01: Introduction

Distributed Software Architectures

Stefan Huber <shuber.lba@fh-salzburg.ac.at>

March 8, 2019

Section 1

Course formalities

Distributed Software Architectures

This course is about Software Architectures of Distributed Systems.

Topics:

- Ways to organize distributed software systems
- Foundations of distributed systems
  - Communication
  - Coordination
  - Virtualization & operating systems
  - Name systems
  - Replication and consistency
  - Security for distributed systems

Related fields and courses:

- Software engineering
- Computer networks
- Operating systems

Course organization

The course is split into lecture (VL) and tutorial (UE) units.

Grading:

- 60% lecture: Two 45 min. tests
- 40% tutorial: Exercises, presentations, coding assignments

FR 08.03.	08:15–09:45 VL		
FR 22.03.	08:15–09:45 VL	10:00–11:30 UE	
TH 04.04.	08:15–09:45 VL	10:00–11:30 UE	
FR 26.04.	08:15–09:45 VL	10:00–11:30 UE	
TH 09.05.	08:15–09:45 VL	10:00–11:30 UE	Test
FR 07.06.	08:15–09:45 VL	10:00–11:30 UE	
FR 14.06.	08:15–09:45 VL		
MO 24.06.	08:15–09:45 UE		Test

Literature:

- van Steen, Tanenbaum (2017): Distributed Systems
  Free digital copy: `https://www.distributed-systems.net`

About me

Short bio:

- B&R Industrial Automation:
  - Headed an R&D team of 5 PhDs in mathematics, computer science and control theory.
  - Invented the software system operating *ACOPOStrak*.
  - Recently switched to a new project *Digital Automation*.
- PostDoc researcher at IST Austria in the field of *computational topology*.
- Senior Scientist (math), PostDoc (CS), and PhD student (CS) at the Univ. of Salzburg in the field of *computational geometry*.
- Doctorate in CS, Master in Math, Master in CS, Bachelor in Math and Bachelor in CS in 2011, 2009, 2008, 2007 and 2006.

Section 2

Introduction to Distributed Systems

What is a Distributed System?

A computing system that is distributed on a computer network.

- Collection of *autonomous* components (nodes).
- Nodes cooperate to form a single *coherent* system.
- Nodes are distributed on and communicate via a computer network.

Characterization (van Steen)

A *distributed system* is a collection of autonomous computing elements that appears to its users as a single coherent system.

Examples for distributed systems

WWW Master-slave. The World Wide Web is the probably the largest distributed system. Web browsers and web servers as nodes that communicate via HTTP protocol over the Internet. For the user the WWW appears as a single system.

Bittorrent A peer-to-peer file sharing protocol using so-called distributed hash tables to find peers.

Examples for distributed systems

WWW Master-slave. The World Wide Web is the probably the largest distributed system. Web browsers and web servers as nodes that communicate via HTTP protocol over the Internet. For the user the WWW appears as a single system.

Bittorrent A peer-to-peer file sharing protocol using so-called distributed hash tables to find peers.

GIMPS Great Internet Mersenne Prime Search is an example for *distributed computing*.

NFS The Network File System is a distributed file system based on Remote Procedure Calls.

Examples for distributed systems

WWW Master-slave. The World Wide Web is the probably the largest distributed system. Web browsers and web servers as nodes that communicate via HTTP protocol over the Internet. For the user the WWW appears as a single system.

Bittorrent A peer-to-peer file sharing protocol using so-called distributed hash tables to find peers.

GIMPS Great Internet Mersenne Prime Search is an example for *distributed computing*.

NFS The Network File System is a distributed file system based on Remote Procedure Calls.

Car A car comprises a distributed system of several dozens ECUs, controlling the engine, brakes, doors, or HMI. A typical automotive network is CAN bus.

Automation An industrial machine comprises controllers, drives, sensors, HMI with realtime communication over a fieldbus.

Characteristics: Collection of autonomous nodes

- Nodes are in principle independent from each other, but cooperate.
- Nodes run concurrently.
- The collection may be very heterogeneous.
- Each node has its own notion of time. There may not be a global clock.
- Cooperation requires communication, e.g., message passing.
- Notion of an overlay network, e.g., the communication topology:
  - Structured overlay: Tree, ring, grid, et cetera.
  - Unstructured overlay: Randomly chosen neighborhood.
- Group management for the collection of nodes:
  - Open group: Any node can join
  - Closed group: Access is restricted

Characteristics: Single coherent system

- Wiktionary on *coherent*: Unified; sticking together; making up a whole.
- Level of coherence versus the level of distribution.
- Distribution transparency:
  - User does not need to know where something is processed, stored, et cetera.
  - Half of system engineering is abstraction! Abstraction means hiding details.
- Fault tolerance

Characterization (Lamport)

[A distributed system is] one in which the failure of a computer you didn't even know to exist can render your computer unusable.

Section 3

Python

Python: Motivation

Python is popular:

- 3rd on TIOBE index 01/2019: Java, C, Python, C++, VB.net, JS, C#

Python is very simple and easy to learn:

- Very popular at MIT courses
- Syntax is simple and expressive, reasonably close to pseudo code
- *Distributed Systems* uses Python for examples

Python is technologically rich:

- Originated in 1990
- Interpreted, platform independent
- Duck typing, very dynamic
- General-purpose, multi-paradigm: object-oriented, functional, aspect-oriented, ...
- A rich standard library and a huge universe of third-party modules
- IPython and friends make it a Matlab alternative

Python: Hello World

This is a Python 3 hello world example:

```
1 #!/usr/bin/env python3
2
3 print("Hello World!")
```

Listing 1: helloworld.py

Learning Python:

- Python tutorial:
  `https://docs.python.org/3/tutorial/`
- Python in 10 minutes:
  `https://www.stavros.io/tutorials/python/`
- Python 101:
  `http://www.davekuhlman.org/python_101.html`
  For Python 2, but mentions the differences to Python 3.

Brief tutorials, but for Python 2:

- Google for Education:
  `https://developers.google.com/edu/python/introduction`
- A very short introduction by scipy:
  `http://scipy-lectures.org/language/python_language.html`

Python: Variables and types

Python can be used in an interactive interpreter:

```
>>> a = 2
>>> b = True
>>> print(a, b)
2 True
>>> type(a), type(b), type("hello"), type('world')
(<class 'int'>, <class 'bool'>, <class 'str'>, <class 'str'>)
>>> "hello" + 'world'
'helloworld'
>>> 2 + 2, 3 * 4, 4**2, 17 / 3, 17 // 3, 3 * "no", True and False or True
(4, 7, 16, 5.666666666666667, 5, 'nonono', True)
>>> r"c:\who\uses\windows\anyways"
'c:\\who\\uses\\windows\\anyways'
>>> """A multiline
string can be
handy"""
'A multiline\nstring can be\nhandy'
>>> c = "slicing"
>>> c[0], c[-1], c[1:4], c[-3:], c[:], c[::2], c[::-1], len(c)
('s', 'g', 'lic', 'ing', 'slicing', 'siig', 'gnicils'), 7
>>> c[:2] + c[2:]
'slicing'
```

Listing 2: Variables and basic types

Python: Lists

A list is a heterogeneous sequence of elements.

```
>>> squares = [1, 4, 9, 16, 25]
>>> type(squares)
<class 'list'>
>>> squares[::-1]        # Indexing and slicing works for all sequence types
[25, 16, 9, 4, 1]
>>> squares + [36, 49, 64, 81, 100]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>> bag = ["hello", 42, [0, 1]]                    # Can be heterogeneous
>>> bag[0] = "hi!"
>>> bag.append(3.14)
>>> bag
['hi!', 42, [0, 1], 3.14]
```

Listing 3: Lists

A tuple is, like a list, a sequence type, but immutable.

```
>>> tri = 23, 42, 100                          # The same as (23, 42, 100)
>>> type(tri)
<class 'tuple'>
>>> tri[0], len(tri)          # Note that this expression is a tuple of two
(23, 3)
>>> a, b, c = tri                          # Assign tri to the tuple (a, b, c)
>>> print(a, b, c)
23 42 100
```

Listing 4: Tuples

Python: A first program

```
#!/usr/bin/env python3

a, b = 0, 1
while a < 10:
    print(a)
    a, b = b, a + b
```

Listing 5: The Fibonacci series

Some remarks:

- We use tuples for multiple assignments and variable definitions
- Indentation defines blocks. It is 4 spaces, see `https://www.python.org/dev/peps/pep-0008/`.
- The function `help()` in the interpreter gives help on anything, e.g., `print()`.

Python: More control statements

```
#!/usr/bin/env python3

x = int(input("Enter a whole number: "))

if x < 0:
    x = 0
    print("Negative changed to zero")
elif x == 0:
    print("Zero")
else:
    print("Positive")


for w in "hello!":                          # Iterating over a sequence
    print(w)
    if w == 'o':
        print("exit loop")
        break

for i in range(5):              # range(5) generates the numbers 0, 1, 2, 3, 4
    print(i)

while True:                             # Endless loop. Exit by pressing ctrl+c
    pass
```

Listing 6: More control statements

Python: Functions

```
#!/usr/bin/env python3

def fib_print(n):
    """Print the Fibonacci up to at most n."""          # The __doc__ string
    a, b = 0, 1
    while a < n:
        print(a, end=', ')
        a, b = b, a + b
    print()

def fib_print_ext(n, start=0):
    """Print the Fibonacci series up to at most n, but at least start."""
    a, b = 0, 1
    while a < n:
        if a >= start:
            print(a, end=', ')
        a, b = b, a + b
    print()

print("Our fancy module", __name__, "loaded")

if __name__ == "__main__":                          # If called from interpreter
    fib_print(30)
    fib_print_ext(30, 4)
```

Listing 7: Functions

Python: Sets and dicts

A set has unique elements and adding, removing, ownership-testing is fast.

```
>>> bucket = {'hi', 0, 1, 2, 2, 2}
>>> len(bucket)
4
>>> 1 in bucket
True
```

Listing 8: set

A dictionary is a key-value map.

```
>>> codes = {5020: "Salzburg", 6010: "Innsbruck", 4010: "Linz"}
>>> len(codes)
3
>>> codes[5020]
'Salzburg'
```

Listing 9: dict

Python: Exceptions

```
#!/usr/bin/env python3

if __name__ == "__main__":
    try:
        x = int(input("Enter whole number: "))
        print("Entered", x)
    except ValueError:
        print("No whole number given")

    def whiney():                    # A function can be defined in any scope
        raise Exception("A whiney function")

    try:
        whiney()
    except Exception as err:
        print(err)

    try:
        f = open("doesnotexist.txt", "r")
    except OSError as err:
        print("Error opening file:", err)
    except:
        print("Some unknown error")
    else:
        print("Close again")
        f.close()
```

Listing 10: Exceptions

Python: Classes

```python
#!/usr/bin/env python3

class Vector:
    def __init__(self, x, y):                                   # Constructor
        self.x, self.y = x, y

    def printit(self):      # All methods are functions with self as 1st arg
        print("({}, {})".format(self.x, self.y))          # String formatting

    def add(self, v):                           # We do not care about type(v)...
        self.x += v.x                     # ... only that it has members x and y
        self.y += v.y                                  # It is called Duck Typing

class Object:
    pass

if __name__ == "__main__":
    v = Vector(1.0, 2.0)
    v.add(Vector(5.0, 6.0))
    v.printit()

    u = Object()                                                # u has no members
    u.x, u.y = 10.0, 20.0           # Now we dynamically add members x and y
    v.add(Vector(5.0, 6.0))
    v.printit()
```

Listing 11: Classes

Python: Class inheritance

```python
#!/usr/bin/env python3

class A:
    def __init__(self, x, y):
        self.x, self.y = x, y

    def f(self):
        print("(%g, %g)" % (self.x, self.y))                # String formatting

class B(A):                                                 # Inheritance
    def __init__(self, x, y, z):
        A.__init__(self, x, y)                  # Calling base class constructor
        self.z = z

    def f(self):                                            # Polymorphism
        print("(%g, %g, %g)" % (self.x, self.y, self.z))

if __name__ == "__main__":
    b = B(1, 2, 3)
    b.f()
```

Listing 12: Inheritance

Python: Beyond Hello World

```python
#!/usr/bin/env python3

# A function
def print_square(n):
    print("{} squared is {}".format(n, n**2))
    if n % 2 == 0:
        print("  even", n)

# Will also be called when this module is imported somewhere
print("module loaded")

# The module name is "__main__" if called by the interpreter rather than
# being imported
if __name__ == "__main__":
    # Loop over the list [0, 1, 2, 3, 4]
    for i in range(5):
        print_square(i)

    li = []
    i = 0
    while i < 5:
        li.append(i*i)
        i += 1

    # List comprehension: Functional programming rocks!
    assert(li == [x*x for x in range(5)])
```

Listing 13: morehelloworld.py

```python
#!/usr/bin/env python3

# A class
class PowerPrinter:
    # A constructor
    def __init__(self, exp=2):
        self.exp = exp

    # A method
    def print(self, n):
        fmt = "{} to the power of {} is {}"
        print(fmt.format(n, self.exp, n**self.exp))

if __name__ == "__main__":
    # Loop over a tuple with two elements
    for p in PowerPrinter(), PowerPrinter(3):
        # Loop over the list of whole numbers [0..5)
        for i in range(5):
            p.print(i)
```

Listing 14: moremorehelloworld.py

Section 4

Exercise formalities

How to submit

Hard facts:

- Exercise sheets on the course wiki page.
- Deadline given on the sheet.
- Exercises submitted via git repo on `gitlab.mediacube.at`.
- Please respect the dictionary name formatting.

Soft facts:

- Use static code analyzers, like `flake8` or `pylint`, to improve your results and avoid trivial mistakes.
- A sample git repository is on `gitlab.mediacube.at` to ease the getting started phase.

On exercise sheet 01

- Work load primarily on getting familiar with Python at home.
- Therefore, only two small warm-up exercises.