FZI

FZI FORSCHUNGSZENTRUM INFORMATIK

Software-Archäologie mithilfe von Microservices am Beispiel von PASIPP

Software Archaeology Using Microservices with the Example of PASIPP

Sascha Alpers

12. Oktober 2018 | Wart, Altensteig | Klausurtagung der PROMATIS software GmbH

Agenda

- Software archaeology
- Microservices
- Example PASIPP
- Conclussion

SOFTWARE ARCHAEOLOGY

Looking at old systems as treasures

- Old systems run beyond their planned useful life.
- Old systems contain knowledge that has been cast into code.
- Artifacts are recovered from old storage media or old publications.
- The reliable knowledge is in the source code itself (documentation is outdated or insufficient).
- Not every old system includes treasures.

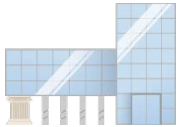
compare G. Chroust (2005): „Software-Archäologie – eine interdisziplinäre Betrachtung“;
I. Philippow, I. Pashov, M. Riebisch (2003): „Application of feature modelling for architecture revovery“

Software archaeology process

Investigate and find old software artefacts.

Prepare software artefacts
(e.g., get runtime environment).

Analyse software artefacts
(understand, extract knowledge, prepare for re-use).



Integrate in new systems.

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

Typical problems of software archaeologists

- Developers are not available.
- Motives or concepts of the development have been forgotten or cannot be understood.
- The knowledge itself was partly forgotten in the meantime, e.g., concerning the treatment of special cases. Source code is the only “documentation” of the knowledge.
- Many characteristics of the legacy system can only be understood in the context of the former time of development (technologies, project framework conditions, etc.).
- The analysis of the system is distorted by preconceived (false) opinions about the purpose and use of the system because the expectations influence the result.

- Parts of the system have been changed (several times). This may have diluted the original software design, making the architecture even more difficult to understand.
- Parts of the system are not available.
- Some parts of the system are not (any longer) relevant. Sometimes there are even components that are no longer accessible. However, analysis and understanding become more complex as a result.

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

“Never change a running system!” ? Why not to use an old system as it is?

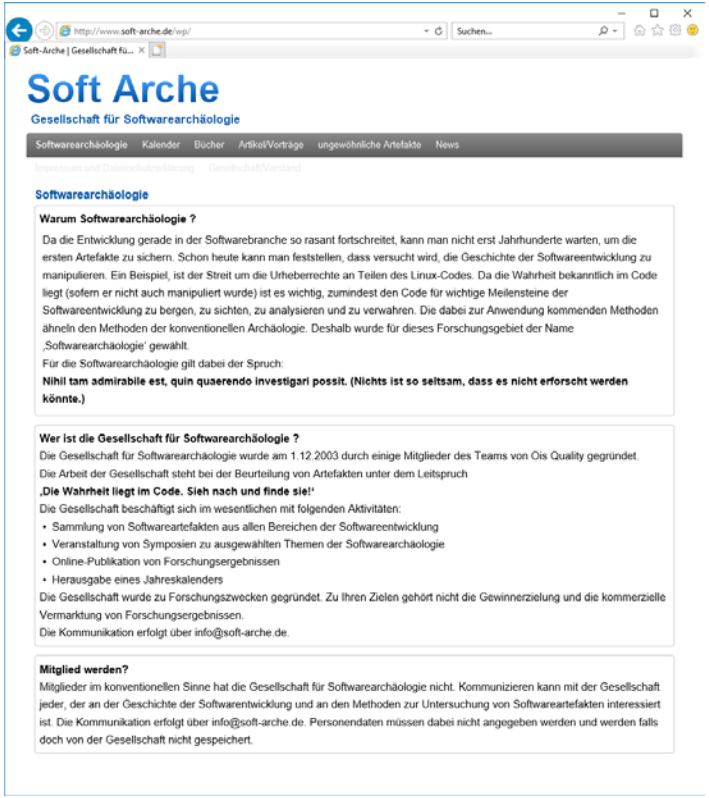
- There is a need for interaction with the changing environment (interfaces, etc.).
- User requirements change (UUX, etc.).
- The valuable functionality is only a part of the legacy system and the value only arises when used in another system
- Technology (programming language, frameworks, etc.) no longer fits (not widely used, not supported, unreliable, insecure, etc.).

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

Goal of Software Archaeology – for the purpose of “historiography”

- Conserve software for posterity and, e.g., clarify questions like, who really contributed what to Linux?

„Gesellschaft für Softwarearchäologie“, Wolfgang Ulbrich, http://www.soft-arche.de



Goal of Software Archaeology – for the purpose of productive use

- Extract and reuse knowledge which has flown into the development at that time (technical, conceptual, etc.).

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

or

- Re-use the system or parts of the system itself.

Five **re** of reuse

- Redo: The component is completely redeveloped without taking over more than the ideas and requirements of the old system.
- Reengineer: Semantic information is filtered out of the existing components and the application is newly developed.

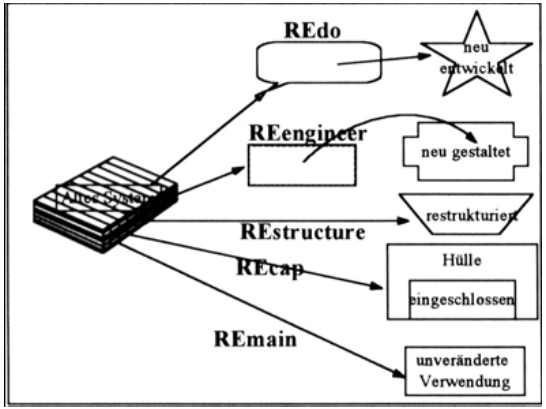


Image also G. Chroust (1996)

- Restructure: The module is mainly restructured automatically and thus adapted to the new requirements.
- Recap: Reuse by enclosing the module with a new envelope. The envelope takes over the adaptation.
- Remain: Reuse without modification.

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

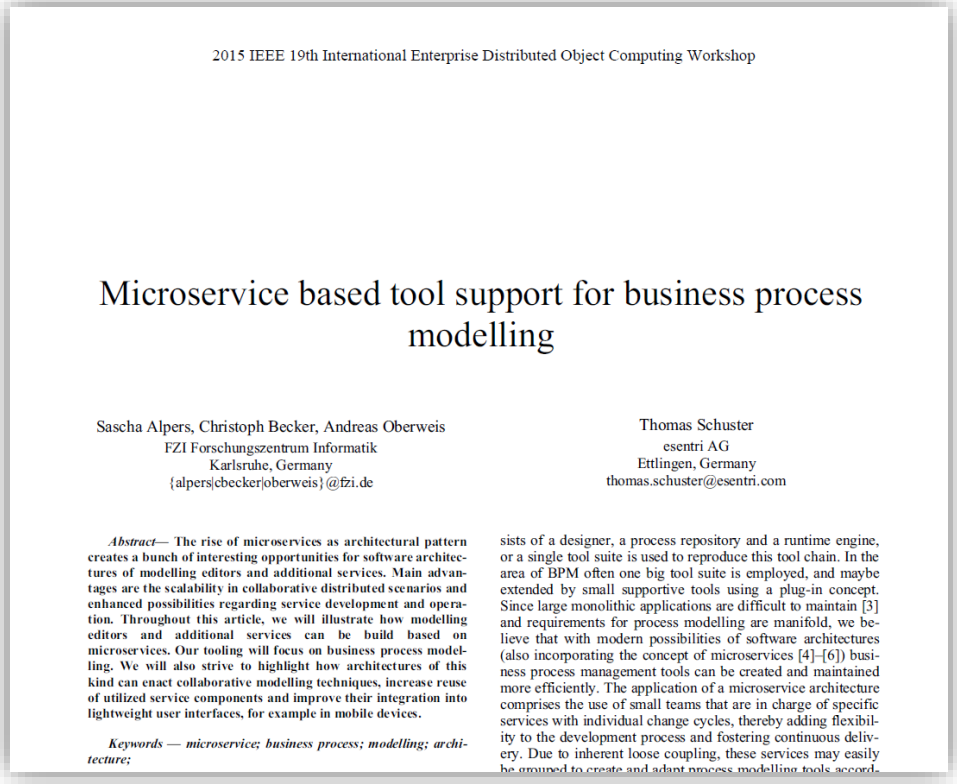
FZI

FZI FORSCHUNGSZENTRUM INFORMATIK

MICROSERVICES

Do they offer a chance for reuse old software (in sense of “recap”)?

One paper and one presentation



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-

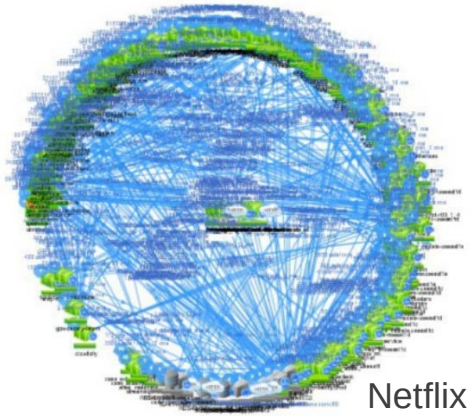
tomorrow here

- Thomas Schuster: “From Micro to Macro service or the other way round - architecture in change?“

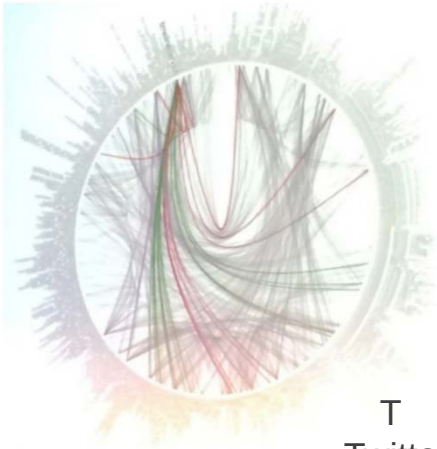
What is a microservice architecture?

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

- Samples:



Netflix



T
Twitte
r

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

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

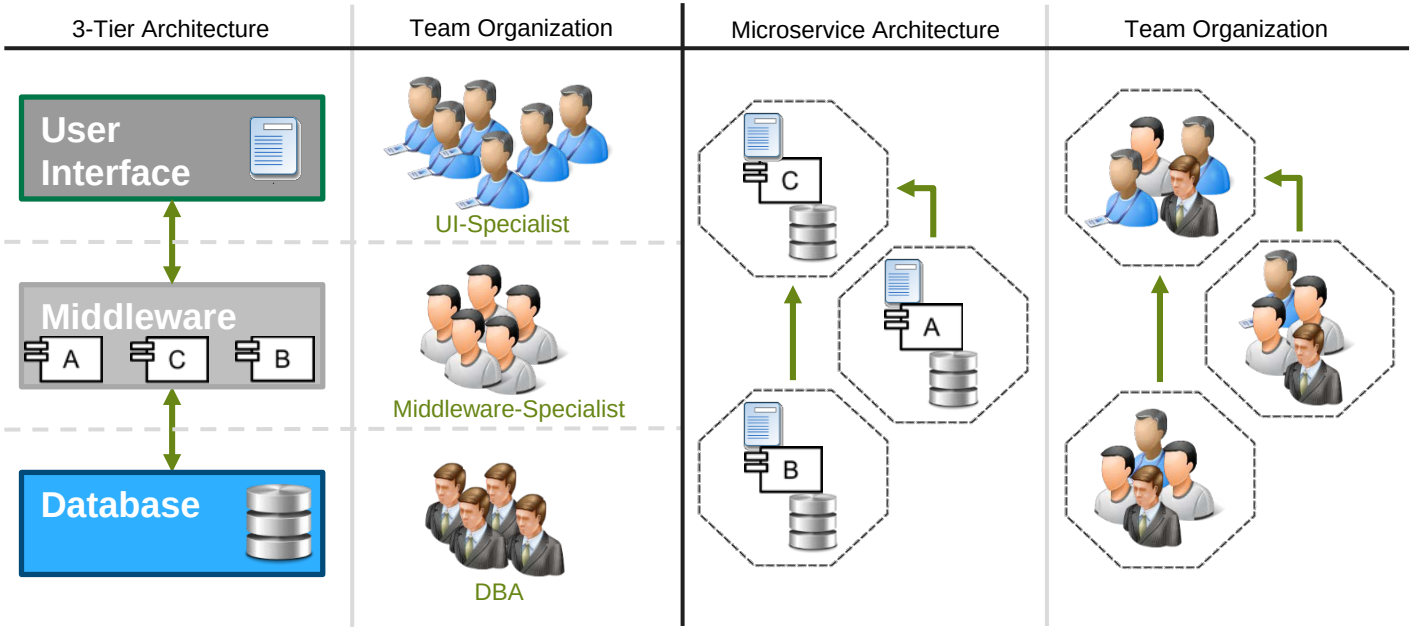
Characteristics of microservices

- 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

Difference from monolithic architecture

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



EXAMPLE PASIPP

Many thanks to ...

- … Fabian Stolz, for the prototype implementation in his master's thesis (supervised by Sascha Alpers, Prof. Dr. Andreas Oberweis, Prof. Dr. Ralf Reussner)
- … Meike Ullrich, for the prototype of the service PetriAnalyzer to convert Horus PNML to ISO/IEC 15909 conform PNML
- … Prof. Dr. Andreas Oberweis, for finding the PASIPP disks and finding a way to read them.

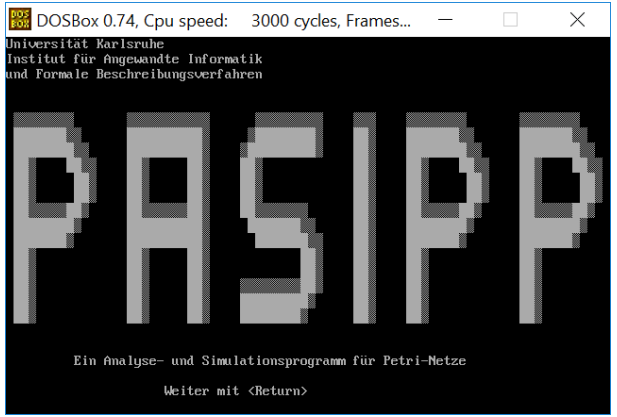
PASIPP

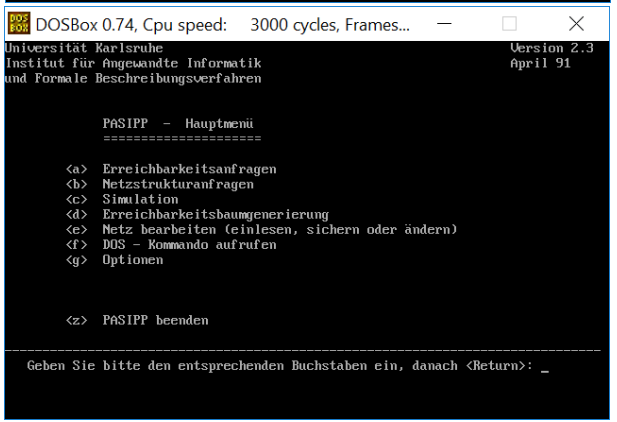
- Run compiled PASIPP in a virtual DOS environment (e.g., DosBox 0.74 https://www.dosbox.com).

- Get the Prolog system: Arity/Prolog32 was available on the website of Peter Gabel (one of the original developers of Arity/Prolog32). It is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0.

PASIPP

- Scientific Publication, e.g.:
  - Andreas Oberweis, Jürgen Seib and Georg Lausen (1991): „PASIPP: Ein Hilfsmittel zur Analyse und Simulation von Prädikate/Transitionen-Netzen“
- A tool from 1991 (Version 2.3)
- Implemented by Andreas Oberweis, Volker Sänger and others
- Implemented in Prolog





PASIPP: Implementation

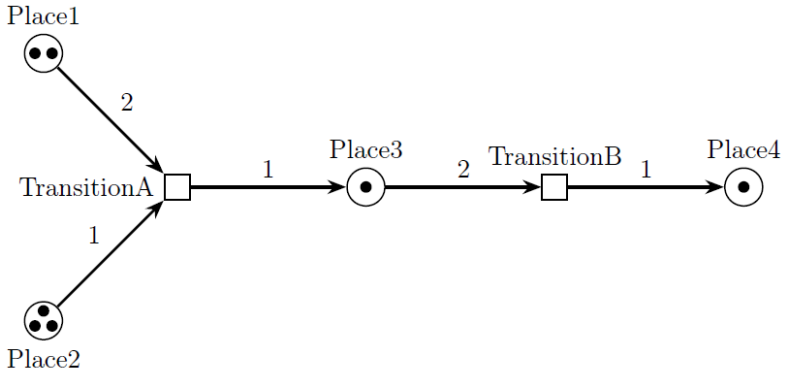
Source code excerpt

```
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])),
```

compare 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: One Petri Net as example



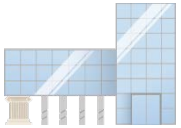
Net structure

```
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').
```

Initial marking

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

Many thanks to Fabian Stolz for the Prolog snippets and the graph.



Wrap the Prolog component

- PASIPP should be available as a microservice with a rest interface.
- Original PASIPP will still be executed as single routines (without GUI) in Prolog.
- Prolog is controlled and evaluated by a Java program as an external process.
- The microservice also converts PNML files to Prolog Petri Net descriptions and responses to expected formats.

Changes in PASIPP

- Small changes in PASIPP sources are made so that PASIPP runs under Arity/Prolog32 instead of Arity/Prolog :

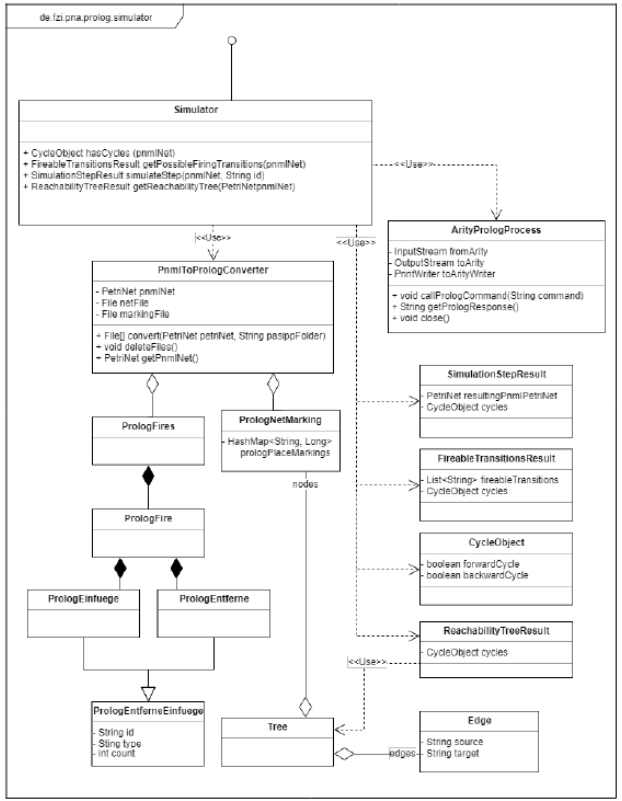
```
-read_string(String) :- read_string(40,List),atom_string(String,List).
+read_string(String) :- read_line(0,String).
```

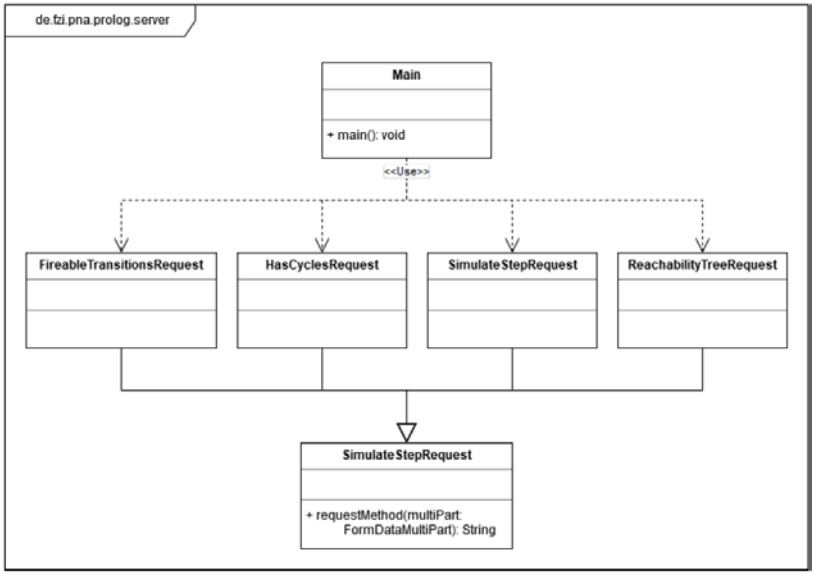
- And that the results are displayed completely and not page-wise:

```
weiter :-
  nl,write('                    Weiter mit <Return> '),nl,
  read_string(_),nl,nl.

[...]

ausgabe_one([],_) :- nl,write('Ende der Ausgabe!'),nl,nl.

ausgabe_one([X|Tail],10) :-
-weiter,
 writeq(X),nl,
 ausgabe_one(Tail,1),!.

 ausgabe_one([X|Tail],Zeilennr) :-
   Neue_Nr is Zeilennr + 1,
   writeq(X), nl,
   ausgabe_one(Tail,Neue_Nr).
```

Many thanks to Fabian Stolz for the Prolog snippets.

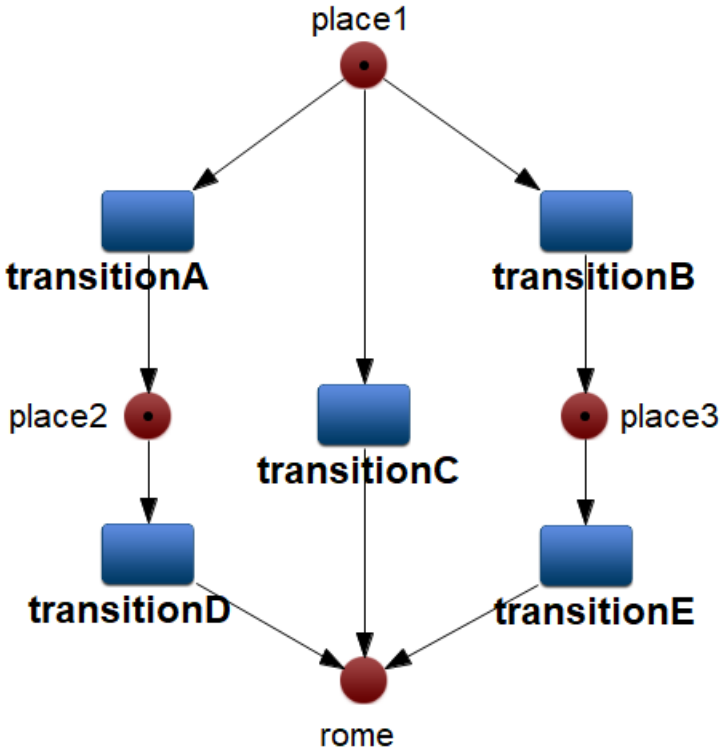
Main components of the microservice



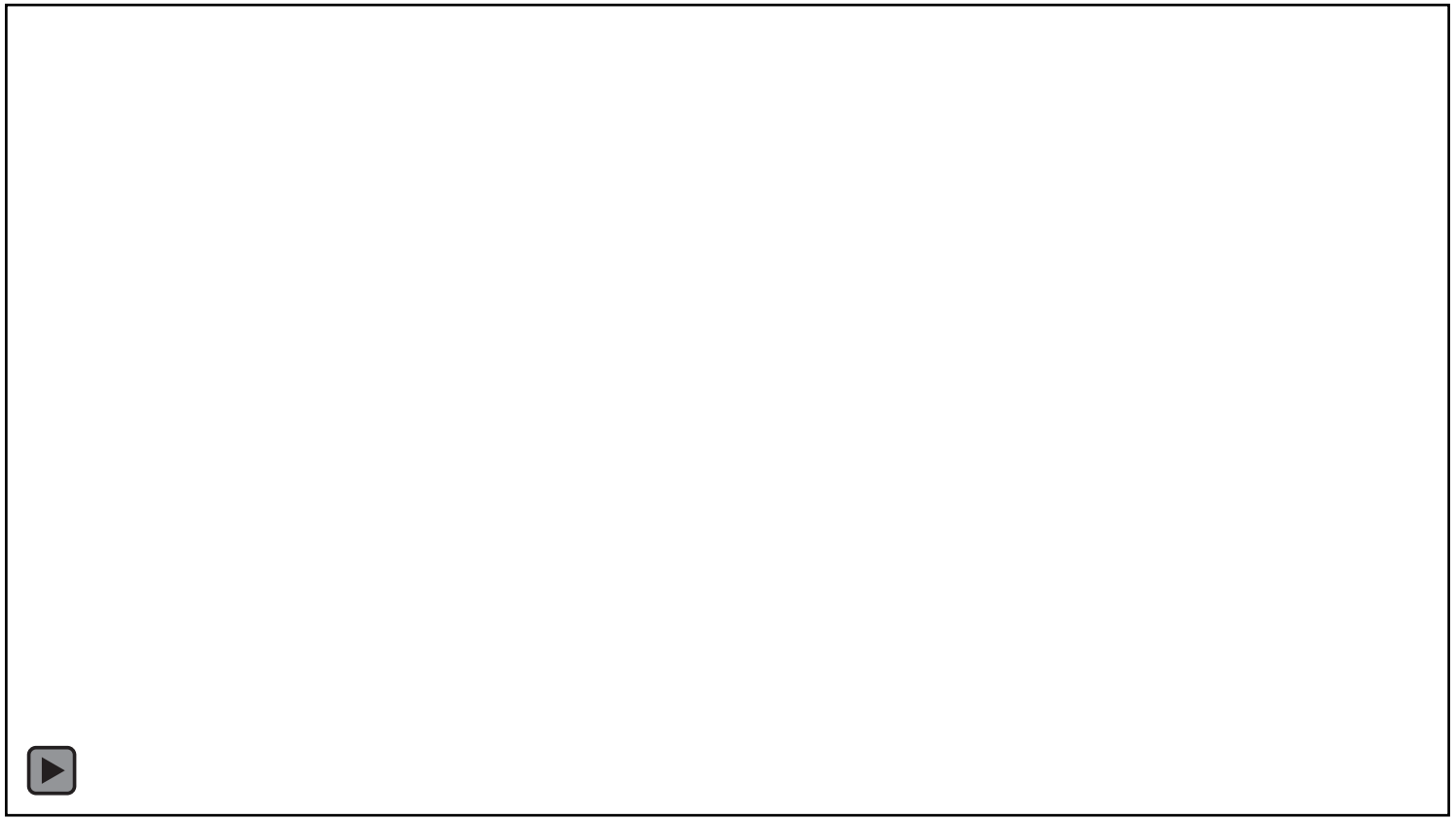


Many thanks to Fabian Stolz for the UML diagrams.

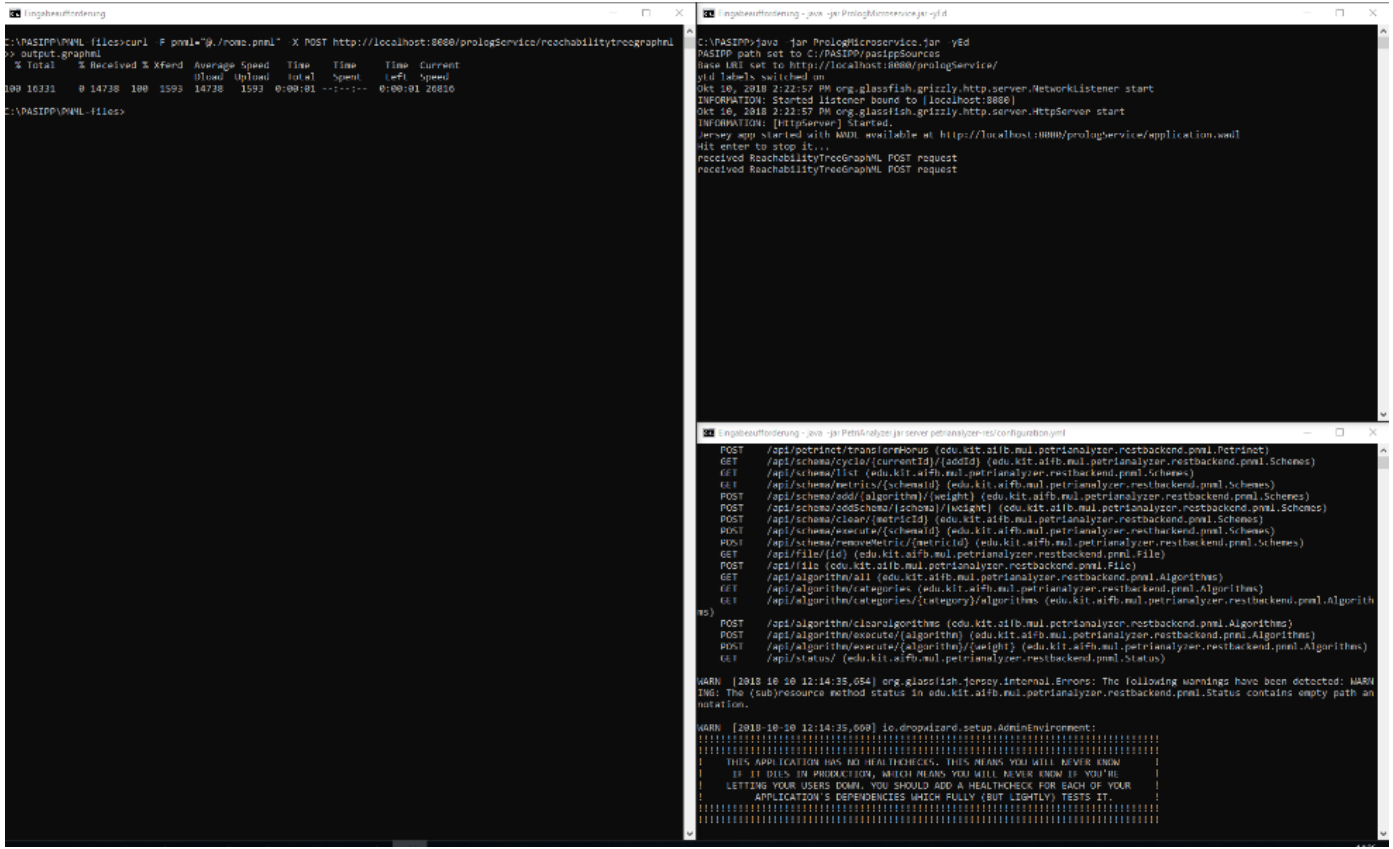
Demo Petri Net „All roads lead to Rome”



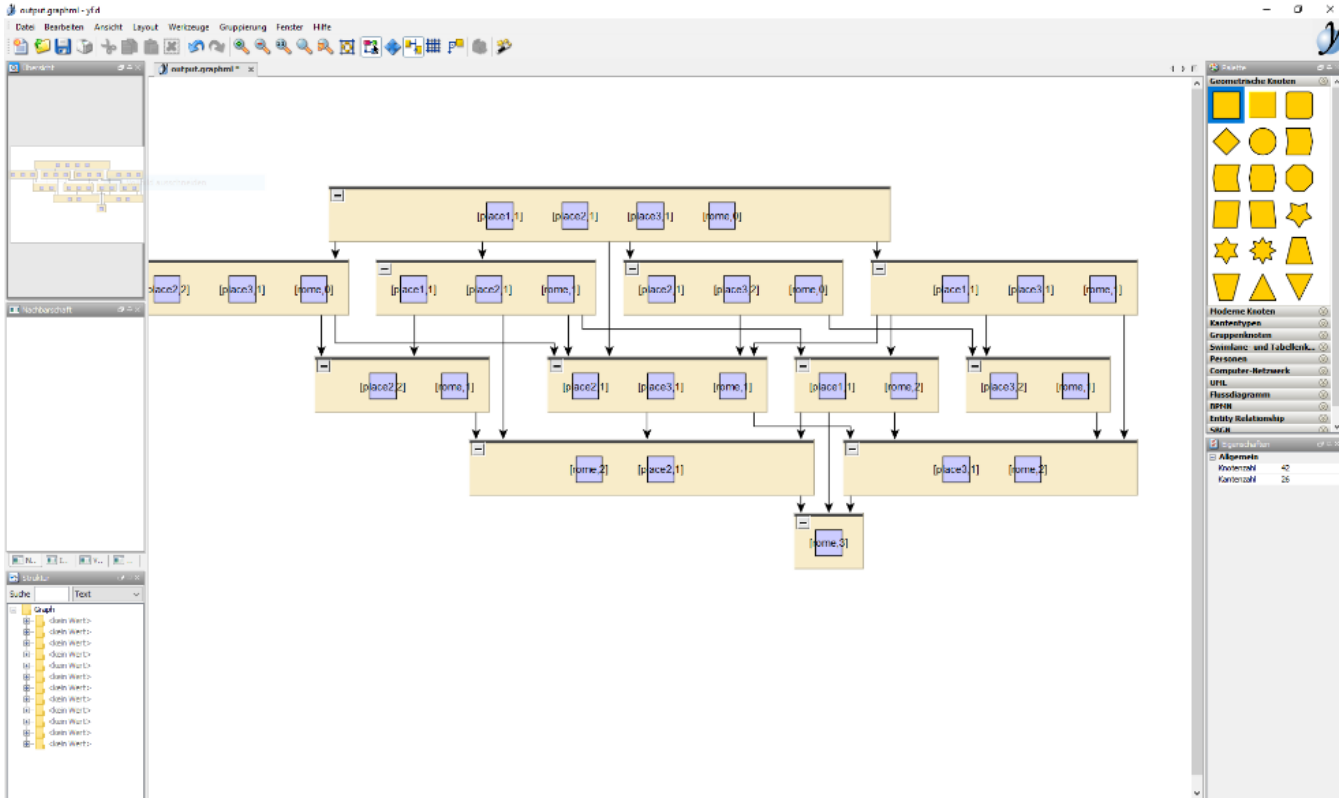
Demo video



Demo screenshots



Demo screenshots



CONCLUSION

Conclusion and Future Work

- Microservices offer a chance to use artefacts from software nearly as they are in actual software systems.

- Example PASIPP is implemented as a prototype.
  - More functionality and further testing in future works.
  - PASIPP as a microservice is a chance for new Hors functionalities
    - PNML must be ISO conform or the service of Meike Ullrich must be used (and further developed)
    - The result format must be specified, and UI must visualize the result for the user.

BACKUP SLIDES: STEP-BY-STEP GUIDE TO SETTING UP THE EXECUTION ENVIRONMENT (IN GERMAN)

Einrichten

1. Verzeichnis anlegen: C:\PASIPP
   - PASIPPsources in das Verzeichnis C:\PASIPP\PASIPPsources kopieren
   - prologMicroservice.jar in das Verzeichnis kopieren
     (Java muss installiert und Systemumgebungsvariablen müssen entsprechend gesetzt sein;
     getestet unter Java 8 und Java 11)
   - PetriAnalyzer.jar (Dienst von Meike Ullrich (AIFB) zur Umwandlung Horus PNML → PNML) und Verzeichnis petrianalyzer-res nach C:\PASIPP kopieren
2. Arity Prolog einrichten
   - Aritiy Prolog in Verzeichnis C:\PASIPP\Arity\ kopieren
   - Systemumgebungsvariablen in Windows setzen/ergänzen:
     Path: C:\PASIPP\Arity\bin;
     lib: C:\PASIPP\Arity\lib
     include: C:\PASIPP\Arity\include
3. Curl (zu Testzwecken) einrichten:
   - Curl in das Verzeichnis C:\PASIPP\curl kopieren
   - Systemumgebungsvariablen in Windows setzen/ergänzen:
     Path: C:\PASIPP\curl\bin

Demo

1. Zur Veranschaulichung Horus (falls noch nicht geschehen) und yEd (https://www.yworks.com/downloads#yEd) installieren
2. Für die Demo, den Ordner PNML-Files in C:\PASIPP kopieren

3. (eigenes cmd / Terminalfenster) PASIPP Microservice starten: `java -jar PrologMicroservice.jar -yEd`
   wird auf localhost:8080 ausgeführt
   Flag yEd erzeugt Labelinformationen ind GraphML für yEd
4. (eigenes cmd / Terminalfenster) PetriAnalyzer starten: `java –jar PetriAnalyzer.jar server petrianalyzer-res/configuration.yml`
   wird auf localhost:8090 ausgeführt

5. Erreichbarkeitsgraph für PNML (in Ordner C:\PASIPP\PNML-files\ ausführen): curl -F pnml="@./rome.pnml" -X POST http://localhost:8080/prologService/reachabilitytreegraphml >> output.graphml

6. Erreichbarkeitsgraph für PNML (im entsprechenden Workspace von Horus ausführen): curl -F pnml="@./horus-example.pnml" -X POST http://localhost:8080/prologService/reachabilitytreegraphml >> outputHorus.graphml