Technische Universität Berlin
Software Engineering for Embedded Systems Group – Prof. Dr. Sabine Glesner
www.pes.tu-berlin.de Secr. TEL 12-4 Ernst-Reuter-Platz 7 10587 Berlin

Softwaretechnik und Programmierparadigmen WiSe 2014/2015

Prof. Dr. Sabine Glesner
Joachim Fellmuth joachim.fellmuth@tu-berlin.de
Dr. Thomas Göthel thomas.goethel@tu-berlin.de
Lydia Mattick lydia.mattick@tu-berlin.de
Tutoren

Hausaufgabenblatt 2 **Ausgabe: 09.01.15** **Abgabe: 25.01.15**

Euer Auftraggeber war sehr zufrieden mit den bisherigen Modellierungen der Software für die Verwaltung der wissenschaftlichen Konferenzen und Reviews von dafür eingereichten Papers. Um eine möglichst hohe Qualität zu erhalten, fordert er die Erstellung und Analyse weiterer Modelle, um Fehler möglichst frühzeitig finden bzw. ausschließen zu können.
Als Erinnerung habt ihr euch erneut die Anforderungen an die Software erarbeitet:

Ein Nutzer kann sich unter Angabe seines Namens und Kontaktdaten in der Software registrieren. Nutzer dürfen Vorsitzende von Konferenzen, Autoren und/oder Reviewer von Papers sein.

Autoren können ihre Papers in Textform hochladen. Dazu muss zusätzlich der Titel, das Verfassungsjahr und das Abstract eingegeben werden. Die Seitenzahl wird für den Inhaltstext automatisch bestimmt und gespeichert. Der Inhalt des Papers darf nachträglich angepasst werden. Ein Paper darf bei nur einer Konferenz bis zur ersten Deadline eingereicht werden, wenn es die vorgesehene maximale Seitenzahl nicht übersteigt und noch nicht veröffentlicht ist. Ein eingereichtes Paper darf zwischen der ersten und zweiten Deadline und, falls es akzeptiert wurde, nach Abschluss der Konferenz nicht angepasst werden.

Nach Ablauf der ersten Deadline kann die Vorsitzende den Papers Nutzer als Reviewer zuweisen. Ein Reviewer darf nicht Autor desselben Papers sein. Die Auswahl der Reviewer und ihre Anzahl liegt im Ermessen der Vorsitzenden und wird vom System nicht modelliert. Bis zur zweiten Deadline hat ein Reviewer Zeit, ein Review für ein ihm zugewiesenes

Paper einzureichen. Es gibt außerdem die Möglichkeit, ein Review abzulehnen, um der Vorsitzenden zu signalisieren, dass ein neuer Reviewer zugewiesen werden sollte. Ein Review beinhaltet einen Text und zusätzlich die Bewertung accepted, borderline oder rejected.

Sobald die Vorsitzende eine Konferenz abschließt, werden vom System mehrere Aufgaben erledigt: Alle Papers, welche die Zusagekriterien erfüllen, bleiben mit der Konferenz verknüpft. Die Autoren dieser Papers werden über die Zusage benachrichtigt. Alle übrigen Papers werden von der Konferenz getrennt. Betroffene Autoren erhalten dann die Nachricht, dass ihr Paper bei einer anderen Konferenz eingereicht werden darf. Die von getrennten Konferenzen erhaltenen Reviews werden nicht vom Paper getrennt, beeinflussen aber nicht die Bewertung bei der momentan verknüpften Konferenz.
Nach dem Konferenztermin werden alle akzeptierten Papers veröffentlicht.

Zusätzliche Anforderung:
Jedes Paper muss mindestens einen Reviewer und ein Review zugewiesen bekommen.

1. Aktivitätsdiagramm

Modelliert den Lebenszyklus eines Papers als Aktivitätsdiagramm. Dabei sollen alle möglichen Aktionen der Nutzer erfasst werden.

- Der workflow eines Papers beginnt mit der Existenz seines Inhaltes.
- Alle erforderlichen Informationen können als Systemabfrage modelliert werden.
- Da die Lebensdauer eines Papers über mehr als einen Tag geht, kann mittels der Variable `TODAY` auf den aktuellen Tag zugegriffen werden.
- Wenn eine Konferenz ausgewählt wurde, kann mittels `first deadline` und `second deadline` auf die Deadline-Zeitpunkte zugegriffen werden.
- Zeitpunkte können mittels $==$, $<$, $>$ verglichen werden.
- Folgende Aktionen müssen vorkommen:
  - `create paper`
  - `add conference`
  - `change content`
  - `add reviewer`
  - `add review`
  - `publish paper`

2. CTL

Überführt euer Aktivitätsdiagramm in eine Kripke Struktur. Für die Kripke Struktur gilt: Die Aktionen werden als Zustände interpretiert, die Kanten als Kanten. Alle weiteren Aktivitätsdiagramm-spezifischen Modellierungselemente werden entweder weggelassen oder als Zustand interpretiert.
Eure Beschriftungsfunktion soll folgende Zustände mit den jeweiligen Annotationen versehen:

- `create paper` : **create**
- `add conference` : **addConference**
- `change content` : **changeContent**
- `add reviewer` : **addReviewer**
- `add review` : **addReview**
- `publish paper` : **publish**

(a) Euer Auftraggeber möchte nun, dass ihr das System bzgl. folgender Anforderungen überprüft:

(i) Wenn ein Reviewer hinzugefügt wurde, wird irgendwann auch mindestens ein Review hinzugefügt.
(ii) Solange ein Paper noch nicht erstellt wurde, sollen die anderen aufgeführten Aktionen nicht ausgeführt werden können.
(iii) Jedes Paper kann nur einmal erstellt werden.
(iv) Jedem Paper kann irgendwann eine Konferenz zugeordnet werden.
(v) Jedem Paper kann irgendwann ein Review zugeordnet werden, nachdem ihm eine Konferenz zugeordnet wurde.
(vi) Jedes Paper kann irgendwann veröffentlicht werden, nachdem ihm eine Konferenz, ein Reviewer und ein Review zugeordnet wurde.
(vii) Solange ein Paper nicht veröffentlicht wurde, kann ihm immer wieder eine Konferenz hinzugefügt werden.
(viii) Immer wenn dem Paper ein Reviewer zugewiesen wird, kann als nächste Aktion ein Review zugewiesen werden.

Überführt die in Textform gegebenen Anforderungen in CTL-Formeln.

(b) Gebt an, welche der Formeln von eurem Zustandsübergangssystem (Kripke-Struktur) erfüllt wird. Begründet eure Antworten.

(c) Überführt die Kripke Struktur in einen jABC-Graphen und überprüft eure Antworten in jABC mit GEAR.

Hinweis: Wir empfehlen für die Modellierung in jABC für diese Aufgabe als Knotentypen ausschließlich Noop- und ChooseRandomBranch-SIBs zu verwenden.
Abzugeben sind die Formeln mit den Begründungen und die jABC-Dateien als exportiertes Projekt mit dem Namen *HA02_Kripke*.

3. jABC-Modellierung

Modelliert ein einfaches Login-System als **Service Logic Graph** in **jABC**. Folgende Anforderungen werden an euer SLG gestellt:

- Ein Nutzer kann sich in eurem System unter Angabe von *username* und *password* registrieren.
- Der Nutzer wird über den Erfolg bzw. Misserfolg der Registrierung informiert.
- Zwei Nutzer dürfen sich nicht den gleichen Benutzernamen teilen.
- Ein Nutzer darf kein leeres Passwort haben.
- [optional] Das Passwort muss mind. fünf Zeichen lang sein.
- [optional] Das Passwort muss mind. einen Buchstaben und eine Zahl enthalten.
- Ein Nutzer kann sich in eurem System unter Angabe seiner Registrierungsdaten einloggen.
- Der Nutzer wird über den Erfolg bzw. Misserfolg des Logins informiert.
- Bei Misserfolg wird der Nutzer erneut aufgefordert seine Registrierungsdaten anzugeben.
- Ein registrierter Nutzer bleibt über die Sitzung hinaus registriert. Dazu soll euer SLG die Nutzerdaten in einer lokalen Textdatei *Logindata.txt* gespeichert halten.
- Das Passwort darf nur verschlüsselt gespeichert werden.

Hinweise zur Modellierung:

- Für die Bearbeitung dieser Aufgabe stehen euch auf ISIS weitere SIBs zur Verfügung. Ladet euch dazu die jar-Datei **ha02-sibs-1.0** herunter und verschiebt diese in euren Projektordner. Fügt nun eurem Projekt den SIB-Pfad zu der jar-Datei durch Rechtsklick auf euer Projekt im Projekt-Fenster hinzu. Alternativ könnt ihr den SIB-Pfad auch über den Menüeintrag *Project* hinzufügen.
  Die jar-Datei **ha02-sibs-1.0** enthält die SIBs *SplitString*, *JoinString*, *EncryptString* und *ContainsCharSequence*.

- Für die Bearbeitung dieser Aufagbe steht euch auf ISIS das Beispielprojekt **HelloWorldDemo** zur Verfügung. Dieses zeigt die Verwendung der SIBs *SplitString* und *JoinString*. Außerdem zeigt es Beispielhaft den Einsatz von IO-SIBs. Um das Beispielprojekt zu benutzen, ladet euch dazu die Datei **HelloWorld** herunter und importiert das Projekt durch Rechtsklick in euer Projekt-Fenster. Alternativ könnt ihr das Projekt auch über den Menüeintrag *Project* importieren.

- Bei aufkommenden Fragen über die Funktionsweise von SIBs oder jABC informiert euch bitte zunächst unter

  `http://jabc.cs.tu-dortmund.de/manual/index.php/FAQ_%28Frequently_Asked_Questions%29` .

- Es ist ratsam das Plugin **Tracer** exzessiv bei der Modellierung einzusetzen. Dieses Plugin erlaubt euch zum einen die schrittweise Ausführung eures Modells und zum anderen die detaillierte Ansicht des Zustandes eures Systems.

- Bei der Speicherung der Nutzerdaten ist zu beachten, dass sich ein Benutzername eindeutig einem Passwort zuordnen lässt. Eine möglich Form der Abspeicherung wäre "Benutzername:Passwort".

- Bei der Verwendung des SIBs *EncryptString* ist zu beachten, dass die Verschlüsselung gerichtet und deterministisch ist. D.h. zum einen kann ein verschlüsselter String nicht wieder entschlüsselt werden, und zum anderen wird ein gegebener String immer auf den gleichen verschlüsselten String abgebildet.

- Bei der eventuellen Verwendung des SIBs *IterateElements* ist zu beachten, dass die Variable *iterator* vom System nur nach einem kompletten Durchlauf der Liste gelöscht wird, nicht aber, wenn ihr vorzeitig den SIB *IterateElements* verzweigt.

- Die Abgabe dieser Aufgabe soll als exportiertes Projekt *HA02_LOGIN* erfolgen.

Generelle Hinweise und Abgabeformalitäten

- Wir vergleichen eure Abgabe **nicht** mit einer Musterlösung, da es meist mehrere gültige Lösungsmöglichkeiten gibt.
- Fügt eurer Abgabe eine Titelseite mit euren Namen, dem Namen eures Tutors und eurem (offiziellen) Tutoriumstermin hinzu.
- Wiederholer von „MPGI 3 - Softwaretechnik“, welche ihre Hausaufgabennote behalten wollen, schreiben ihren Namen / ihre Matrikelnummer **nicht** auf die Abgabe.
- Diese Hausaufgabe ist eine Prüfungsleistung! Wenn ein (gruppenübergreifendes) Plagiat entdeckt wird, resultiert dies in einem Fehlversuch.
- Die Abgabe erfolgt pünktlich über Isis, als .pdf, .asta und exportierte jABC-Projekte.