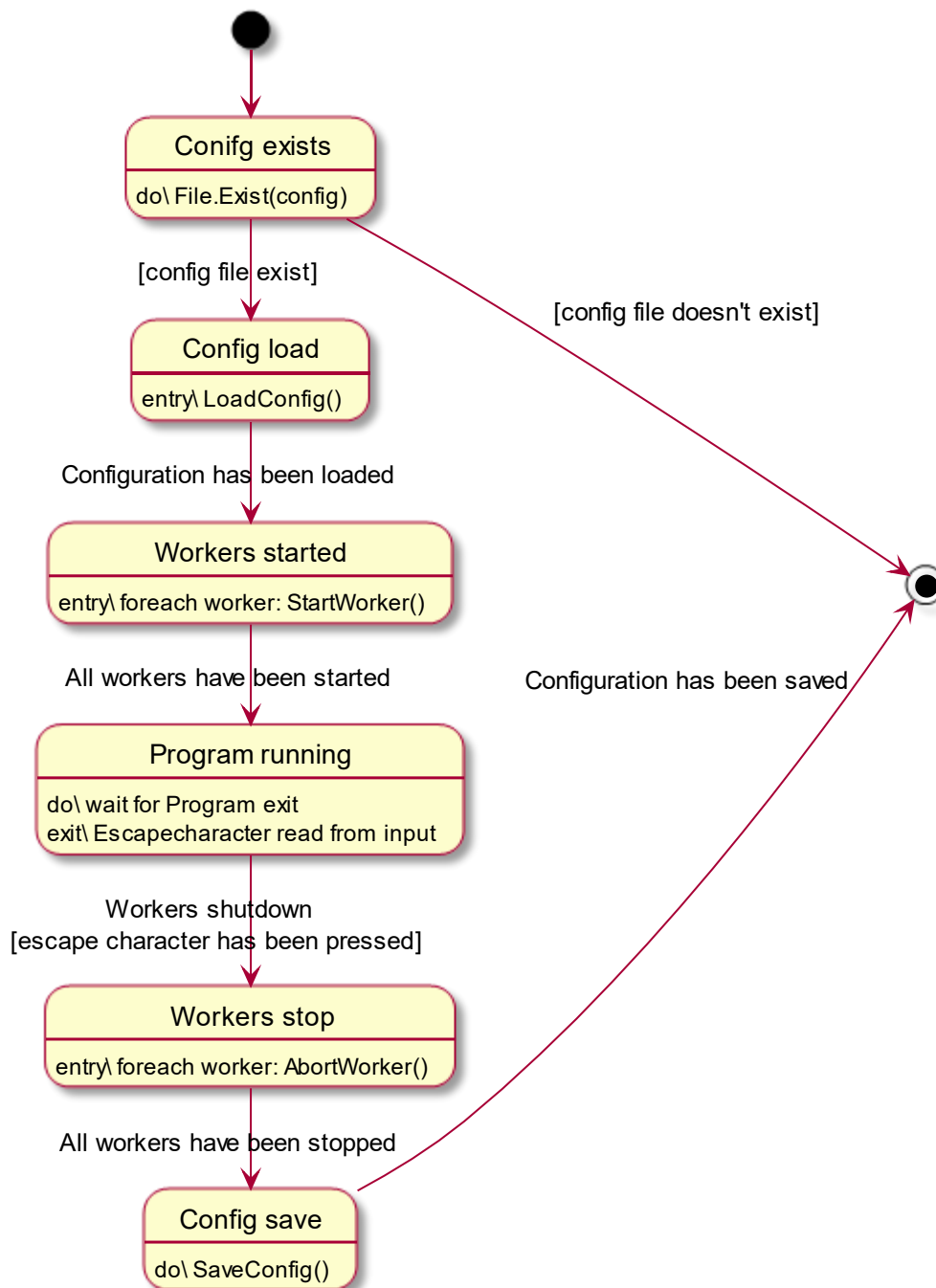


Diagramm ist auch auf GitHub zu finden, für eine genauere Betrachtung:

<https://github.com/Louis9902/Sosse19-SE/blob/master/Diagramme/Zustand/TinyTasks.svg>

Implementierung

Für die Implementierung des Projektes haben wir uns für das .NET Framework entschieden da dieses auf unterschiedlichen Betriebssystemen läuft. Hierdurch gibt es eine größere Flexibilität bei der Anwendung.

Toolkits wurden für dieses Projekte keine externen hinzugezogen lediglich die Standardbibliothek von .NET.

Erfüllte Anforderungen

- Anforderung 1: Wurde voll und ganz erfüllt (siehe Benutzeroberflächen Beispiele).
- Anforderung 2: Wurde ebenfalls erfüllt, wenn auch die aktuelle Implementierung nur einen Typ von Aufgabe zu lässt. Dies ist aber einfach zu erweitern da das gesamte System relative allgemein gehalten ist.
- Anforderung 3: Diese ist für manche Datentypen erfüllt, wobei das System zum Überprüfen ebenfalls wieder sehr offengehalten ist, was eine einfache Erweiterung zu einem späteren Zeitpunkt zu lässt.
- Anforderung 5: Wurde im vollen Umfang implementiert und ist weitestgehend unabhängig von der Laufzeit Plattform.

Nicht erfüllte Anforderungen

- Anforderung 4: diese ist prinzipiell möglich, jedoch nicht aktiv implementiert. Die nötigen Strukturen bestehen jedoch und die Implementierung kann erfolgen, wenn diese benötigt werden sollte.

Beiträge der Teilnehmer

Jens:

- Erste Implementierung der Benutzeroberfläche
- Implementierungserweiterungen der Benutzeroberfläche
- Ideenfindung für User Stories, Use Case (Diagramm/Tabelle), Klassendiagramm

Lukas:

- Erste Version des Klassendiagrammes
- Erstellen des Sequenzdiagrammes
- Erstellen des Zustandsdiagrammes
- Ideenfindung für User Stories, Use Case, Klassendiagramm

Louis:

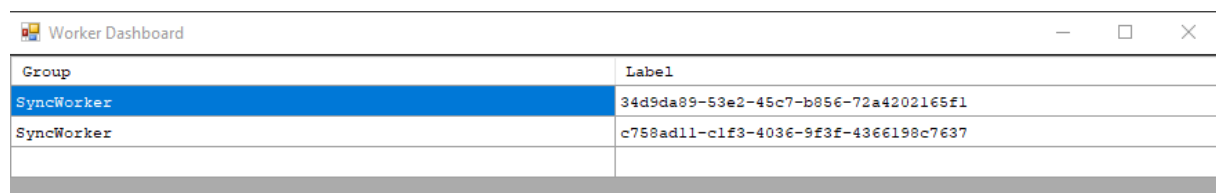
- Projekt Management
- Implementierung der Programmierschnittstelle und des Hintergrundprozesses
- Überarbeitung der ersten Version der Benutzeroberfläche¹
- Erstellen des Klassendiagrammes (Überarbeitung und Spezifizierung)
- Erstellen einer Überarbeiteten Version des Anwendungsfall-Diagramm + Tabelle
- Erstellen und Zusammentragen dieses Dokumentes

Moritz:

- Erstellen der Objektdiagramm
- Ideenfindung und Gruppendiskussion

¹ Aufgrund eines Missverständnisses bezüglich des Modularen Aufbaus für eine spätere Erweiterung

User Interface



Group	Label
SyncWorker	34d9da89-53e2-45c7-b856-72a4202165f1
SyncWorker	c758ad11-c1f3-4036-9f3f-4366198c7637

Abbildung 1: Allgemeine Übersichtsliste

Nach dem Öffnen der Benutzeroberfläche erscheint eine Übersicht der geladenen Aufgaben, diese wurden aus der im User Home liegenden Datei erstellt.

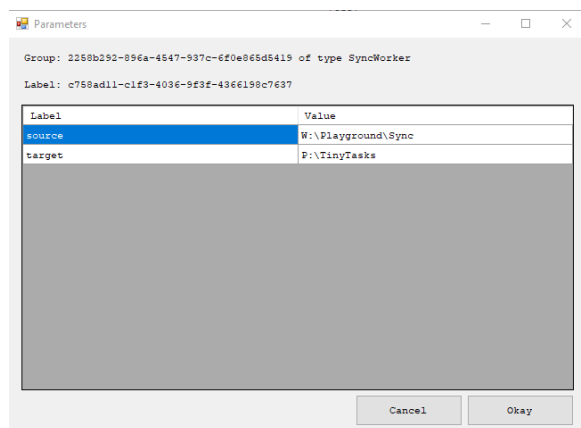


Abbildung 2: Menu mit weiteren Einstellungen

Bei dem Schließen der Benutzeroberfläche wird der Benutzer gefragt ob die Eingaben gespeichert werden sollen. Sollte der Benutzer dem zustimmen wird die in User Home liegende Datei mit den neuen Werten überschrieben.

Nun kann man durch einen Doppelklick auf ein Element in dieser Liste weitere Einstellungen vornehmen. Dabei ist zu beachten das je nach Einstellung der verlangten Werte eine unterschiedliche Eingabe erfolgen kann. In diesem Fall wird hier ein Ordner Pfad verlangt. Dafür öffnet sich beim Klicken auf das Werte Feld ein Ordner Auswahl Dialog.

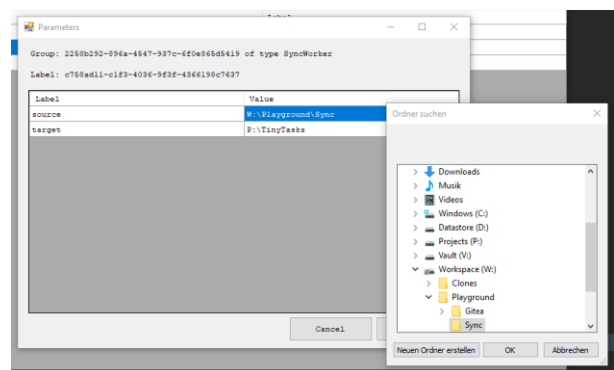


Abbildung 3: Der geöffnete Ordner wähl Dialog

Reflexion

Das Ergebnis unseres Projektes entspricht zu Teilen der Idee des Ursprünglichen Projektes. Jedoch gab es im Laufe der Entwicklung einige Hindernisse, welche zu einer Neuorientierung der eigentlichen Ziele geführt haben. So ist die Ursprüngliche Idee eines Backup Programmes relative schnell verworfen worden, da dies einen zu geringen Nutzen gegenüber des Entwicklungsaufwands hätte. Anstatt dessen ist ein Framework artiges Projekt entstanden, welches beliebig mit weiteren Aufgaben Typen erweitert werden kann.

Bei der Implementierung des Projektes sind dann an einigen Stellen auch Probleme aufgetreten bezüglich der Neuorientierung, da für die Erweiterbarkeit einige Strukturen anders definiert werden mussten als geplant.

Aufgrund dessen kam es bei der Arbeitsteilung im Projekt auch zu kleineren Missverständnissen, welche dazu geführt haben das manche Abschnitte erneuert werden mussten. Dies wurde von uns leider nicht optimal umgesetzt, da es gewisse Missverständnisse gab. Um dies in der Zukunft zu vermeiden sollten wir in der Planung des Projektes auch auf allgemeine Standards eingehen. An diese müssten sich dann alle Projektmitglieder halten.

Diese Standards sollten vor allem eine genaue Arbeitsaufteilung festlegen, sowie gewährleisten, dass ein einheitliches Quellcodebild entsteht.

Für ein zukünftiges Projekt müsste man die Versionsverwaltung Git besser nutzen und ein damit verbundenes Review System benutzen, um Änderungen gemeinsam zu beschließen. Dies ist bei unserem Projekt aufgrund der geringen Git Kenntnisse leider nicht möglich gewesen.