Karlsruher Institut für Technologie
Institut für Theoretische Informatik
Prof. Dr. Bernhard Beckert
Dr. Mattias Ulbrich, Alexander Weigl

Praxis der Softwareentwicklung, WS 2016/17

Aufgabenbeschreibung

Eine kurze Bemerkung vorab

Dieses PSE-Projekt ist *Ihr* Projekt, in dem *Sie* eine Software-Anwendung konzipieren, entwerfen, umsetzen, testen und ausliefern werden.

Sie müssen und sollen selbst kreativ entscheiden, wie Sie *Ihr* Produkt gestalten. Dieses Dokument ist daher nicht als Katalog von Aufgaben zu verstehen, die Punkt für Punkt abgearbeitet werden müssen, um das Modul zu bestehen. Sie erhalten hier von uns Hinweise bezüglich unserer Vorstellungen und Erwartungen.

1 Hintergrund

Im Zeitalter von Industrie 4.0 sollen Industrieanlagen individuell konfigurierte Produkte vollautomatisch und autonom herstellen können. Diese Anforderung lässt die Software in Anlagensteuerungen zunehmend komplexer werden. Dennoch muss sie Sicherheitsaspekte, wie zum Beispiel das korrekte Verhalten im Falle eines Nothalts, gewährleisten, um Leib und Leben der Angestellten zu schützen. Formale Verifikation kann hier eingesetzt werden, um bereits während der Entwicklung zu beweisen, dass Software definierte Sicherheitsspezifikationen einhält.

Automatisierungssteuerungen lesen in zyklischen Abständen Sensorwerte ein und berechnen daraufhin Aktuatorwerte. In die Berechnung können die Ergebnisse vorangegangener Zyklen einfließen. Daher müssen Sicherheitsspezifikationen über *Abfolgen* von Ein- und Ausgaben formuliert werden, was ihren Umfang und ihre Komplexität erhöht und damit ihre Verständlichkeit verringert.

2 Ziel

Ziel des Projekts ist es, eine grafische Entwicklungsumgebung zu entwickeln, die es dem Automatisierungsingenieur erlaubt, komfortabel komplizierte Verhaltens-Spezifikationen von Anlagensoftware zu erstellen, zu bearbeiten und zu analysieren. Fehler in Spezifikationen sollen (in Form von Gegenbeispielen) dargestellt werden.

3 Aufgabenstellung

Ihr Programm soll die Bearbeitung von Programmen und die Eingabe von Spezifikationen erlauben. Die Routine, die für ein Programme überprüft eine Spezifikation erfüllt, ist *nicht* Teil Ihrer Aufgabe, sondern sie wird Ihnen als Schnittstellt zur Verfügung gestellt, die Sie ansteuern können.

Die Software soll möglichst robust gegen fehlerhafte und unerwartete Eingaben sein. So sollen z.B. zyklische Referenzen oder unerfüllbare Bedingungen durch Ihr System abgefangen werden, damit der Benutzer frühzeitig auf potentielle Probleme hingewiesen wird.

3.1 Minimale Leistungsmerkmale

Das User-Interface soll aus drei Komponenten bestehen:

1. Ein Code-Editor für die Programmierung in der Programmiersprache *Structured Text* mit Syntax-Highlightning.
2. Eine tabellarischen Ansicht für Spezifikation, die auch Bearbeitung zulässt, die an bekannte Tabellenprogrammen (Excel) orientiert.
3. Eine grafische Ansicht einer Spezifikation als *Timing-Diagramm*.

Darüber hinaus benötigen Sie folgende Funktionalitäten:

1. Ansteuerung der Verifikationsmaschine
2. Eine grafische Darstelung von Gegenbeispielen als Timing-Diagramme.

Software-Architektur. Für ein Projekt, in dem Korrektheit von Algorithmen eine extrem hohe Bedeutung hat, ist eine saubere Trennung zwischen kritischen und unkritischen (z.B. GUI, aber auch Berechnung anderer Daten wie Statistiken) Komponenten von immenser Bedeutung. Entwerfen Sie austauschbare Komponenten mit minimalen, wohldefinierten Schnittstellen.

3.2 Zusatzmerkmale über die Mindestanforderungen hinaus

Es gibt eine Reihe von weiteren sinnvollen Merkmalen. Entscheiden Sie selbst und erörtern Sie im Pflichtenheft, welche davon für Sie in Frage kommen. Grundsätzlich gilt: Was im Pflichtenheft erwähnt wird, muss später auch umgesetzt werden.

Mögliche Zusatzmerkmale sind:

- Übersichtansicht (Overview) über die Toplevel-Elemente im Source Code.
- Autocompletion
- Konsistenzprüfung für Source Code
  - Zugriff auf definierte Variablen
  - Typkonformität innerhalb von Audrücken
- Konsistenzprüfung der Spezifikation
  - Zyklenfreiheit
  - Typkonformität
  - Konformität zum Source Code
- Erzeugen von konkreten Instanzen von Test-Tabellen (beschränkter Funktionsumfang)
- Editieren in Timing-Diagrammen
- Simulation des Programs
- weitere Merkmale, die *Sie* sich für das Produkt als sinnvoll vorstellen.

3.3 Abgrenzung an Structured Text

Um dieses Projekt im Rahmen zu halten, schränken wir den Sprachumfang von Structured Text ein. Achten Sie weiterhin auf die Erweiterbarkeit Ihres System, um spätere Erweiterungen zu unterstützen.

Wir schränken Structured Text wie folgt:

1. Wir erlauben nur Integer (signed/unsigned), Bit-Datentypen (BOOL, BYTE, WORD, DWORD, LWORD) und Enumerations.
2. Wir schränken die erlaubten Funktion auf die Grundrechenarten ein.
3. Die Programme sind immer geschlossen, d. h. alle verwendeten Blöcke sind vorhanden und liegen in einer Datei. (Kein include oder Modulsystem).

3.4 Technische Eckpunkte

- Programmiersprache: Java
- GUI-Bibliothek: Swing, JavaFX
- Versionsverwaltung
- Entwurfs- und Entwicklungsumgebung, z.B. ArgoUML oder Papyrus,
- Unit-Test-Rahmenwerk: JUnit
- Dokumenterstellung: Ihre Entscheidung, bevorzugt LaTeX (Abgabe aber immer als PDF)

4 Organisatorisches

4.1 Bewertung

Die Benotung Ihres Systems richtet sich nach folgenden Kriterien:[1]

- Qualität aller abgegebenen Dokumente
- Qualität der Kolloquien
- Qualität der Abschlusspräsentation
- Erfüllen der minimalen Leistungsmerkmale (s.o.)
- *Sinnvolle*[2] Erweiterungen über diese Merkmale hinaus
- Qualität des erstellten Programms (das schließt u.A. Benutzbarkeit und Robustheit ein)

Die Gesamtnote errechnet sich dann nach der im Modulhandbuch genannten Gewichtung:

- Pflichtenheft 10%
- Entwurf 30%
- Implementierung 30%
- Qualitätssicherung 20%
- Abschlusspräsentation 10%

Nach jeder Phase findet (im Rahmen der regelmäßigen Treffen) ein Kolloquium statt, in dem die Ergebnisse der Phase *selbstständig* (in rund 20 Minuten) vorgestellt werden. Jedes Team-Mitglied muss einmal präsentieren.[3] Die Benotung jeder einzelnen Phase wird im entsprechenden Kolloquium besprochen.

[1] Diese Liste hat keine Reihenfolge, die einer Gewichtung entspricht. Es gibt sicherlich weitere Punkte, die als selbstverständlich gelten und sich bei Nichterfüllen negativ auswirken.

[2] Sie tun Sich selbst und dem Projekt nichts Gutes wenn Sie sich zu sehr verkünsteln.

[3] Dennoch müssen sich selbstverständlich *alle* Team-Mitglieder in *allen* Phasen einbringen.

4.2 Treffen

Treffen finden wöchentlich im Raum 211 statt. Auch wenn gerade kein Kolloquium ansteht, raten wir Ihnen dringend, jede Woche über Ihren Fortschritt zu berichten. Nur dann können wir Hilfestellung geben. Sorgen Sie daher bitte auch dafür, dass alle erforderlichen Dokumente rechtzeitig per E-Mail abgegeben sind. Für den Abschluss jeder Phase gilt ein *strikter* Abgabeschluss um 9 Uhr am Montag nach der Beendigung der Phase.

Darüber hinaus sollten Sie sich natürlich auch noch regelmäßig in der Gruppe besprechen. Sie können dazu Raum 211 nutzen falls er frei ist (vor der Tür hängt ein Kalender mit Reservierungen; bitte nehmen Sie Rücksicht auf die Einträge und beachten Sie dass Belegungen von Mitarbeiter und -innen Priorität haben).