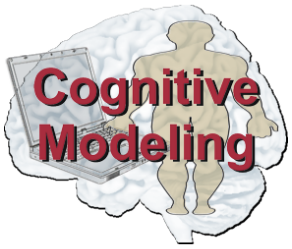
EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Technical Report
No. COBOSLAB_Y2013_N002
21. November 2013

# SimulationManager 1.0 - Abstraktion und Parallelisierung von Simulationen

Deutsche Dokumentation

Fabian Schrodt




Wilhelm-Schickard-Institut für Informatik
University of Tübingen
Sand 14
72076 Tübingen, Germany
http://www.cm.inf.uni-tuebingen.de

# SimulationManager 1.0 - Abstraktion und Parallelisierung von Simulationen

Fabian Schrodt*

**Kurzfassung**

Diese Dokumentation dient der technischen Beschreibung des SimulationManager-Frameworks. SimulationManager ist ein quelloffenes Framework für die Abstraktion und automatische Parallelisierung von Simulationen. Das Framework ist im Rahmen der Masterthese von Fabian Schrodt („Imitation von Verhalten mittels neuronaler Netze“) entstanden und verfolgt die Zielsetzung, öffentliche Algorithmen und Implementierungen bereitzustellen, welche die Anbindungen verschiedener Lernalgorithmen an beliebige Simulationen erleichtern und die Auswertung derselben beschleunigen. Das Framework ist für Linux 32/64 Bit unter GPL 3.0 verfügbar. Eine Kopie der Lizenz befindet sich im Anhang dieses Dokuments.

*FSchrodt@gmx.de

# 1 SimulationManager-Framework

## 1.1 Einführung

Beim SimulationManager handelt sich um ein Simulations-Verwaltungsframework, das als einheitliche Schnittstelle für die Kommunikation von beliebigen Programmen oder Programmmodulen mit beliebigen Simulationen konzipiert ist. Er dient dazu, die Regelung und Auswertung von Simulationen und das Anbinden von Lernalgorithmen zu verallgemeinern und zu vereinfachen. Das Framework gewährleistet die konfliktfreie Nebenläufigkeit aller Simulationen mittels einer einheitlichen Kontrollstruktur.

Der Funktionsumfang des SimulationManagers kann leicht um eigene Komponenten erweitert werden: Der Manager kann simulationskontrollierende Programmbibliotheken beim Start oder zur Laufzeit dynamisch und in beliebiger Zahl laden, daher werden diese im Folgenden als *Plugins* bezeichnet. Alle Plugins laufen als Threads im Adressraum des Managers und implementieren eine beliebige Funktionalität wie einen Lerner, einen Regler, eine Netzwerkschnittstelle oder eine GUI. Plugins können über eine definierte Kommandostruktur Simulationen starten, auf diese zugreifen und diese wieder beenden. Die Verwaltung, Ansteuerung und Parallelisierung der Simulationen fällt dabei in den Aufgabenbereichen des Managers, so dass dieser als Abstraktionsebene für Simulationsauswertungen anzusehen ist.

Da es sich in bestimmten Fällen als vorteilhaft oder gar notwendig erweist, eine adressraumunabhängige Ausführung von Simulationen sicherzustellen, werden diese zur Laufzeit als eigenständige Prozesse gestartet. Die dadurch bedingte Interprozesskommunikation (inter-process communication, IPC) zwischen Manager und Simulationen ist minimalistisch, mehrfach gepuffert und parallelisiert, um eine verzögerungsfreie Ansteuerung von Simulationen zu ermöglichen. Aus der Pufferung und Zuweisung von Simulationskommandos folgt zudem die Eigenschaft, dass Simulationsansteuerungen vorausgeplant und automatisch parallelisiert werden können.

In Abbildung 1 ist die Modularisierung des Frameworks aufgezeigt. Für die Integration von Simulationen und Plugins sind Programmierschnittstellen bereitgestellt.

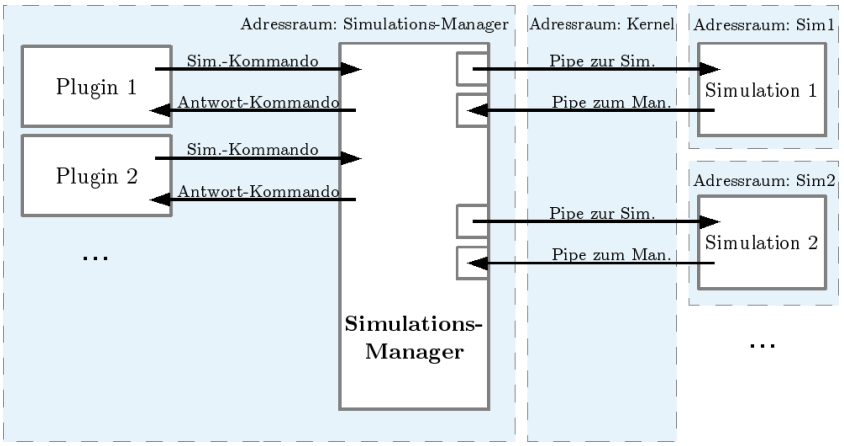



**Abbildung 1:** Diese Abbildung zeigt den modularen Aufbau des SimulationManagers und die Kommunikationsregelung zwischen Plugins und Simulationen. Alle grauen Kästen im Bild entsprechen einem Thread im jeweiligen Prozess. Aus Sicht eines Plugins verläuft die Kommunikation mit Simulationen ausschließlich über Simulations-Kommandos, welche in Kapitel 1.3 vorgestellt werden. Der Manager verwaltet pro Simulation je einen Puffer für die Abarbeitung von Kommandos und Annahme von Antworten. Die aus der adressraumunabhängigen Ausführung von Simulationen folgende Interprozesskommunikation wird vollständig vom Manager und dessen API übernommen und ist über bidirektionales FIFO-Piping realisiert.

## 1.2 Plugins

Ein Plugin ist im Sinne dieser Dokumentation ein Shared Object, das ausgehend von einer vorgegebenen Schnittstelle eine beliebige Funktionalität implementiert, um die Möglichkeiten des Managers zu erweitern oder von ihm kontrollierte Simulationen anzusprechen und auszuwerten.

Plugins können direkt beim Start des SimulationManagers geladen werden: Jede nicht mit `#` auskommentiere Zeile der Datei `plugins.txt` wird als Dateiname eines Shared Objects inklusive Pfad relativ zur Manager-Executable interpretiert und per Linux dynamic loader (siehe [1]) geladen und ausgeführt. Das Laden von Plugins kann ebenso zur Laufzeit über die Manager-Funktion `PluginInterface* loadPlugin( const char* so_name)` geschehen. Auf diese Weise kann der Manager bei Bedarf angewiesen werden, weitere Plugins zu laden, was zum Aufbau von Plugin-Hierarchien verhilft. Analog weist die Funktion `int unloadPlugin(PluginInterface* plugin)` den Manager an, ein Plugin zu entladen. Da alle Plugins im selben Adressraum ausgeführt werden ist eine Kommunikation untereinander grundsätzlich möglich, diese fällt jedoch nicht in den Aufgabenbereich des Managers. Aus selbigen Grund müssen nebenläufige Plugins so implementiert werden, dass keine Konflikte zwischen statischen oder globalen Objekten entstehen.

Jedes Plugin muss die externe Funktion `extern "C" PluginInterface * createPlugin(SimulationManager* man)` implementieren, welche als als Eintrittspunkt für den SimulationManager dient und diesem eine von `PluginInterface` vererbte Instanz zurückliefert (siehe Abbildung 2). Das Interface erzwingt die Implementierung von Funktionen zum Starten und Beenden des Plugins und stellt zudem den Zugriff auf den Manager bereit: Wenn ein Plugin erfolgreich geladen wurde, wird die Implementierung der virtuellen Funktion `virtual void run()` als unabhängiger Thread aufgerufen, welche als Haupt-Routine des Plugins dient. Initialisierungen von Plugin-internen Objekten können hier oder auch im Konstruktor des Plugins vorgenommen werden. Handelt es sich um ein permanentes Plugin, ist dies anhand einer Schleife zu realisieren, da `run()` nur einmalig aufgerufen wird. Sobald der Manager das Beenden und Entladen eines Plugins fordert, wird die Implementierung der virtuellen Funktion `virtual void exit()` vom Manager aufgerufen. Diese Funktion muss die Ausführung der `run()`-Funktion beenden und allen verwendeten Speicher freigeben. Zu beachten ist, dass `exit()` vom Manager aus aufgerufen werden kann, während `run()` in einem separaten Thread läuft, daher muss das Abschließen der `run()`-Funktion threadsicher geschehen (zum Beispiel über eine per `pthread_mutex_lock` gesicherte Indikatorvariable). Plugins können untereinander anhand der Funktion `bool isRunning()` prüfen, ob ein Plugin noch läuft oder bereits beendet wurde.

Ein vollständig geladenes und ausgeführtes Plugin kann fortan jede

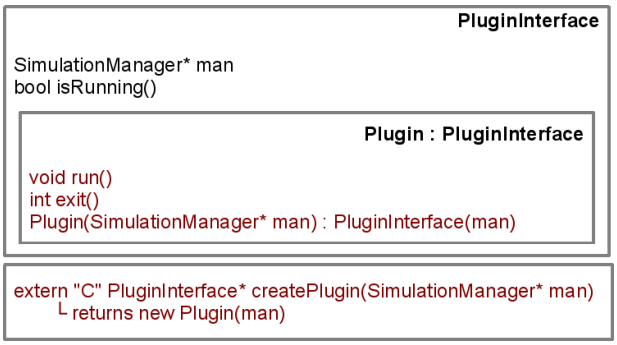



**Abbildung 2:** Aufbau eines Plugins im Pseudo-Klassendiagramm. Virtuelle Funktionen sind in rot dargestellt. Ein Plugin wird von `PluginInterface` abgeleitet und muss die virtuellen Funktionen `run()`, `exit()` und den eigenen Konstruktor implementieren, welcher einen Pointer auf den Manager an das Plugin-Interface übergibt. Das externe Symbol `createPlugin` dient als Eintrittspunkt für den dynamischen Loader und gibt einen Pointer auf eine Instanz der abgeleiteten Plugin-Klasse zurück.

öffentliche Funktion des Managers aufrufen, ohne auf simultanen Zugriff anderer Plugins zu achten, da die Threadsicherheit managerintern sichergestellt wird. Von besonderem Interesse ist in diesem Zusammenhang die Funktion `SimulationCommand SimulationManager::applyCommand( SimulationCommand* cmd)`, welche an Simulationen gerichtete Kommandos entgegennimmt und verarbeitet. Sie dient als alleinige Schnittstelle zwischen Plugins und Simulationen und wird im folgenden Kapitel erläutert.

## 1.3 Simulations-Kommandos

### 1.3.1 Datenstruktur der Kommandos

Ein Simulations-Kommando (Klasse `SimulationCommand`) ist eine dynamische Datenstruktur, die der Manager für die Kommunikation mit Simulationen vorschreibt. Kommandos, die der Manager empfängt, werden von diesem entweder direkt verarbeitet oder einer Simulation zugewiesen und gepuffert. Ebenso verläuft die Antwort des Managers an Plugins über dieselbe Kommandostruktur. Wenn ein Simulations-Kommando an die Manager-Funktion `SimulationCommand applyCommand(SimulationCommand* cmd)` übergeben wird, liefert diese in jedem Fall eine sofortige Rückgabe in Form eines Simulations-Kommandos. Generell sind jedoch blockierende und nicht-blockierende Kommandos zu unterscheiden: Während blockierende Kommandos den aufrufenden Thread bis zur Abarbeitung blockieren, kehren nicht-blockierende Kommandos immer unmittelbar zurück, und liefern

gegebenenfalls verspätet eine weitere Antwort an das aufrufende Plugin. Erst dadurch wird eine effiziente Parallelisierung der Simulationsansteuerung ermöglicht. Generell gilt: Sind alle Kommandos, die zur Durchführung mehrerer Simulationsauswertungen verwendet werden, nicht-blockierend, können alle Simulationen selbst bei sequenzieller Übergabe der Kommandos parallel laufen.

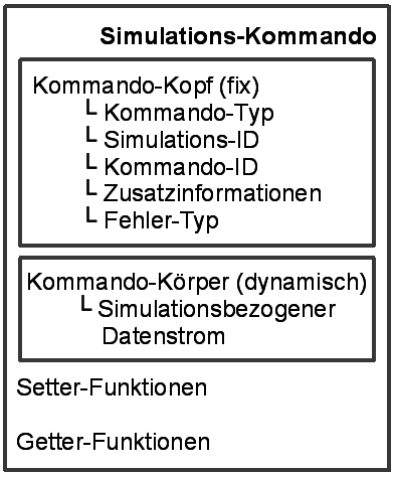



**Abbildung 3:** Ein Simulations-Kommando besteht aus Kopf und Körper. Der Kopf ist eine fixe Informationsstruktur zur Auswertung des Kommandos durch den Manager, der Körper kann Sensor- oder Aktordaten enthalten oder leer sein.

Jedes Kommando setzt sich aus einem Kopf und einem Körper zusammen. Während der Kopf eine Datenstruktur von fester Länge ist und alle eventuell zur Abarbeitung eines Kommandos benötigten Informationen beinhaltet (siehe Abbildung 3), ist der Körper je nach Kommando-Typ entweder leer oder verweist auf einen Datenstrom von simulations- bzw. kommandospezifischen Nutzdaten.

Der Kopf eines Kommandos besteht aus den folgenden Feldern:

- `MessageType messageType`: Enumerator, der den Typ des Kommandos angibt.
- `unsigned int simulationID` (simID): Enthält die ID der Simulation, die das Kommando betrifft. Die Simulations-ID wird dem Erzeuger einer Simulation einmalig als Antwort zurückgeliefert, und muss von diesem vermerkt werden.
- `unsigned int messageID` (mID): Eine Nachrichten-Identifikationsnummer kann optional mit nicht-blockierenden Kommandos, die eine verzögerte Antwort liefern, übergeben werden. Verzögert eintreffende Antworten beinhalten immer die ID des Kommandos, das die Antwort ausgelöst hat, so dass diese eindeutig zugeordnet werden können.
- `unsigned int additionalInfo` (aI): Enthält optionale Informationen zur weiteren Vereinfachung der Zuordnung verzögerter Antwort-Kommandos. Dieses Feld wird vom Manager nicht ausgewertet, aber in jeder bezüglichen Antwort zurückgegeben. Realisiert werden kann

hiermit beispielsweise die Angabe einer Speicher-Adresse, an welche Daten aus einer verspäteten Antwort geschrieben werden sollen.

- `unsigned int errorType`: Enthält Informationen über eventuell aufgetretene Fehler (die zugehörige Fehlermeldung wird ebenso in der Konsole ausgegeben). Dieses Feld ist nur bei Antwortkommandos des Typs `ERR` zu beachten.

Simulations-Kommandos sind als serielle Datentypen implementiert, sie gewährleisten also die zusammenhängende Allokation des Speichers jeweils für Kopf und Körper eines Kommandos, was über die Klasse `StreamableDataContainer` realisiert ist (siehe Kapitel 1.6.1). Diese ermöglicht das Streamen einer beliebigen Komposition fundamentaler Datentypen (d.h. alle im Speicher zusammenhängende Datentypen die nicht auf andere Speicherbereiche verweisen) beispielsweise über das Netzwerk oder Pipes. Zum einen wird diese Eigenschaft benötigt, um die IPC des Managers mit Simulationen (siehe Kapitel 1.5) zu ermöglichen, und zum anderen um das Versenden von Simulations-Kommandos über Netzwerk-Plugins effizient zu gestalten. Die Verwendung einer Serialisierung on-demand (beispielsweise per `boost`) ist möglich, kann aber Performanz-Nachteile mit sich bringen.

### 1.3.2 Kommandotypen

Nach der Erzeugung eines Kommando-Objekts wird über die entsprechende Setter-Funktion dessen Typ eingestellt (anhand der Funktionen `void SimulationCommand::setMessage...(...)`). Jeder Kommando-Typ fordert eine Mindestzahl genau definierter Parameter, die für dessen Abarbeitung ausreichen (siehe Übersichtstabellen 1 und 2). Desweiteren bietet die Datenstruktur diverse Operatoren, Konstruktoren und einen sauberen Destruktor, die einmal erzeugte Kommando-Objekte wiederverwendbar machen, Speicherlecks vermeiden und die Verwendung für netzwerkfähige Plugins erleichtern.

Für das Holen der hier angegebenen Informationen aus Kommando-Antworten stehen entsprechende Getter-Funktionen bereit.

Im Folgenden werden die einzelnen Kommando-Typen und ihre Funktionen einzeln erläutert. Die Zusätze **(b)** und **(nb)** beschreiben, ob es sich um blockierende oder nicht-blockierende Kommandos handelt. Sollen beispielsweise Sensoren ausgelesen werden, stehen je nach Aufgabenbereich eine blockierende und eine nicht-blockierende Kommando-Variante zur Verfügung. Selbiges gilt für das Beenden von Simulationen. Die meisten Kommandos sind generell nicht-blockierend, so dass die Auswertung von Simulationen selbst bei Bedarf unmittelbarer Sensordaten teilweise parallel stattfinden werden kann.

| **Kommando-Typ** | **Parameter** | **Optional** | **Antwort-Typ** |
|---|---|---|---|
| `StartSimulation` | Systembefehl | — | `ACK/ERR` |
| `StopSimulation` | simID | — | `ACK/ERR` |
| `StopSimulationNow` | simID | aI | `ACK/ERR` |
| `Step` | simID | #Schritte | `ACK/ERR` |
| `SetAction` | simID,Aktoren | — | `ACK/ERR` |
| `GetSensors` | simID | mID,aI | `ACK/ERR` |
| `GetSensorsNow` | simID | — | `ACKGetSensors/ERR` |

**Tabelle 1:** Übersicht über die Kommando-Typen, die der SimulationManager verarbeitet. Die meisten Befehle benötigen lediglich die Simulations-ID (simID). Bei Durchführung eines Simulations-Schritts (`step`) kann zusätzlich die maximale Anzahl der Schritte angegeben werden, wobei die Simulation selbst entscheidet, wann diese Anzahl erreicht ist. Zu beachten ist, dass die Felder `messageID` (mID) und `additionalInfo` (aI) optional angegeben werden können, sofern es sich um einen nicht-blockierenden Befehl handelt, der eine verzögerte Antwort liefert.

| **Kommando-Antwort-Typ** | **Rückgabe** |
|---|---|
| `ACK` | simID, mID, aI |
| `ACKGetSensors` | simID, mID, aI, Sensoren |
| `ERR` | simID, mID, aI, Fehlermeldung |

**Tabelle 2:** Übersicht über die Kommando-Antworten des SimulationManagers. Optional gegebene Informationen werden immer zurückgeliefert, falls diese gegeben wurden, andernfalls wird der Standardwert 0 benutzt.

### Simulations-Kommandos

- `StartSimulation` **(nb)**: Startet eine Simulation, die das vom Manager vorgegebene Interface implementiert (siehe Kapitel 1.6), als eigenständigen Prozess. Es muss der Systembefehl übergeben werden, der die Simulation startet (inklusive Pfad relativ zur Executable des Managers), welcher im Körper des Kommandos abgelegt wird. Bei Erfolg liefert der Manager ein Kommando des Typs ACK zurück, das insbesondere die ID der soeben gestarteten Simulation beinhaltet. Die Simulations-ID wird benötigt, um die Simulation subsequent ansprechen zu können.

- `StopSimulation` **(nb)**: Merkt eine Simulation zur Beendung vor. Die Simulation wird erst beendet, sobald alle zuvor übermittelten Kommandos abgearbeitet wurden. Hierbei handelt es sich um die nicht-blockierende Variante des Kommandos. Es ist zu verwenden, wenn mehrere Simulationsdurchläufe im Hintergrund parallelisiert werden sollen.

- `StopSimulationNow` **(b)**: Soll eine Simulation blockierend beendet

oder nach Ablauf einer Frist abgebrochen werden (auch wenn noch nicht alle Befehle abgearbeitet wurden), eignet sich dieser Befehl. Im Parameter additionalInfo (aI) wird dazu die Anzahl der Millisekunden angegeben, nach der die Beendung einer Simulation erzwungen wird. In der Kommando-Antwort beinhaltet selbiger Parameter die Anzahl der Millisekunden, die anschließend von der Frist verblieben sind (¿0, wenn die Simulation in dieser Zeit regulär beendet wurde). Durch Weiterreichen dieser verbleibenden Zeit an das nächste Kommando desselben Typs ist es einem Plugin möglich, ausgewählte parallele Simulationen innerhalb einer Gesamtfrist zu beenden. Eine fristlose Beendung wird dagegen mittels aI=0 gewählt.

Ein Plugin kann dieses Kommando beispielsweise nutzen, um mit der Auswertung von Simulationsdaten zu warten, bis die Simulation auf alle gepufferten Sensoranfragen geantwortet hat und beendet wird. Ebenfalls sichert dieses Kommando gegen Abstürze von Simulationen ab. Die Vorgehensweise ähnelt der eines Prozessmanagers: Alle gestarteten Simulationen werden anhand ihrer Programm-ID (PID) und ihrer Prozesskennung vermerkt. Wird eine Simulation innerhalb der Frist regulär beendet, wird diese nicht weiter kontrolliert. Wenn die Frist der zu beendenden Simulation jedoch abgelaufen ist, und ein im System unter der PID vermerkter Prozess existiert und der Prozesskennung entspricht, wird dieser per Signal `SIGKILL` beendet. Wurde aufgrund eines frühen Absturzes der Simulation keine PID zurückgeliefert, wird die Simulation als beendet betrachtet und nicht weiter kontrolliert.

- `Step` **(nb)**: Führt einen oder mehrere Simulationsschritte durch. Die gewählte Simulation bekommt dabei optional die gewünschte Anzahl der Schritte (Default: 1) aus dem Parameter additionalInfo (aI) übergeben und kann diesen nach eigenem Ermessen interpretieren. Beispielsweise kann so lange simuliert werden, bis die zuletzt übergebene Aktion nach einem bestimmten Kriterium ausgeführt wurde, oder maximal aI Schritte durchgeführt wurden.

- `SetAction` **(nb)**: Löst das Setzen von Aktoren auf Simulationsseite aus. Der Körper des Kommandos wird auf einen Datenstrom gesetzt der die Aktorwerte repräsentiert. Dieser Datenstrom kann von beliebiger Länge sein, muss jedoch vom Typ `StreamableDataContainer` oder einem davon abgeleiteten Typ sein. Die Konvention dieses Datenstroms muss in jedem Fall sowohl dem Erzeuger der Aktor-Daten (das heißt i.d.R. dem Plugin) als auch der angesprochenen Simulation bekannt sein.

- `GetSensorsNow` **(b)**: Fordert die Simulation auf, nach Abarbeitung aller zuvor gepufferten Kommandos ihre Sensoren auszulesen und zurückzugeben. Der Sender dieses Kommandos wird so lange

blockiert, bis die Sensordaten bereitstehen. Da der Kommando-Puffer der Simulation hierzu eventuell erst geleert werden muss, kann dies eine nennenswerte Zeit in Anspruch nehmen. Es ist jedoch zum Beispiel dann unumgänglich, wenn die nächste Aktion von den aktuellen Sensordaten abhängig ist. Der Sender erhält die Sensordaten im Körper der Kommando-Antwort vom Typ `ACKGetSensors`, welche ihm als direkte Rückgabe der Funktion `SimulationManager::applyCommand` übermittelt wird. Der Körper ist ferner vom Typ `StreamableDataContainer` oder einem Subtyp. Wie im Fall von `SetAction` müssen sowohl dem Verwender der Sensor-Daten (das heißt i.d.R. dem Plugin) als auch der Simulation die Konvention der Daten bekannt sein.

- `GetSensors` **(nb)**: Dies ist die nicht-blockierende Variante des Sensor-Kommandos und liefert in der Folge eine verzögerte Antwort an das Plugin. Das nicht-blockierende Auslesen von Sensordaten kann zum Beispiel nützlich sein, wenn keine unmittelbaren Sensorrückmeldungen nötig sind, um die nächste Aktion zu generieren. Zur Nutzung dieser Funktion muss das Plugin oder eine ihm bekannte Klasse das Interface `SensorHandlerInterface` implementieren. Dieses schreibt eine Funktion vor, die der Manager aufrufen kann um die Sensordaten zu übergeben, sobald diese bereitstehen. Da der Aufruf aus einem anderen Thread erfolgt, muss diese Funktion auf Threadsicherheit achten. Zudem muss dem Manager zuvor per `SimulationManager::setSensorHandler( SensorHandlerInterface* set_sensorHandler, unsigned int simID)` mindestens einmalig mitgeteilt werden, an welchen Handler er die verzögerten Sensordaten der jeweiligen Simulation versenden soll. Zu beachten ist, dass dabei keine 1:1 Zuordnung zwischen `GetSensors`-Kommandos und Sensor-Handlern stattfindet. Die verzögerte Antwort ist vom Typ `ACKGetSensors` und enthält die Daten im Körper.

**Kommando-Antworten**

- `ACK`: Generelle Kommando-Antwort bei erfolgreicher Übermittlung/Ausführung eines Kommandos.
- `ERR`: Generelle Kommando-Antwort bei fehlgeschlagener Übermittelung/Ausführung eines Kommandos. Diese Antwort verweist auf eine Fehler-Nachricht.
- `ACKGetSensors`: Antwort auf die Kommandos `GetSensors` bzw. `GetSensorsNow`. Diese Kommando-Antwort enthält im Körper die Sensordaten einer Simulation und wird entweder direkt oder verzögert zurückgegeben.

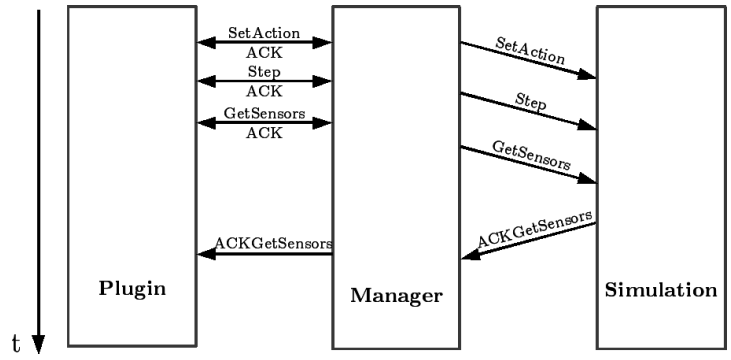



**Abbildung 4:** Beispielhafte Ansteuerung einer Simulation über den SimulationManager anhand nicht-blockierender Kommandos. Vertikal ist die Zeit-Achse aufgetragen.

Der SimulationManager stellt über die obige Kommandostruktur eine einfach zu handhabende Schnittstelle bereit, die die Ansteuerung und Auswertung von Simulationen jeder Art vereinheitlicht und somit vielfältig Anwendung finden kann. Selbstredend verbirgt sich dahinter ein aufwändiges System aus Parallelisierung, Pufferung, Prozess-Management und IPC. In den folgenden Kapiteln werden daher diese Kernpunkte des Frameworks näher ausgeführt.

## 1.4 SimulationManager

Die Klasse `SimulationManager` stellt das Kernstück der hier vorgestellten Architektur dar. Ihr obliegt die Aufgabe, Plugins und Simulationen zu parallelisieren und einen kontrollierten Ablauf derselben gewährleisten. Hierzu müssen alle gestarteten Plugins und Simulationen in eine Verwaltungsstruktur integriert werden. Der Manager muss weiterhin die verzögerungsfreie Annahme von Kommandos sicherstellen, was eine Verwendung von Puffern bedingt. Die Methodologie dieser Punkte wird in den folgenden Kapiteln dargelegt.

Ferner sei angemerkt, dass eine einzelne Instanz des SimulationManagers bei der Ausführung automatisch initialisiert wird. Da dieses Manager-Objekt sowohl mehrere Simulationen als auch mehrere Plugins handhaben kann, besteht keine Notwendigkeit, weitere Instanzen zu erzeugen. Um die Prinzipien der Objektorientierung zu wahren, bleibt diese Möglichkeit jedoch offen, weswegen der Manager nicht als Singleton [2] implementiert ist, und keine statischen oder globalen Funktionen oder Variablen erstellt.

### 1.4.1 Manager-Schnittstelle

Wie angedeutet verfügt der SimulationManager nur über wenige öffentliche Funktionen, da jegliche Interaktion mit Simulationen den oben erklärten Simulations-Kommandos vorbehalten ist, welche von der Funktion `applyCommand` verarbeitet werden. Alle weiteren öffentlichen Funktionen erfüllen daher das Kriterium, dass sie die Ausführung von Code auf Simulationsseite weder direkt bewirken noch beeinflussen. Im Folgenden werden die öffentlichen Funktionen des Managers aufgeschlüsselt:

- `SimulationCommand applyCommand(SimulationCommand* cmd)`: Diese Funktion führt ein Kommando wie zuvor angegeben aus. Sie überprüft dazu dessen Gültigkeit und klassifiziert es zunächst: Nicht-blockierende Kommandos werden zur späteren Verarbeitung an den Kommando-Puffer der betreffenden Simulation gehängt, welcher näher in Kapitel 1.4.2 beschrieben wird. Die Funktion kehrt umgehend zurück. Bei blockierenden Kommandos kehrt die Funktion erst nach vollständiger Abarbeitung des Kommandos zurück. Die Abarbeitung zuvor übermittelter Kommandos der betreffenden Simulation wird dabei vorgezogen. In beiden Fällen gibt `applyCommand` eine Kommando-Antwort zurück, die entweder das Kommando bestätigt, oder einen adäquaten Fehlertyp angibt.

- `PluginInterface* loadPlugin(const char* soname)`: Diese Funktion läd ein Shared Object, das die oben angegebenen Plugin-Kriterien erfüllt. Der Name des Plugins muss als Dateiname inklusive Pfad relativ zur Executable des Managers angegeben werden. Wenn das Plugin erfolgreich geladen wurde, wird es einer Liste kontrollierter Plugins hinzugefügt. Die Liste ist als assoziatives Mapping zwischen Zeigern auf Plugins und Shared-Object-Handles realisiert. Kann ein Plugin nicht geladen werden, wird `NULL` zurückgegeben und eine Fehlermeldung ausgegeben, andernfalls enthält die Rückgabe einen Pointer auf das geladene Plugin.

- `int unloadPlugin(PluginInterface* plugin)`: Diese Funktion erzwingt das Beenden der Ausführung eines Plugins (per `PluginInterface::exit()`), destruiert das Plugin und dekrementiert den zugehörigen Shared-Object-Handle. Falls das übergebene Plugin nicht vom Manager kontrolliert wurde, wird ein Fehler zurückgegeben.

- `void setSensorHandler(SensorHandlerInterface* set_sensorHandler, unsigned int simID)`: Wie zuvor beschrieben dient diese Funktion der Angabe eines Handlers pro Simulation, der verzögert eintreffende Sensordaten entgegennimmt und verarbeitet.

- Statistische Funktionen: Alle statistischen Funktionen befinden sich im öffentlichen Unterobjekt `SimulationManager::stats`, beispielsweise:
  - `const char* stats::version()`: Versionsnummer des SimulationManagers.
  - `int stats::numSlots()`: Maximale Anzahl Simulationen.
  - `int stats::numBusySlots()`: Anzahl aktiver Simulationen.
  - `double stats::getMemFreeMB()`: Freier Speicher des Systems in MB.
  - `double stats::currentLoad()`: Angabe der aktuellen Systemlast (current load nach BSD-Standard [3]).
  - `double stats::averageLoad(int numBusySlots)`: Angabe der mittleren Systemlast (average load nach BSD-Standard [3]).

### 1.4.2 Parallele Simulationskontrolle

Die mechanisierte Parallelisierung sequenziell angesteuerter Simulationen basiert auf der Einsortierung nicht-blockierender Kommandos in parallel abgearbeitete Kommando-Puffer. Jede Simulation wird auf Manager-Seite durch eine Kontrollstruktur, bestehend aus einem solchen Kommando-Puffer (double-ended queue nach C++11-Standard [4]), einem Interface zur bidirektionalen IPC (mittels FIFO-Pipes bzw. named pipes nach Linux-Standard [5]) sowie einem Schreib- und einem Lese-Thread (Native POSIX Thread Library [6]), repräsentiert und verwaltet.

Beim Starten einer Simulation werden von der Kontrollstruktur zwei Pipes angelegt: Eine Pipe, die von dieser beschrieben und vom Simulationsprozess ausgelesen werden wird, und eine Pipe in die entgegengesetzte Richtung. Dann wird die Simulation gestartet, wobei ihm die Dateinamen der Pipes per Kommandozeilenparameter mitgeteilt werden. Es besteht fortan die Möglichkeit der bidirektionalen Kommunikation zwischen den beiden Prozessen.

Während im weiteren Verlauf die Funktion `SimulationManager::applyCommand` die Kommando-Puffer der Simulations-Kontrollen befüllt, ist es Aufgabe des Schreib-Threads (`QueueWriter`), diesen zu entleeren, bzw. die dort hinterlegten Kommandos zu entpacken und die benötigten Daten auf die an die Simulation gerichtete Pipe zu schreiben. Dieser Prozess wird näher in Kapitel 1.5 beschrieben. Analog liest der Lese-Thread die an die Kontrollstruktur gerichtete Pipe aus um eventuelle Antworten der Simulation entgegenzunehmen. Je nach verursachendem Kommando werden die entsprechenden Daten dann entweder an einen Handler weitergereicht, zur direkten Antwort an die Manager-Funktion `applyCommand` übergeben, oder von der Kontrollstruktur selbst verarbeitet.

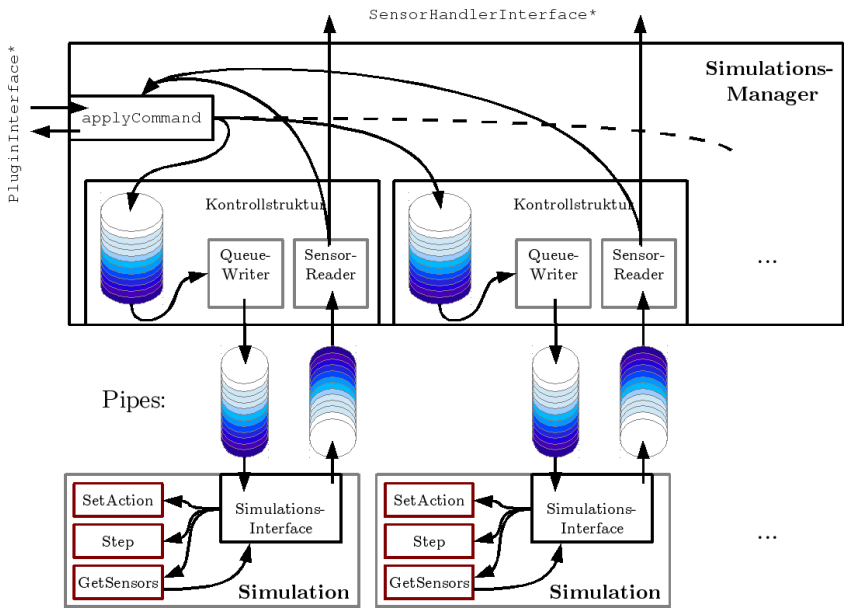



**Abbildung 5:** Diese Abbildung zeigt die Verarbeitungswege von Kommandos durch parallele Simulations-Kontrollstrukturen. Kommandos werden generell von der Manager-Funktion `applyCommand` angenommen. Anschließend werden diese einer Simulation zugewiesen und gepuffert (breitere blaue Zylinder). Pro Simulation arbeiten zwei Threads (in grau dargestellt): Der eine legt die gepufferten Kommandos auf die Pipe zur Simulation, welche für sich wiederum als Puffer angesehen werden kann (schmalere blaue Zylinder), der zweite liest die an den Manager gerichtete Pipe aus und handhabt die Antworten entsprechend. Letzterer benötigt Kenntnis über einen Sensor-Handler zur Abarbeitung nicht-blockierender Sensoranfragen. In rot sind die zu implementierenden Funktionen der Simulations-Schnittstelle gezeigt.

Jede Simulation wird also allein auf Manager-Seite von mindestens drei Threads verwaltet: Dem Plugin-Thread, welcher den Manager bedient, sowie dem Schreib- und dem Lese-Thread der Kontrollstruktur. Die durch diese Herangehensweise bedingte Threadkommunikation wird wie die Pufferkontrolle (Vermeidung von overflow/underrun) soweit möglich über POSIX mutex- und condition-locks gelöst, so dass unnötige Idle-Zyklen der CPU vermieden werden. Da die Simulation zusätzlich als Prozess ausgeführt wird, sind insgesamt vier parallele Strukturen am Management einer Simulation beteiligt.

Die Simulations-Kontrollstruktur beinhaltet ferner alle simulationsbezogenen, auf Manager-Seite benötigten und von dort aus kontrollierbaren Informationen (SimID, PID, Kommandozeilenparameter, Informationen über die Beendung oder Beendungsanforderung der Simulation, etc.).

## 1.5 Interprozesskommunikation

### 1.5.1 FIFO-Piping und IPC-Diskurs

Der SimulationManager verwendet für die IPC Pipes nach dem FIFO-Prinzip (First-In First-Out). Pipes werden in Unix-basierten Systemen (nach Douglas McIlroy, der die Grundidee schon 1964 beschrieb [7]) als spezielle Dateien mit einem Schreibanfang und einem Leseende repräsentiert. Anders als bei normalen Dateien bedingt ein Zugriff auf eine Pipe keinerlei Operationen auf dem zugrundeliegenden Festspeicher, da ihr Inhalt im Betriebssystem-Adressraum liegt. Am Ende der Pipe abgelesene Daten fallen außerdem aus dem Inhalt der Pipe heraus.

Pipes weisen damit ebenso wie Dateien eine (per Dateideskriptor einstellbare) Kapazität auf und eignen sich daher wie der zuvor dargestellte Kommando-Puffer zum Zwischenspeichern von Daten. Für die doppelte Pufferung von Kommandos sprechen dennoch Gründe: Zum einen werden auf diese Weise gleichzeitige Zugriffe auf kommunizierte Daten vermieden, falls der Pipe-Puffer mehr als ein Kommando enthält, was den Prozesssynchronisierungsaufwand verringert (Vergleich siehe Watson [8]). Dadurch kann eine eine kontinuierlichere Kommando-Frequenz gewährleistet werden. Zum anderen wird dadurch die Abhängigkeit der Kommandolatenz von der Auslastung des Betriebssystem-Schedulers verringert.

Plugins sollen bei Übergabe eines Kommandos zur Erhöhung der Parallelität möglichst kurz blockiert werden. Zur Verringerung des Kommunikationsoverheads werden daher niemals die gesamten Kommandos auf die Pipe geschrieben, sondern nur die benötigten Nutzdaten der jeweiligen Kommandos. Auch lässt sich der Manager dadurch leicht - falls vorteilhaft - auf eine alternative, nicht-puffernde IPC-Methode adaptieren.

Als solche Alternative ist vor Allem das schnellere Shared Memory [9] in Betracht zu ziehen. Indessen spricht für die Verwendung von Pipes, dass die-

se im Gegensatz zu Shared Memory sämtliche Prozesssynchronisations- und Puffermechanismen übernehmen, was die IPC-Implementierung vereinfacht. Eine äquivalente Pufferung mittels Shared Memory zu realisieren könnte sich derweil als schwer herausstellen, da Shared Memory Blöcke immer mit einer vorab bekannten Größe initialisiert werden müssen. Sowohl der oben aufgeführte Kommando-Datentyp als auch die Kommando-Puffer sind hingegen von dynamischer und vorab unbekannter Größe. Implementierungen dynamischen Shared Memorys sind zudem eher als experimentell einzustufen und im C++-Standard nicht vorgesehen.

Erwägenswert ist auch, dass Pipes keinen direkt gemeinsamen Speicherbereich zur IPC benötigen. So ist es möglich, den hier vorgestellten Manager ohne wesentliche Erweiterungen auf verteilten Systemen einzusetzen: Einige Implementierungen des Server Message Block Kommunikationsprotokolls (SMB [10]) - so zum Beispiel Samba [11] - unterstützen das Piping zwischen im lokalen Netzwerk verteilten Prozessen [12]. Auf diese Weise könnten Simulationen auf entfernten Systemen ausgeführt werden, während der SimulationManager diese von einem zentralen Kontrollrechner aus anspricht. In Verbindung mit netzwerkfähigen Plugins ergäbe sich die Möglichkeit, sämtliche Komponenten des Frameworks zu dezentralisieren.

### 1.5.2 Kommando-Codierung

Programmiertechnisch wurde das Piping anhand einer Klasse (`PipeRW`) umgesetzt, welche sowohl in der Simulationskontrolle auf Manager-Seite als auch auf Simulationsseite zur Anwendung kommt. Sie codiert die Daten minimalistisch und garantiert zudem die Atomizität von Schreib- und Leseoperationen auf den Pipes. Bei Letzterem ist zu beachten, dass Pipes sowohl über einen Schreib-, als auch über einen Lesepuffer verfügen. Diese sind unabhängig vom eigentlichen Inhalt der Pipe, welcher im Adressraum des Kernels liegt. Der Schreibpuffer einer Pipe wird (bis auf vernachlässigbare Ausnahmen) dann ausgeschrieben („geflusht“), sobald dieser die maximale Puffergröße erreicht. Um also auf der Gegenseite der Pipe das Auslesen von unvollständig geschriebenen Daten zu vermeiden, muss der Schreibpuffer mindestens von der Größe der zu schreibenden Daten sein, was über die Modifikation des entsprechenden Dateideskiptors erreicht werden kann. Durch manuelles Flushen nach dem Befüllen des Schreibpuffers der Pipe wird dann ein atomares Ausschreiben der Daten garantiert. Zur Beschleunigung des Pipings kann das Flushing in Abhängigkeit vom Zustand des zuvor beschriebenen Kommando-Puffers sowie des Pipe-Inhalts in unregelmäßigen Abständen erfolgen.

Ein atomarer Schreibbefehl (im Folgenden „Block“) besteht bei `PipeRW` allgemein aus Angabe der Anzahl Bytes der nachfolgend gesendeten Daten (codiert als `unsigned int64`), sowie der Daten selbst (n `chars`). In der Folge wird von der Pipe immer zuerst die Länge abgelesen, anschließend Daten von

entsprechender Länge. Bei Übermittlung der Codierung eines Kommandos oder einer Kommando-Antwort wird als erster Block („Block 1“) immer der Kommando- bzw. Antwort-Typ gesendet, anschließend eventuell benötigte Daten („Block 2“). Die Übermittlung eines Kommandos besteht daher immer mindestens aus einem Block.

Tabelle 3 gibt eine Übersicht über die beim jeweiligen Kommando gesendeten Daten:

| **Kommando** | **Block 2** | **Kommando-Antwort** | **Block 2** |
|---|---|---|---|
| `StartSimulation` | — | `StartSimulation` | PID |
| `StopSimulation` | — | `StopSimulation` | — |
| `Step` | #Schritte | — | — |
| `SetAction` | Aktoren | — | — |
| `GetSensors` | — | `ACKGetSensors` | Sensoren |

**Tabelle 3:** Die blockierenden Kommandos entsprechen bei der IPC ihren nichtblockierenden Derivaten und werden nur vom Manager anders gehandhabt. Das Kommando `StopSimulation` wird dem Manager quittiert um diesem den Zeitpunkt für die Beendung der Simulations-Kontrolle anzuzeigen. Diverse Kommandos benötigen keine Antwort.

## 1.6 Simulationen

Nach Beschreibung aller Komponenten von den Plugins bis zur IPC verbleibt einzig die Erklärung der allgemeinen Programmierschnittstelle für Simulationen, welche diesem Kapitel erfolgt.

### 1.6.1 Verwendung serieller Datentypen

Die Übertragung von Simulations-Kommandos, Sensor- und Aktordaten zwischen Prozessen bedingt entweder eine Serialisierung der kommunizierten Daten, oder von vornherein seriell erzeugte Daten. Da einerseits die Kommunikation mit Simulationen generell über IPC realisiert ist, und Simulations-Kommandos andererseits über das Netzwerk übertragbar sein sollen, wurde sich für letzteres Mittel entschieden, um den bei regelmäßiger Serialisierung anfallenden Overhead zu vermeiden. Zudem begünstigt dies die Auslesegeschwindigkeit jener Daten durch Plugins, den Manager und Simulationen, da die Lokalitätseigenschaften serieller Daten die Effizienz des Prozessorcachings stark begünstigen.

Die konsequente Serialität von Datenstrukturen ist jedoch im Allgemeinen nicht gegeben: In der Regel werden Datenelemente an ganzzahlige Vielfache der Wortbreite der Prozessorarchitektur (auch: Busbreite [13]) adressiert. Da Leseoperationen des Prozessors immer mindestens ein ganzes Wort lesen, wird so garantiert, dass der Prozessor zum Auslesen eines Datums

die minimale Anzahl Wörter liest. Diese Speicherausrichtung (auch „Data Structure Alignment“ [14]) hat also zur Folge, dass auf Datenelemente, deren Bitbreite kein ganzzahliges Vielfaches der Wortbreite ist, ungenutzte Füllbytes folgen (sog. „pad-bytes“).

Hat ein Datentyp zum Beispiel 10 Bytes, und wird dieser auf einem 64-Bit System an der Wortbreite ausgerichtet („aligned“), dann werden zur Rekonstruktion des Datums 2 Wörter benötigt, und es befinden sich darauffolgend 6 Füllbytes im Speicher. Ist dieser jedoch nicht ausgerichtet („misaligned“), so müssen eventuell drei Wörter gelesen werden, und das Ergebnis muss anhand weiterer Routinen zusammengesetzt werden, die je nach Hardware eventuell nicht (wie bei den meisten RISC-Prozessoren [15]) oder nicht beschleunigt (bei x86 Architektur nur ohne SSE2 [16]) unterstützt werden. Daher ist die Speicherausrichtung generell wichtig für die Performanz der Datenauswertung.

Auf der anderen Seite können nicht-ausgerichtete serielle Daten jedoch so gepackt werden, dass sie keine Füllbytes aufweisen, indem diese an jedem Vielfachen *eines* Bytes anstelle der Wortbreite ausgerichtet werden. Da die IPC bei hohem Overhead einen Flaschenhals bilden kann, kann es also sinnvoll sein, Daten nicht ausgerichtet zu übertragen. Um diese Für und Wider zu beachten, bietet der SimulationManager die Möglichkeit, serielle Daten entweder mit oder ohne Padding zu generieren. Dies erfolgt anhand der Definition `ALIGN_DATA` in der Header-Datei `SimulationData.h`. Zudem können die seriellen Datentypen auch manuell ausgerichtet werden, was insbesondere vonnöten sein kann, falls diese zwischen Architekturen mit verschiedenen Wortbreiten ausgetauscht werden müssen: In diesem Fall sind das größte gemeinsame Vielfache der Wortbreiten sowie Datentypen mit definierter Breite (z.B. `int64`) zu wählen.

Das SimulationManager-Framework stellt für die seriellen Daten und deren Kommunikation eine Klasse (`StreamableDataContainer`) bereit, die allgemeine Funktionen für das Speichermanagement, das Streaming und die Interpretation serieller Datentypen beinhaltet, ohne Kenntnisse über die verwalteten Daten oder die Implementierung weiterer Funktionen zu fordern. Datenstrukturen sind in C++ dann seriell, wenn sie ausschließlich aus fundamentalen, nicht-verweisenden Elementen (also auch Feldern) bestehen. Sowohl Kopf und Körper von Simulations-Kommandos, als auch jeglichen Sensor- und Aktordaten wird die Verwendung dieser Klasse vorgeschrieben, wobei das Datenformat bis auf den Kommando-Header frei gewählt werden kann. Das bedeutet, dass jede gewöhnliche Struktur (C++ `struct`) ohne Pointer direkt als Datenstrom für die Aktorik oder Sensorik verwendet werden kann. Simulationen und Plugins müssen dazu die selbe Konvention für die Interpretation der Sensor-und Aktordaten verwenden. Die Interpretation wird über Template-Typisierung eines Datenstroms mittels der Funktion `template <typename T> T* StreamableDataContainer ::dat()` erzielt, wobei `T` eine beliebige zusammenhängende Klasse von der

Größe des Stroms sein kann.

Da für jede serielle Datenstruktur ein Speicherbereich von fester Größe vor der Belegung alloziert werden muss, ist die effiziente Implementierung dynamischer serieller Datentypen nicht möglich. Dennoch stellt die Container-Klasse `StreamableDataContainer` zur Laufzeit Funktionen für die implizite und explizite Rekonfiguration des verwalteten Speichers bereit, so dass Daten theoretisch nicht nur mit variablem Typ, sondern auch mit variabler Größe kommuniziert werden können. Es ist jedoch in gewissen Fällen nicht trivial, eine Neuabsprache der Interpretationskonvention für dynamische Daten zur Laufzeit zu ermöglichen.

Ist es unumgänglich, nicht-zusammenhängende oder komplexe dynamische Sensor- oder Aktorstrukturen zu kommunizieren, bietet sich eine Serialisierung der Daten mittels `boost` an. Boost-Archive können problemlos mittels des Managers verschickt und auf der Empfängerseite entpackt werden.

### 1.6.2 Simulationsparallelität auf Prozessebene

Wie beschrieben werden vom SimulationManager kontrollierte Simulationen als eigenständige Prozesse ausgeführt, was die Verwendung von IPC unvermeidlich macht. Dieser Schritt ist in Anbetracht der zuvor gestellten Ansprüche allgemein vonnöten, wenn die Simulationen nicht im selben Prozessraum parallelisierbar sind, oder aber eine verteilte Ausführung von Simulationen gewünscht ist.

Überdies kann die Ausführung von Simulationen als eigenständige Prozesse zweckdienlich sein, da ein Nachladen (verschiedenartiger) Simulationen zur Laufzeit zu Symbolkollisionen führen könnte oder die Prozessstabilität durch Exceptions oder Speicherlecks insgesamt beeinträchtigt werden könnte. Sie ermöglicht außerdem theoretisch das Verwalten von Simulationen anderer Programmiersprachen, sofern diese ein äquivalentes Kommunikationsinterface implementieren.

### 1.6.3 Simulations-Programmierschnittstelle

Jede Simulation, die vom Manager kontrolliert werden soll, muss die Schnittstelle `PipeClientSimulation` implementieren, welche im Wesentlichen die IPC-Methoden der Klasse `PipeRW` erbt und die Implementierung der auch auf Kontrollseite verwendeten Schnittstelle `Simulation` vorschreibt.

Der Konstruktor der Schnittstelle fordert die Übergabe aller beim Start der Simulation übergebenen Kommandozeilenparameter `char** argv` sowie deren Anzahl `unsigned int argc`. Diese müssen also gemäß Abbildung 6 an das Simulations-Interface weitergereicht werden. Sie werden für den Aufbau der IPC benötigt, da der Manager beim Ausführen der Simulation die Namen der zu verwendenden Pipes an die Kommandozeile anhängt. Das vollständige

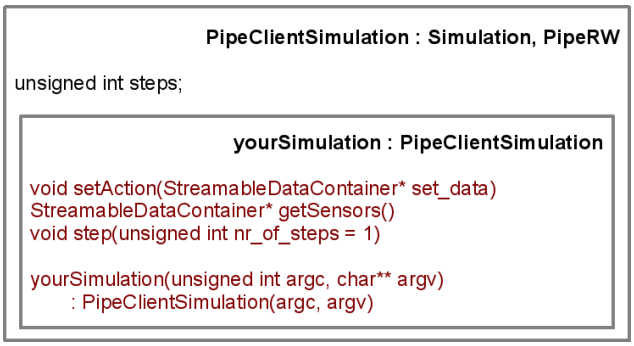



**Abbildung 6:** Die Programmierschnittstelle für Simulationen sieht die Implementierung der Funktionen `setAction`, `getSensors` und `step` vor. Die Kommandozeilenargumente müssen beim Start an den Konstruktor der Schnittstelle weitergereicht werden.

Kommandozeilenformat lautet daher

```
<Executable> [Parameter] <Pipe zum Manager> <Pipe zur
Simulation> &
```

,

wobei die Ausführung als Hintergrundprozess mittels „`&`“ ebenso vom Manager angehängt wird. Die optionalen Simulations-Parameter (`[Parameter]`) werden im Startkommando der Simulation definiert und können von dieser frei interpretiert werden.

Nach dem Starten einer Simulation liest obige Schnittstelle kontinuierlich Kommandos von der Pipe ab und ruft die entsprechenden Implementierungen der Simulation direkt auf. Es ist dazu kein Threading auf Simulationsseite von Vorteil. Diese von der Simulation zu implementierenden Funktionen umfassen:

- `void Simulation::setAction(StreamableDataContainer* set_data)`: Konvertiert den übergebenen Aktor-Datenstrom in ein zuvor definiertes, simulationsspezifisches Format, und setzt die Aktoren der Simulation entsprechend der gestellten Anforderungen. Die Interpretation des Datenstroms wird über Template-Typisierung (bzw. pointer typecasting) vorgenommen, wie in Kapitel 1.6.1 beschrieben wurde.

- `void Simulation::step(unsigned int nr_of_steps = 1)`: Führt einen Simulationsschritt aus. Der Parameter `nr_of_steps`, der optional mit dem auslösenden `step`-Kommando übergeben wurde, kann frei interpretiert werden, also zum Beispiel als minimale oder maximale Anzahl Schritte. Je nach Ermessen können so lange

Simulationsschritte berechnet werden, bis die Simulation einen bestimmten Zustand erreicht hat. Es ist an dieser Stelle auch möglich, die Funktion `Simulation::getSensors` dazu zu verwenden.

- `StreamableDataContainer* Simulation::getSensors()`:
  Erzeugt ein zuvor definiertes und simulationsspezifisches, von `StreamableDataContainer` abgeleitetes Objekt, und gibt einen Zeiger auf selbiges zurück. Gemäß Kapitel 1.6.1 kann es alle benötigten Sensorwerte enthalten. Eine Übereinstimmung von Aktor- und Sensor-Datentyp kann vorteilhaft sein, sollten sich die enthaltenen Daten überschneiden.

# Tabellenverzeichnis



# Abbildungsverzeichnis



# Literatur

[1] LD.SO(8) - Linux Programmer's Manual. `http://man7.org/linux/man-pages/man8/ld.so.8.html`, 21. November 2013.

[2] Singleton (Entwurfsmuster) - Wikipedia. `http://de.wikipedia.org/wiki/Singleton_%28Entwurfsmuster%29`, 21. November 2013.

[3] getloadavg(3) - Linux man page. `http://linux.die.net/man/3/getloadavg`, 21. November 2013.

[4] std::deque - cppreference.com. `http://en.cppreference.com/w/cpp/container/deque`, 21. November 2013.

[5] PIPE(7) - Linux Programmer's Manual. `http://man7.org/linux/man-pages/man7/pipe.7.html`, 21. November 2013.

[6] PTHREADS(7) - Linux Programmer's Manual. `http://man7.org/linux/man-pages/man7/pthreads.7.html`, 21. November 2013.

[7] M Douglas McIlroy. Pipes and filters. *Internal Bell Labs memo, original title lost*, 11, 1964.

[8] WJ Watson. The ti asc: a highly modular and flexible super computer architecture. In *Proceedings of the December 5-7, 1972, fall joint computer conference, part I*, pages 221–228. ACM, 1972.

[9] Shared Memory - Wikipedia. `http://de.wikipedia.org/wiki/Shared_Memory`, 21. November 2013.

[10] Server Message Block - Wikipedia. `http://de.wikipedia.org/wiki/Server_Message_Block`, 21. November 2013.

[11] Samba. `http://www.samba.org/`, 21. November 2013.

[12] Jay Ts, Robert Eckstein, and David Collier-Brown. *Using Samba.* O'Reilly, 2003.

[13] Datenwort - Wikipedia. `http://de.wikipedia.org/wiki/Datenwort`, 21. November 2013.

[14] Data structure alignment - Wikipedia. `http://en.wikipedia.org/wiki/Data_structure_alignment`, 21. November 2013.

[15] Data structure alignment - Wikipedia. `http://en.wikipedia.org/wiki/Data_structure_alignment#RISC`, 21. November 2013.

[16] Data structure alignment - Wikipedia. `http://en.wikipedia.org/wiki/Data_structure_alignment#x86`, 21. November 2013.

# GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007



**Preamble**

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

## Terms and Conditions

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

   The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   (a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

   (b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

   (c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless

of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

(d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

(a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

(b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

(c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

(d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

(e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install

and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part

of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

(a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

(b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

(c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

(d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

(e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

(f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

   You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express

agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

    The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing

courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

## End of Terms and Conditions

### How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <textyear>  <name of author>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program>  Copyright (C) <year>  <name of author>
```

```
This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `http://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `http://www.gnu.org/philosophy/why-not-lgpl.html`.