Future-Proof Software-Systems (FPSS)

Part 3C: Special Topics (1)

Lecture WS 2017/18: Prof. Dr. Frank J. Furrer
Future-Proof Software-Systems [Part 3C]

Our journey:

«Software everywhere» ➤ Systems & Software Engineering ➤ Three devils of Systems Engineering

Managed Evolution Strategy ➤ Architecting for Dependability ➤ Special Topics

Architecture ➤ Architecting for Changeability ➤ The Future-Proof Software-Systems Engineer

Technical Debt ➤ Architecture Erosion ➤ Future-Proof Software-Systems

© Prof. Dr. Frank J. Furrer: FPSS - WS 17/18
Special Topics = Specific IT-Architecture Areas related to IT-Systems

- Cyber-Physical Systems (CPS)
- Systems-of-Systems (SoS)
- Cyber-Physical Systems-of-Systems (CPSoS)
- Cloud Computing
- Microservices
- Agile Manifesto and Future-Proof Software-Systems?
- Domain Software Engineering
- Legacy System Migration/Modernization
- Software Product Lines
Cyber-Physical Systems (CPS)
A **cyber-physical system** (CPS) consists of a computing device interacting with the physical world in a feedback loop

Rajeev Alur, 2015

**Sensors:** Read plant information

**Actuators:** Control plant
CPS-Example: ESC

Feedback Loop

ESCAPE

ECU (SW)

Knowledge Base

WS 17/18
Future-Proof Software-Systems [Part 3C]

Control Computer (ECU)

Real-World Model

Actuators

Software

Sensors

Cyber-Part

Physical-Part

© Prof. Dr. Frank J. Furrer: FPSS - WS 17/18
Cyber-physical systems control more and more of our physical devices (Cars, planes, trains, …)

⇒ **Dependability** becomes the key issue

- Dependable **architecture** (resilience)
- Dependable **software** (trustworthiness)
- Dependable **implementation** (correctness)
Risk: Cyber-Physical Attacks

Cyber-part Interaction

Physical part

Sensors: Read plant information
Actuators: Control plant

Risk: Cyber-Physical Attacks
Trustworthy Cyber Physical Systems

Trustworthiness of a system is the expectation that the system will do what it should do, and not do what it should not do, both in expected and in unexpected conditions.

Dependability Architecture Principles (Part 4)
Cyber-Physical Systems (CPS)

1. *Dependability properties* (safety, security, …) always superse functionality requirements – in specifications, architecture, design, implementation, and operation

2. Implement *monitoring capabilities* to predict or detect abnormal or dangerous behaviour
Rajeev Alur: 
**Principles of Cyber-Physical Systems**

Raj Rajkumar, Dionisio De Niz, Mark Klein: 
**Cyber-Physical Systems**
Addison Wesley Inc., USA, 2016 
USA National Academies of Sciences, Engineering, and Medicine:  
**21st Century Cyber-Physical Systems Education**  
Downloadable from:  
[https://www.nap.edu/download/23686](https://www.nap.edu/download/23686)
Systems-of-Systems (SoS)
A **system of systems (SoS)** brings together a set of cooperating systems for a task that none of the systems can accomplish on its own (= emergent property).

Each constituent system keeps its own management, goals, and resources while coordinating within the SoS and adapting to meet SoS goals.

ISO/IEC/IEEE 15288 Annex G
Example 1 for emerging properties: „flying“

Constituent systems (CS) of an aircraft:
- engines
- body
- wings
- cockpit
- etc.

... none of the constituent systems is able to fly!

Assemble the essential constituent systems:

Emerging property: the assembly (= airplane) is able to fly!
Example 2 for emerging properties: „AlphaGo Zero“

AlphaGo's team published an article in the journal Nature on 19 October 2017, introducing AlphaGo Zero, a version created without using data from human games, and stronger than any previous version.

AlphaGo Zero is so powerful because it is "no longer constrained by the limits of human knowledge“ [Demis Hassabis, 2017]
Example 3 for emerging properties: „Landing Crash“

**Cause:** „Interface“ between the runway and the airplane
- Landing when braking action is less than good
- Crew mistakes

**Constituent systems:**
- Airplane (DC-8)
- Airport (Runway)

October 8, 1979: Swiss Air Flight 316 overran the Athens runway – 14 deaths
## SoS Emergent Behaviour Classification

<table>
<thead>
<tr>
<th>Emergence</th>
<th>Desirable/positive</th>
<th>Undesirable/negative</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Expected</strong></td>
<td>Reason for building the SoS (<strong>SoS objective</strong>)</td>
<td><strong>Mitigate</strong> by appropriate design measures, such as threat/risk analysis and countermeasures</td>
</tr>
<tr>
<td><strong>emergent behavior</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Unexpected</strong></td>
<td>Sometimes (however, quite rarely) an SoS shows unexpected, <strong>beneficial behaviour</strong></td>
<td><strong>Unexpected &amp; undesirable negative emergent behavior is one of the critical risks of most SoS</strong></td>
</tr>
<tr>
<td><strong>emergent behavior</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What is a system-of-systems?

Monolithic systems ↔ Systems-of-systems:
1. Operational Independence of the Elements
2. Managerial Independence of the Elements
3. Evolutionary Development
4. Emergent Behavior
5. Geographic Distribution

[5 Maier criteria, 1998]
Systems-of-systems characteristics:
1. Operational Independence of the Elements
2. Managerial Independence of the Elements
3. Evolutionary Development
4. Emergent Behaviour
5. Geographic Distribution

"The whole is greater than the sum of its parts."
-Aristotle
Managerial Independence of the Elements

**SoS Example:** AMAZON Business Model

- Governance Boundary 1
- Governance Authority 1
- Governance Boundary 2
- Governance Authority 2
- Governance Boundary 3
- Governance Authority 3

[Diagram showing governance boundaries and authorities in the context of an SoS example related to Amazon's business model.]
**SoS (Systems-of-systems) Terminology**

- **Constituent Systems (CS)**
- **Dependency**
- **System-of-Systems (SoS)**
- **Stakeholders**
## SoS (Systems-of-systems) Classification

<table>
<thead>
<tr>
<th>Type of SoS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directed</td>
<td>Directed SoS are those in which the integrated system-of-systems <em>is built and managed to fulfill specific purposes</em>. It is <em>centrally managed</em> during long-term operation to continue to fulfill those purposes as well as any new ones the system owners might wish to address. The constituent systems maintain an ability to operate independently, but their normal operational mode is <em>subordinated to the central managed SoS purpose</em>.</td>
</tr>
<tr>
<td>Acknowledged</td>
<td>Acknowledged SoS have recognized objectives, a designated manager, and resources for the SoS. However, the <em>constituent systems retain their independent ownership</em>, objectives, funding, and development and sustainment approaches. Changes in the systems are <em>based on collaboration</em> between the SoS and the systems.</td>
</tr>
<tr>
<td>Collaborative</td>
<td>In collaborative SoS the <em>constituent systems interact more or less voluntarily</em> to fulfill agreed upon central purposes. The central players <em>collectively decide</em> how to provide or deny service, thereby providing some means of enforcing and maintaining standards.</td>
</tr>
<tr>
<td>Virtual</td>
<td>Virtual SoS <em>lack a central management authority</em> and a centrally agreed upon purpose for the system-of-systems. Large-scale behavior emerges – and may be desirable – but this type of SoS must rely upon <em>relatively invisible mechanisms</em> to maintain it.</td>
</tr>
</tbody>
</table>
Managerial Independence: Governance

Governance Boundary 1

Governance Boundary 2

Governance Boundary 3

Governance Authority 1

Governance Authority 2

Governance Authority 3
Managerial Independence: **Governance by Contract**

- **Governance Authority 1**
- **Governance Authority 2**
- **Governance Authority 3**

**Collaboration Contract**
- functional
- operational
- commercial
- legal
... now we understand SoS’s and their challenges

... what does it mean for our future-proof software?

1) Transparent architecture
2) Explicit dependencies
3) Complete contracts
4) Risk mitigation
5) Monitoring and early response
SoS Architect’s Responsibility:
1) Transparent architecture
2) Explicit dependencies
3) Complete contracts
4) Risk mitigation
5) Monitoring and early response

Systems-of-Systems Engineering (SoSE)

Structure & Behaviour Model

Identify, document and understand all dependencies (i.e. make them explicit, no implicit dependencies!)

Define all dependencies by contracts:
- Functional
- Operational
- Commercial

Identify, assess and mitigate all risks in the SoS in relationship with the SoS mission

Define and implement mechanisms to monitor the correct operation of the SoS and to react early to deviations
Recommendations

**Systems-of-Systems (SoS)**

1. Develop and maintain a transparent, complete, up-to-date, well documented architecture for the SoS
   2. Fully understand and (formally) specify all dependencies
   3. Fully understand and (legally) specify the governance of the SoS
   4. Define all dependencies by formal contracts
   5. Use effective risk analysis and mitigation for the early detection of operational faults, errors or unwanted emergent behaviour
   6. Implement monitoring capabilities to detect unwanted emergent behaviour

**Justification**: Due to the fragmented governance/ownership in a system-of-systems, the management, evolution and operation of a SoS are more demanding. Therefore new procedures, engineering processes and operational measures must be used.
Mo Jamshidi:  
*Systems of Systems Engineering – Principles and Applications*  

Thrishantha Nanayakkara, Ferat Sahin, Mo Jamshidi:  
*Intelligent Control Systems with an Introduction to Systems of Systems Engineering*  
Cyber-Physical Systems-of-Systems (CPSoS)
A **cyber-physical system of systems (CPSoS)** brings together a set of **cyber-physical operating systems** for a task that none of the systems can accomplish on its own (= emergent property).

Each constituent cyber-physical system keeps its own management, goals, and resources while coordinating within the CPSoS and adapting to meet CPSoS goals.

Adapted from ISO/IEC/IEEE 15288 Annex G
**Example**: A modern car as CPSoS

A modern high-end car contains more than 100 networked ECUs (Electronic Control Units)
**CPSoS Example**: Roborace [Unmanned Automobile Racing]

- 24 mechanically identical cars / 12 teams / F1-race circuits
- **Winner**: Cognitive and autonomic CPSoS-SW (24-Teraflops-Computers on-board)
- **Fully electric cars,** $V_{\text{max}} = 300 \text{ km/h}$
Top Level Real-time requirement (e.g. Autonomous Driving)

Real-time requirement
CPSoS Example: Autonomous earthquake search & rescue robots
Recommendations

Cyber-Physical Systems-of-Systems (CPSoS)

1. Dependability properties (safety, security, ...) always superse functionality requirements – in specifications, architecture, design, implementation, and operation

2. Implement monitoring capabilities – especially on the interfaces – to predict or detect abnormal or dangerous behaviour
**Mo Jamshidi (Editor):**
*Systems of Systems Engineering – Innovations for the 21st Century*

**Andrea Bondavalli, Sara Bouchenak, Hermann Kopetz (Editors):**
Springer-Verlag, Germany, 2016. ISBN 978-3-319-47589-9
Cloud Computing
Cloud computing ...

- is a **model** for enabling ubiquitous, convenient, on-demand access:
- to a shared pool of configurable **computing resources** (e.g., networks, servers, storage, applications, and services)
- that can be rapidly provisioned and released
- with minimal management effort or service provider interaction

US National Institute of Standards and Technology
Future-Proof Software-Systems [Part 3C]

**Iron Age**

**Cloud Age**

**Application Landscape**

http://www.freiburg.de

https://www.snelserver.com

http://blog.schneider-electric.com

© Prof. Dr. Frank J. Furrer: FPSS - WS 17/18
Cloud-based computing

Enterprise Cloud Computing

Embedded Cloud Computing
Enterprise Cloud Computing

**Enterprise Cloud Computing** is a business model with promising commercial and technical advantages. It relies on using 3rd party IT-services delivered via the Internet instead of in-house IT-installations.

⇒ Enterprise IT **deployment-architecture** for the next decades
- On-demand use of IT-capabilities
- Massive reduction of IT capital investment
- Significant reduction of (low level) IT staff
- Access to modern technology and services
Advantages of Cloud Computing
Risks of Cloud Computing

Security

Loss of Control

Loss/Theft of Company Data

Disaster Recovery

© Prof. Dr. Frank J. Furrer: FPSS - WS 17/18
Risks of Cloud Computing

Own protection mechanisms

Cloud operator protection mechanisms

http://www.itbriefcase.net

© Prof. Dr. Frank J. Furrer: FPSS - WS 17/18
Distributed systems communicate and synchronize in real-time via the cloud.

Distributed systems use the power of cloud resources.
Cloud Robotics is the application of the cloud computing concept to robots. This means using the Internet to augment the robots capabilities by off-loading computation and providing services on demand.

⇒ Interesting and active research area:

- Cloud Robotics
- Collaborative Manufacturing
- Intelligent Traffic Management
- etc.
The cloud will enable many useful and interesting applications (e.g. optimized traffic management)
Cloud Definitions:

- **Software as a Service**
  [http://www.salesforce.com/eu](http://www.salesforce.com/eu)

- **Platform as a Service**

- **Infrastructure as a Service**
  [http://www.rackspace.com](http://www.rackspace.com)
Cloud Definitions:
Cloud **Benefits:**

**Benefits for the Cloud Service Consumers:**
- Minimal investment
- Low running/operational cost
- Technology-independent
- No vendor lock-in
- Short time-to-market ("assemble" instead of "build")
- Choose "best of breed"
- Simple up-/downscaling

**Benefits for the Cloud Service Providers:**
- Large market
- No copying, no IPR disclosure
- Economy of scale
- Interesting new business models
- Specialization

**Cloud** = Execution and Delivery Structure

**Cloud Usage** = Commercial decision (Business model)
Cloud **Risks:**

- Company & Personal Data
  - Unauthorized access, hacking
  - Privacy
  - Data Loss
  - Availability

- SaaS
- PaaS
- IaaS

- Large Scale Disaster
- Local Disaster

- Industrial Espionage

© Prof. Dr. Frank J. Furrer: FPSS - WS 17/18
Cloud Risks:

- Security,
- Dependability,
- Resilience,
- ...

Loss of direct control

compensate by:

Cloud Service Level Agreements (Cloud-SLA)

SaaS

PaaS

IaaS

Attacks

Contract

SLA: Service Level Agreement

- functional
- operational
- commercial
- legal

http://pssiusa.wordpress.com

Security, Dependability, Resilience, …
Cloud Service Level Agreements (SLAs)

- Automatic service discovery
- SLA negotiation
- QoS-based SLA search

Cloud-SLA Template

Contract

**SLA**: Service Level Agreement
- functional
- operational
- commercial
- legal

Machine-readable SLA

Cloud-SLA Modeling & Description Language
Cloud Definitions:

- Business as a Service (BaaS)
- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)

End-User
Application
Developer
System Manager

Business Process
Cloud Computing Stack:

- **Cloud Clients** [Presentation Layer]
  - Browser, Mobile Devices
  - CMS, Bookkeeping, …

- **Cloud Applications** [Software as a Service (SaaS)]
  - Payments, Trading, Address Book, Directory, …

- **Cloud Services** [Components as a Service]
  - Technical Infrastructure

- **Cloud Platform** [Platform as a Service (PaaS)]

- **Cloud Storage** [Storage as a Service]
Cloud Responsibilities

**Provider**
- Service architecture
- Industry standards
- Effective & efficient
- Agile
- Security
- Disaster recovery
- Cost models

**Consumer**
- Business model
- Enterprise cloud architecture
- Risk analysis
- Contracts
- Privacy
- Business Continuity Planning
- Monitoring

**Responsible Cloud Infrastructure**

**Enterprise Cloud Service Architecture**
Cloud: Influence on Architecture

- **Technical Architecture**: (Technical Infrastructure)
- **Integration Architecture**: (Cooperation Mechanisms)
- **Applications Architecture**: (Functionality)
- **Services**:
- **Information (Data) Architecture**: (Information & Data)
- **Business Architecture**: (Business Processes)

Cloud: Influence on Architecture
The cloud paradigm offers very interesting commercial and technical **benefits**

When using IaaS, PaaS, SaaS or BaaS („cloudification“) a **loss** of direct control results.

Many of the quality properties (Security, availability, privacy etc.) are mainly under the control of the cloud service provider.

The loss of direct control must be compensated by clear, explicit, and legally binding **contracts**

Focus shifts from „building“ to „assembling“ – and needs new, different engineering + architecture processes.
**Architectural Recommendation for Cloud Service Providers:**

1. Implement the architecture principles as presented in this lecture
2. Deliver the cloud-services via established, accepted industry-standards
3. Provide transparency on your architecture, implementation and evolution
4. Give factual & contractual assurance for the quality properties (dependability, availability, privacy, disaster-recovery, performance etc.)

**Architectural Recommendation for Cloud Service Consumers:**

1. Compensate the loss of transparency by requiring sufficient information about cloud service provider architecture, quality properties etc.
2. Compensate the lack of control by clear, explicit, legally binding Cloud-SLAs (Cloud service level agreements)
3. Insist on established, accepted industry-standards for the delivery of all cloud-services
Thomas Erl, Ricardo Puttini, Zaigham Mahmood: *Cloud Computing – Concepts, Technology & Architecture*

Bill Wilder: *Cloud Architecture Patterns*
Microservices
Future-Proof Software-Systems [Part 3C]

- Functionality
- Operational parameters
- Commercial Conditions

Service Contract

System Part A

Interface

Service

System Part B

Interface
**Services** implement:
- Good partitioning
- Excellent encapsulation
- Loose coupling

**Granularity**

Coarse-grained service

Fine-grained service
**Example:** Coarse-grained service (1/4)

Interbank Electronic Money-Transfer:
Example: Coarse-grained service (2/4)
Example: Coarse-grained service (3/4)

All databases are correct and consistent
**Example:** Coarse-grained service (4/4)

- **Service: Request**
  - Authorization
  - Check balance
  - Transfer
  - Process feedback
  - Rollback

- **Service: Response**
  - Authorization
  - Check account
  - Transfer denied
  - Generate feedback
  - Rollback

**System Consistency**
Example: Fine-grained service (1/x)
Granularity

Coarse-grained service

Fine-grained service

Service Usage

Functionality
- Computation
- Data access
- Data transfer
- Control transfer

Non-functional properties
- Error handling
- Fault processing
- Security
- Safety, ...
- Parallelism
- Scalability
- ...

System-Properties
- Synchronization
- Consistency
- Monitoring
- Auditability
- Accountability
- Versioning
- Recovery
- ...

© Prof. Dr. Frank J. Furrer: FPSS - WS 17/18
Future-Proof Software-Systems [Part 3C]

**Functionality**
- Computation
- Data access
- Data transfer
- Control transfer

**Non-functional properties**
- Error handling
- Fault processing
- Security
- Safety, ...
- Parallelism
- Scalability
- ...

**System-Properties**
- Synchronization
- Consistency
- Monitoring
- Auditability
- Accountability
- Versioning
- Recovery
- ...

**Service**

- 2’000 SLOCs
- 10’000 SLOCs
- 20’000 SLOCs

© Prof. Dr. Frank J. Furrer: FPSS - WS 17/18
Microservice Principles

Microservices are *small, autonomous, independently deployable* services that work together using an *integration infrastructure*.

Orchestration ↔ centrally controlled
Choreography ↔ triggered
Future-Proof Software-Systems [Part 3C]

Specification -> Development Evolution -> Operation (24/7)

Teams

Management

http://publishinginsider.net
Microservice Principles

- A microservice is «small» (various, fuzzy definitions)
- A microservice does one thing – and does it well
- A microservice works with business objects
- A microservice respects all architecture principles (especially partitioning, encapsulation, loose coupling and no reduncancy)
- A microservice is consistently embedded into the enterprise architecture ⇐ Importance of overall architecture!
  (Structure, technology, security, safety, …)
- A microservice is independently evolvable and deployable
- A microservice can be monitored as part of the whole
- A microservice has a defined owner, embedded in the company governance structure
CAUTION: Adopting microservices requires an adequate company organization

- **Maximize team autonomy**: Create an environment where teams can get more done without having to coordinate with other teams.
- **Optimize for development speed**: Hardware is cheap, people are not. Empower teams to build powerful services easily and quickly.
- **Focus on automation**: People make mistakes. More systems to operate also means more things that can go wrong. Automate everything.
- **Provide flexibility without compromising consistency**: Give teams the freedom to do what's right for their services, but have a set of standardized building blocks to keep things sane in the long run.
- **Built for resilience**: Systems can fail for a number of reasons. A distributed system introduces a whole set of new failure scenarios. Ensure measures are in place to minimize impact.
- **Simplified maintenance**: Instead of one codebase, you'll have many. Have guidelines and tools in place to ensure consistency.
**Recommendations**

**Architecture Recommendations for Microservices**

1. Use microservices only if they fit into the *overall architecture* and strategy (enterprise strategy)

2. Respect all *architecture principles* while building and evolving microservices

3. Establish a working governance structure for microservices

4. Base microservices on the *domain model* (domain driven engineering)

5. *Automate* the management/deployment of microservices and the integration infrastructure
Sam Newman: 
**Building Microservices**

Mike Amundsen, Matt Mclarty:
**Microservice Architecture – Aligning Principles, Practices, and Culture**
Software Product Lines

1. Product lines make use of planned, massive reuse

2. The product line approach promises significant advantages in development cost, time-to-market and quality of the products = strong amplifier for agility

3. Product line engineering requires specific organizational structures and a new software development process

4. The product line approach is a mature, proven technology which leads to considerable competitive advantages for companies
Paul Clements, Linda Northrop: 
**Software Product Lines – Practices and Patterns**

Klaus Pohl, Gunter Bockle, Frank J. Linden: 
**Software Product Line Engineering – Foundations, Principles and Techniques**
Part 3 C

The End