Übersicht *Player*

- *Player* ist ein Framework (Middleware) für reale und simulierte Roboter basierend auf IP-basierter Kommunikation (Client-Server)
- Grundidee: Aufspaltung der SW eines komplexen Systems in übersichtliche und unabhängige SW-Komponenten (Driver)
- Viele Standardfunktionen (Steueralgorithmen, Auslesen von Sensor-Daten und Ansteuerung der Aktorik) sind als *Komponenten* verfügbar
- Bestimmte *Komponenten* (*Driver*) wechselwirken direkt mit HW
- Eigene *Komponenten* können in das Framework integriert werden
- Echtzeitbedingungen können wegen TCP/IP nicht garantiert werden
- *Komponenten* werden von *Player* als Plug-ins eingebunden und kommunizieren miteinander über standardisierte *Schnittstellen*

Player-Schnittstellen

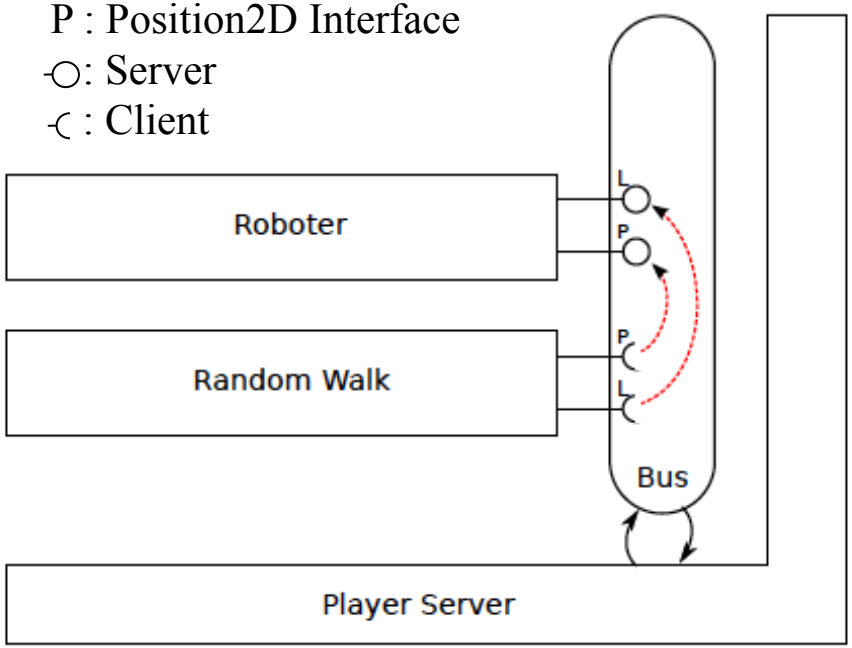
- *Schnittstellen* ermöglichen in *Player* die Kapselung von anwendungs-spezifischen Modulen (*Komponenten/Driver* bzw. *Player Clients*)
- Dadurch leichter Austausch von HW sowie weitestgehend unabhängige SW-Entwicklung und Tests möglich
- Über *Schnittstellen* können Parameter und Daten zwischen Modulen in standardisierten Formaten ausgelesen und gesetzt werden
- *Schnittstellen* sind definiert als spezifische Nachrichten-Protokolle mit festgelegten Datentypen
- In Player sind Schnittstellen für verschiedene Programmiersprachen (C, C++, Python, Ruby, Ada, Java, Lisp, Matlab) enthalten
- Viele Standardschnittstellen sind verfügbar, z. B: ***Position2d***, *Laser,* ***Ranger***, *Fiducial, Map, Camera, Blobfinder, Localize ...*

Verknüpfung von *Komponenten* über *Schnittstellen*

Einfaches Beispiel zur Steuerung einer Zufallsbewegung mit Player

Verbindungen der *Komponenten*:

mit:
L : Laser Interface
P : Position2D Interface
-○: Server
-(: Client



Quelle: Langbehn, „Das Player / Stage System“

Zugehörige Konfigurationsdatei:

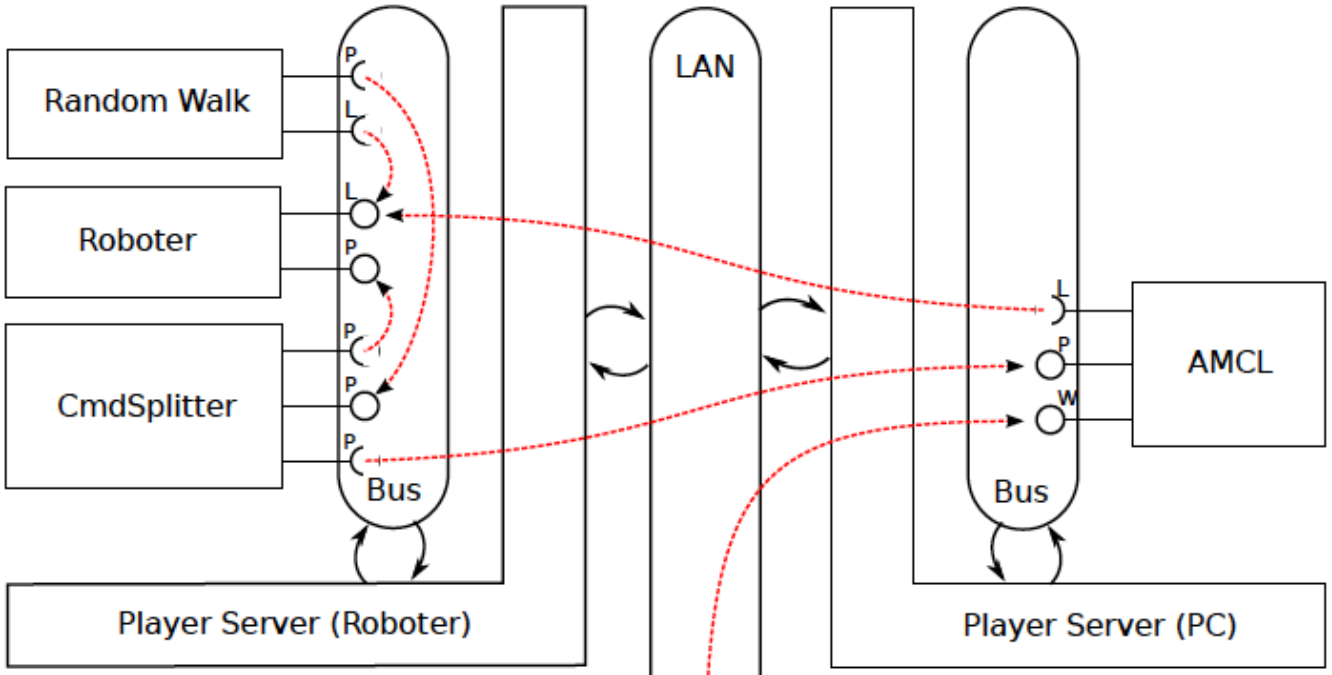
```
driver
(
  name "Roboter"
  provides ["position2d:0" "laser:0" ]
)

driver
(
  name "RandomWalk"
  requires ["position2d:0" "laser:0" ]
  alwayson 1
)
```

Aufruf mit:

```
player name.cfg
```

Komplexere Implementierung im Netzwerk



Quelle: Langbehn, „Das Player / Stage System“

Als Clienten dienen in *Player* integrierte Tools oder eigene Programme

mit:
L : Laser Interface
P : Position2D Interface
W: Localize Interface
-○: Server
-ᑕ : Client

Verbinden eigener C++ Clients mit *Player*

- Im Client-Programm dienen lokale Klassenobjekte als Service Proxies für die Verbindung mit entfernten *Player*-Komponenten
- Diese Objekte stellen Methoden im Client zur Verfügung und kommunizieren über *Schnittstellen*-spezifische Protokolle
- Voraussetzung ist die Einbindung von *libplayerc++/playerc++.h*
- Zunächst wird ein Proxy vom Typ *PlayerClient* erzeugt, welches den Client mit einem Player-Host/-Port koppelt und über Methoden Übertragungsparameter festlegt sowie Daten einliest
- Mit diesem *PlayerClient* werden anschließend die erforderlichen *Schnittstellen*-Proxies für den Datenaustausch mit den *Player-Komponenten* entsprechend der Konfigurationsdatei verbunden

Grundstruktur eines *PlayerClients* in C++

```
#include <libplayerc++/playerc++.h>

int main(int argc, char *argv[]) {

        using namespace PlayerCc; // Player-spezifischer namespace
        // Erzeugung eines Objektes als Player-Proxy
        PlayerClient robot(host, port);

        // Einbindung der Schnittstellen-Proxies
        Position2dProxy p2dProxy(&robot,0);
        SonarProxy sonarProxy(&robot,0);
        BlobfinderProxy blobProxy(&robot,0);
        LaserProxy laserProxy(&robot,0);

        // eigener Quellcode für den Client

        return 0;
}
```

Mehr Infos unter: http://playerstage.sourceforge.net

Übersicht *Stage*

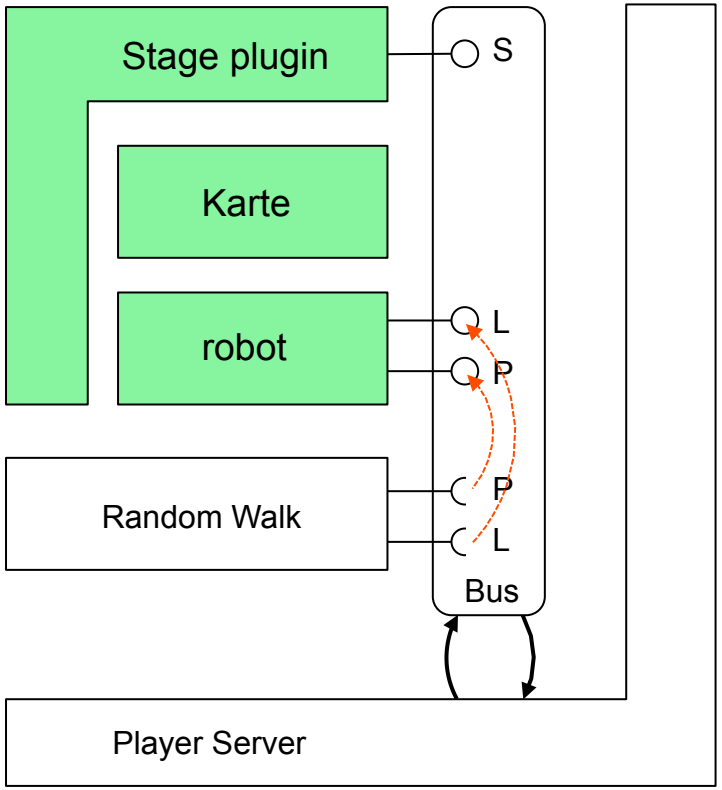
- Stage ist ein Simulationsprogramm für Roboter mit ihren Sensoren
- Insbesondere auch einsetzbar für eine größere Anzahl von Robotern
- Komplexe Roboter- und Kartenkonfigurationen können über eine sogenannte *world*-Datei flexibel vorgegeben werden
- Beschränkung auf 2 Dimensionen, Darstellung und Sensorauswertung aber auch in z-Richtung möglich (2.5D Simulator)
- Als Plugin *Komponente* für Player verfügbar, wird beim Start von Player automatisch geladen
- Unterstützt Standardschnittstellen von Player, Simulationen können daher einfach auf reale Roboter portiert werden
- Ermöglicht über Player das Zeichnen in die Simulationsumgebung

Einbindung von *Stage* in das *Player*-Framework

Beispiel: Zufallsbewegung mit simuliertem Roboter

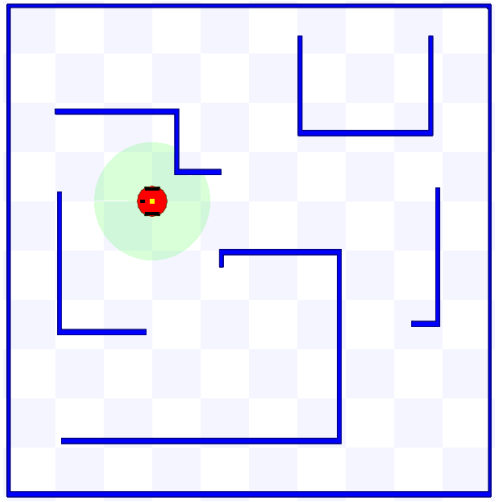
Player-Konfiguration (cfg-Datei):

```
driver (
  name "stage"
  provides [ "simulation:0" ]
  plugin "stageplugin"
  worldfile „virtual.world“
)
driver (
  name "stage"
  provides ["position2d:0" "laser:0"]
  model "robot"
  alwayson 1
)
driver
(
  name "RandomWalk"
  requires ["position2d:0" "laser:0" ]
  alwayson 1
)
```



Player / *Stage* Konfiguration für AMS

Simulierte Karte mit Roboter:



cfg-Datei:

```
driver (
  name "stage"
  provides [ "simulation:0" ]
  plugin "stageplugin"
  worldfile "AMS.world“
)
driver (
  name "stage"
  provides [ "graphics2d:1" ]
  alwayson 1
)
driver (
  name "stage"
  provides ["position2d:0" "ranger:0" "graphics2d:0“]
  model "r0"
  alwayson 1
)
```