Entwicklung der Forschungssoftware RCE

DLR Simulations- und Softwaretechnik (SC)

Abteilung Intelligente und Verteilte Systeme, Köln

RCE

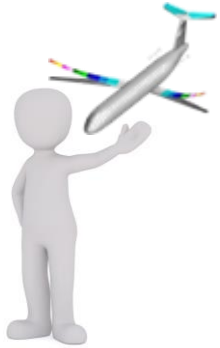
Brigitte Boden
Robert Mischke

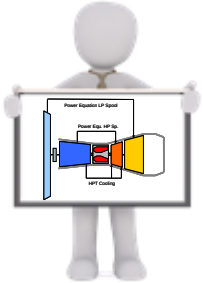
DLR

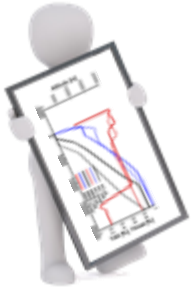
Übersicht

- Einführung RCE
- Forschungsprototyp vs. Produktivsoftware
- Finanzierung und Praxisprobleme
- Kompatibilität: pro und contra
- Feature-Breite vs. Projektmanagement

Multidisziplinäre Entwurfswerkzeuge

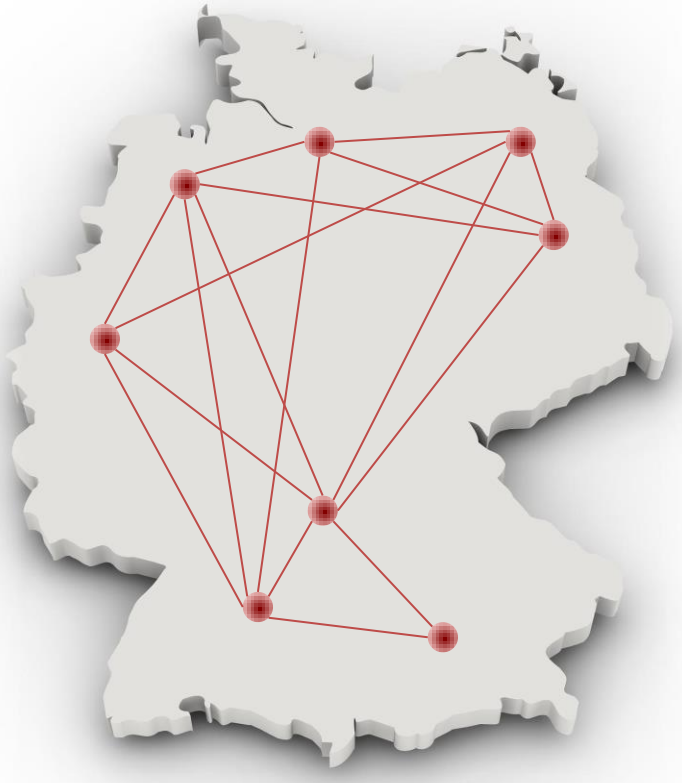






Kopplung multidisziplinärer Entwurfswerkzeuge

Kopplung verteilter, multidisziplinärer Entwurfswerkzeuge



Überblick RCE

***R**emote **C**omponent **E**nvironment*

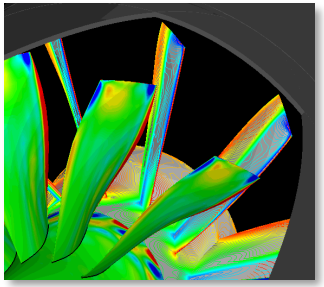
- Integrationsumgebung
- Verteilte, datengetriebene Simulationsworkflows
- Grafische Benutzerschnittstelle
- Batch-Modus
- Werkzeug-Integration
- Dezentrales Datenmanagement
- Workflow-Monitoring
- Remote Access (SSH)
- …

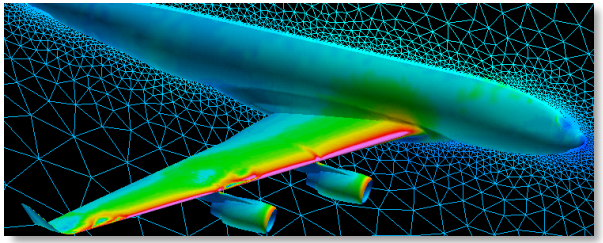
- Java, Eclipse RCP
- Open Source

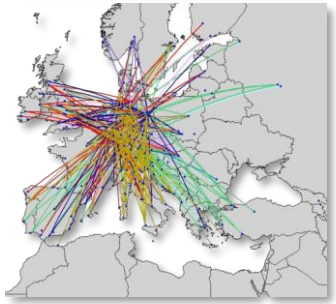
Überblick RCE

Einsatzbereiche von RCE - Beispiele

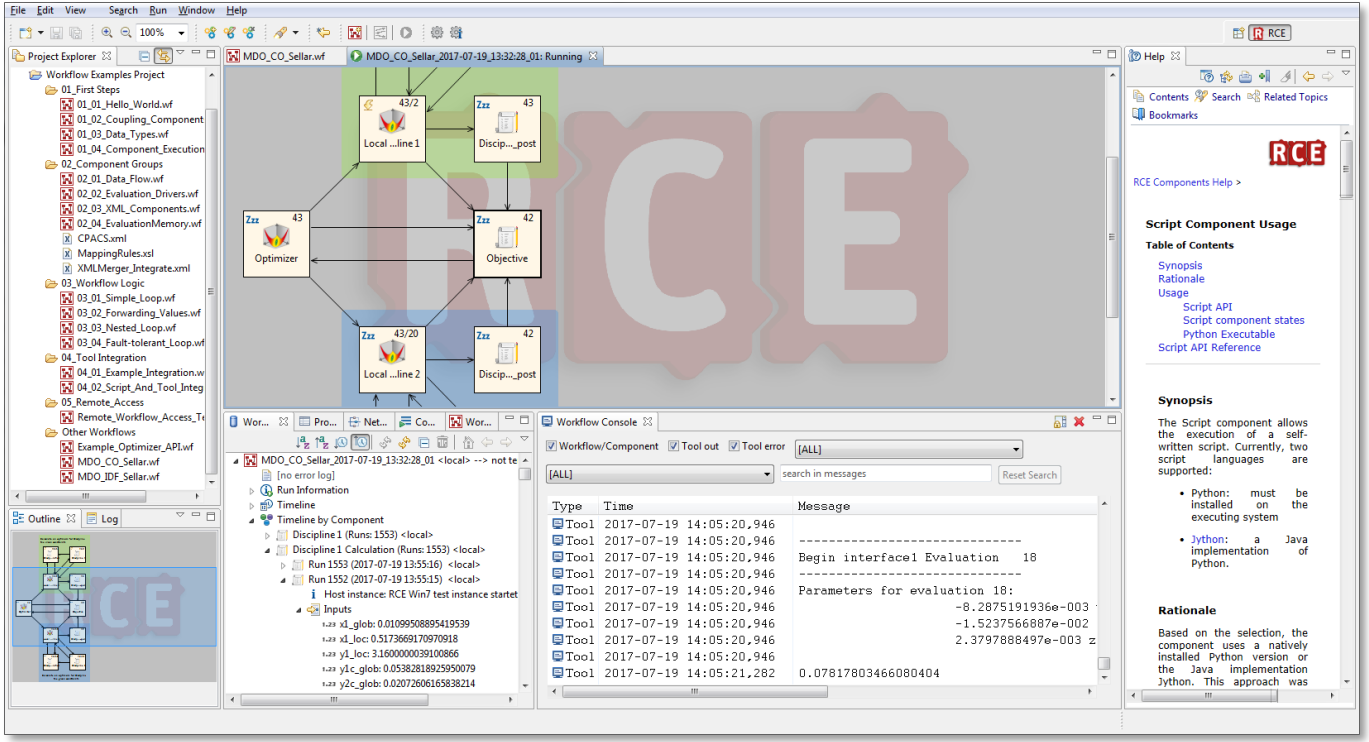
- Flugzeugentwurf
- Modellierung des Luftverkehrs in Deutschland
- Klimaoptimiertes Fliegen
- Design von Gas-Turbinen
- Schiffsentwurf



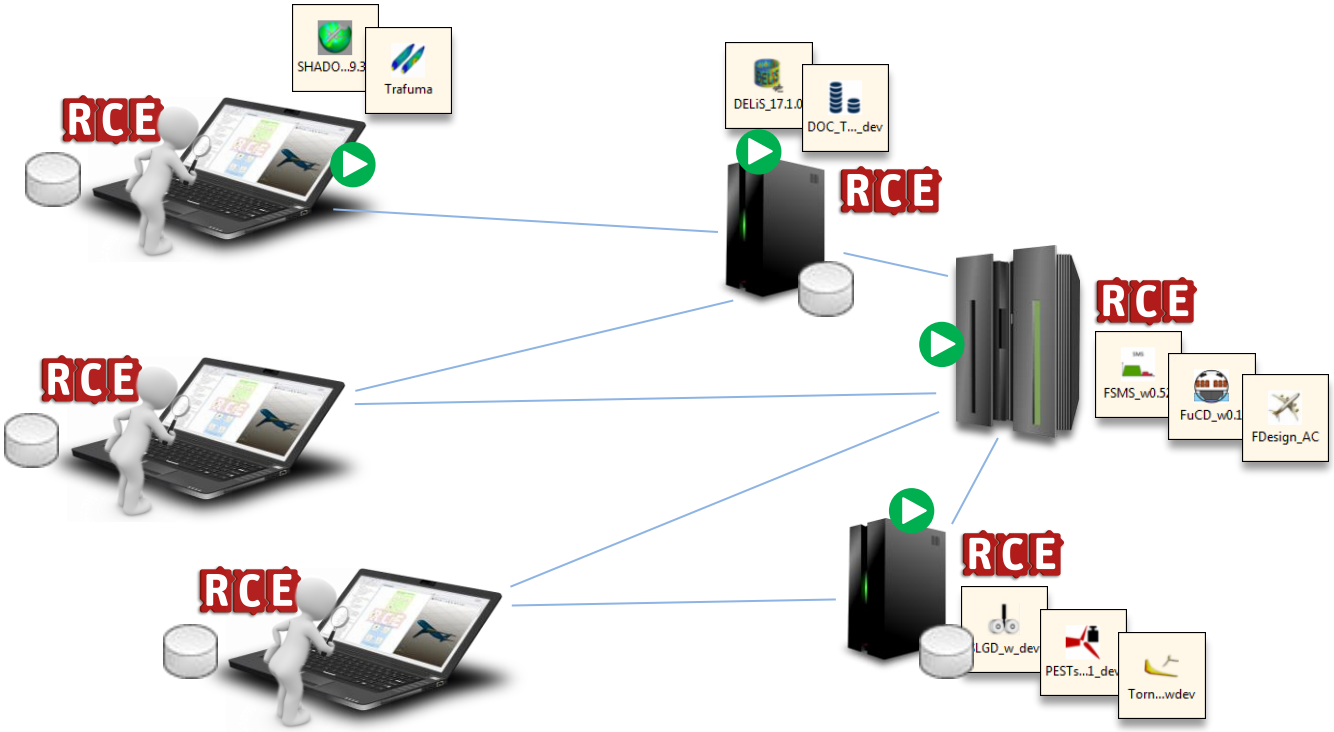




Grafische Benutzungsschnittstelle

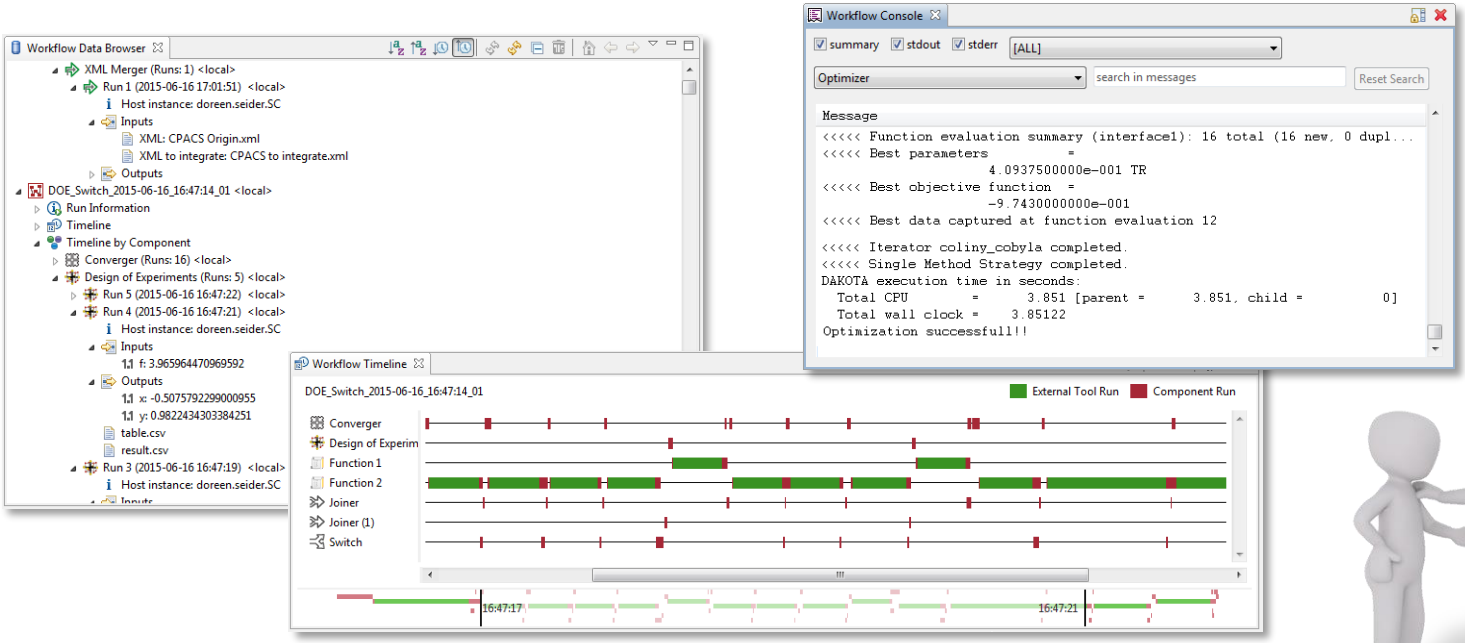


Mit RCE zusammenarbeiten



Mit RCE zusammenarbeiten

Workflowausführung gemeinsam monitoren



Zusammenarbeit im Entwickler-Team

- Heterogenes Team
- Jedes Teammitglied hat eigene Themenschwerpunkte bei Entwicklung
- Keine festen Rollen wie Entwickler, Tester etc.
- Ein fester Ansprechpartner für jedes Forschungsprojekt
- Häufiges Betreuen von Studenten, Praktikanten…

Softwareentwicklung in einer wissenschaftlichen Einrichtung

- Wissenschaftlicher Anteil vs. Produktivsoftware
- Eher wenige Gelegenheiten zu Publikationen
  - Teilweise Publikationen zu Projekten
  - Thema "Software publizierbar/zitierbar machen"
- Bewertung durch typische Kennzahlen schwierig

RCE als Open Source Software

- RCE ist Open Source, bisher gibt es aber keine „Developer Community“ außerhalb des DLR

Herausforderungen auf dem Weg zu „offenerer“ Entwicklung:

- Wie funktioniert die Koordinierung verschiedener Features?
- Schaffung von Infrastrukturen nötig, z.B.
  - Plattform für Konzeptdiskussionen
  - Mailingliste, Forum etc. (teils erschwert durch IT-Richtlinien)
- Projekt bekannt machen/etablieren vs. Projekt-Breite handhabbar halten

Interaktion mit Nutzern

- Benutzer-Workshops ca. alle 1-1,5 Jahre
  - Mischung aus Präsentationen und Gruppendiskussionen (Anforderungsanalyse, geplante Konzepte validieren)
  - Erfahrungsaustausch zwischen den Nutzern (Präsentationen)
  - Feedback an uns als Entwickler
- Bug-Reports, Vorschläge, Support-Anfragen von Nutzern
- Test-Snapshots für neue Features (oft für einzelne Projekte)
- Projekttreffen
- Hands-On-Workshops (Einführung in RCE)

Forschungsprototyp vs. wissenschaftliche Produktivsoftware

- Forschungsprototyp:
  - Entwickler oft selbst die Anwender (oder im direkten Kontakt)
    - in der Regel wenige Personen
  - flexible Anpassung nach Bedarf; Ansätze „ausprobieren“
  - Software **ist** oft Forschung

- Wissenschaftliche Produktivsoftware:
  - Entwickler und Anwender sind oft im Kontakt, aber nicht unbedingt direkt
    - tendenziell größerer Kreis
  - mehr Fokus auf Robustheit, Rückwärtskompatibilität, Usability, Dokumentation, Administrierbarkeit, …
  - Software **ermöglicht** oft Forschung

Finanzierung wissenschaftlicher (OS-)Produktivsoftware

- Finanzierung über Forschungsprojekte
  - Verhandlungen mit potenziellen Projektpartnern
  - neue Features oder fachliche Unterstützung ↔ Finanzierungsumfang

- Projektziele oft offen/unscharf
  - nicht nur **wie** (normal in der Softwareentwicklung), sondern **was**; Interaktion mit Forschungsaktivitäten!
  - Lösung: agile Methoden?
    - *kann* funktionieren, aber auch kollidieren
    - Ressourcen und Roadmaps i.d.R. fixiert
  - Zielerreichung selten objektiv messbar / sanktionierbar
    - Korrektiv: der „gute Ruf“ als Projektpartner → Folgeprojekte!

Finanzierung wissenschaftlicher Produktivsoftware: die Probleme

- Alles wunderbar?
  Mit Projektpartnern reden, neue Features finanziert bekommen, gute Arbeit abliefern?

- …leider nicht ganz

- zentrales Problem: Produktivsoftware vs. Prototyp!
  - Maintenance (z.B. neue OS-Versionen, Library-Upgrades, Security, CI-Setup, …)
  - Qualitätssicherung *bestehender* Features
    (Rückwärtskompatibilität, Konsistenz zu neuen Features, …)
  - Support-Anfragen
  - größeres Team → mehr Overhead
  - mehr „Kathedrale“ als „Basar“

Finanzierung wissenschaftlicher Produktivsoftware: die Probleme

- gleichzeitig immer weniger Neuerungsbedarf
  - reife Software oft „gut genug“!
  - …aber woher kommt dann die Finanzierung?

- zum Teil klassisches Problem von Open-Source-Software
  - aber: typische OSS-Finanzierungsmodelle oft nicht möglich
  - gleichzeitig ist Open Source für Wissenschaft extrem sinnvoll!
    - Reproduzierbarkeit, Prüfbarkeit der Verfahren, …

- wichtig: Projektplanung und Wartungsaufwand kommunizieren
  - User-Workshops, offener Issue-Tracker, …

Kompatibilität: pro und contra

- wichtig v.a. bei verteilter Software (Protokolle) und Dateiformaten
- pro:
  - Wissenschaftsanwender oft konservativ mit Updates
    - …kooperieren aber über Institutionen hinweg
  - Kompatibilität macht Koordination in Projekten einfacher!
- contra:
  - sehr hoher Mehraufwand! (QA)
  - Einfügen neuer Features deutlich verzögert („breaking changes“)
    - auch Verbesserungen und teilweise Bugfixes
- Empfehlung: genau abwägen, wie viel Kompatibilität den Aufwand wert ist

Feature-Breite vs. Projektmanagement

- wichtige Frage aus Projektmanagementsicht: welche Features einbauen/verfolgen?

- Problem der Projektfinanzierung: Anforderungen oft speziell
  - besonders bei vielen kleinen Projekten; teilweise <0,5 Stelle/Projekt
  - Gefahr: „Flickenteppich“ / „Zick-Zack-Kurs“
  - ideal: Projektpartner überzeugen, breit anwendbare Features zu finanzieren

- manche Features will jeder haben, aber sie passen in kein Projekt
- Priorisierung Projektwünsche ↔ Kernentwicklung

Feature-Breite vs. Projektmanagement

- Features vs. Maintenance-Aufwand (v.a. Testen)
- kann man Features jemals wieder entfernen, und wenn ja, wann?
  - tatsächliche Nutzung oft schwer abschätzbar
  - was wird noch benutzt?
    - Telemetrie → Datenschutz

Zusammenfassung

- Mit RCE werden Programme zu Workflows verbunden und verteilt ausgeführt
- Einsatz in Forschung und Industrie
- weitgehend über Forschungsprojekte finanziert
- Open Source
- Beteiligungen jeglicher Art willkommen

Fazit/Emfehlungen

- Forschungsprototyp vs. Produktivsoftware → möglichst früh Ziel überlegen
- Feature-Strategie (Scope) überlegen und so gut wie möglich beibehalten
- Kompatibilität gut abwägen: wie viel wird wirklich gebraucht?
- Projektmanagement für Benutzer transparent machen

Q&A

Download: http://rcenvironment.de
Twitter: http://twitter.com/rcenvironment
YouTube: http://www.youtube.com/rcenvironment
GitHub: https://github.com/rcenvironment