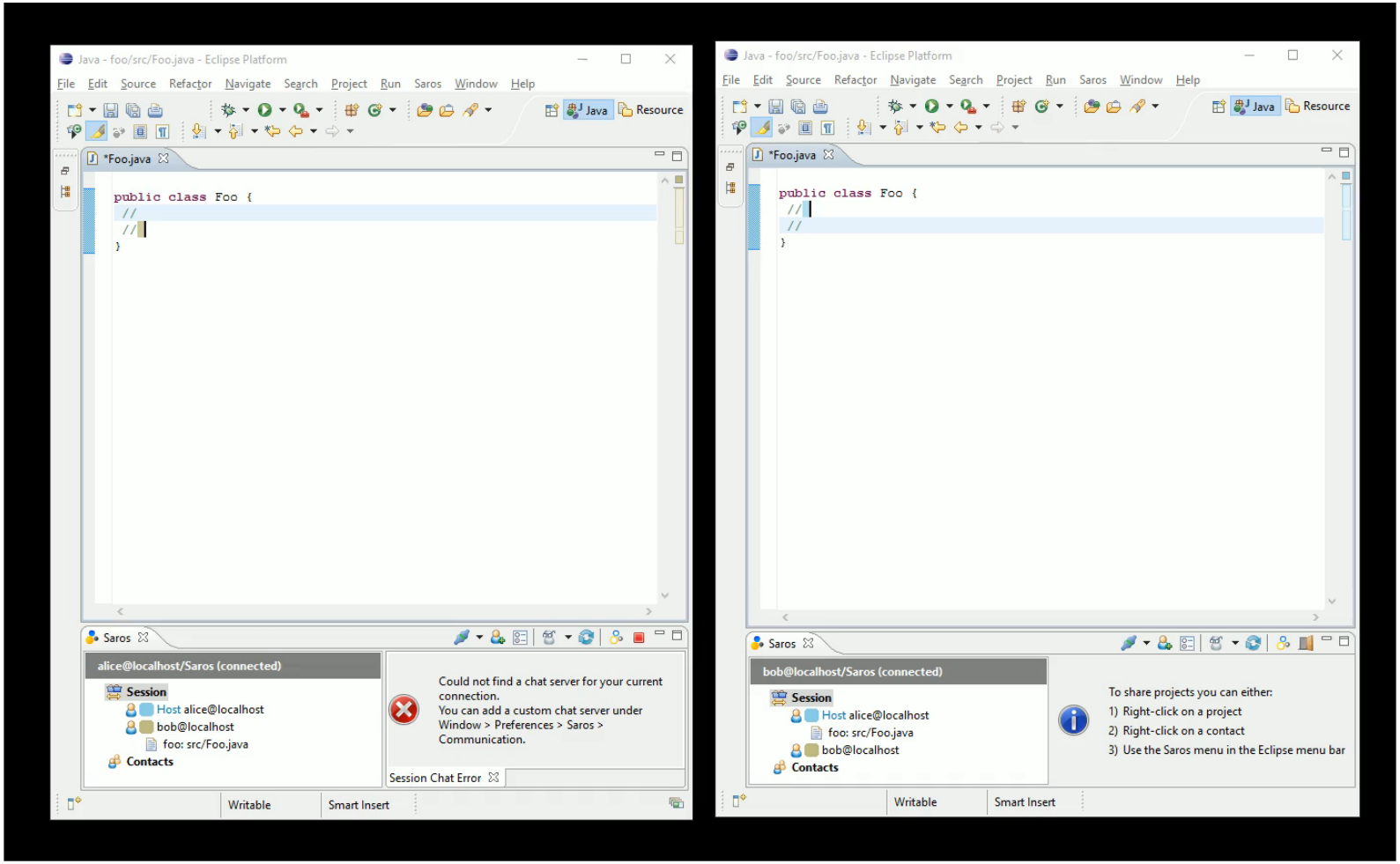
Ein Standalone-Server für das SAROS-System

Denis Washington

SAROS – Distributed Party Programming



Verteilte Code-Kollaboration für **Eclipse** (und bald **IntelliJ**)

Dieser Vortrag

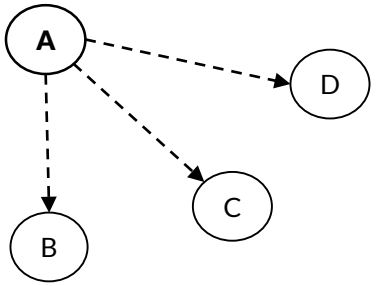
- Motivation und Ziele
- Umsetzung und offene Fragen
  - inkl. derzeitigem Status

Motivation und Ziele

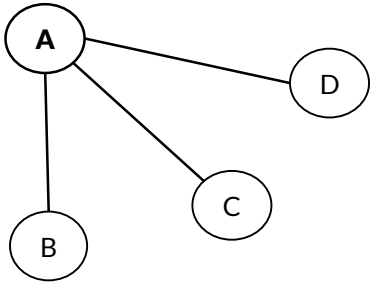
Heute: **Host-basierte Sitzungen**

- Initiator einer Sitzung ist **Host**
  - Hat Sonderrechte
    - Projekt teilen
    - Benutzer einladen
  - Mittelpunkt aller Kommunikation
  - „Quelle der Wahrheit“
- Sitzung ohne Host zuende
  - single point of failure

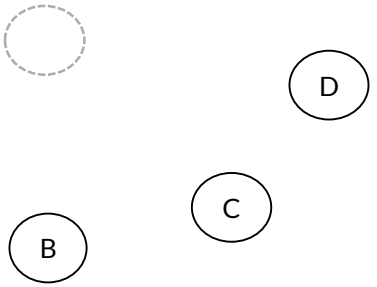
Einladungen



Sitzung



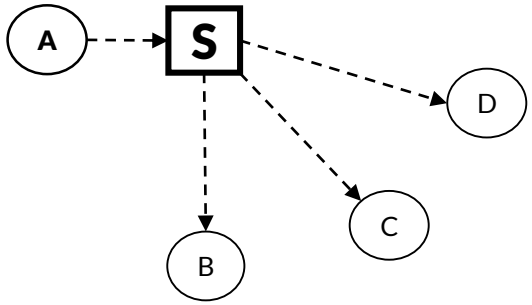
A weg, Sitzung weg



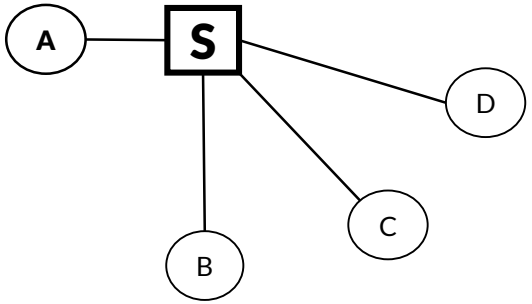
Idee: **Server-basierte Sitzungen**

- **Server** übernimmt Host-Rolle
  - Initiator ohne Sonderstatus
  - Teilnehmer können Server bitten
    - eine Sitzung zu erstellen
    - Benutzer einzuladen
    - Projekte zu teilen
- Jeder kann Sitzung verlassen
  - Auch alle
  - Ermöglicht “Sitzung fortsetzen”-Feature

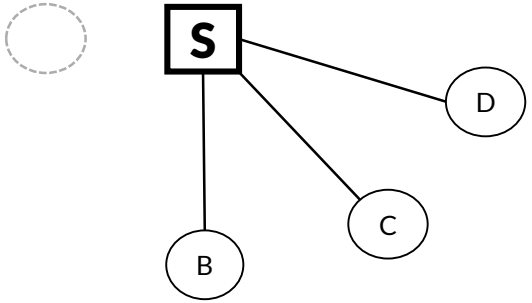
Einladungen über Server



Sitzung über Server

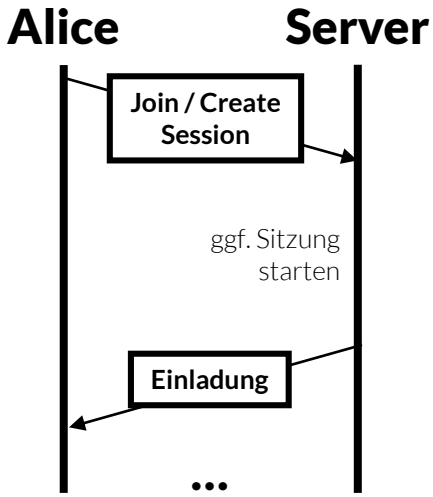


A weg, Sitzung bleibt



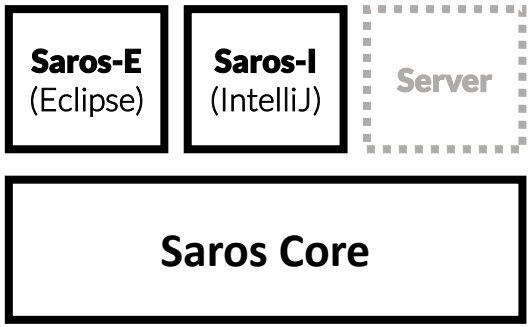
Vorarbeit: **Nils Bussas (2014)**

- Bachelorarbeit: **„Entwicklung eines Server-Prototypen für Saros“**
- Protokoll zur Initiierung einer Sitzung auf Server
  1. Benutzer bittet Server, Sitzung zu erstellen (“*Create Session*”) oder laufender Sitzung beizutreten (“*Join Session*”)
  2. Server lädt Benutzer zur Sitzung ein
- Server-Prototyp in Eclipse-Plugin integriert
  - *„Dies soll ein Server sein“*-Checkbox in Saros-Einstellungen
- **Einschränkungen:**
  - **Noch keine Möglichkeit, Projekte zu teilen**
  - Nur eine Sitzung pro Server
  - Server muss als Eclipse-Instanz betrieben werden



Vorarbeit: **Saros-Kern**

- Wiederverwendbaren, IDE-unabhängige Teile von Saros
  - z.B. Netzwerkkommunkiation, Sitzungslogik, Dateisynchronisation
- Weniger Duplikation zwischen Eclipse- und IntelliJ-Plugin
  - siehe Bachelorarbeit von Arndt Lasarzik (2015)
- Gute Grundlage für selbständigen Saros-Server
  - Allerdings noch nicht alles Notwendige im Kern (z.B. Projektverteilung, Sitzungsverwaltung)



Ziele der Arbeit

- Protokoll für Server-Sitzungen vervollständigen
  - Projekt teilen als Nicht-Host
- Eigenständig lauffähigen Saros-Server entwickeln
  - Nötige Komponenten vom Eclipse-Plugin in den Kern verschieben
  - Nötige Kern-Interfaces implementieren
  - Server-spezifischen Code schreiben
- Zuverlässigkeit des Servers evaluieren
  - Automatisierter Stresstest
  - Gezielte Codeinspektion, insbesondere bzgl. Gefahren für Langzeitbetrieb (Speicher-/Ressourcenlecks)

Nicht-Ziele

- Vollständiger Funktionsumfang
  - z.B. nur eine Sitzung auf einmal
- Produktionsreife
  - Aber: gute Grundlage für Weiterentwicklung
  - Evaluation soll Hindernisse für Produktionsreife zeigen

Umsetzung und offene Fragen

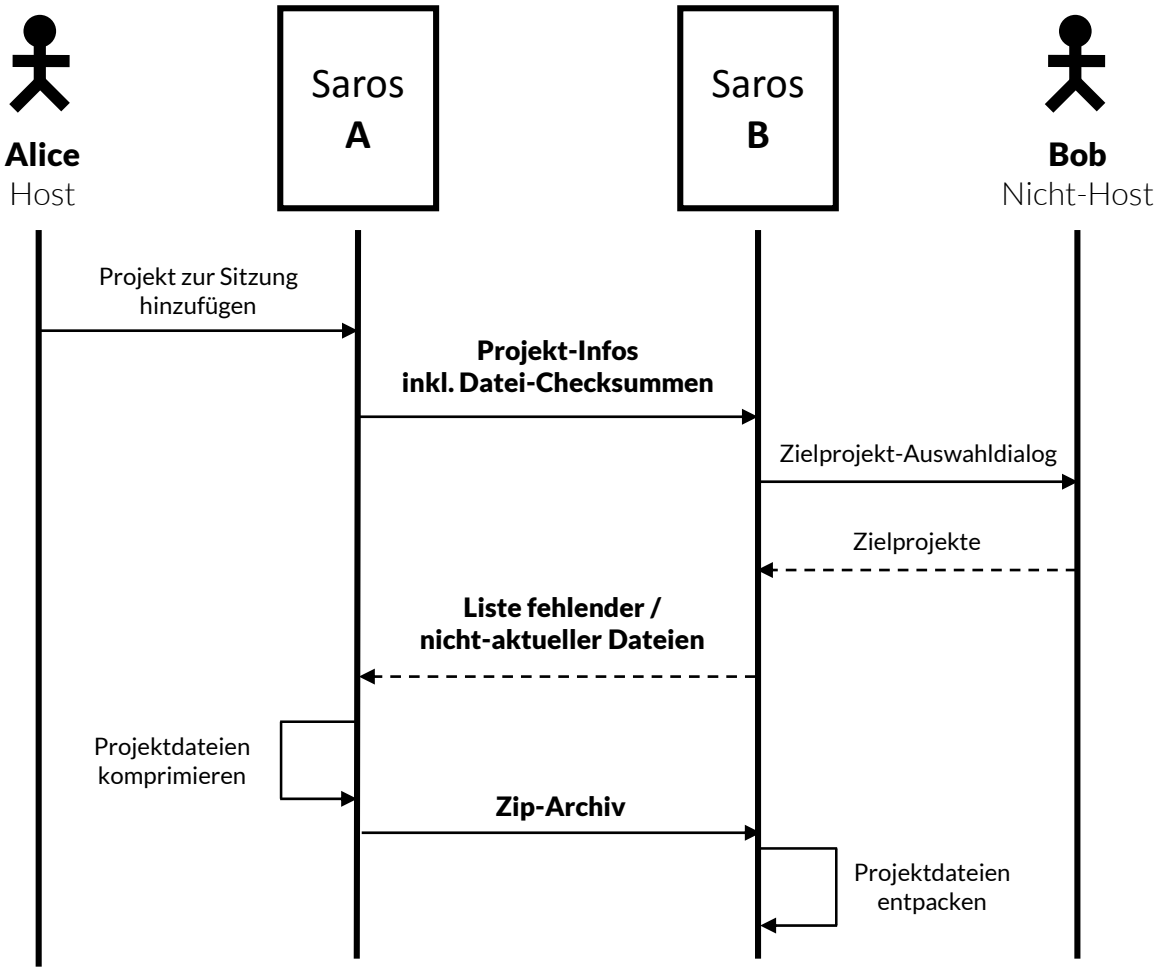
Projekt teilen als Nicht-Host

in Zusammenarbeit mit **Ute Neise**

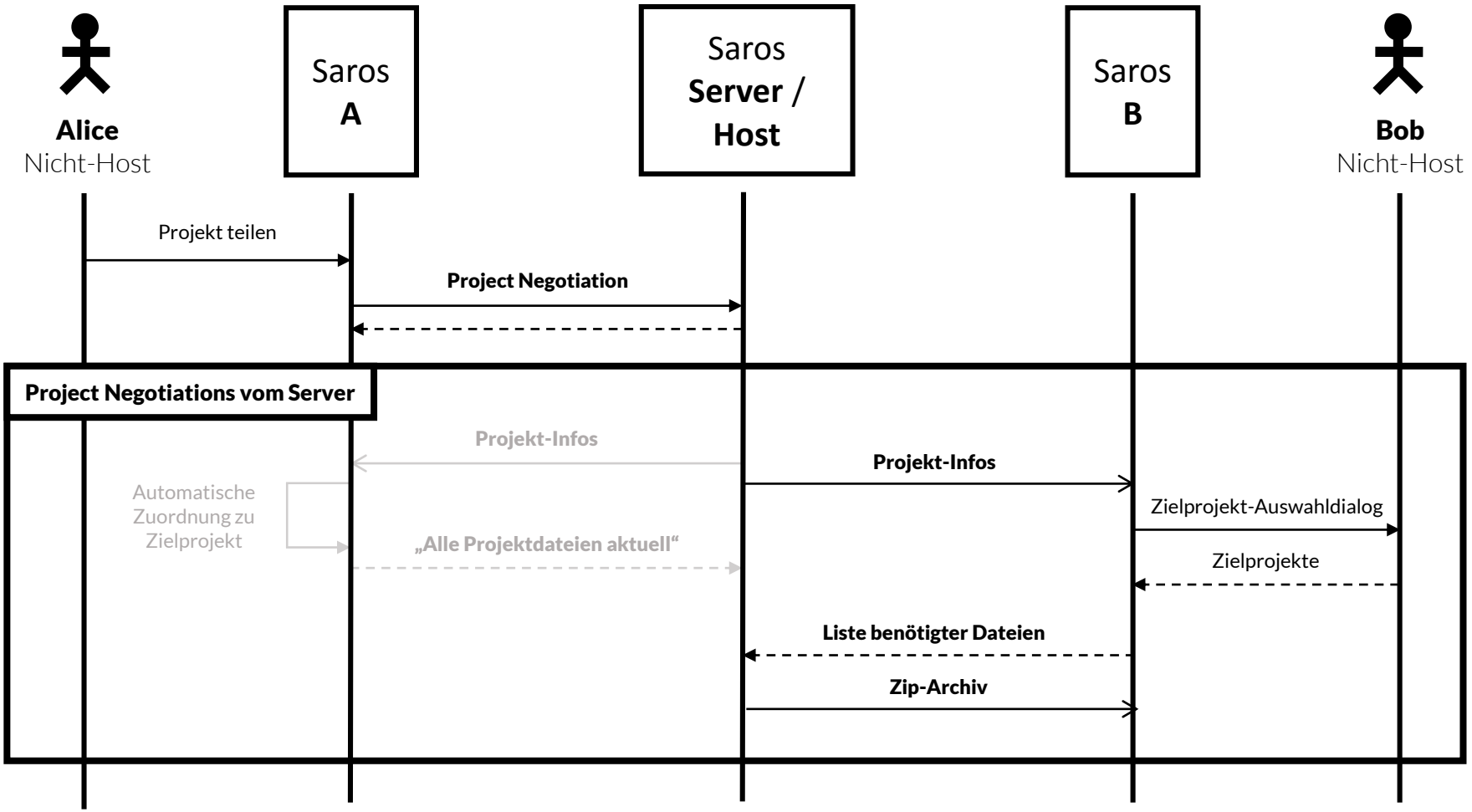
- **Problem:** Server-Sitzungsteilnehmer können keine Projekte verteilen
  - Kein Mensch ist Host
- **Lösungsansatz:** Teilnehmer schickt Projekt an Server,

  Server verteilt es an alle
- **Erkenntnis:** Ansatz auch für Host-basierte Sitzungen verwendbar
  - Entfernt Einschränkung, dass nur Host Projekte teilen kann

Der Projektaustausch (Project Negotiation)



Neu: **Project Negotiation von Nicht-Host zu Host**



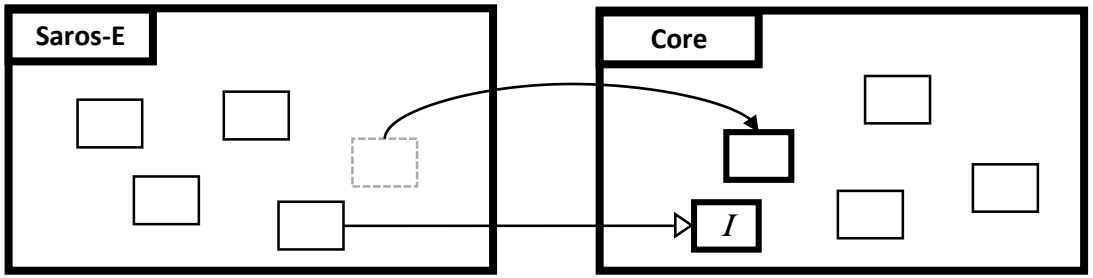
Projekt teilen als Nicht-Host: **Status**

- Implementiert (Patch #2264)
- Mit Bussas‘ Server-Prototyp erfolgreich Projekt in Server-Sitzung geteilt

Projekt teilen als Nicht-Host: **Offene Fragen**

- Sollte Rück-Projektaustausch von Host zu Initiator vermieden werden?
  - Vermeidet unnötigen Netzwerkverkehr
  - Unter Umständen weniger verwirrend
  - **Aber:** erfordert mehr Special Casing im Code, mehr Potenzial für Programmierfehler

Erweiterung des SAROS-Kerns

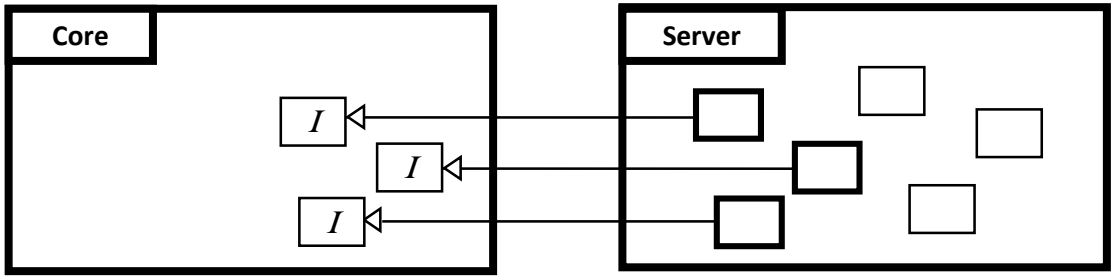


- **Methode:** Benötigte Klasse aus Eclipse-Plugin ermitteln; Baum der Nicht-Kern-Abhängigkeitsbaum erstellen; von den Blättern aufwärts zum Kern portieren
  - Eclipse-Klassenversionen besser gepflegt als IntelliJ-Äquivalente
  - Einführung von Interfaces, wenn Portierung zu aufwendig oder nicht sinnvoll
- Inkrementell
  - Erst nur absolut nötige Klassen in den Kern bringen
  - Dann weitere während der Entwicklung des Servers, falls nötig

Erweiterung des SAROS-Kerns: **Status**

- Alle bisher ermittelten essenziellen Klassen zum Kern portiert
  - `OutgoingProjectNegotiation` und `IncomingProjectNegotiation`
  - `SarosSession` und `SarosSessionManager`
- Die meisten Patches bereits in Master, einige letzte noch im Review

Implementation des Servers

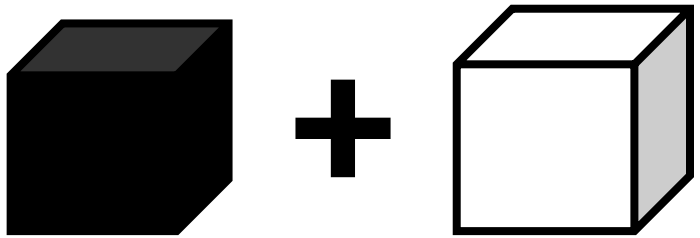


- Implementation von Kern-Interfaces
  - Projekt- und Dateisystemmodell (`IWorkspace`, `IResource`, `IPath`, …)
  - Einstelllungen (`IPreferencesStore`)
  - …
- Bootstrapping Code (Main-Klasse usw.)

Implementation des Servers: **Status**

- Dateisystem-Interfaces implementiert
- Rudimentäre Main-Klasse
- Gerade: Ermittlung und Implementation fehlender Komponenten
  1. Laufen lassen
  2. Aufbau einer Server-Sitzung von Saros-E aus versuchen
  3. Gucken was schief geht, Code ergänzen
  4. Zurück zu Schritt 1

Evaluation des Servers



- **Frage:** Wie zuverlässig sind Saros-Kern und -Server im Langzeitbetrieb?
- Kombination aus Black- und White-Box-Verfahren
  - **Black:** Automatisierter Stresstest auf Basis des STF-Framework für Saros-Oberflächentests
  - **White:** Codeinspektionen in kritischen Bereichen, um z.B. Speicher-/Ressourcenlecks oder mögliche Verklemmungen aufzuspüren
- Fokus auf Identifikation von Problemen, nicht deren Behebung

Evaluation des Servers: **Offene Fragen**

- Was genau bei Stresstest beobachten / messen?
  - Konsistenz des Synchronisationszustands?
  - Log-Dateien?
  - Ressourcenverbrauch (mittels Monitoring-Tools wie VisualVM)?
- Wie möglicherweise kritische Stellen für Code-Inspektionen ermitteln?
  - Erfahrene Saros-Entwickler (Stefan Rossbach) fragen?
  - Statische Analysetools wie FindBugs, PMD?
- Wie Einzelergebnisse zu Gesamtbild integrieren?

Zusammenfassung

Status

- **Projekt hinzufügen für Nicht-Hosts** implementiert und mit Server-Prototyp erfolgreich getestet
- Wesentliche fehlende **Komponenten in den Saros-Kern** verschoben
- **Kern-Dateisystem-Interfaces** für den Server implementiert
- **Kurz vor erster lauffähigen Version** des Servers

Ausstehend

- Server lauffähig, funktionierend machen
- Evaluation
  - Klärung offener Fragen
  - Schreiben eines automatisierten Stresstests
  - Messungen
  - Codeinspektionen
  - Auswertung der Ergebnisse

Danke fürs Zuhören

Fragen? Anregungen?