1 Designtreffen 2: Roadmap

1. Einleitung
2. Name der Anwendung?
3. Ergebnisse Phasenprotoyp
   - Wie teilen wir Phasen in Client/Server ein?
     - Möglichkeit 1: wir haben eine ClientXPhase und eine ServerX-Phase mit genau einer execute-methode. Das ergibt kleine Klassen, aber schwer wartbare Methoden und relativ grosse Redundanzen bei der Konstrktion der Phasen
     - Möglichkeit 2: wir haben eine Phasenklasse mit Methoden clientExecute, serverExecute. Das ergibt etwas groessere Klassen, ist aber wesentlich übersichtlicher und erlaubt es, genau einmal die Phasen zusammenzusetzen.
   - Symmetrische Phasen? (Hier muesste man sich vmtl einfach einmal ein Kollaborations-Diagramm einer symmetrischen Phase hinmalen, clientseitig und serverseitig und gucken, was dann passiert und ob man das zusammenfassen kann)
   - Genaue Methoden der Verbindung? (Hier muss man vermutlich die Phasen alle einmal durchgehen und untersuchen, welche Daten ausgetauscht werden, um so die Methoden der Verbindung zu bekommen.
   - Kommunikation von Phase zu Phase?
     - Möglichkeit 1: mittels getOutputFoo und setInputBar in der uebergeordneten Klasse Wissen von A nach B schieben. Das ist relativ einfach umzusetzen, wird bei viel Wissen und viel Bewegung etwas unübersichtlich.
     - Möglichkeit 2: mittles einem Mediator werden die Daten verschoben – der Vorgänger rutf put(wissen) auf, der Nachfolger ruft get(wissen) auf. Das ist etwas anspruchsvoller zu implementieren, macht aber das Zusammensetzen der Phasen sehr einfach und modelliert erstaunlich dicht die Phasendiagramme aus der Spezifikation.
     - Möglichkeit 3: Observer. Die Nachfolgephasen registrieren sich bei der Vorgaengerphase als Ergebnis-observer und behandeln dann die Objekte. Hier gab es im Prototypen relativ grosse Probleme, das ans Laufen zu kriegen und die Skalierbarkeit ist fragwürdig.
4. CRC-Karten abschliessen
   - Phasen für Yaos Protokoll
   - Klassifikator

- Schaltkreise, garbled Schaltkreise

5. Feindesign Schaltkreis
   - Interne darstellung?
   - Api?
   - Builder?
   - Verwendung in der Spezifikation:
     - Seite 5: Definition
     - Seite 16: Yaos Protokoll
     - Seite 19: Dominierende Ausgabe
     - Seite 20: eindeutige Ausgabe
     - Seite 22: Erweiterung eines Schaltkreises: Separate Ausgabe
     - Seite 23: Erweiterung eines Schaltkreises: Share-Zerlegung
     - Seite 24: Anfangsapproximation
     - Seite 24: Attribut mit maximalem Informationsgewinnn
6. Feindesign Polynom
   - interne Darstellung?
   - API?
   - Verwendung in der Spezifikation:
     - Seite 27: Private Multiplikation
     - Seite 24: Polynomauswertung
     - Seite 24: Verbesserung der Approximation
7. weitere Umsetzung
   - wöchentliche Treffen?
   - ISP Rechnerpool ist frei
   - Anfrage an Admin ist noch nicht beantwortet
8. Deployment
   - Aufteilung in Klassifkator und Lerner: mit einem Parameter wie client oder server? als separates Programm?
   - bei Aufteilung brauchen wir Package-Diagramme und Deployment-Diagramme
   - 2 Achsen entlang denen sich das ändert Anzahl jarfiles und Anzahl main-Methoden.
   - Jarfile fü Server, Client, Klassifizerer, oder entsprechende Teilmengen davon

- Anzahl Main-Methoden: Main-Methode für Client, Server, Klassifizierer, oder Teilmengen davon
- Anzahl Main-Methoden ¿= Anzahl Jarfiles
- mehr Jarfiles = mehr Flexibilitaet und mehr Aufwand (wegen mehr Files)
- mehr Main-Methodne = einfacherer Code, aber mehr Wissen, was der Anwender behalten muss

9. Testing
    - Testen wir?
    - Was testen wir?
    - Mocks?
    - Random? (muesste man Seeden und injecten)
10. Logging
    - Loggen wir?
    - Was loggen wir? (Phasen-Schritte, Daten senden, ...?)
    - Framework? (logback ist das aktuellste freie)
11. Code-Dokumentation
    - Dokumentieren wir?
    - Womit?
    - Was?
12. Handbuch
    - Verlorene Information
    - Ausführung der Programme
    - Fehlermeldungen, ihre Bedeutung, beheben der Ursache
    - LoggingKonfiguration erklaeren, wenn wir das tun