Studienarbeit

Prototypimplementierung
für einen
asynchronen entfernten Methodenaufruf
in Java mit Future-Objekten

05.05.2006

Eka Gelashvili

Inhalt

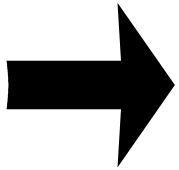
- Motivation
- Ziel
- Future-Konzept
- Nachrichtenkanal-Netzwerk
- Implementierung
- Bewertung
- Ideen

Motivation

- Future-Konzept
  - Flexibilität in verteilten Anwendungen
- Nachrichtenkanal-Netzwerk
  - Beispiel für die Nutzung
  - Integration in Standard-API

Ziel

- Asynchrones Kommunikationsmuster
  - mehr Flexibilität und Performanz
- Nachrichtenkommunikation mit Rahmenwerk
  - Java 5 ähnliche API



Future mit Nachrichtenkanal-Netzwerk

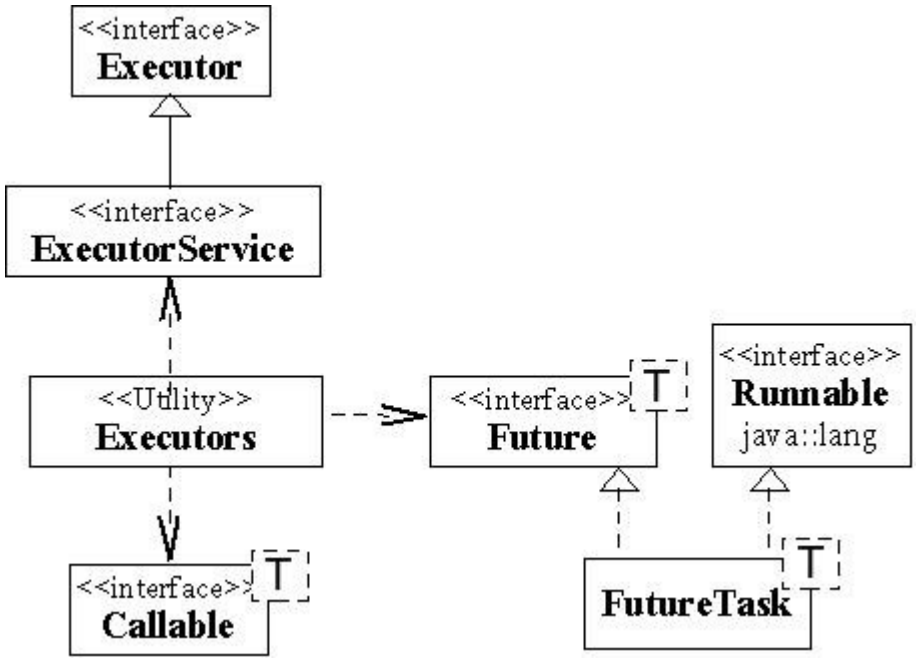
Future-Konzept

Future – Platzhalter für ein Ergebnis, das noch nicht bekannt ist, weil seine Berechnung noch nicht abgeschlossen ist

- Basiert auf nebenläufiger Programmierung
- Realisiert asynchronen Methodenaufruf
- Seit Java 5 Teil der Standard-API: `java.util.concurrent` Paket

Future-Konzept

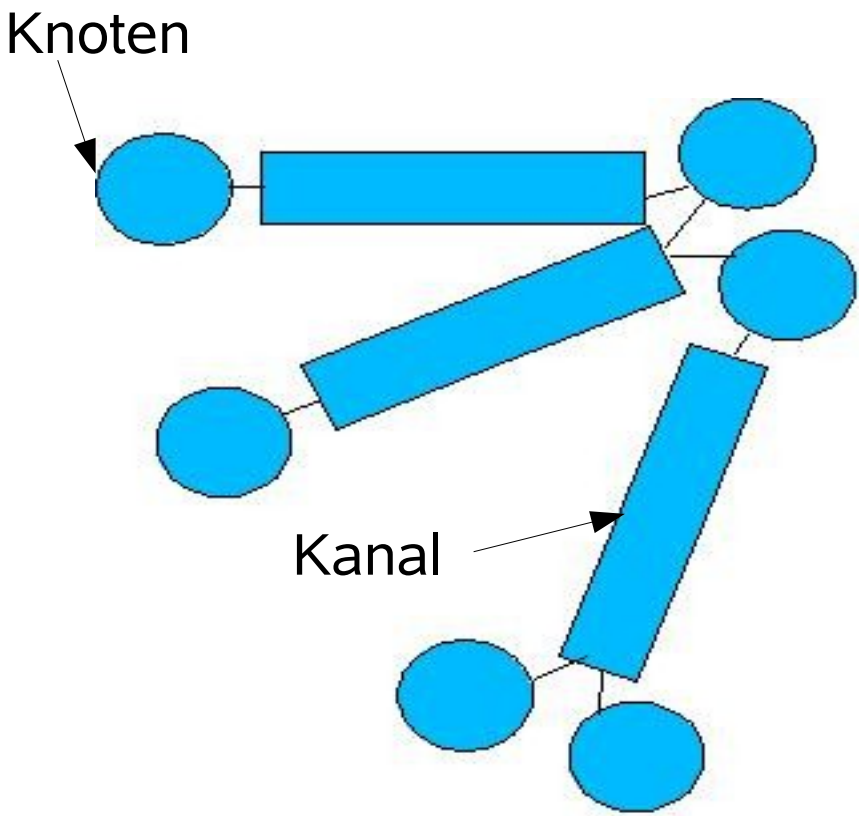
- Interessante Klassen und Schnittstellen aus dem `java.util.concurrent` Paket



Nachrichtenkanal-Netzwerk

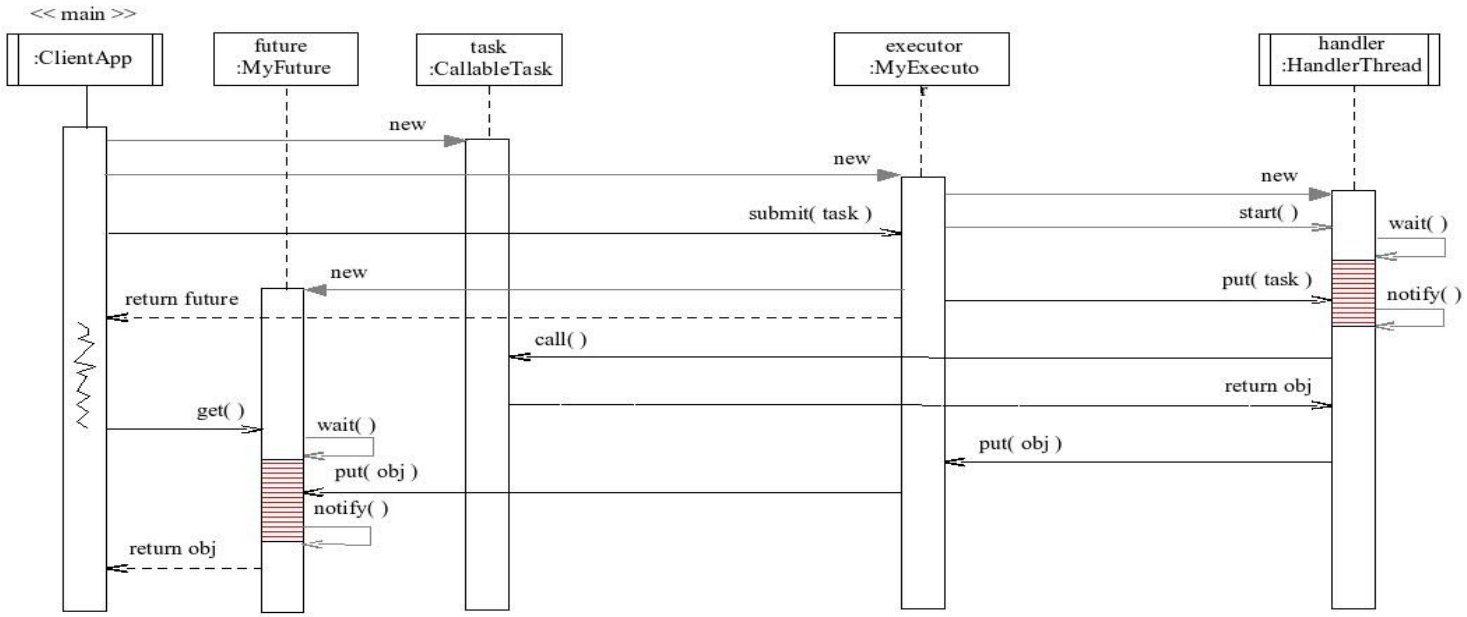
- Asynchrone Nachrichtenkommunikation
- Für eingebettete Systeme (nicht nur !)
  - Echtzeitanforderungen (Echtzeit-Java, RTSJ)
  - Rundruf-Netzwerke (Viele-zu-Viele)
- Logische Nachrichtenkanäle
  - Publiziere/Abonniere-Muster
  - Anonyme Adressierung
  - Prioritätsbasierte Verarbeitung

Nachrichtenkanal-Netzwerk



- Mehrere Knoten an einem Kanal
- Ein Knoten an mehreren Kanälen

lokale asynchrone Kommunikation



- Auftraggeber erzeugt Auftrag (`task`), übergibt diesen an Ausführungsdienst (`MyExecutor`) mit `submit(task)`
- Ausführungsdienst leitet ihn zur Ausführung an Behandler weiter
- `submit(task)`-Methode liefert ein `Future`-Objekt zurück
- Auftraggeber kann von `Future`-Objekt das Ergebnis asynchron lesen

Vorteile von Future

- Auftraggeber wartet nicht auf den Behandler
- Bekommt gleich ein Future-Objekt zurück
- Wird nicht blockiert
- Kann andere Aufgaben erledigen

Nachrichtenkanal-Netzwerk

- Zentrale Komponente `EventChannelNetwork`
  - Einzelstück (singleton)
  - XML-Datei für Konfiguration
  - Erzeugt alle für die Kommunikation notwendige Objekte (Kontrollfäden, Behandler, Warteschlangen)

Nachrichtenkanal-Netzwerk

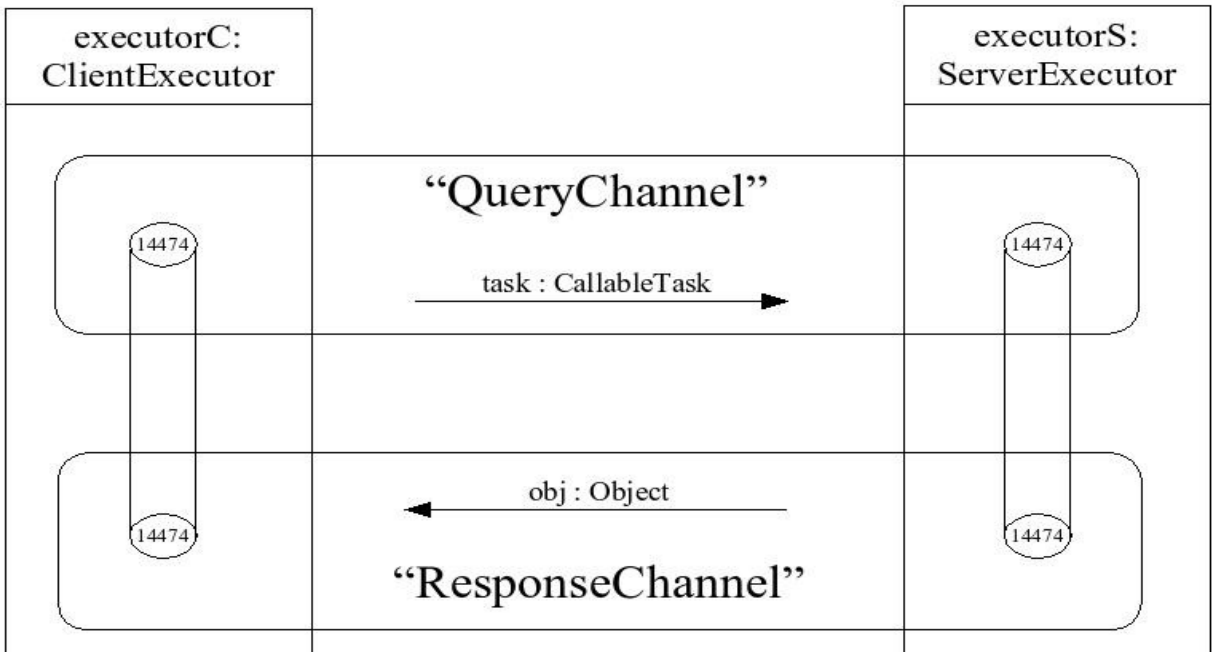
- Ein Beispiel für XML-Beschreibung:

```
...
<server name="Server">
  ...
  <channel name="querychannel" exclusive="false">
    <description>Query Channel</description>
    <priority>20</priority>
    <socket>S.udp14474</socket>
    ...
```

- Ein Codebeispiel für Erzeugung des `EventChannelNetwork`-Objekts:

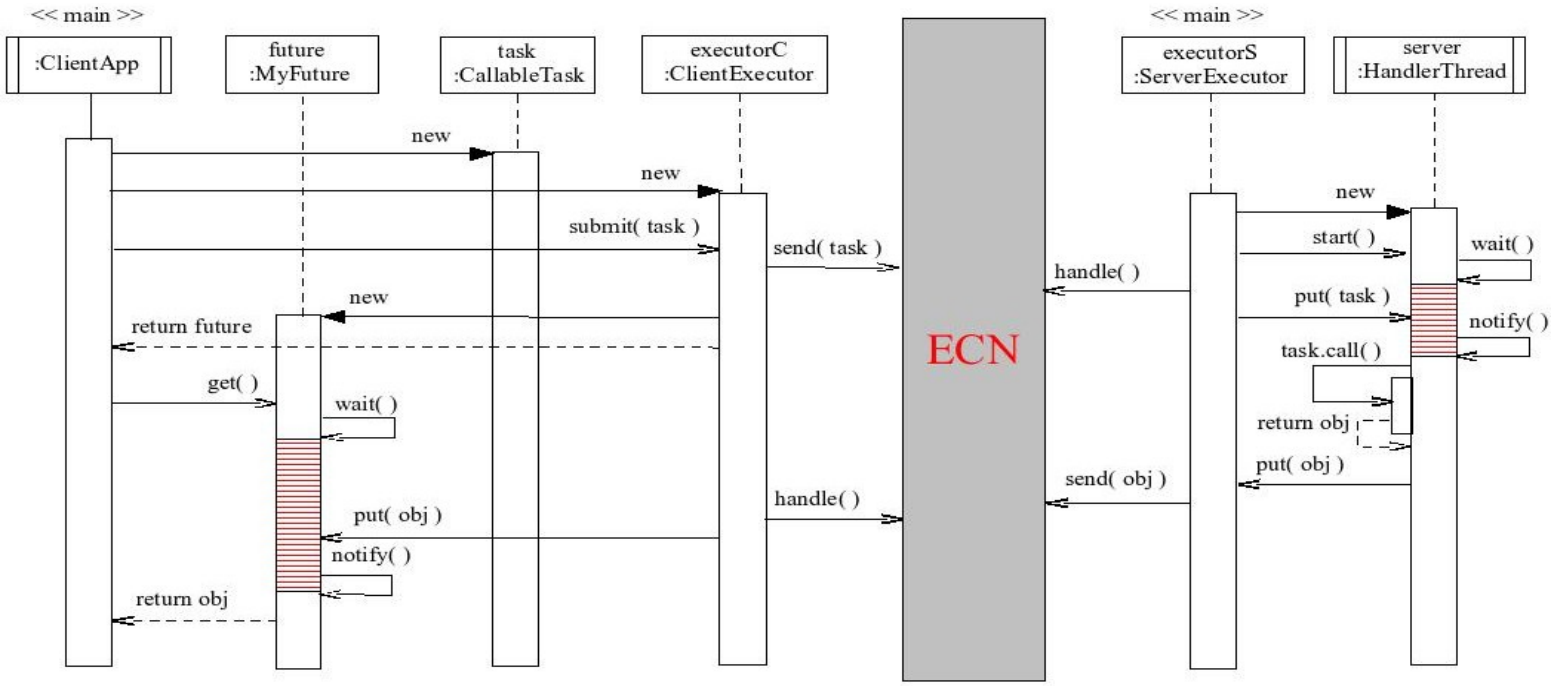
```
EventChannelNetwork system =
  EventChannelNetwork.createSystem(new File(
  "src/de/fzi/ecn/test/future/ecn/channel_server.xml"));
```

Nachrichtenkanal-Netzwerk



- "QueryChannel“ für Senden/Empfang des Auftrags (`task`)
- "ResponseChannel“ für Senden/Empfang des Ergebnisses (`obj`)

entfernte asynchrone Kommunikation



- Executor verteilt
  - `ClientExecutor`
  - `ServerExecutor`
- Fassaden für die Kommunikation

Bewertung

- Future-API für verteilte Anwendungen
- Basierend auf dem ECN-Rahmenwerk
  - Beispiel für transparente ECN-Nutzung
  - ECN in Java 5 integrierbar
- Kaum Unterschiede zwischen lokalem und verteiltem Future
- Einfache Nutzung asynchroner Kommunikation
  - Einfache Adaption des Rahmenwerks an andere Rundruf-Netze

Weitere Ideen

- Vorteile für Future mit Nachrichtenkanal-Netzwerk
  - Ausweitung der 1-zu-1 Kommunikation auf 1-zu-N
    - Erste Antwort ausreichend
    - Mehrheitsentscheidung (in sicherheitskritischen Anwendungen sinnvoll)
  - N-zu-M Kommunikation
    - Mehrere interessieren sich für das Ergebnis
- Einsatz von einem asymetrischen kryptographischen Verfahren, um Sicherheit beim Empfang der Nachrichten zu gewährleisten

Danke für Ihre Aufmerksamkeit

Fragen?