

Robotica – HBO-ICT



Agenda

Workshop 1

- Waarom deze workshops?
- Waarom WeBots?
- Introductie WeBots
- Zelf aan de slag
- Analyseer business logica vs hardware interfacing

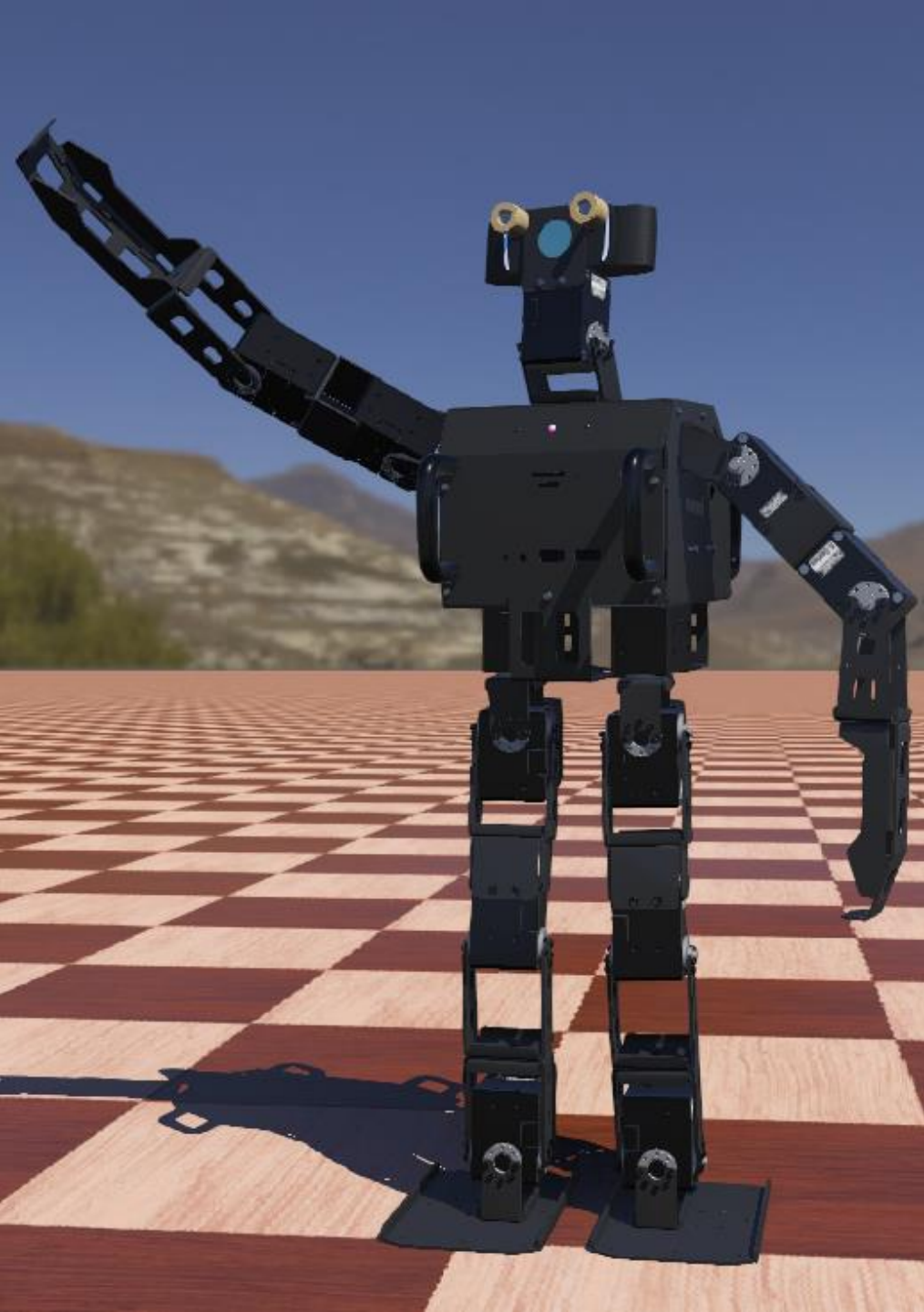
Workshop 2 – 28 april

- Façade Design Pattern
- Interne communicatie + Remote Controller
- Voorbeeld: logger
- Voorbeeld: Communicatie via pipes (C++ / Python)



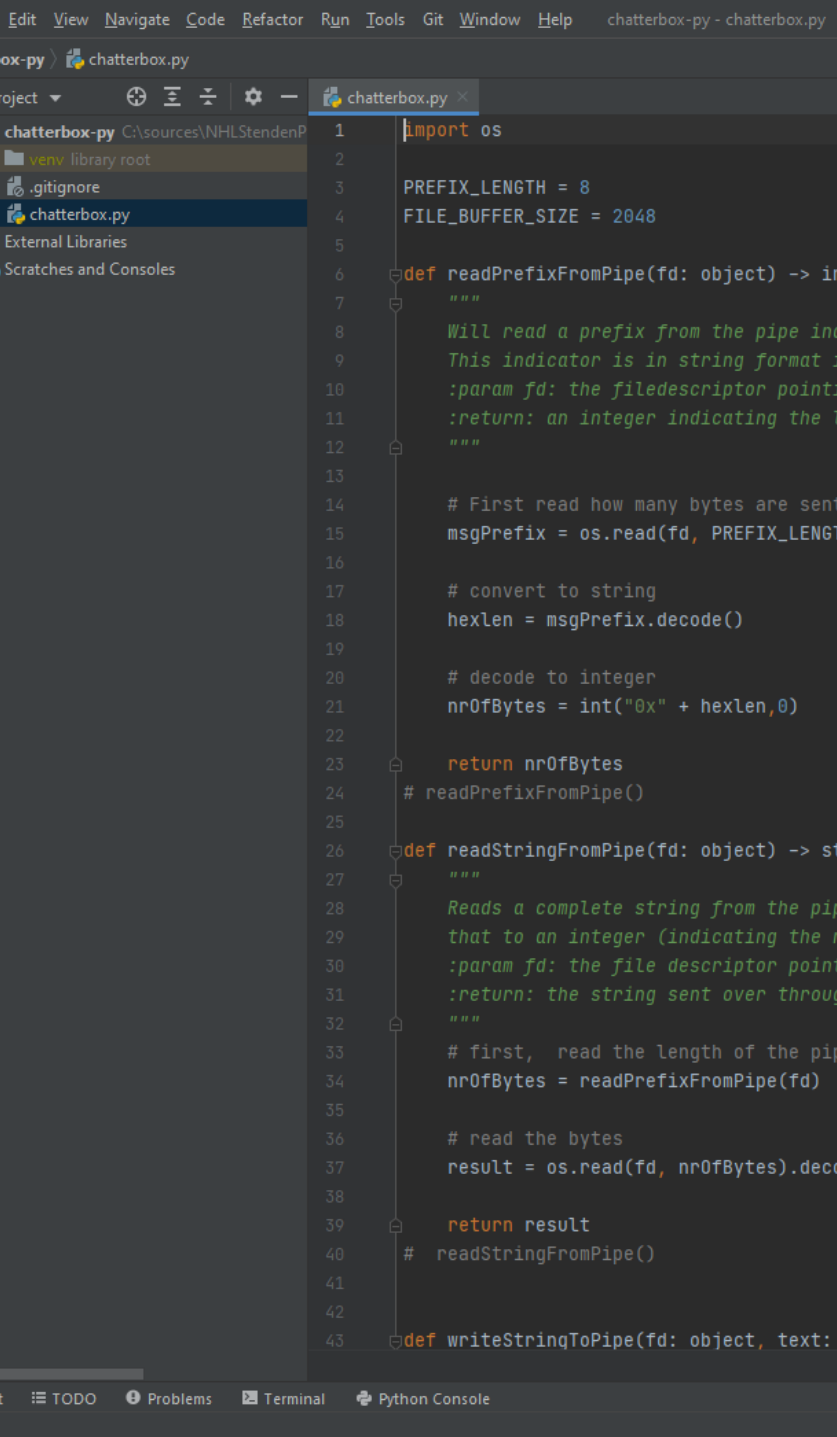
Code op GitHub

<https://github.com/NHLStenden/Robotica-IDP>



Waarom deze workshops?

- Snel kunnen starten
- Niet wachten op beschikbaar komen van fysieke robot
- Voorkomen dat je code schrijft die niet herbruikbaar is
- Toepassing van Design Patterns
- Opstap voor onderzoek 4+1



```
1 import os
2
3 PREFIX_LENGTH = 8
4 FILE_BUFFER_SIZE = 2048
5
6 def readPrefixFromPipe(fd: object) -> int:
7     """
8     Will read a prefix from the pipe into a string.
9     This indicator is in string format.
10    :param fd: the file descriptor pointing to the pipe
11    :return: an integer indicating the length of the prefix
12    """
13
14    # First read how many bytes are sent
15    msgPrefix = os.read(fd, PREFIX_LENGTH)
16
17    # convert to string
18    hexlen = msgPrefix.decode()
19
20    # decode to integer
21    nrOfBytes = int("0x" + hexlen, 0)
22
23    return nrOfBytes
24    # readPrefixFromPipe()
25
26 def readStringFromPipe(fd: object) -> str:
27     """
28     Reads a complete string from the pipe.
29     Converts the string to an integer (indicating the length of the string).
30    :param fd: the file descriptor pointing to the pipe
31    :return: the string sent over through the pipe
32    """
33
34    # first, read the length of the prefix
35    nrOfBytes = readPrefixFromPipe(fd)
36
37    # read the bytes
38    result = os.read(fd, nrOfBytes).decode()
39
40    return result
41    # readStringFromPipe()
42
43 def writeStringToPipe(fd: object, text: str):
```

Waarom WeBots?

- Snelle Prototyping
- Maak gebruik van bestaande robots om zaken uit te proberen
- Bewezen platform

Note:

- WeBots is niet altijd even simpel
- WeBots is veel eisend
- We gaan je helpen
- Tutorials doorlopen.
- Gebruik je IDE
- <https://cyberbotics.com/doc/guide/using-your-ide>



Introductie WeBots

- Demonstratie



Zelf aan de slag

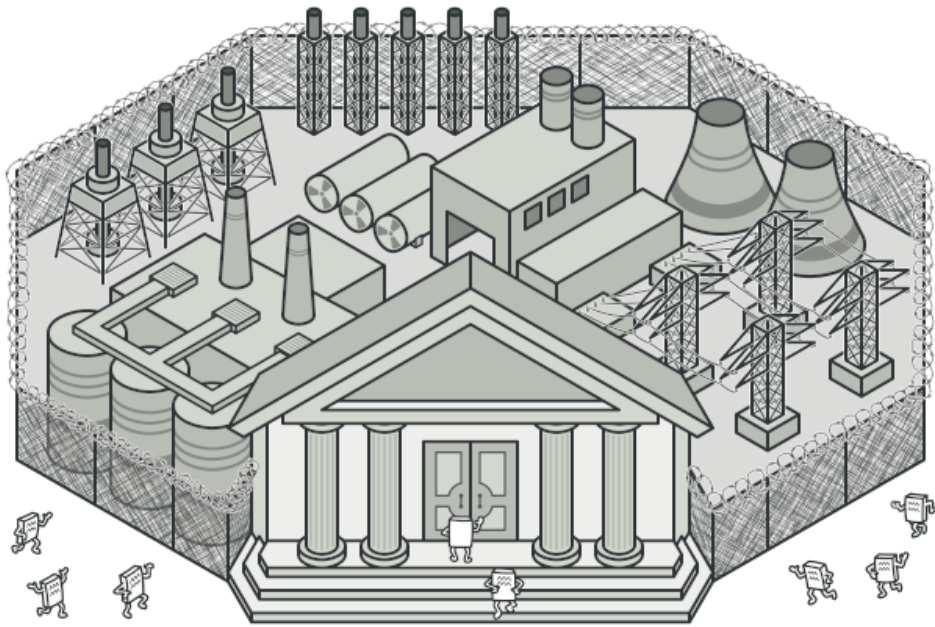
- Welke tutorials?

<https://cyberbotics.com/doc/guide/tutorials>

- Tutorial 1: Simpele robot controller
- Tutorial 4: Motor en sensor controller
- Tutorial 6: Advanced controllers
- (Tutorial 2/3/5 is voor het bouwen van eigen models)

Workshop 2

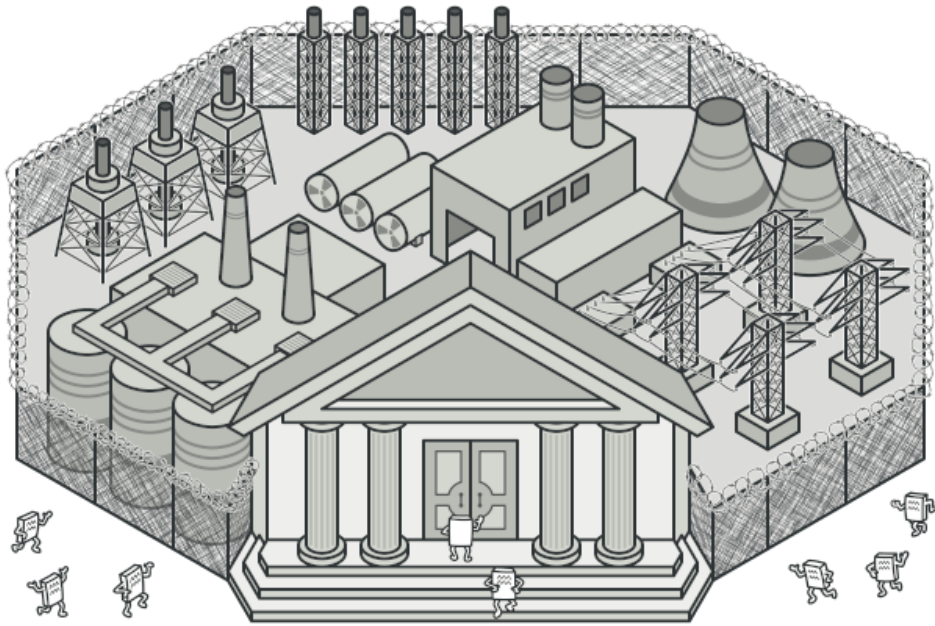
- Façade Design Pattern
- Interne communicatie + Remote Controller
- Voorbeelden



Façade Design Pattern - Problem

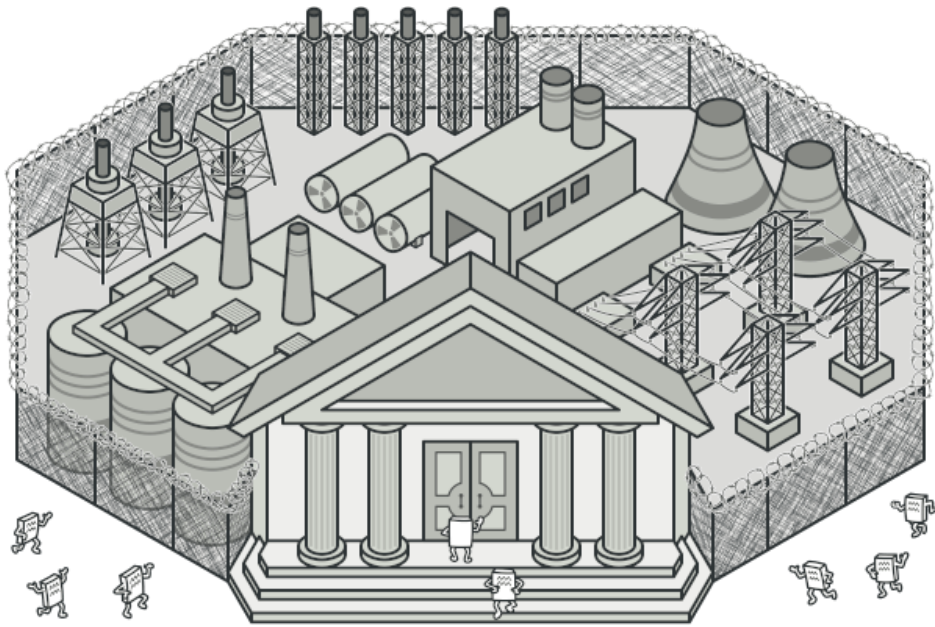
Imagine that you must make your code work with a broad set of objects that belong to a sophisticated library or framework. Ordinarily, you'd need to initialize all of those objects, keep track of dependencies, execute methods in the correct order, and so on.

As a result, the business logic of your classes would become tightly coupled to the implementation details of 3rd-party classes, making it hard to comprehend and maintain.

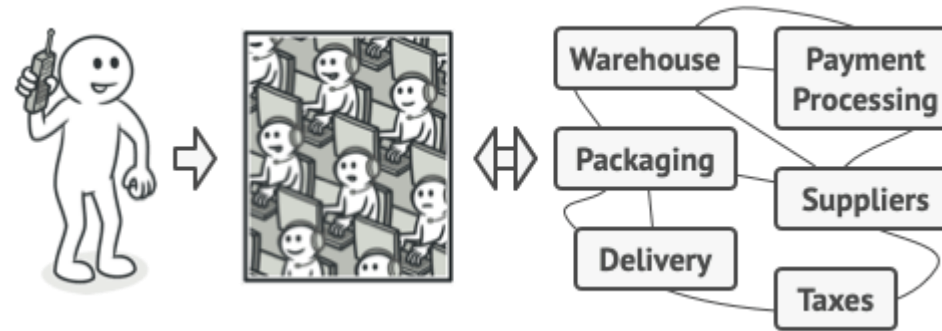


Façade Design Pattern - Solution

A facade is a class that provides a simple interface to a complex subsystem which contains lots of moving parts. A facade might provide limited functionality in comparison to working with the subsystem directly. However, it includes only those features that clients really care about.



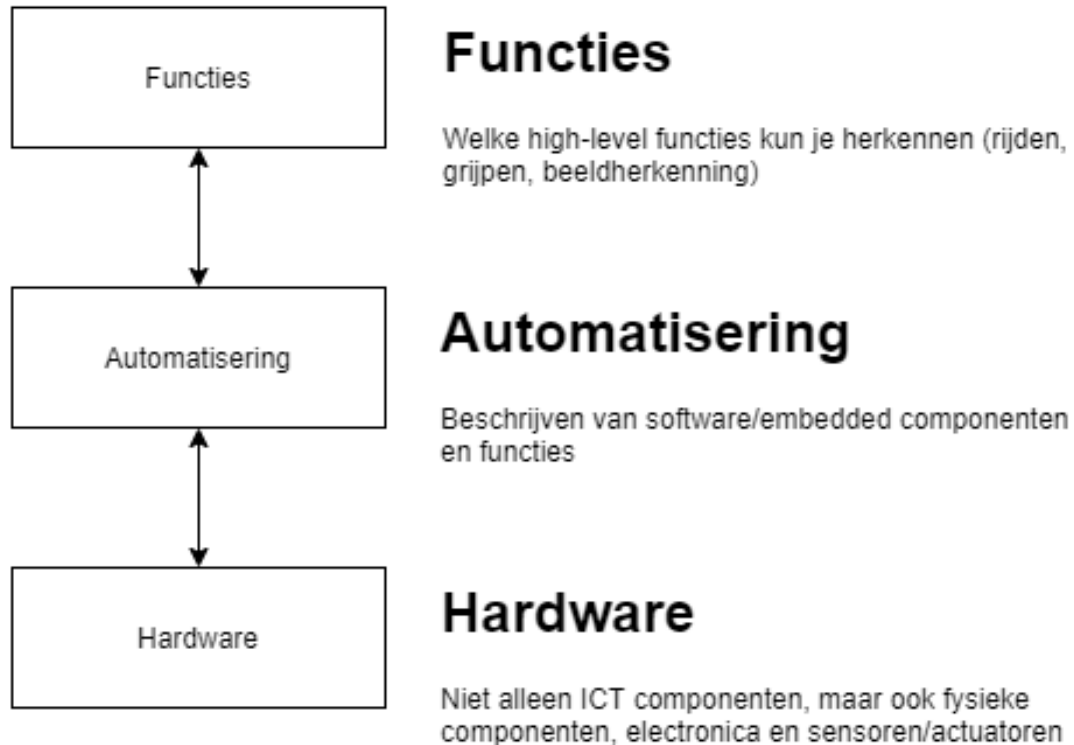
Façade Design Pattern – Real world example



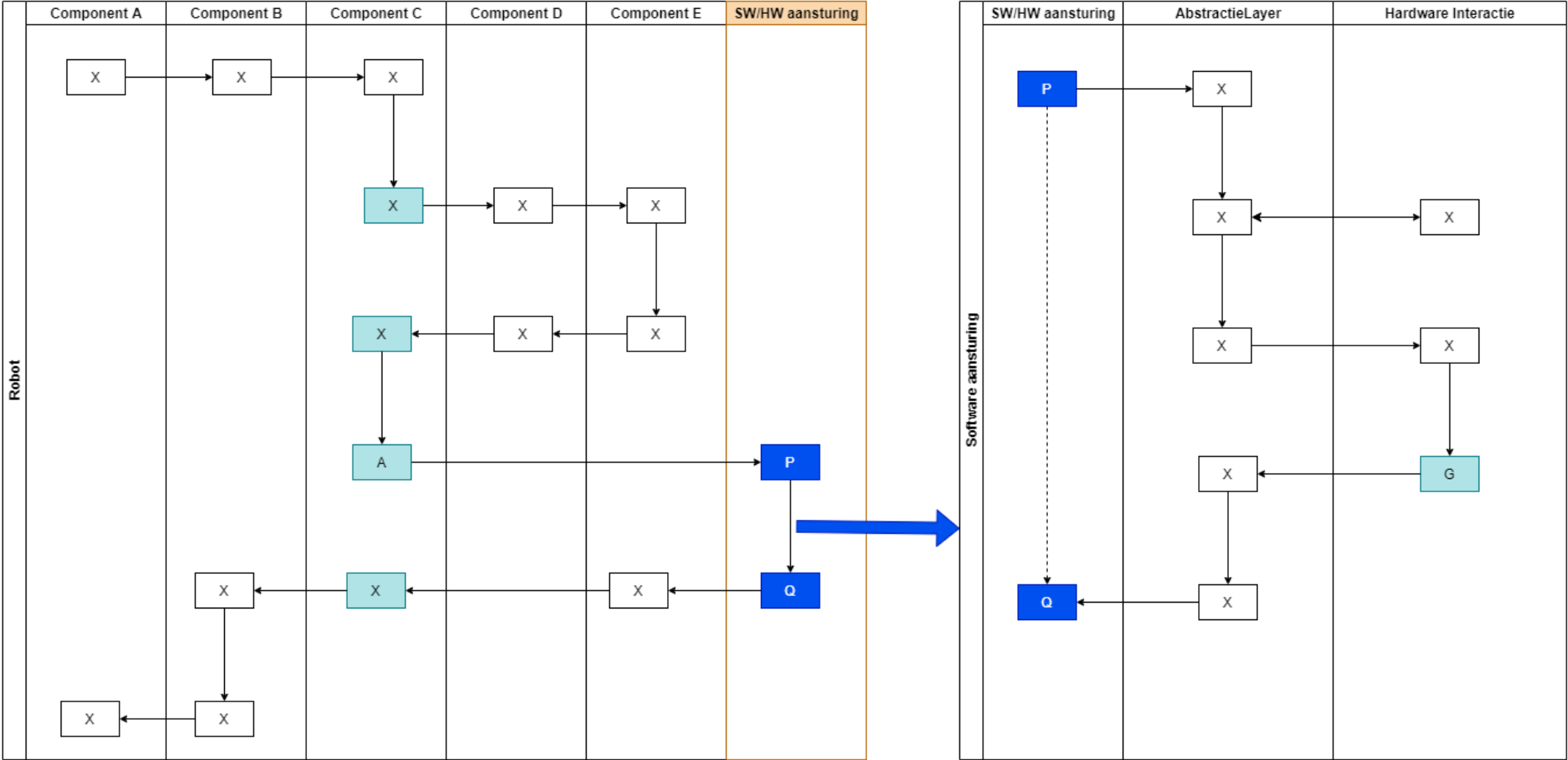
When you call a shop to place a phone order, an operator is your facade to all services and departments of the shop. The operator provides you with a simple voice interface to the ordering system, payment gateways, and various delivery services.

Abstractie van de hardware laag

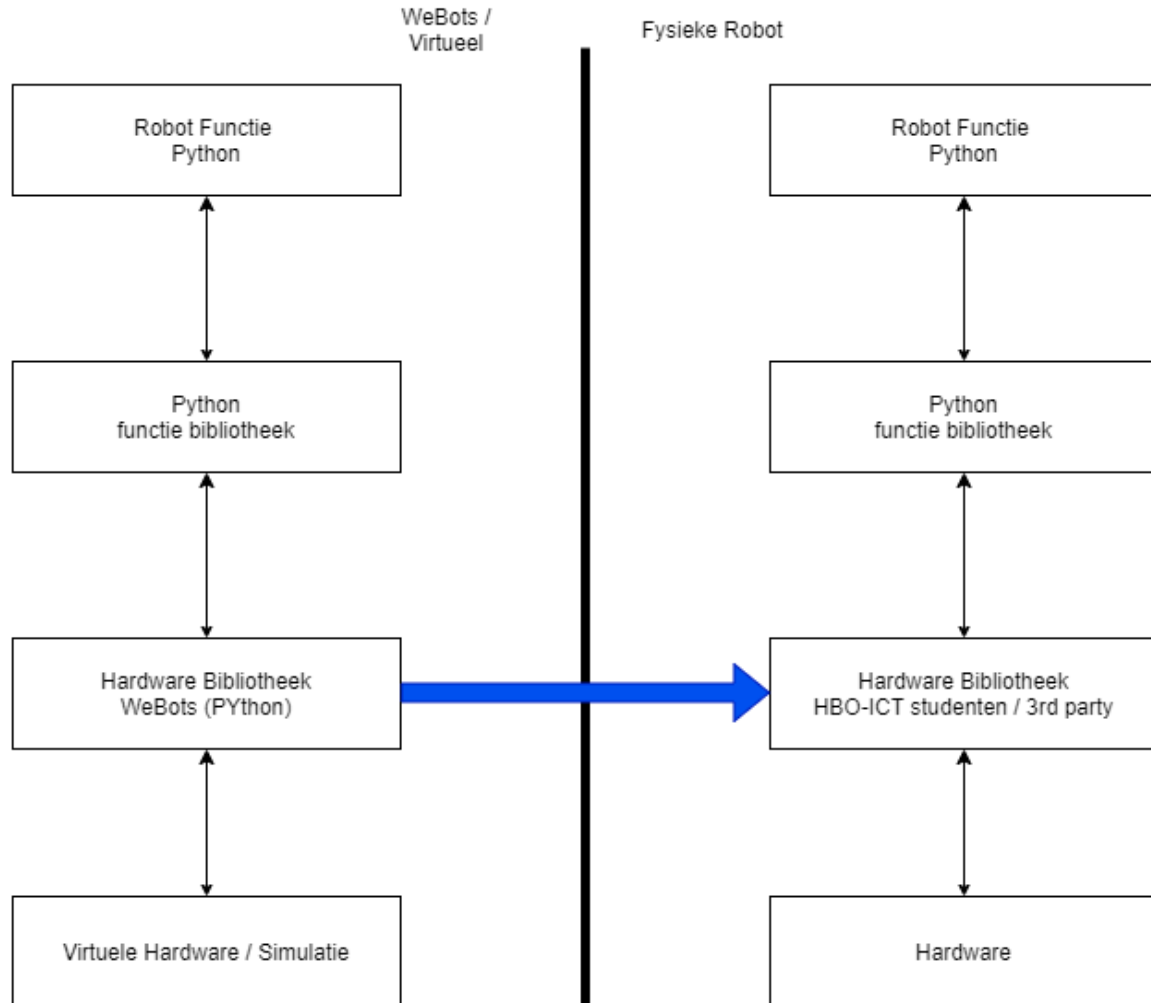
- Veel gebruikt mechanisme
- Windows : “Hardware Abstraction Layer”
- Scheidt de Business Logica van de Hardware aansturing



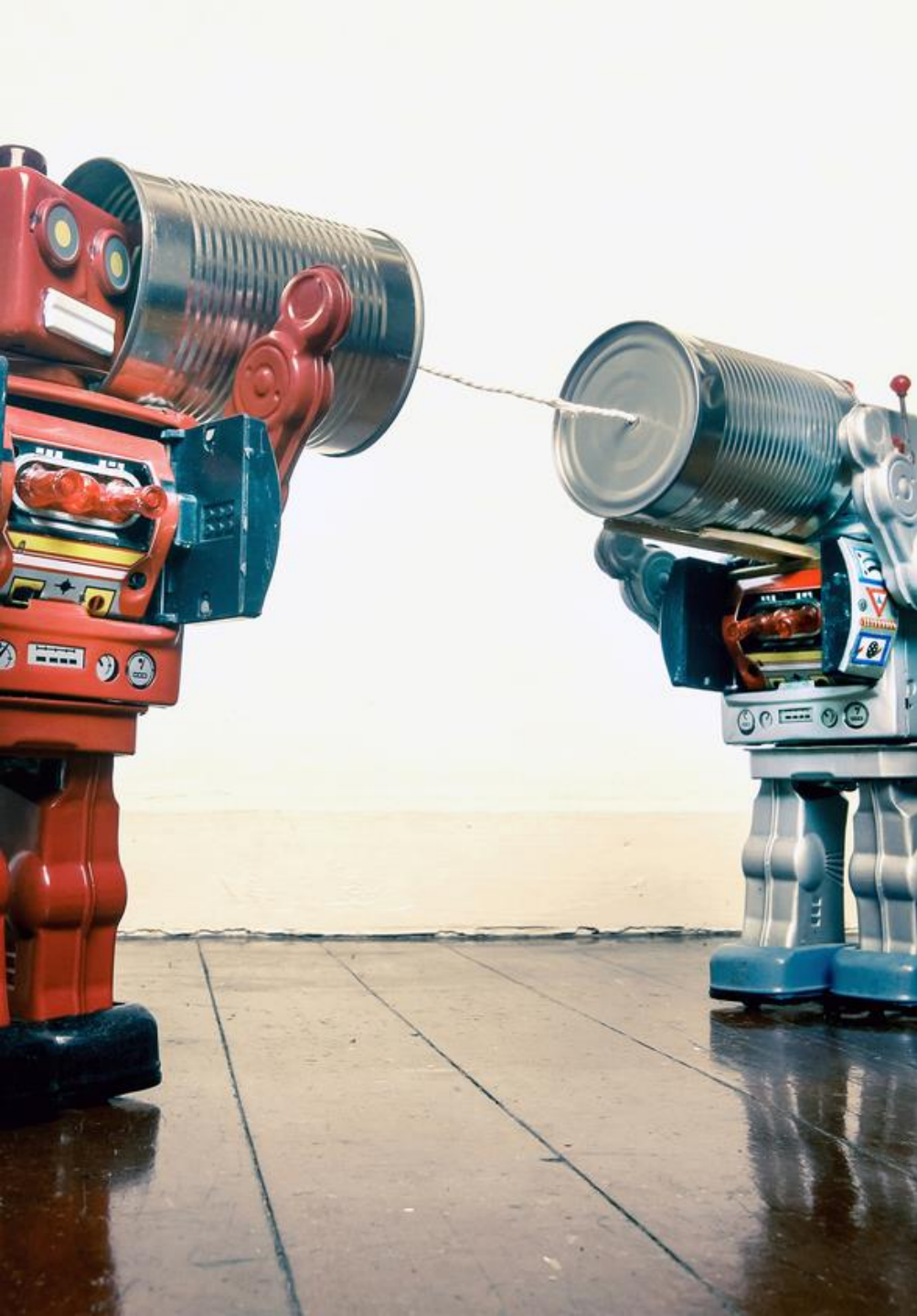
Abstractie van de hardware laag



Van WeBots naar Fysieke Robot

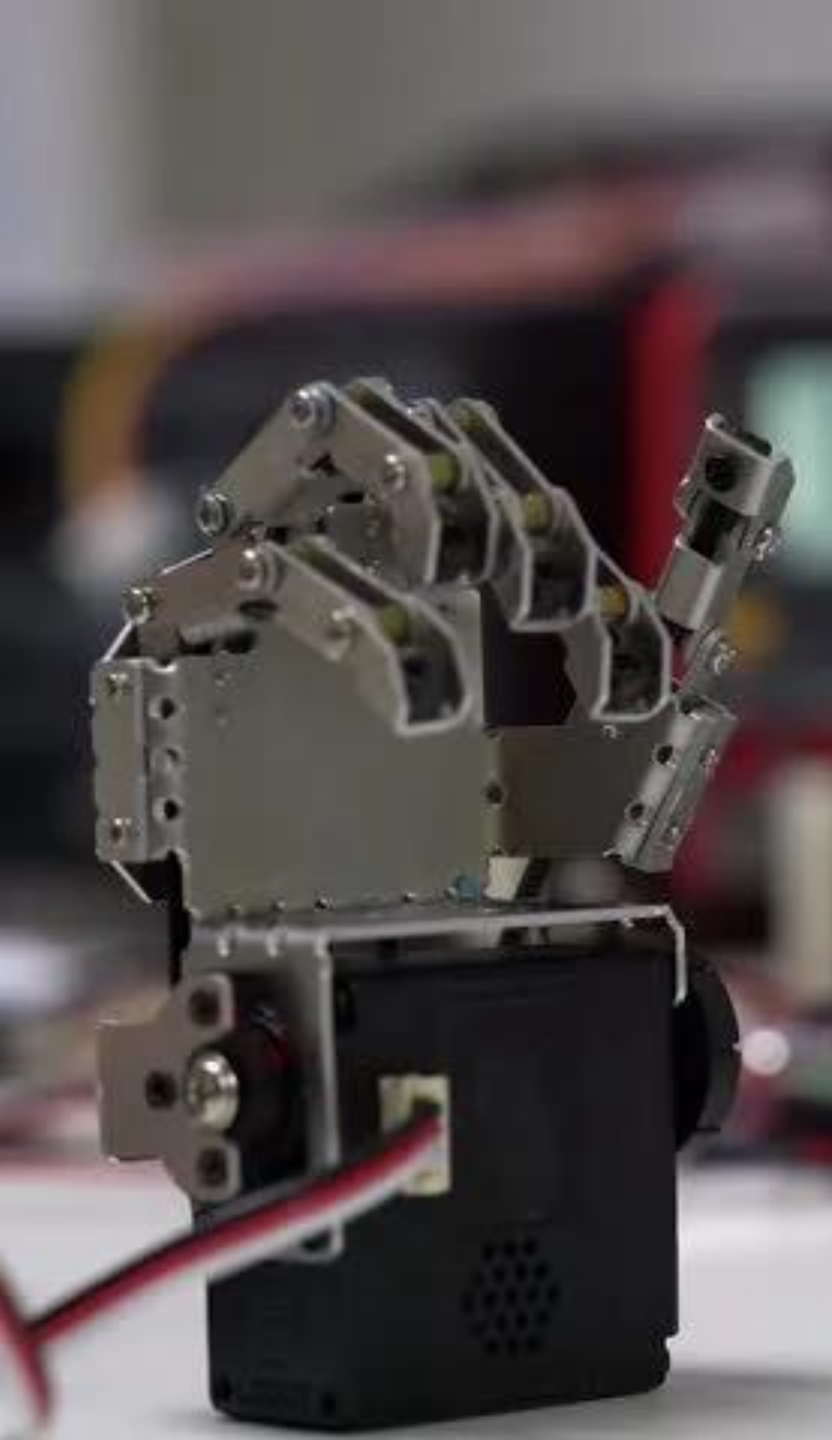


- Als je code hebt geschreven in WeBots kun je deze meenemen naar de fysieke robot
- Hardware bibliotheek opnieuw samenstellen
 - 3rd party bibliotheken
 - Eigen code

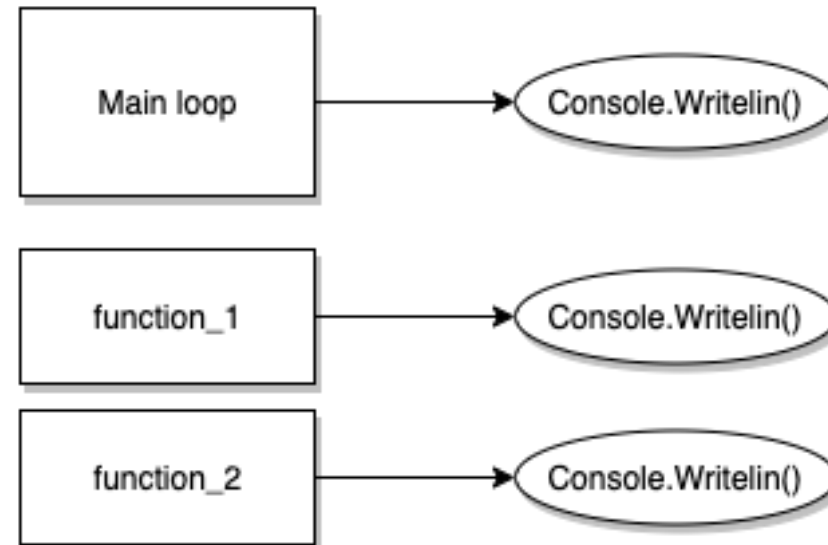


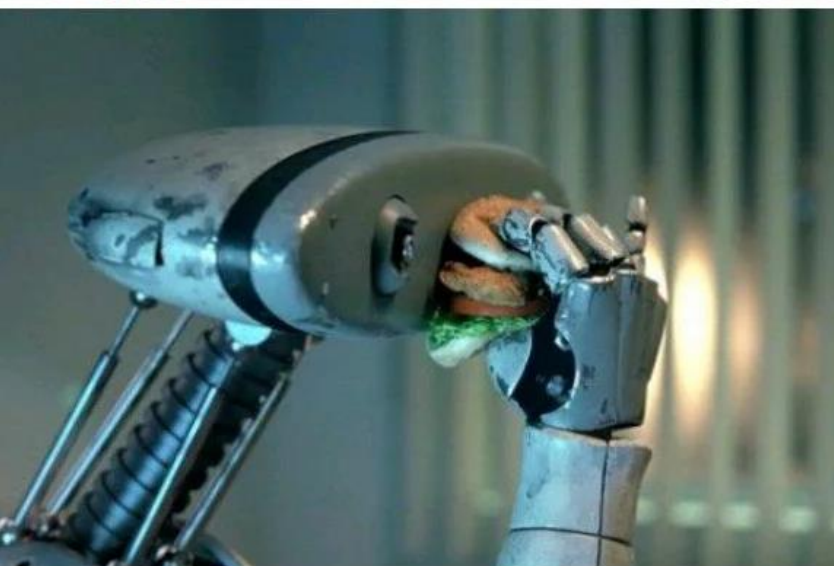
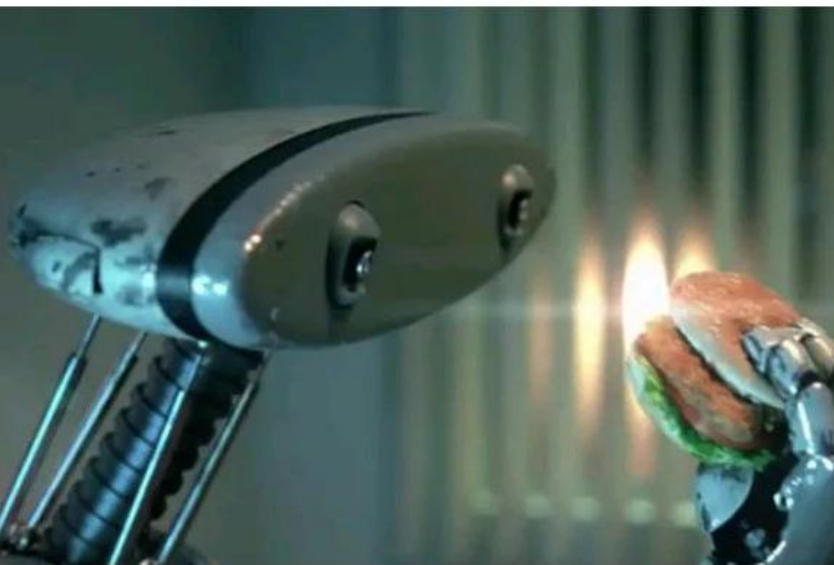
Interne communicatie + Remote Controller

- Communicatie tussen componenten
- Communicatie tussen afstandsbediening en remote controller

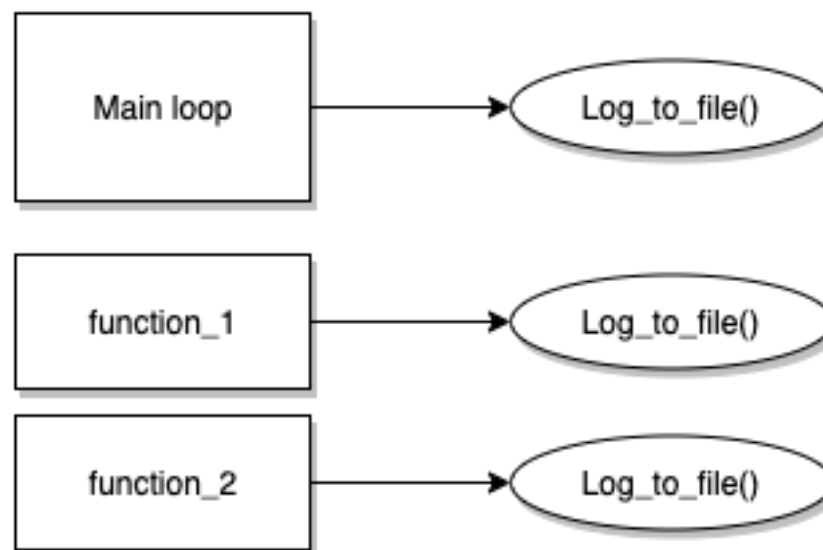


Voorbeeld: logger



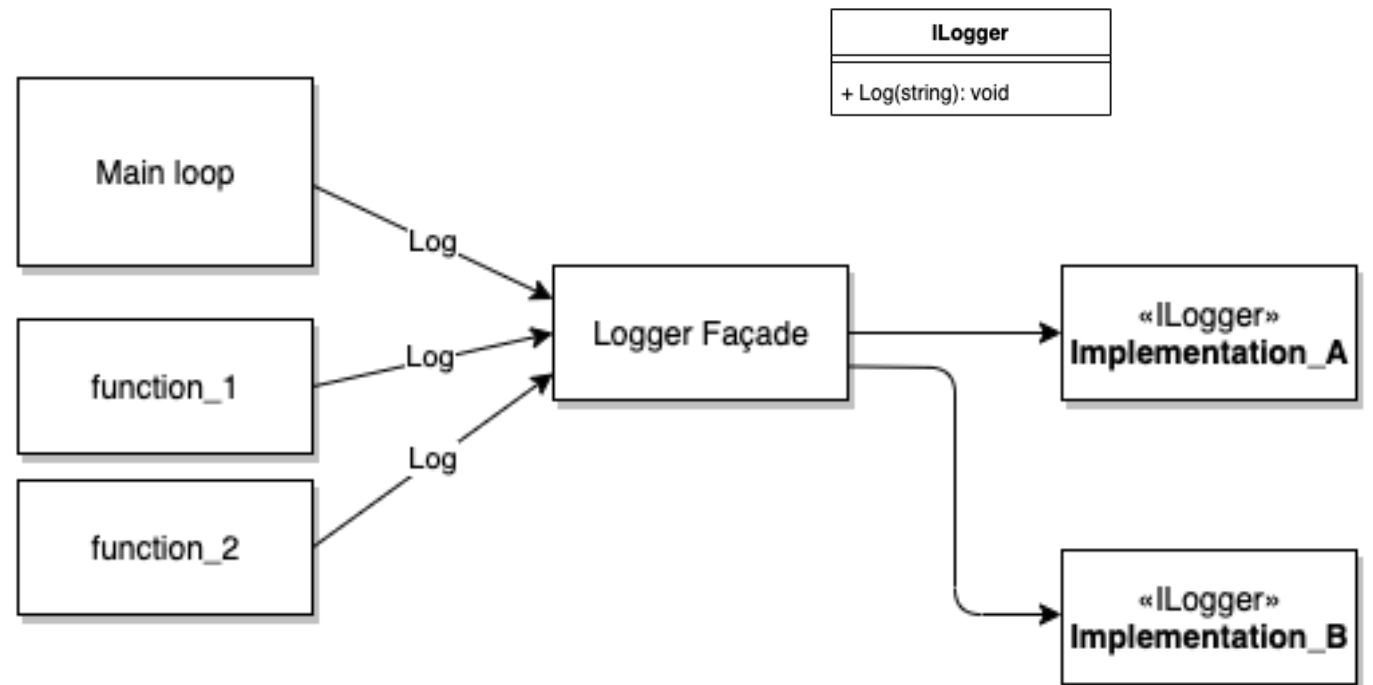


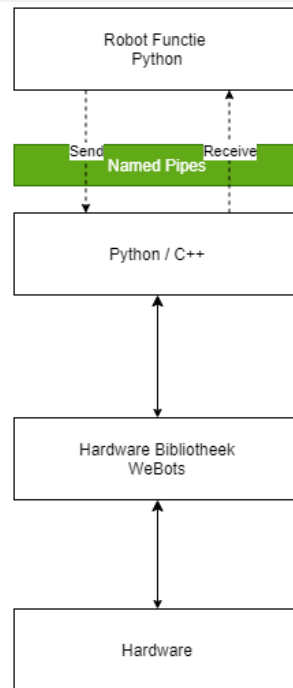
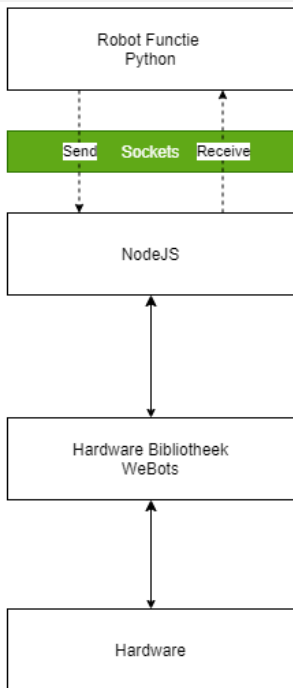
Voorbeeld: logger





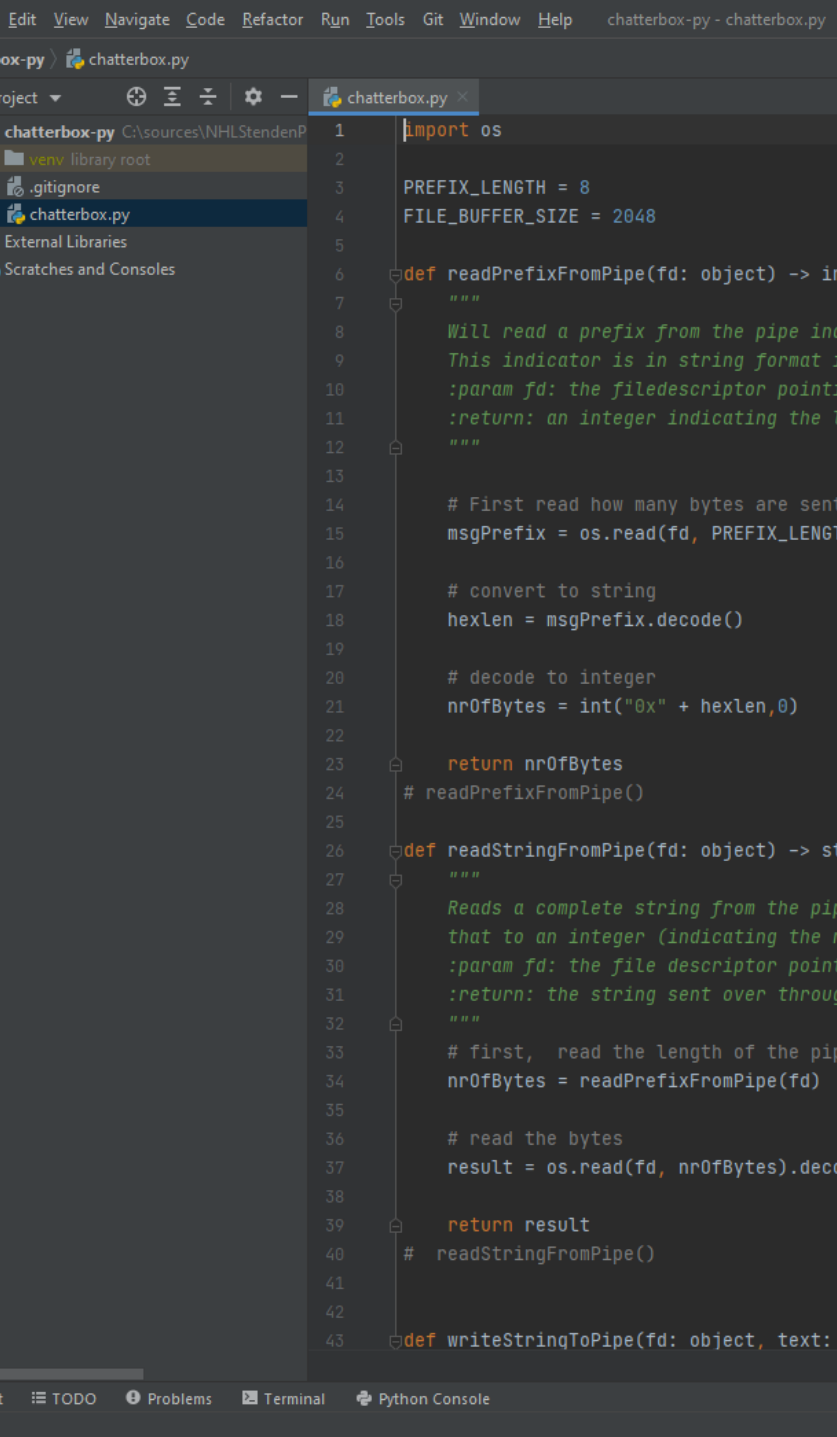
Voorbeeld: logger





Voorbeeld: Pipes (C++ / Python)

- Soms werk je met incompatible talen
- Geen mogelijkheid om functies aan te roepen
 - Aansturing van hardware in C++
 - Business Logica / robot functies in Python
- Hoe laat je deze met elkaar communiceren
- Mogelijke uitkomsten
 - Named pipes
 - Sockets
- Kijk naar de OSI-layers
 - In welke laag zit je werken
 - Stel een protocol op



```
1 import os
2
3 PREFIX_LENGTH = 8
4 FILE_BUFFER_SIZE = 2048
5
6 def readPrefixFromPipe(fd: object) -> int:
7     """
8     Will read a prefix from the pipe into a string.
9     This indicator is in string format.
10    :param fd: the filedescriptor pointing to the pipe
11    :return: an integer indicating the length of the prefix
12    """
13
14    # First read how many bytes are sent
15    msgPrefix = os.read(fd, PREFIX_LENGTH)
16
17    # convert to string
18    hexlen = msgPrefix.decode()
19
20    # decode to integer
21    nrOfBytes = int("0x" + hexlen, 0)
22
23    return nrOfBytes
24    # readPrefixFromPipe()
25
26 def readStringFromPipe(fd: object) -> str:
27     """
28     Reads a complete string from the pipe.
29     Converts the string to an integer (indicating the length of the string)
30    :param fd: the file descriptor pointing to the pipe
31    :return: the string sent over through the pipe
32    """
33
34    # first, read the length of the prefix
35    nrOfBytes = readPrefixFromPipe(fd)
36
37    # read the bytes
38    result = os.read(fd, nrOfBytes).decode()
39
40    return result
41    # readStringFromPipe()
42
43 def writeStringToPipe(fd: object, text: str):
```

Voorbeeld code

- Bestandsuitwisseling tussen C++ en Python via Pipes
- Zie GitHub

**Bedankt voor uw
aandacht**