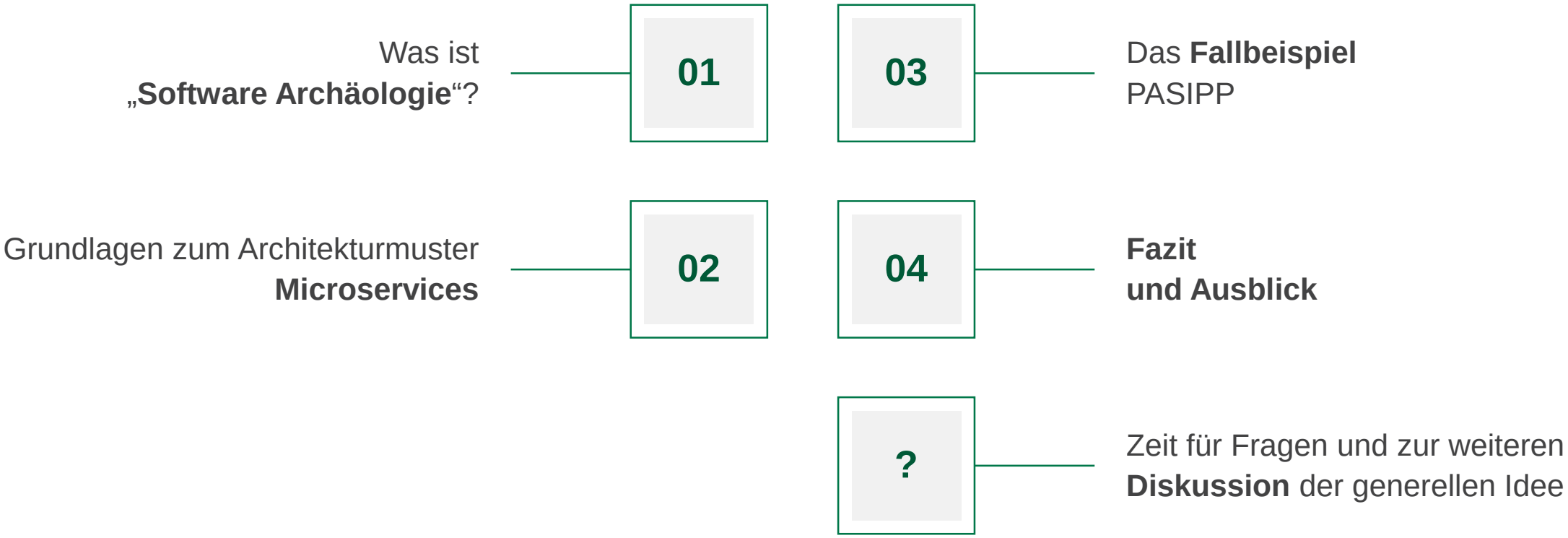
Software-Archäologie mithilfe von Microservices am Beispiel von PASIPP

Software Archaeology Using Microservices with the Example of PASIPP

Sascha Alpers

14. Juni 2019 | Jahrestagung der GI-Fachgruppe „Architekturen“ | Karlsruhe

Agenda


Was ist „**Software Archäologie**“?
01
Grundlagen zum Architekturmuster **Microservices**
02
03
Das **Fallbeispiel** PASIPP
04
Fazit und Ausblick
?
Zeit für Fragen und zur weiteren **Diskussion** der generellen Idee

Software Archäologie

Alte Software-Systeme als Schätze betrachten

- Alte Systeme laufen / liefen über ihre geplante Nutzungsdauer hinaus.
- Alte Systeme enthalten Wissen, welches in Code umgewandelt wurde.
- Artefakte werden von alten Speichermedien oder alten Publikationen wiederhergestellt.
- Das zuverlässige Wissen steckt im Quellcode selbst (Dokumentation ist veraltet oder unzureichend).
- Nicht jedes alte System enthält Schätze.

Vgl. G. Chroust (2005): „Software-Archäologie – eine interdisziplinäre Betrachtung“;
I. Philippow, I. Pashov, M. Riebisch (2003): „Application of feature modelling for architecture revovery“

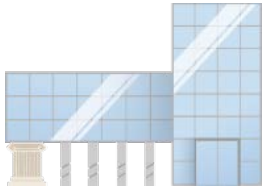
Vorgehensmodell Software Archäologie

Suche von (potentiell) wertvollen alten Software Artefakten

Präparieren der Software Artefakte

(bspw. Laufzeitumgebung beziehen, installieren und einrichten)

Analyse der Software Artefakte
(Verstehen, Wissen extrahieren, Vorbereiten des Wieder-Einsatzes).



Integrieren in neue Systeme

(extrahiertes Wissen in neue Systeme gießen oder alte Artefakte nutzbar machen)

Vgl. A. Hunt; D. Thomas (2002): „Software archaeology“

Typische Probleme eines Software Archäologen

- Entwickler sind nicht (mehr) verfügbar.
- Motive oder Konzepte der Entwicklung wurden vergessen oder können nicht nachvollzogen bzw. verstanden werden.
- Das Wissen selbst ist inzwischen teilweise vergessen worden, z.B. über die Behandlung von Sonderfällen. Der Quellcode ist die einzige "Dokumentation" des Wissens.
- Viele Merkmale des Altsystems lassen sich nur im Kontext der früheren Entwicklungszeit (Technologien, Projektrahmenbedingungen, etc.) verstehen.
- Die Analyse des Systems wird durch vorgefasste (falsche) Meinungen über den Zweck und die Nutzung des Systems verzerrt, da die Erwartungen des „Archäologen“ das Ergebnis seiner Betrachtung beeinflussen.

teils aus G. Chroust (2005): „Software-Archäologie – eine interdisziplinäre Betrachtung“;

weitere typische Probleme

- Teile des Systems wurden (mehrfach) geändert. Dies hat möglicherweise das ursprüngliche Software-Design verwässert, was die Architektur noch schwieriger zu verstehen macht.
- Teile des Systems sind nicht verfügbar.
- Einige Teile des Systems sind nicht (mehr) relevant. Manchmal gibt es sogar Komponenten, die nicht mehr zugänglich sind. Allerdings werden Analyse und Verständnis dadurch immer komplexer.

teils aus G. Chroust (2005): „Software-Archäologie – eine interdisziplinäre Betrachtung“;

„Never change a running system!“ ?

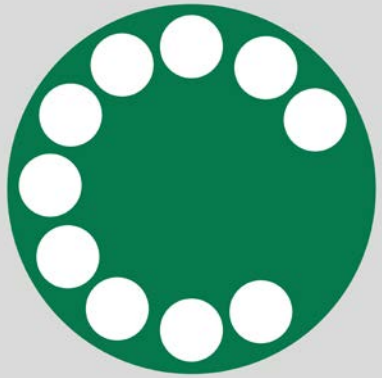
Warum nicht das Altsystem unverändert weiter betreiben und nutzen?

Die Anforderungen und Erwartungen der Benutzer ändern sich (UUX, etc.).

Technologie (Programmiersprache, Frameworks, etc.) passt nicht mehr (nicht weit verbreitet, nicht unterstützt, unzuverlässig, unsicher, etc.).

Es besteht ein Bedarf an Interaktion mit der sich verändernden Umgebung (Schnittstellen, etc.).

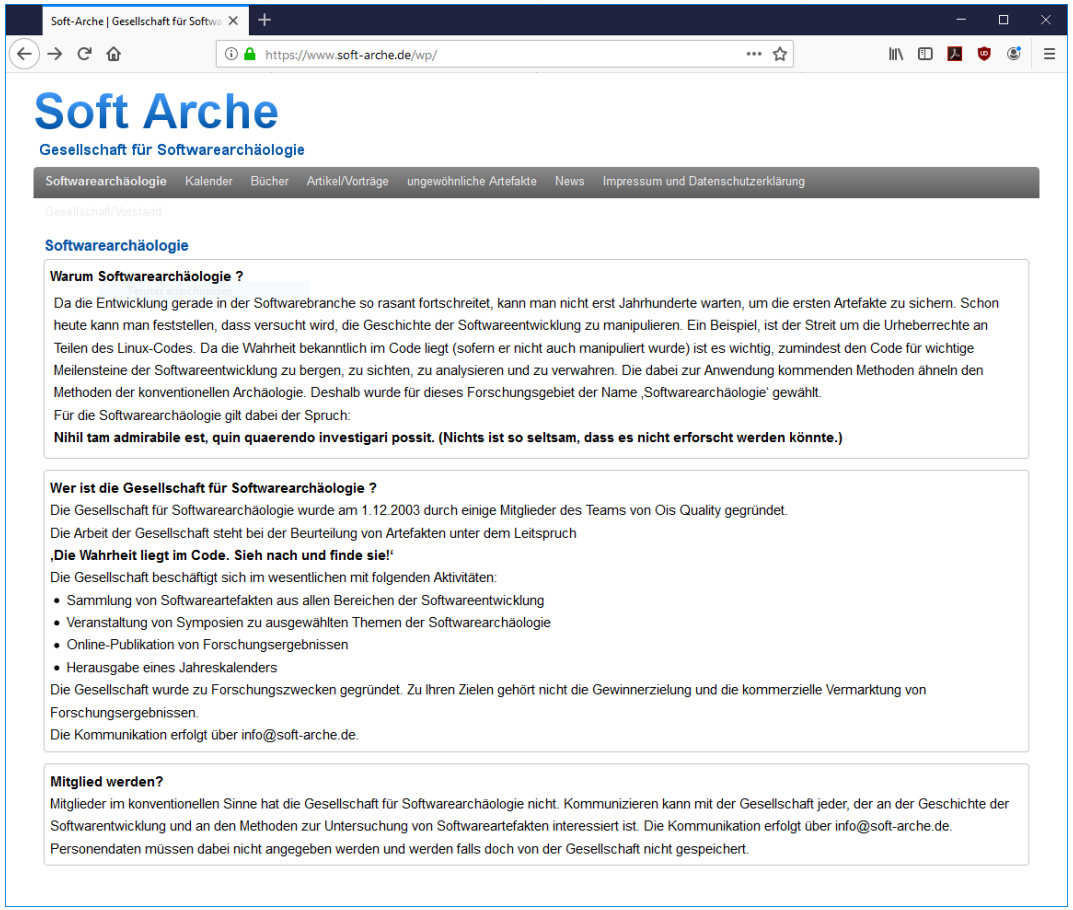
Die wertvolle Funktionalität ist nur ein Teil des Altsystems und der Wert entsteht nur bei Verwendung in einem anderen System.



teils aus G. Chroust (2005): „Software-Archäologie – eine interdisziplinäre Betrachtung“;

Ziel der Softwarearchäologie
- im Sinne der "Geschichtsschreibung "

- Software aus geschichtlichem Interesse für die Nachwelt erhalten? (So sah die erste Version von Windows, Tetris, … aus!)
- Klärung von hsitorischen Fragen wie z.B. „Wie ist Windows entstanden?“ oder strittiger „Wer hat was genau zu der Entwicklung von Linux beigetragen?“



Screenshoot vom 12. Juni 2019 Webseite der „Gesellschaft für Softwarearchäologie“, Wolfgang Ulbrich, http://www.soft-arche.de

Ziel der Softwarearchäologie
- im Sinne einer produktiven Verwendung

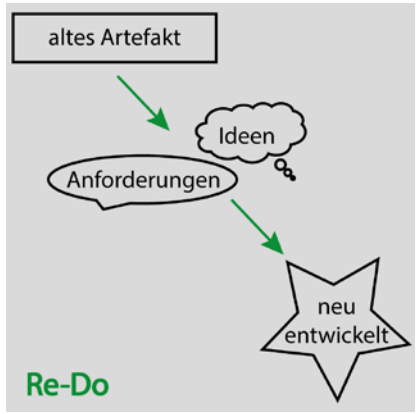
- Extrahieren und Wiederverwenden von Wissen, dass zur Entwicklungs- und Wartungszeit in das Artefakt eingeflossen ist (technisch, konzeptionell, etc.).

vgl. G. Chroust (2005): „Software-Archäologie – eine interdisziplinäre Betrachtung“

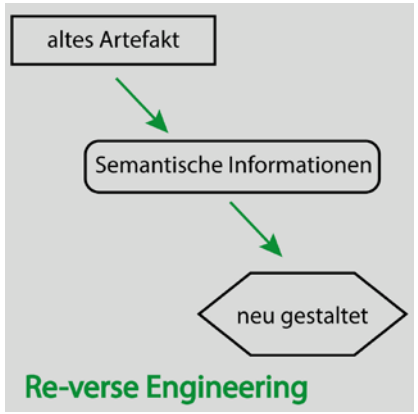
oder

- Das System oder Teile des Systems selbst wiederverwenden.

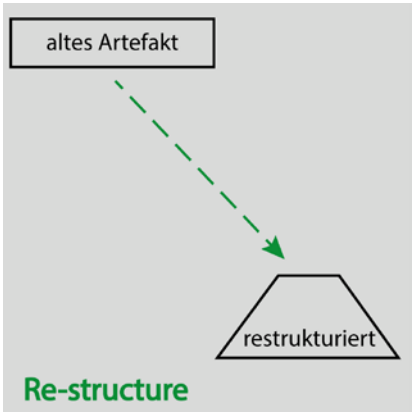
Fünf „Re“ der Wiederverwendung (Re-Use)



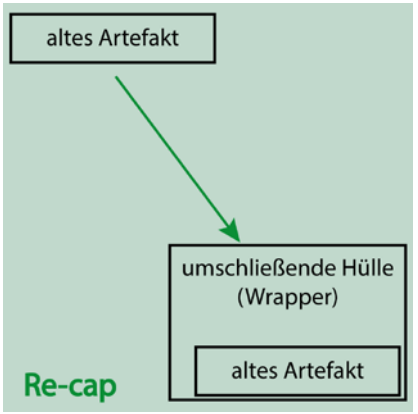
Die Komponente wird **komplett neu entwickelt**, ohne mehr als die Ideen und Anforderungen des alten Systems zu übernehmen.



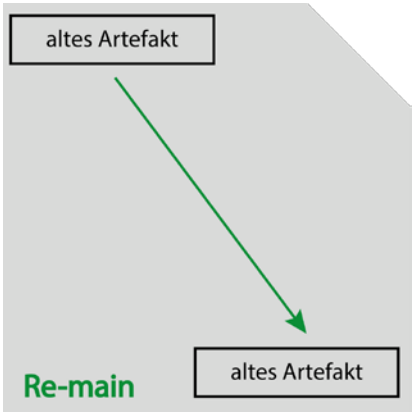
Semantische Informationen werden aus den bestehenden Komponenten gewonnen (Quellcode-Analyse, Suche nach Architektur-entscheidungen, Prozessen, Ideen der Algorithmen, …) und die Anwendung **neu gestaltet**.



Das Modul wird weitgehend automatisch **neu strukturiert** und damit an die neuen Anforderungen angepasst.

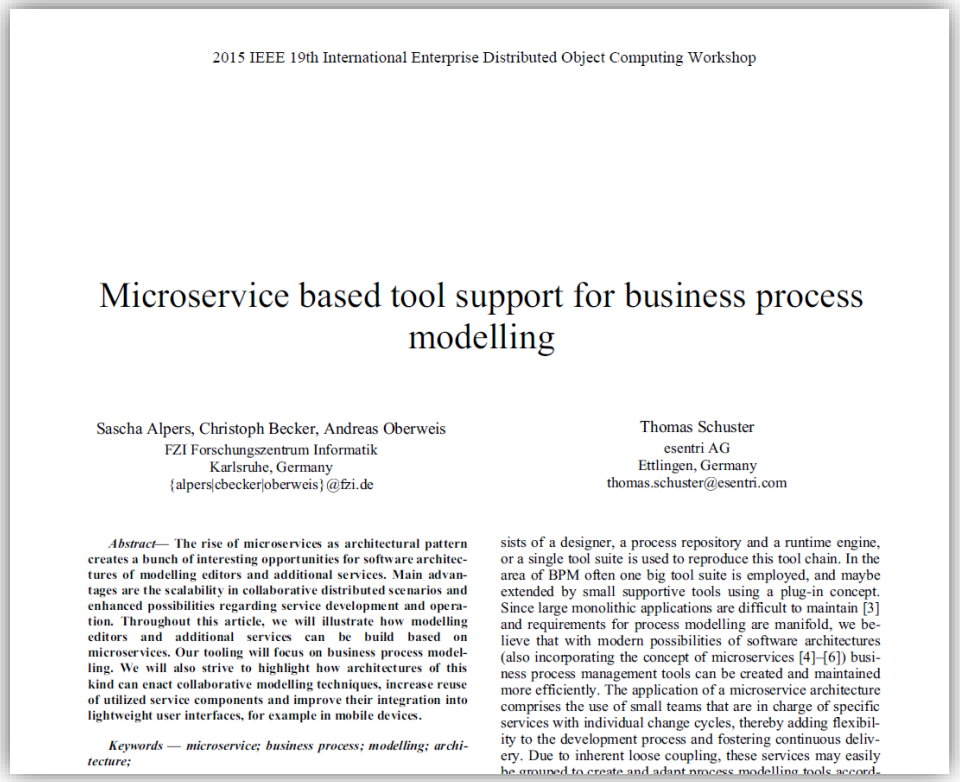


Wiederverwendung durch **Einschließen des Moduls** mit einer neuen Hülle. Die Hülle übernimmt die Funktion eines Wrappers.



Wiederverwendung **ohne Modifikation**.

teils aus / nach G. Chroust (1996): „Software 2001: Ein Weg in die Wiederverwendungswelt“;



2015 IEEE 19th International Enterprise Distributed Object Computing Workshop

Microservice based tool support for business process modelling

Sascha Alpers, Christoph Becker, Andreas Oberweis
FZI Forschungszentrum Informatik
Karlsruhe, Germany
{alpers|cbecker|oberweis}@fzi.de

Thomas Schuster
esentri AG
Ettlingen, Germany
thomas.schuster@esentri.com

***Abstract*— The rise of microservices as architectural pattern creates a bunch of interesting opportunities for software architectures of modelling editors and additional services. Main advantages are the scalability in collaborative distributed scenarios and enhanced possibilities regarding service development and operation. Throughout this article, we will illustrate how modelling editors and additional services can be build based on microservices. Our tooling will focus on business process modelling. We will also strive to highlight how architectures of this kind can enact collaborative modelling techniques, increase reuse of utilized service components and improve their integration into lightweight user interfaces, for example in mobile devices.**

Keywords — microservice; business process; modelling; architecture;

sists of a designer, a process repository and a runtime engine, or a single tool suite is used to reproduce this tool chain. In the area of BPM often one big tool suite is employed, and maybe extended by small supportive tools using a plug-in concept. Since large monolithic applications are difficult to maintain [3] and requirements for process modelling are manifold, we believe that with modern possibilities of software architectures (also incorporating the concept of microservices [4]–[6]) business process management tools can be created and maintained more efficiently. The application of a microservice architecture comprises the use of small teams that are in charge of specific services with individual change cycles, thereby adding flexibility to the development process and fostering continuous delivery. Due to inherent loose coupling, these services may easily be grouped to create and adapt process modelling tools accord-

Microservices

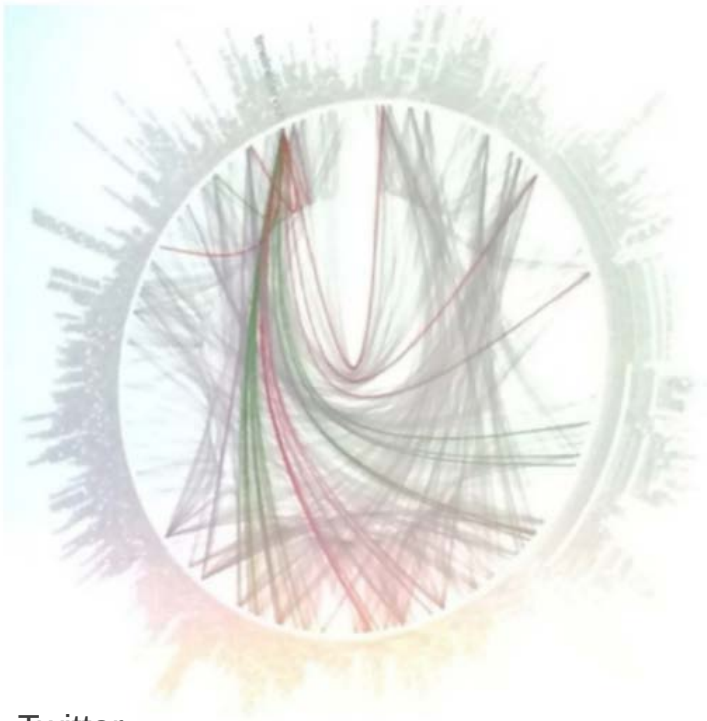
Bieten Sie eine Chance der Wiederverwendung (im Sinne von „recap“)?

What is a microservice architecture?

- Suite of “small”-sized services
  - Executed in its process space
  - Own database
  - Lightweight communication

- Samples:

Netflix



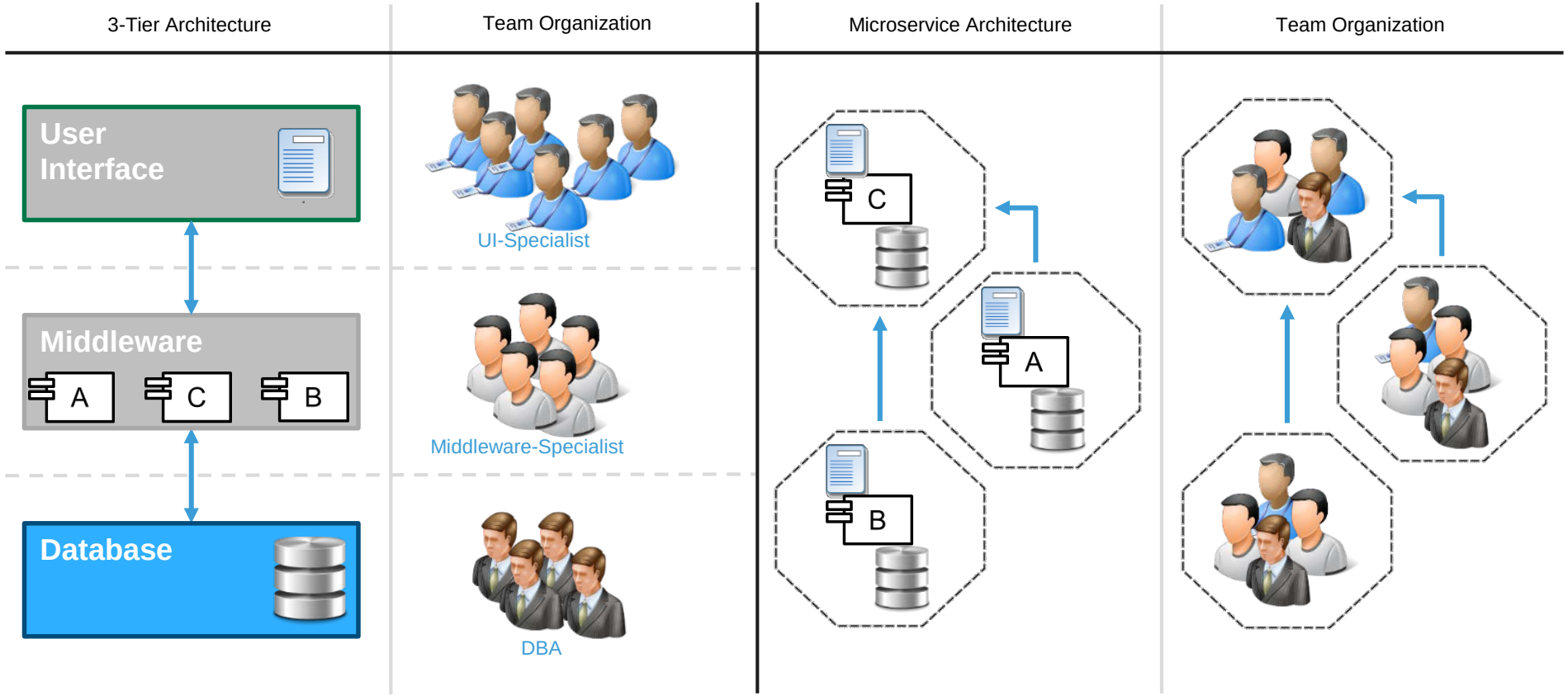
Twitter

Slide of the presentation of the paper at EDOC 2015 by Christoph Becker

http://de.slideshare.net/adriancockcroft/microxchg-microservices

Difference from monolithic architecture

- Dependencies between team organization and software architecture (Conway's Law)

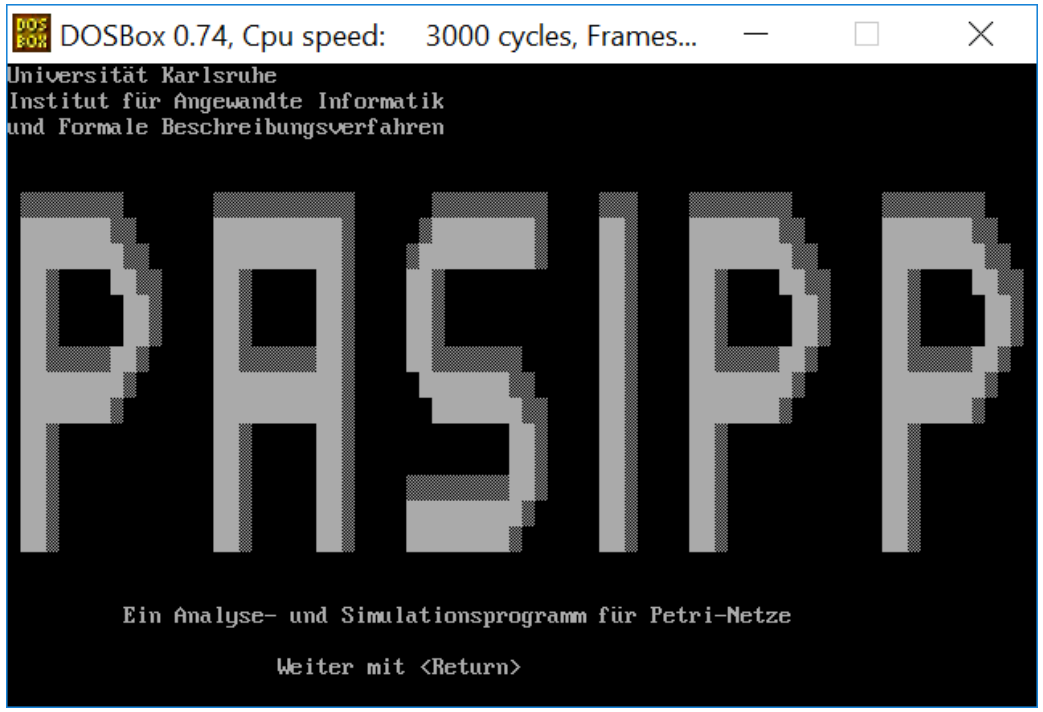


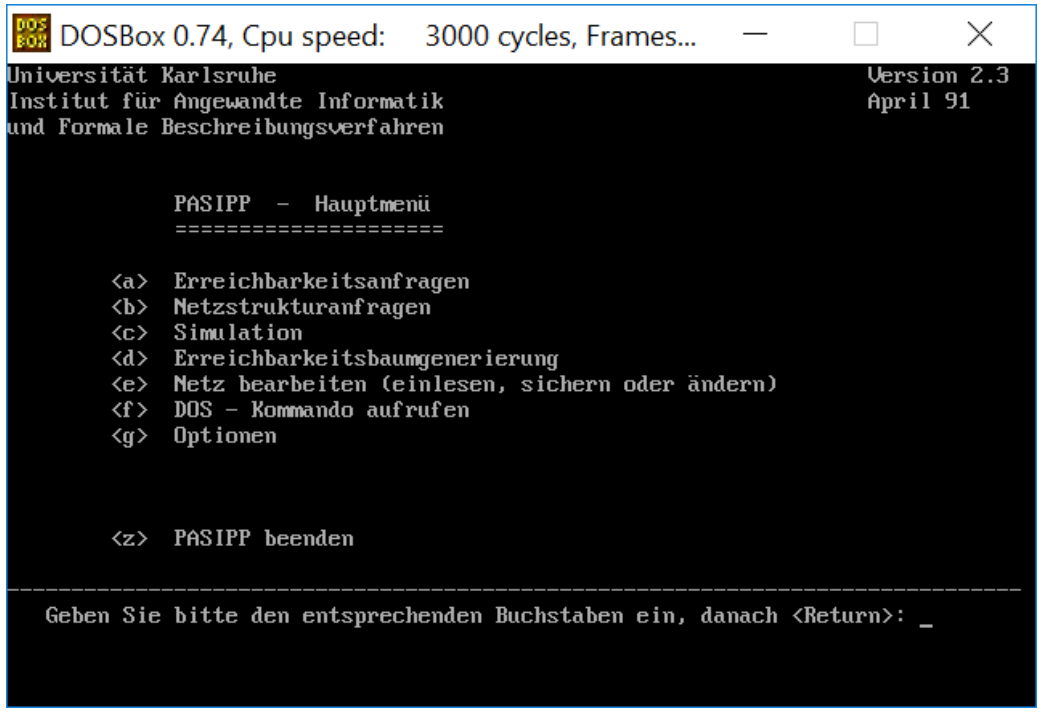
Slide of the presentation of the paper at EDOC 2015 by Christoph Becker

Characteristics of microservices

- Decentralized governance
- Shared nothing
- Decentralized data management
- Smart endpoints and dumb pipes
- Infrastructure automation
- Evolutionary design

Slide of the presentation of the paper at EDOC 2015 by Christoph Becker





Fallbeispiel PASIPP

Vielen Dank an

Fabian Stolz	▪ für die prototypische Implementierung der Idee in seiner Masterarbeit (betreut von Sascha Alpers, Andreas Oberweis, Ralf Reussner)
Meike Ullrich	▪ für ihren prototypischen Dienst zur Konvertierung von Horus PNML Dateien zu ISO/IEC 15909 konformen PNML Dateien
Andreas Oberweis	▪ für das Wiederfinden und Auslesen der PASSIP-Disketten sowie die Bereitstellung des Quellcodes für den Prototyp

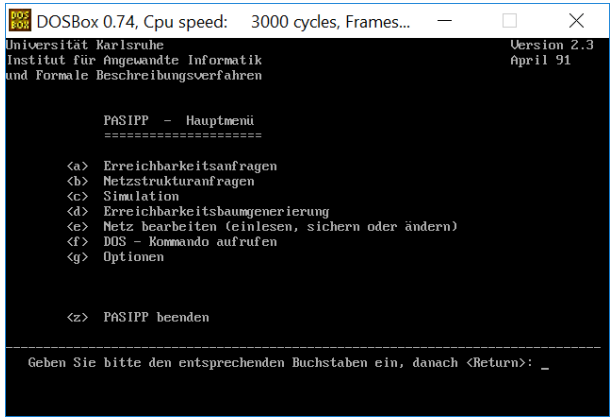
PASIPP - Präparieren des Software Artefaktes

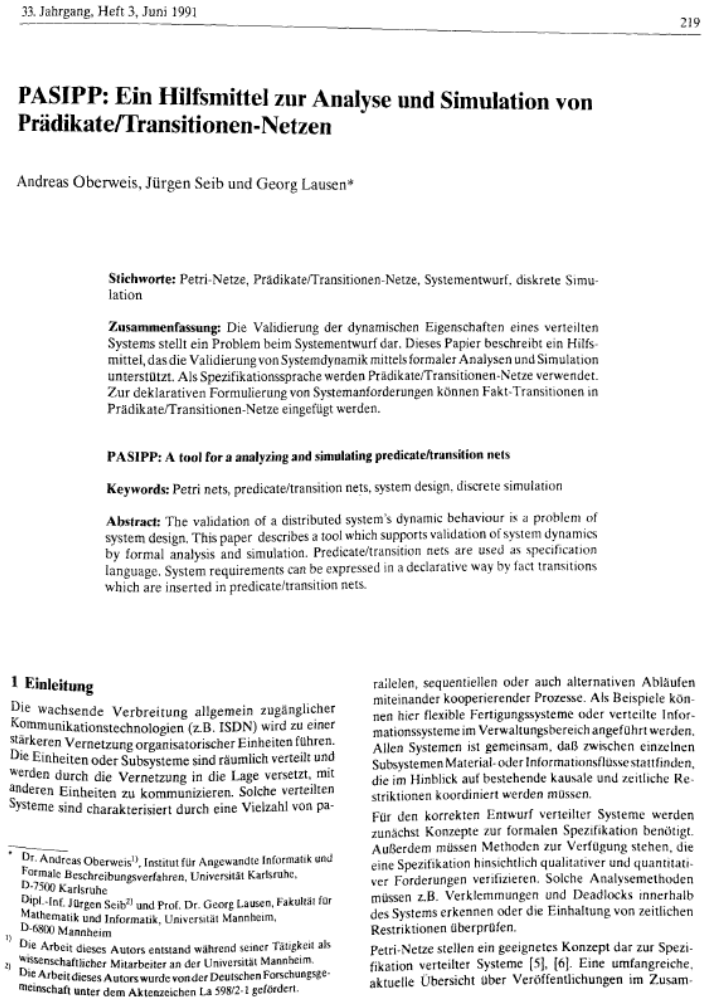
- Kompiliertes PASIPP in einer virtuellen DOS Umgebung ausgeführt
  (z.B. DosBox 0.74, https://www.dosbox.com)

- Recherche und Einrichten eines Prolog-Systems (Entwicklungs- und Laufzeitumgebung)
  - Arity/Prolog32 war auf der Webseite von Peter Gabel (einem der Entwickler von Arity/Prolog32 und Mitgründer der Arity Corporation) verfügbar (http://www.peter-gabel.com, Stand 2017).
  - Lizenziert unter der „Creative Commons Attribution-NonCommercial-NoDerivs 3.0”-Lizenz
  - Aktuell (12. Juni 2019) ist die Webseite von Peter Gabel leider nicht mehr erreichbar, wir haben jedoch eine Kopie von Arity/Prolog32 unter https://github.com/fzi-forschungszentrum-informatik/PasippMicroservice/tree/master/ArityProlog32 veröffentlicht.

PASIPP - Analysieren des Software Artefaktes

- Ein Werkzeug von 1991 (Version 2.3)
- Entwickelt von Andreas Oberweis, Volker Sänger und anderen
- Implementiert in Prolog
- Sekundärquellen: Wissenschaftliche Veröffentlichungen bspw. Andreas Oberweis, Jürgen Seib und Georg Lausen (1991): „PASIPP: Ein Hilfsmittel zur Analyse und Simulation von Prädikate/Transitionen-Netzen“. In Wirtschaftsinformatik Jahrgang 33, Heft 3, S. 219-230





33. Jahrgang, Heft 3, Juni 1991 219

PASIPP: Ein Hilfsmittel zur Analyse und Simulation von Prädikate/Transitionen-Netzen

Andreas Oberweis, Jürgen Seib und Georg Lausen*

Stichworte: Petri-Netze, Prädikate/Transitionen-Netze, Systementwurf, diskrete Simulation

Zusammenfassung: Die Validierung der dynamischen Eigenschaften eines verteilten Systems stellt ein Problem beim Systementwurf dar. Dieses Papier beschreibt ein Hilfsmittel, das die Validierung von Systemdynamik mittels formaler Analysen und Simulation unterstützt. Als Spezifikationssprache werden Prädikate/Transitionen-Netze verwendet. Zur deklarativen Formulierung von Systemanforderungen können Fakt-Transitionen in Prädikate/Transitionen-Netze eingefügt werden.

PASIPP: A tool for a analyzing and simulating predicate/transition nets

Keywords: Petri nets, predicate/transition nets, system design, discrete simulation

Abstract: The validation of a distributed system's dynamic behaviour is a problem of system design. This paper describes a tool which supports validation of system dynamics by formal analysis and simulation. Predicate/transition nets are used as specification language. System requirements can be expressed in a declarative way by fact transitions which are inserted in predicate/transition nets.

1 Einleitung

Die wachsende Verbreitung allgemein zugänglicher Kommunikationstechnologien (z.B. ISDN) wird zu einer stärkeren Vernetzung organisatorischer Einheiten führen. Die Einheiten oder Subsysteme sind räumlich verteilt und werden durch die Vernetzung in die Lage versetzt, mit anderen Einheiten zu kommunizieren. Solche verteilten Systeme sind charakterisiert durch eine Vielzahl von parallelen, sequentiellen oder auch alternativen Abläufen miteinander kooperierender Prozesse. Als Beispiele können hier flexible Fertigungssysteme oder verteilte Informationssysteme im Verwaltungsbereich angeführt werden. Allen Systemen ist gemeinsam, daß zwischen einzelnen Subsystemen Material- oder Informationsflüsse stattfinden, die im Hinblick auf bestehende kausale und zeitliche Restriktionen koordiniert werden müssen.

Für den korrekten Entwurf verteilter Systeme werden zunächst Konzepte zur formalen Spezifikation benötigt. Außerdem müssen Methoden zur Verfügung stehen, die eine Spezifikation hinsichtlich qualitativer und quantitativer Forderungen verifizieren. Solche Analysemethoden müssen z.B. Verklemmungen und Deadlocks innerhalb des Systems erkennen oder die Einhaltung von zeitlichen Restriktionen überprüfen.

Petri-Netze stellen ein geeignetes Konzept dar zur Spezifikation verteilter Systeme [5], [6]. Eine umfangreiche, aktuelle Übersicht über Veröffentlichungen im Zusam-

* Dr. Andreas Oberweis[1], Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Universität Karlsruhe, D-7500 Karlsruhe
Dipl.-Inf. Jürgen Seib[2] und Prof. Dr. Georg Lausen, Fakultät für Mathematik und Informatik, Universität Mannheim, D-6800 Mannheim
[1] Die Arbeit dieses Autors entstand während seiner Tätigkeit als wissenschaftlicher Mitarbeiter an der Universität Mannheim.
[2] Die Arbeit dieses Autors wurde von der Deutschen Forschungsgemeinschaft unter dem Aktenzeichen La 598/2-1 gefördert.

PASIPP - Analysieren des Software Artefaktes

Quellcode-Ausschnitt

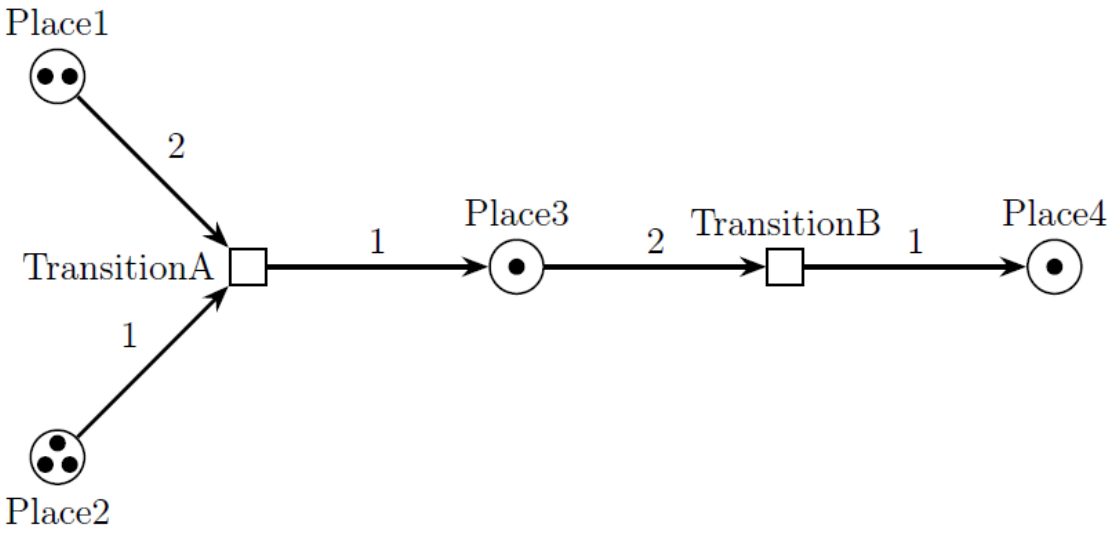
```
1  fire([transitionsname,Inputvariablen,Outputvariablen]) :-
2      entferne(anzahl[1],stellenname[1](marke[1])),
3      ...
4      entferne(anzahl[i],stellenname[i](marke[i])),
5      transitionsbeschriftung,
6      einfuege(anzahl[i+1],stellenname[i+1](marke[i+1])),
7      ...
8      einfuege(anzahl[n],stellenname[n](marke[n])),
```

vgl. A. Oberweis, J. Seib and G. Lausen (1991): „PASIPP: Ein Hilfsmittel zur Analyse und Simulation von Prädikate/Transitionen-Netzen“

FILE	LINES OF CODE (incl. comments)
ANALYSE.ARI	237
BUILD.ARI	344
DYNAMIC.ARI	330
HELP.ARI	416
HELP1.ARI	376
INIT.ARI	1.007
NET_ANA.ARI	217
NETWORK.ARI	156
OPERAT.ARI	771
OPTIONS.ARI	287
PASIPP.ARI	204
SIMULAT.ARI	933
SPECIAL.ARI	204
STATIC.ARI	285
TREE.ARI	526
TOTAL	**6.293**

PASIPP - Analysieren des Software Artefaktes

Beispiel: Repräsentation eines Petri Netzes

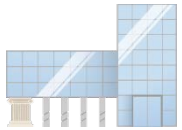


Netzsturktur

```
1 fire(['transitionB']) :-
2         entferne(2,'place3'),
3         einfuege(1,'place4').
4 fire(['transitionA']) :-
5         entferne(2,'place1'),
6         entferne(1,'place2'),
7         einfuege(1,'place3').
```

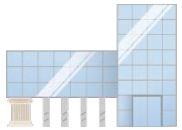
Startmarkierung

```
1 marke(1,'place3').
2 marke(2,'place1').
3 marke(3,'place2').
```



Wiederverwendung von PASIPP

- PASIPP soll als Microservice mit einer Restschnittstelle verfügbar sein.
- Original PASIPP wird weiterhin als einzelne Routinen (ohne GUI) in Prolog ausgeführt.
- Prolog wird von einem Java-Programm als externer Prozess gesteuert und ausgewertet.
- Der Microservice konvertiert
  - PNML-Dateien in Prolog Petri Netz Beschreibungen für PASIPP
  - PASIPP Antworten auf erwartete Formate



Änderungen in PASIPP

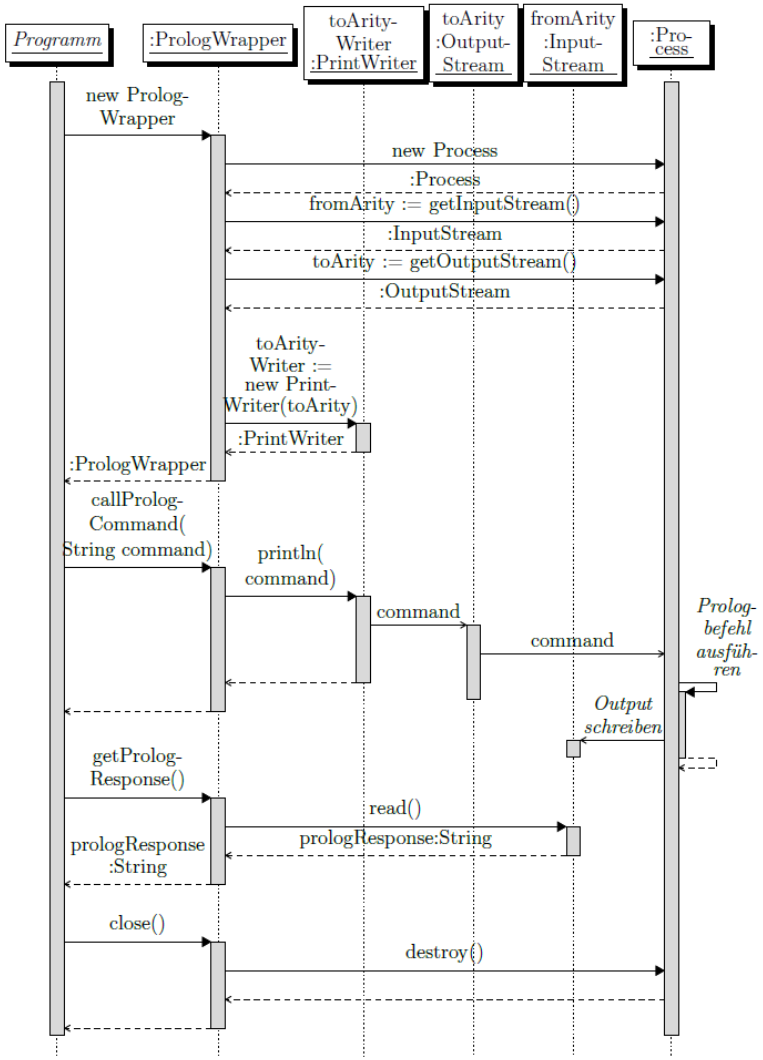
- Kleine Änderungen in PASIPP Quelldateien, damit PASIPP unter Arity/Prolog32 (statt Arity/Prolog) läuft:

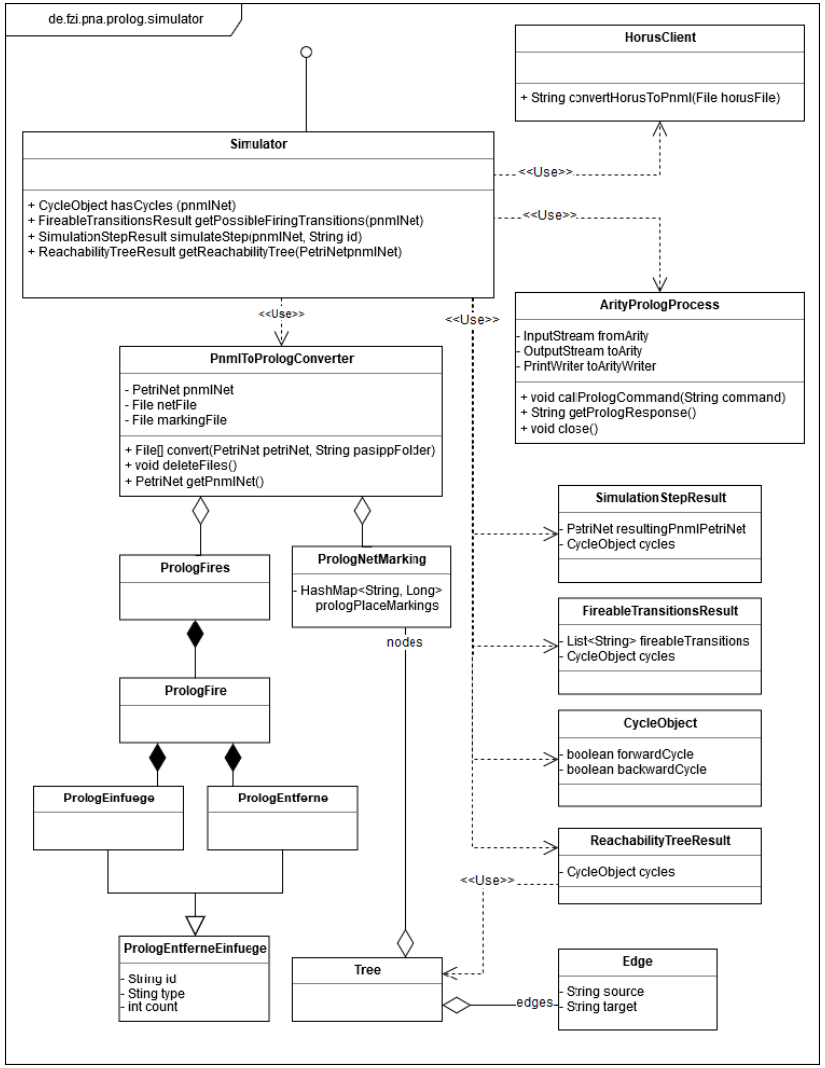
```
-read_string(String) :- read_string(40,List),atom_string(String,List).
+read_string(String) :- read_line(0,String).
```

- und die Ergebnisse komplett statt seitenweise ausgegeben werden

```
1  weiter :-
2    nl,write('                    Weiter mit <Return> '),nl,
3    read_string(_),nl,nl.
4
5  [...]
6
7  ausgabe_one([],_) :- nl,write('Ende der Ausgabe!'),nl,nl.
8
9  ausgabe_one([X|Tail],10) :-
10 -weiter,
11  writeq(X),nl,
12  ausgabe_one(Tail,1),!.
13
14  ausgabe_one([X|Tail],Zeilennr) :-
15    Neue_Nr is Zeilennr + 1,
16    writeq(X), nl,
17    ausgabe_one(Tail,Neue_Nr).
```

Hauptkomponenten des neuen Microservice





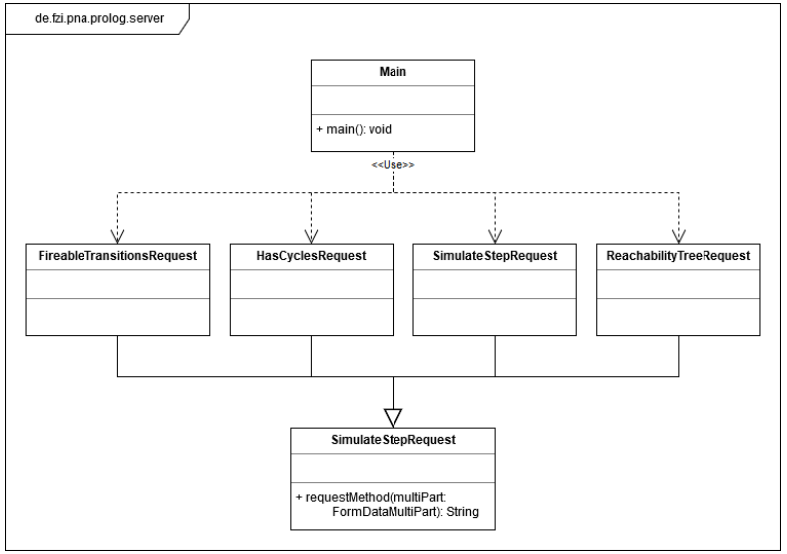
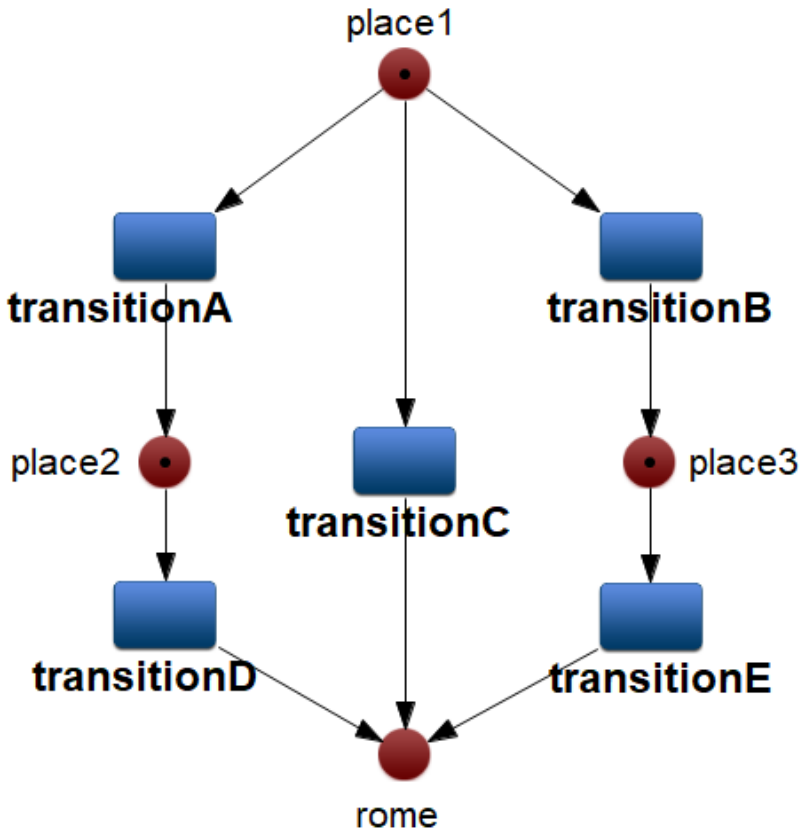
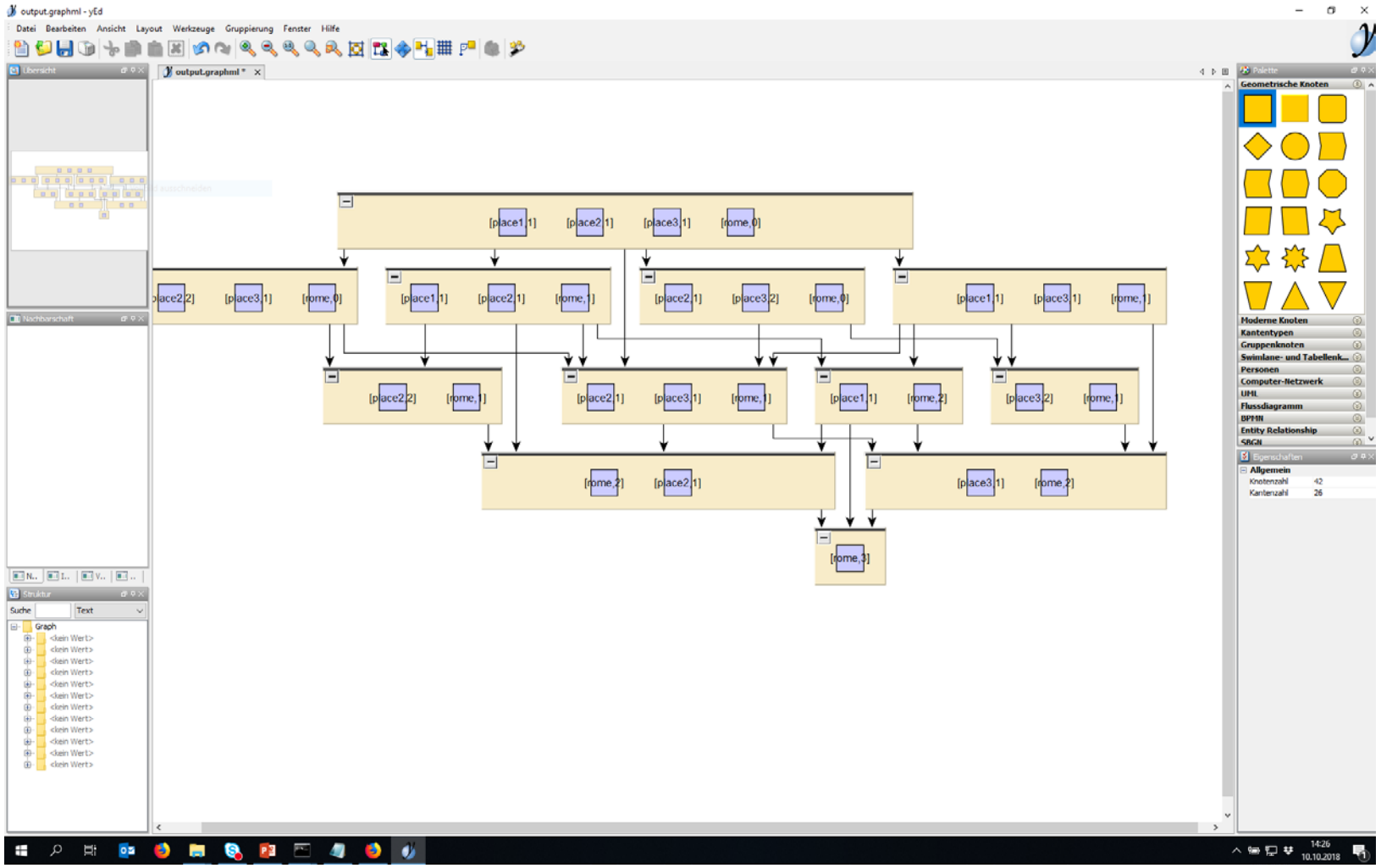


Diagramme von Fabian Stolz

Demo: am Beispielnetz „Alle Wege führen nach Rom“



Demo Screenshots



Fazit und Ausblick

Fazit und Ausblick

- Microservices bieten die Möglichkeit, Artefakte aus Software nahezu wie in realen Softwaresystemen zu nutzen.
  - Schwierigkeit kleine Module aus Altsystemen herauszulösen bleibt bestehen (bspw. bzgl. Datenhaltung)

- Beispiel PASIPP ist als Prototyp implementiert.
  - Mehr Funktionalität und weitere Tests in zukünftigen Arbeiten.
  - PASIPP als Mikroservice ist eine Chance für neue Horus-Funktionalitäten.
  - Das Ergebnisformat muss angegeben werden, und die Benutzeroberfläche muss das Ergebnis für den Benutzer visualisieren.


VIELEN
DANK

Kontakt

FZI Forschungszentrum Informatik

Sascha Alpers
Forschungsbereich Software Engineering

Haid-und-Neu-Str. 10-14
76131 Karlsruhe

Tel: +49 721 9654 - 616
E-Mail: alpers@fzi.de
Web: www.fzi.de