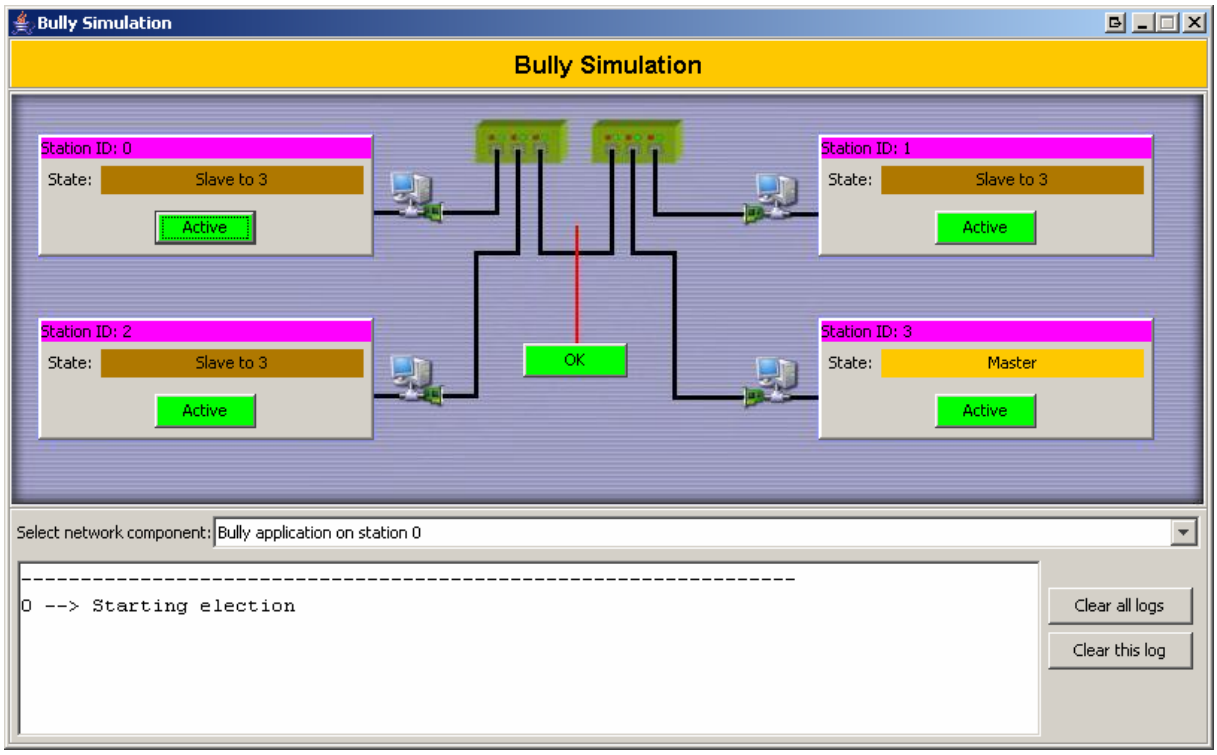
Ü Netzwerke und verteilte Systeme	Übung #9	WS 2005/2006
Name:	Matr-Nr:	
Abgabe: 10.1.2006		Gruppe:

Einführende Informationen zur Übung

Für die folgenden Übungsaufgaben wird ein Java-Framework verwendet, mit welchem eine konkrete Netzwerk-Infrastruktur nachgebildet wird (siehe Übung 3!). Das Framework selbst ist nicht an spezielle Protokolle, Geräte oder ähnliches gebunden und ermöglicht die Simulation von Datenübertragung mittels einer ISO/OSI 7 Schichten-Infrastruktur. Auf jeder Schicht können eine Reihe so genannter „Filter“ eingehängt werden, die die Aufgaben der jeweiligen Schicht übernehmen. In dieser Übung liegt der Schwerpunkt auf Layer 7, also dem Application Layer.

Vorgegebenes Szenario

In dem vorgegebenem Übungs-Szenario existieren vier Rechner, welche mittels zweier Hubs (Layer 1) in einem Broadcast-Netz miteinander verbunden sind und somit ein Netzwerk bilden. Die Rechner (stations) haben eine eindeutige (Layer 2) Identifikationsnummer (sie Layer 3-6 werden nicht verwendet).
Die Applikationen führen keine eigentliche Arbeit durch, sondern sind nur in der Lage, das Bully-Protokoll auszuführen um einen "Master" zu bestimmen.
Nun ergibt es sich, dass die beiden Hubs mit einem fehlerhaften Kabel verbunden sind, und sich dieses (auf Knopfdruck) unterbrechen lässt (Benutzer stecken aus, Kabel bricht, etc.).



Übungsmodalität

- Download des Frameworks und der Dokumentation von der Übungshomepage
- Abgabe des Programmcodes (nur die neue Klasse(n); Quellcode + compiliert) zusätzlich via E-Mail an sonntag@fim.uni-linz.ac.at (Deadline: 10.1.2006, 15:30 Uhr; Subject: "NuVS-Uebung 9")
- Fragen und Probleme bzgl. der Übung? Mail an sonntag@fim.uni-linz.ac.at !

Tipps und Hinweise

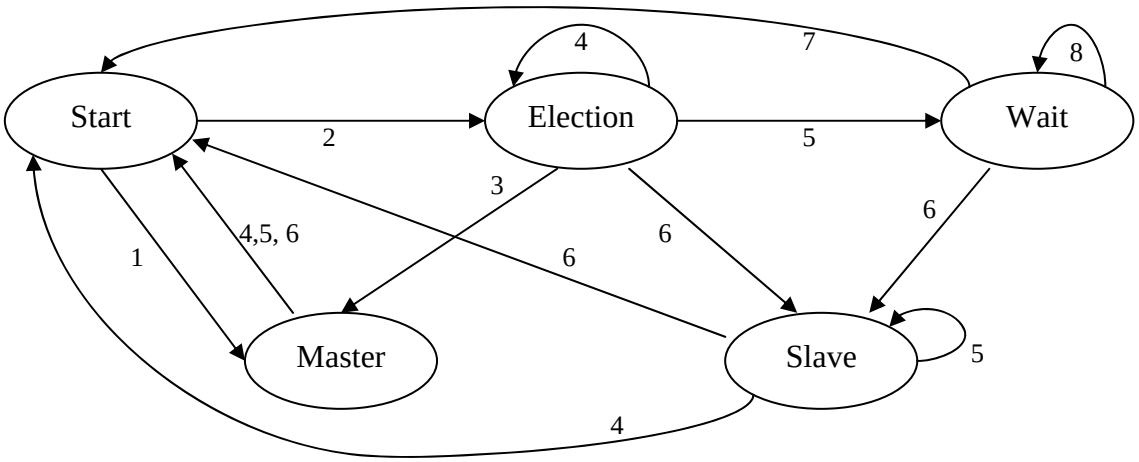
- Zu startende Klasse:
  `at.jku.fim.datalinksimulation.scenario.Bully.BullySimulationApplication`
- Die einzelnen Komponenten und Stacks führen Ihr eigenes Log. Sie sind ein erster Anlaufpunkt für die Fehlersuche. Loggen Sie in den Übungsbeispielen ebenfalls alle wichtigen Aktionen, Zustände (z.B. den internen Zustand des Bully-Protokolls) und Ergebnisse mit. In den Logs können Sie überprüfen, ob Ihre Implementierung auch tatsächlich funktioniert.
- Sehen Sie regelmäßig auf der Übungshomepage nach, ob es neue Hinweise oder Versionen des Frameworks gibt.

Beispiel 23-25

Programmieren Sie den Bully Algorithmus (siehe Skriptum) in Form der Klasse "BullyApplication.java". Diese führt den Algorithmus in einem eigenen Thread aus und zeigt den jeweiligen Zustand über das GUI an. Über dieses GUI kann auch ein Prozess beendet werden, sodass ev. ein neuer Master ausgewählt werden muss. Wird ein Prozess neu gestartet, so wird von diesem eine Election ausgelöst. Sonstige Tätigkeiten (=eigentliche Verarbeitungen) werden nicht durchgeführt!
Einige Hinweise:

- Der "Zustandsautomat" des Algorithmus ist als Hilfe unten angeführt. Hierbei kommen zwei Timeouts vor.
- Timeouts müssen auch dann funktionieren und beachtet werden, wenn zwischenzeitlich Nachrichten eintreffen (welche in der Zwischenzeit aber sehr wohl zu bearbeiten sind!). Nicht mehr aktuelle Timeouts dürfen jedoch nicht beachtet werden (z.B. Timeout von "Election" muss bei Wechsel in den Zustand "Wait" durch das neue, und neu zu startende, Timeout ersetzt werden).
- Jeder Slave soll alle Sekunden einen Ping an den Master schicken um zu überprüfen, ob dieser noch lebt und erreichbar ist. Ist dies nicht der Fall, wird eine neue Election ausgelöst. Dies ist im Zustandsdiagramm **nicht** angeführt!
- Die Methode "stop()" der Klasse Thread darf nicht verwendet werden: Als Ersatz dient "terminate()", welche die laufende Applikation so schnell wie möglich auf ordnungsgemäße Weise (Synchronisation, Kommunikation zwischen Threads, ...) beenden muss. Hierbei wird der Thread tatsächlich beendet und nicht nur auf irgendeine Weise "deaktiviert".
- Synchronisation ist erforderlich: Nachrichten werden in einem anderen Thread in eine Warteschlange (hier erforderlich) eingestellt, als sie von der Applikation ausgelesen werden.
- Sie brauchen keine zusätzlichen Layer (z.B. 3) programmieren: Verschicken Sie direkt selbst definierte Layer 2 Nachrichten.
- Die Schnittstelle der Klasse darf auf keinen Fall verändert werden: Sie dürfen nur private/protected Felder/Methoden hinzufügen. Die öffentlichen Teile müssen gleich bleiben.

- Die Methode "setGui(BullyPanel gui)" dient nur zur Übergabe des "BullyPanel" an ihre Applikation zur Visualisierung, damit sie anschließend mit "setState(state)" bei einer Zustandsänderung den Zustand der Station im Benutzerinterface aktualisieren können.



i ..... Eigene Stationsnummer (0...N-1)
N ..... Anzahl der Stationen
master .. Nummer des Masters

1) i==N-1 / Sende "Coordinator" an 0..N-2
2) i<N-1 / Sende "Election" an i+1..N-1
3) 3 Sek. / Sende "Coordinator" an 0..i-1
4) "Election" von j / Sende "OK" an j
5) "OK" von j / -
6) "Coordinator" von j / master=j
7) 6 Sek. / -
8) "Election" oder "OK" von j / -

Beispiel 26

Beantworten Sie folgende Fragen schriftlich:

- Was passiert, wenn durch Kabelbruch die beiden Hubs getrennt werden?
- Was passiert, wenn dieser Kommunikationsfehler wieder behoben wurde?
- Wann wird das dadurch entstehende Problem automatisch gelöst?
- Wie könnte man dieses Problem sonst noch lösen?
- Warum muss ein Master (bzw. allgemein jeder Knoten), der eine "Election" Nachricht bekommt, selbst eine Election starten? Es würde doch eigentlich ausreichen, dem anfragenden (oder allen mit niedrigerer Nummer) mitzuteilen, dass man selbst der Master ist!