Projektplan für die Diplomarbeit „Entwicklung von Objekt und Petri Netzen im Rahmen des CS-Frameworks“

Alexander Schwinn

23. April 2007

1 Initiierung

Zum Kontrollieren verteilter Geräte und Steuerung mit Hilfe parallel laufender Prozesse an der GSI wurde vor geraumer Zeit „ObjectVIEW“, eine kommerzielle, objektorientierte Erweiterung für LabVIEW genutzt. LabVIEW als Programmiersprache bietet sich an, da parallele Prozesse damit unkompliziert verwirklicht werden können.

Da ObjectVIEW für den Anwendungs-Rahmen an der GSI nicht ausreichend war, beschloss man ein eigenes, objektorientiertes Kontrollsystem-Framework zu entwickeln, das CS-Kontrollsystem.

CS ist ein objektorientiertes, ereigniss-gesteuertes verteiltes Kontrollsystem mit einem sicheren Attributzugriff. Die größten Unterschiede zu ObjectVIEW is ein schneller Objektattribut-Zugriff, und die Möglichkeit Anwendungen in einem Netzwerk einfach zu verteilen.

ObjectVIEW bietet mit Hilfe vorhandener Basisklassen die Möglichkeit Objekt-, und Petri-Netze zu verwirklichen. Beide dienen als ausführbares Designdokument, Objektnetze zum Starten einer verteilten Anwendung, zum Zuweisen der Kommunikationskanäle (Ports) der verteilten Objekte und zur Ausfallkontrolle. Petrinetze in ObjectVIEW hingegen dienen zum Modellieren von Abhängigkeiten zwischen Objekten, bishin zur Implementierung von komplexen Prozessen.

2 Definition

2.1 Grundlegene Anforderungen

Sowohl Objekt-, wie auch Petrinetze sind ausfürbare Designdokumente. Der User hat somit die Möglichkeit auf einen Blick die Funktionsweise eines Systems zu erfassen und gleichzeitig die Möglichkeit diese an seine Bedürfniss an zu passen.

2.1.1 Objektnetze

Ein CS-Objektnetz besteht hauptsächlich aus Methoden, welche Objekte instantiieren (diese werden im weiteren „Launch.vi´s “genannt) und Methoden, welche den Datenfluss zwischen Objekten grafisch Darstellen (sog. „Observer.vi´s “). Hierbei wird der Datenfluss duch die Verdrahtung der einzelnen Ports definiert. Will man ein CS-Objekt in einem Objektnetz verwenden, so lässt man dessen Klasse von der Objektnetz-Basisklasse erben. Somit erhällt dieses Objekt ein eigenes Launch.vi. Das Netz selbst ist Bestandteil eines Objektes, welches von der Netz-Klasse geerbt hat.

- Das Lauch.vi soll beim Start des Objektnetzes eine Instanz der Klasse auf einem beliebigen CSSystem erzeugen.
- Weiterhin soll die Möglichkeit bestehen, Launch.vi´s untereinander zu verbinden. Über die Verbindung der Ports wird die URL einer Datenquelle an die Datensenken vermittelt, OUT-Port ist Publisher, IN-Port ist Subscriber. Auch sind Datentyp und Portname für den Anwender Variabel zu halten.
- Zwischen zwei Lauch.vi´s, von Objekten welche miteinander kommunizieren kann man bei Bedarf einen Port Observer plazieren. Verknüpft man diesen mit einer Anzeige-Referenz, so visualisiert der Observer die Daten auf der dazugehörigen Anzeige.
- Für alle DIM-Standartdatentypen sollen Observer.vi´s existieren, genauso soll aber auch die Möglichkeit bestehen eigene Observer zu bauen, welche beliebige Datentypen visualisieren können.
- Auf jedem CS-System, zu dem Objekte im Netz plaziert sind, soll eine Instanz existieren, welche die ON-Objekte überwacht, sprich sie beim Netzstart inizialisiert, sie im Falle einer Störung neu startet und sie beim Beenden der Anwendung ordnungsgemäß vernichtet.
- Die Möglichkeit, Objektnetze hierarchich anzuordnen soll bestehen
- Auch sollen die Objektnetze selbst an eine Klasse gebunden werden, damit die Möglichkeit besteht mehrere Instanzen des gleichen Netzes zu generieren

- Die Objektnetze sollen als Basis für die Petrinetze dienen, sind also so zu gestalten, dass man die Petrinetze leicht auf sie aufsetzen kann.

2.1.2 Petrinetze

Das klassische CS-Petrinetz wird ein Stellen-Transitions Netz sein, welches ohne Marken-Kapazitäten auskommt. Transitionen schalten **nicht** beliebig verzögert, wie im gewöhlichen Petrinetz (may) sondern haben den Zwang sofort dann zu schalten, sobald alle Marken der Vorgänger-Plätze vorhanden sind. Bei dieser Form der Netze wird keine Rücksicht auf Nachfolge-Plätze genommen, da Kapazitätennetze eine Unterart der hier erläuterten Netze sind, welche sich ohne Schwierigkeiten als Addon implementieren lassen.

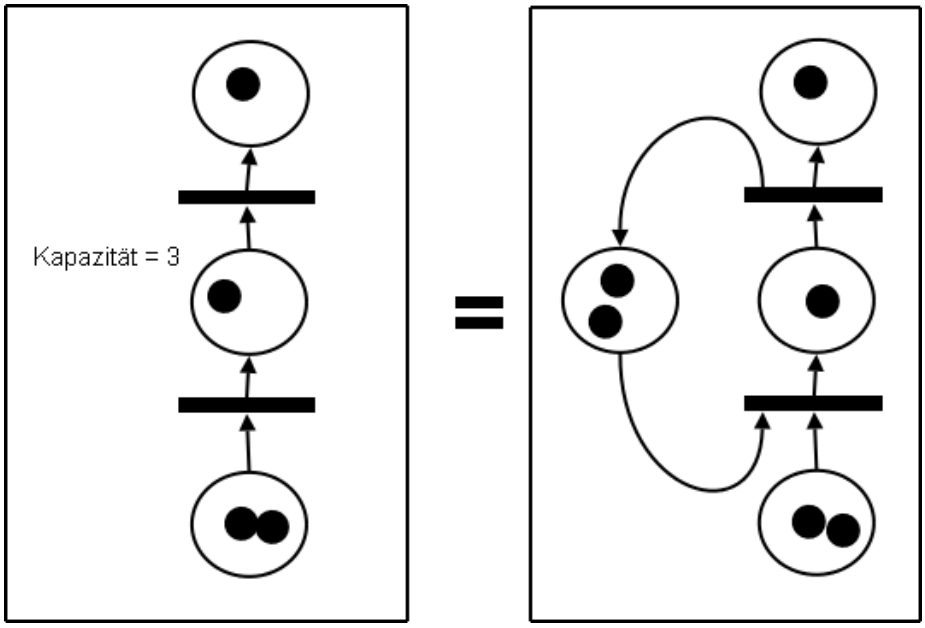

Abbildung 1: die Unterart der Kapazitätsnetze

Als weiterer Zusatz sollen im CS Timing Netze verwirklicht werden. Transitionen dieser Netze haben die Option nicht sofort, sondern in einem beliebigen Zeitfenster, oder nach Ablauf einer Zeit zu schalten. Wie zuvor bleibt die Marken-Kapazität dieser Netze vorerst unbeschränkt.

- Plätze haben eine Eintrittsaktion, eine periodische Aktion und eine Austrittsaktion
- Eintritts- und Austrittsaktion lösen je ein Ereignis aus, welches entweder direkt, oder durch ein sub-vi versendet wird

- Transitionen sollen genau dann schalten, wenn die erforderlichen Marken vorhanden sind oder der externe Trigger ankommt.
- Ob Plätze und Transitionen als eigene CS-Objekte gestartet werden, oder nur vi-templates Darstellen muss noch abgewogen und geprüft werden.
- Auch Petrinetze sollen hierarchich organisiert werden können.
- Wie bei den Objektnetzen sollen Petrinetze an CS-Klassen gebunden werden, um das Erzeugen von Instanzen zu erleichtern.

2.2 Machbarkeit

Prototypen der Objektnetze wurden bereits in einer Studie implementiert. Diese müssen nun an die obigen Anforderungen angepasst werden.
Ein Risiko liegt darin, einen Semaphor-Mechanismus für die Petrinetze zu finden. Wahrscheinlich kann man hier den im CS implementierten Objekt-Reservierungsmechanismus verwenden, was allerdings noch zu prüfen ist.
Desweiteren sollte der Implementierung nichts im Wege stehen.

2.3 Projektverlauf

Der Verlauf des Projektes ist im Wesentlichen an den Abgabetermin der Diplomarbeit und die Urlaubszeiten der Betreuer gebunden.

Zeitraum	**Aktivität**
bis 13.04.07	„Vorgehensweise für die Diplomarbeit, gemäß stdSEM“schreiben
	und Implementierung der Objektnetze, aufbauend auf dem vorhandenen Prototypen
bis 01.05.07	Erstellen eines Theoretischen Konstruktes der Petrinetze,
	ausarbeiten genauen Vorgehensweise unter Betrachtung aller geforderter Aspekte
23.05.07	Ausgabe der Diplomarbeit
bis 01.06.07	Implementieren eines Prototyps der Petrinetze
bis 01.07.07	Schreiben einer Vorabversion der Diplomarbeit
	(Um von Holger Brand geprüft zu werden, bevor dieser seinen Urlaub antritt)
bis 01.08.07	Nachbearbeitung der Petrinetze, Fehlerbehebung, Anpassung an die Forderungen der Anwend
bis 22.08.07	Fertigstellung der Diplomarbeit
23.08.07	Abgabe der Diplomarbeit

3 Entwurf

3.1 Systematischer Aufbau

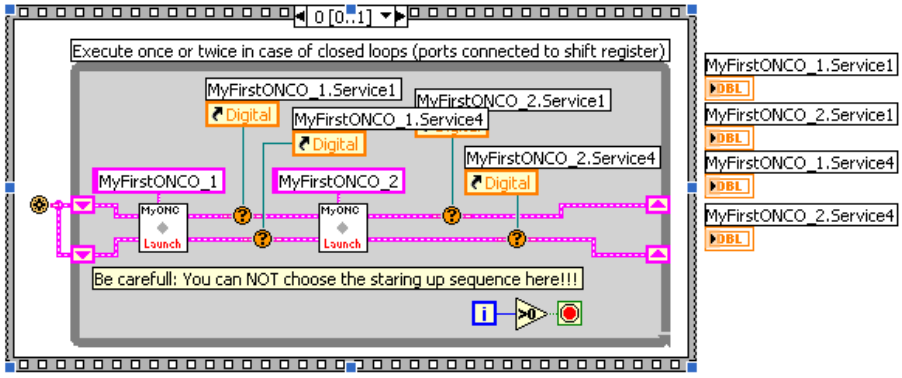

Abbildung 2: einfaches CS-Objektnetz

3.1.1 Objektnetze

Beim Durchlaufen des Netzes werden alle relevanten Informationen (Klassennamen, Objektnamen, CS-System, Servicename...) im Attribut des Netz-Objektes gespeichert. Ferner startet jedes Observer.vi einen DIM-Listener, welcher die Daten des verbundenen OUT-Ports direkt auf die verbundene Anzeige ausgibt. Ein Flussdiagramm zu dieser Thematik ist in „Abbildung 3 “dargestellt.

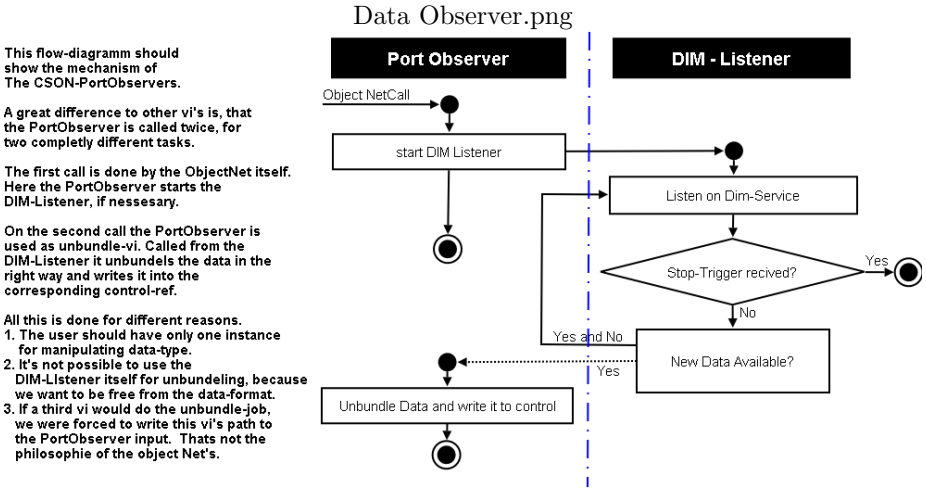

Abbildung 3: Beziehung zwischen DIM-Listener und PortObserver

Nun werden im nächsten Teil der Sequenz alle ONLocalControl-Objekte gestartet, auf jedem beteiligtem CS-System einer. Der Datensatz des Objektnetzes wird nun via DIM veröffentlicht (publishService) und je ein Netupdate- und ein Shutdown-Service werden ins Leben gerufen. Das Netz selbst ist nun fertig durchlaufen und wartet auf die Betätigung des „Stop Serving for Editing“bzw des „Shutdown all Objects“Knopfes.
Sobald die ONLocalControl-Objekte vom Netz erzeugt werden, empfangen diese die Netupdate-Nachricht, zusammen mit dem Datensatz des ObjektNetzes. Das ONLocalControl durchsucht nun das Netz nach Objekten, welche auf dessen System laufen und startet diese wenn nötig. Des weiteren überwacht es nun die Lebenszeichen dieser Objekte und startet diese bei Störungen neu.
Das Netz existiert nun so lange, bis der Knopf „Shutdown all Objects“betätigt wird. Dadurch fährt jedes ONLocalControl all seine Objekte herunter.

Ein detailiertes Flussdiagramm ist in „Abbildung 4 “dargestellt.

3.1.2 Petrinetze

Beim Durchlaufen wird für jede Transition und jeden Platz ein Prozess gestarted (evtl auch ein Objekt einer eigenen Klasse), welche auf Änderungen der Vorgänger-Plätze lauscht und externe Kommandos registriert.
Bevor eine Schaltung im Netz erfolgen kann, müssen die Plätze, die an eine Transition angeschlossen sind, mit Hilfe des CS-Reservierungsmechanismus, für den exclusiven Zugriff reserviert werden. Sind nun alle Marken, die zum Schalten einer Transition erforderlich sind vorhanden, so kann diese Transition das Netz sperren und den Schaltvorgang auslößen. Dabei werden zunächst alle beteiligten Objekte reserviert, dann die exit action der Vorgänger Plätze und die entry action aller Nachfolger-Plätze ausgeführt und zuletzt werden die Objekte und das Netz wieder freigegeben. All dies hängt allerdings von der Verwendbarkeit des CS-Reservierungs-Mechanismuses ab. Sollte dieser für die Petrinetze nicht verwendbar sein, so muss hier ein anderer Weg gefunden werden, Plätze und Objekte zu reservieren.

3.2 Benutzung von existierender Software

Vorrausgesetzt wird die Nutzung des CS-Framworks, welches den objektorientierten Ansatz für beide Netze bietet. Des weiteren sind die Netze darauf spezialisiert mit Objekten des CS-Frameworks zu arbeiten.

Zur Rechner-übergreifenden Kommunikation wird DIM (Distributed Information Management System) genutzt. Was sich nicht zuletzt auch durch die Nutzung im CS-Kontrollsystem anbietet.

Tools wie z.B. der CSSQLServer, welcher auf die MSAccess Datenbank zu greift kommen im Rahmen des CS zur Anwendung.

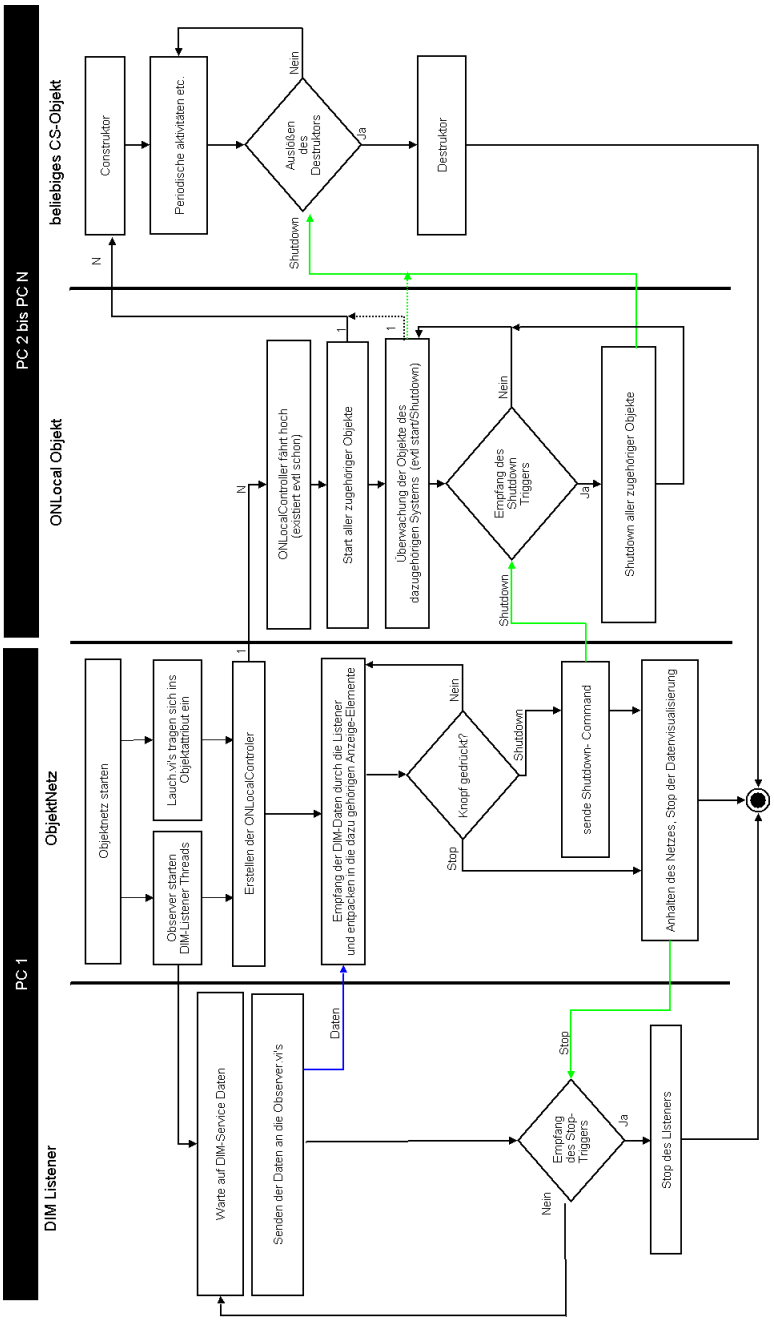

Abbildung 4: Flussdiagramm des Objekt Netzes

4 Realisierung

4.1 Erzeugen des Quellcodes

Begonnen wird mit der Abarbeitung der Stichpunktliste, zu den Änderungen der Objektnetze.
Danach werden die ersten Petrinetze erzeugt, welche ohne Hintergrund-Objekte laufen. D.h. Implementieren der vi-templates für die Transitionen und Wiederverwenden des ObjektNetz-Mechanismuses zum Speichern der Netzdaten.
Nach ausreichenden Tests des Systems kommt als nächster Arbeitsschritt die Implementierung externer Events, und dem Auslößen von Events durch entry und exit action, sowie das starten einer periodic action (evtl als CS-Objekt)
Als letztes folgt das Hinzufügen zeitkritischer Transitionen (Sonderart der schon existierenden Transitionen) und das Testen dieser Transitionen.

4.2 Einbinden existierender Software

Da externe Software schon hinreichend gut ins CS eingebunden wurde, braucht man keine besonderen Maßnahmen zum Einbinden zusätzlicher Software zu ergreifen

4.3 Testen

Objekt Netze Die überarbeiteten Objektnetze werden auf einem verteilten System mit über 100 Prozessvariablen getestet. Wobei Netze auch dynamisch erzeugt werden sollen.

Petri Netze I Die erste Vorabversion der Petrinetze soll die Grundfunktionallität des Schaltens eines Petrinetzes erfüllen. Dazu werden einige Testnetze nachgebaut, deren Funtionalität bekannt ist, und deren Schalteigenschaften geprüft.

Petri Netze II Die zweite Version der Netze sollte externe Objekte im verteilten Netzwerk unterstützen und bekommt demnach im Test Signale von diesen. Genauso versenden die Plätze nun Events beim Ausführen der entry, periodic und exit action. Diese werden wiederum mit externen Objekten empfangen und geprüft.

Petri Netze III Zum Test der Timernetze stehen einige Beispiele zur Verfügung, welche nach Vorgabe implementiert und getestet werden. Unter Zeitkritischen Gesichtspunkten wird nun auch die maximale Schaltzeit eines Petrinetzes festgestellt

5 Abschluss

5.1 Sichern des Quellcodes

Während des Projektes wird der Code in unregelmäßigen Abständen ins LVSCC (LabVIEW Source Code Controll) System eingecheckt. Die finale Version wird dann zu einem Packet gebündelt und zusätzlich auf der CS-SourceForge Website veröfentlicht.

5.2 Vervollständigen und Archivieren der Dokumente

Der gesamte Code wird noch mehrmals von verschiedenen Personen geprüft, um fehlende Funktionen und Anmerkungen zu ergänzen. Zusammen mit der Diplomarbeit werden die Dateien an der GSI archiviert.