HTBLuVA Wiener Neustadt
Höhere Lehranstalt für Informatik

AUSBESSERUNGSARBEIT

Simple Raft-Simulator

Ausgeführt im Schuljahr 2019/20 von:

Recherche, Programmierung, Testing
Matthias Guzmits 5AHIF

Betreuer / Betreuerin:

none

Wiener Neustadt, am 24. April 2019/20

Abgabevermerk: Übernommen von:

Eidestattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die im Literaturverzeichnis angegeben Quellen und Hilfsmittel verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Wiener Neustadt am 24. April 2019/20

Verfasser / Verfasserinnen:

Matthias Guzmits

Inhaltsverzeichnis

Eidestattliche Erklärung **i**

1 Introduction **1**

2 Installation von Sobjectizer **2**
2.1 Kompilieren von Sobjectizer 2
2.2 Setup zum Verwenden des Projekts 3

3 Benutzte Sobjektizer-Funktionen **4**
3.1 Funktionen 4
3.1.1 so_5::launch(*[](so_5::environment_t&){}*) 4
3.1.2 introduce_coop(*[](so_5::coop_t&){}*) 4
3.1.3 make_agent<*Klasse*>(*Paramliste des Konstruktors der Klasse*) 4
3.1.4 so_direct_mbox() 5
3.1.5 so_5::send<*Klasse/Struct*>(so_5::mbox_t, *Paramliste des Konstruktors der Klasse*) 5
3.1.6 on_enter(*Funktion*) 5
3.1.7 on_exit(*Funktion*) 6
3.1.8 event(*Funktion*) 6
3.2 Typen/Klassen 7
3.2.1 so_5::environment_t 7
3.2.2 so_5::coop_t 7
3.2.3 so_5::mbox_t 7
3.2.4 so_5::agent_t 7
3.2.5 so_5::signal_t 7
3.2.6 state_t 7

Kapitel 1

Introduction

Author: Matthias Guzmits

Dieses Dokument beschäftigt sich im Allgemeinen mit dem Raft Consensus Algorithmus und seiner Implementation. Zusätzlich ist eine Beschreibung der Projektverwendung enthalten. Da die Installation von sobjectizer nicht trivial ist wird diese neben den verwendeten Funktionen ebenfalls beschrieben.

Kapitel 2

Installation von Sobjectizer

Sobjectizer ist ein Framework für C++ welches das Verwenden des Actor-Models vereinfachen soll. Das einbinden von diesem Framework in mein Projekt geschieht über eine Shared-Library. Um Sobjectizer jedoch verwenden zu können ist es erst einmmal notwendig die Libraries zu kompilieren.

2.1 Kompilieren von Sobjectizer

Für das kompilieren von Sobjectizer wird von den Herstellern Stiffstream CMake vorausgesetzt. Das builden sieht unter Linux folgendermaßen aus:

```
git clone https://github.com/stiffstream/sobjectizer
cd sobjectizer
mkdir cmake_build
cmake -DCMAKE_INSTALL_PREFIX=target -DCMAKE_BUILD_TYPE=Release ../dev
cmake --build . --config Release
cmake --build . --config Release --target install
```

Nun sollten sich in einem neu erstellten target-Folder zwei Ordner befinden. Einmal include und einmal lib. In dem lib-Folder befinden sich die kompilierte Static- und Dynamic-Library. Im include-Folder können alle notwendigen header-Files gefunden werden.

2.2 Setup zum Verwenden des Projekts

Um das Projekt verwenden zu können muss ersteinmal die richtige Arbeitsumgebung geschaffen werden. Im include-Verzeichnis des Projekts befindet sich eine vorkompilierte Version von Sobjectizer. Dafür muss folgendes gemacht werden:

- Kompilieren wie oben beschrieben oder vorkompiliertes Material verwenden
- Shared Library in /usr/lib verschieben
- Ordner so_5 in /usr/lib anlegen
- Den Ordner so_5 aus target/include in /usr/lib/so_5 verschieben

Meson kann leider mit der find_library-Funktion keine relativen Pfade verwenden weshalb dieser Umweg notwendig ist. Ein Pfad der bei jedem Linux-User gleich ist ist nunmal /usr/lib.

Kapitel 3

Benutzte Sobjektizer-Funktionen

SObjectizer besitzt an sich eigentlich eine Dokumentation. Leider ist diese nicht sonderlich detailreich und in manchen Fällen auch nicht aufschlussreich. Der einzige Weg herauszufinden wie etwas funktioniert war es in einem Beispiel auszuprobieren.

3.1 Funktionen

3.1.1 so_5::launch(*[](so_5::environment_t&){}*)

Diese Funktion ist essentiell für die Verwaltung von Agents. Prinzipiell wird nur ein Argument übernommen. Dieses ist eine Funktion welche eine so_5::environment_t Instanz verwaltet. Auf Basis dieser Instanz werden die Agents angelegt und verwaltet. Erst wenn diese Umgebung beendet wird wird die launch-Funktion beendet.

3.1.2 introduce_coop(*[](so_5::coop_t&){}*)

introduce_coop() ist eine Member-Function von so_5::environment_t. Diese Funktion wird in der Dokumentation von SObjectizer nicht erklärt. Sie wird einfach als Code-Snippet als notwendiges Übel bereitgestellt. Der einzige Parameter ist wieder wie in so_5::launch() eine Funktion welche sich wiederum um das eigentliche Anlegen der Agents kümmert. Der Parameter der übergebenen Funktion ist ein so_5::coop_t Objekt.

3.1.3 make_agent<*Klasse*>(*Paramliste des Konstruktors der Klasse*)

Die verwendete Klasse muss immer von so_5::agent_t abgeleitet sein um make_agent verwenden zu können. Es wird ein Zeiger auf die Klasse angelegt. Der einfachste Weg das Resultat zu speichern ist "auto" zu verwenden.

3.1.4 so_direct_mbox()

Im Kontext zu meinem Programm ist dies die wichtigste Funktion. Mit ihr wird die Inbox des erstellten Agents zurückgeliefert. Mithilfe dieser können wie bereits erwähnt die einzelnen Agents miteinander Kommunizieren. Wenn der Agent mittels make_agent aufgerufen wurde ist es notwendig darauf zu achten, dass das erstellte Objekt ein Pointer ist.

3.1.5 so_5::send<*Klasse/Struct*>(so_5::mbox_t, *Paramliste des Konstruktors der Klasse*)

Der erste Parameter ist immer die Inbox des Agents den man ansprechen möchte. Alle weiteren Parameter werden dazu genutzt die Klasse zu bilden. Die Funktion kann sowohl innerhalb von launch als auch in diversen Agents verwendet werden. Mithilfe dieser Funktion wird die Kommunikation der Agents ermöglicht. Die Klasse wird als "Brief" verwendet.

3.1.6 on_enter(*Funktion*)

Diese Funktion kann in Kombination von States verwendet werden. Wenn der State eines Agent zum spezifizierten State geändert wird, wird sofort der Code aus der übergebenen Funktion ausgeführt. Das kann benutzt werden um Variablen zu initialisieren oder um anderen Agents mitzuteilen, dass sich dieser jetzt in jenem State befindet.

```
class test final : public so_5::agent_t{
public:

  void so_define_agent() override {

    off.on_enter([this]{
      cout << "Dieser Agent ist gerade inaktiv geworden!!!" << endl;
    });

    on.on_enter([this]{
      cout << "Dieser Agent ist gerade aktiv geworden!!!" << endl;
    });

  }

private:
  state_t server_state,
  on{substate_of(server_state)},
  off{initial_substate_of(server_state)};
}
```

Als Eklärung: "so_define_agent()" ist eine Funktion, mit welcher das Verhalten eines Agents definiert wird. Diese Funktion ist von der Parent-Class "agent_t" vererbt worden und muss überschrieben werden.

3.1.7 on_exit(*Funktion*)

Ist das Gegenstück zu "on_enter" und wird aufgerufen wenn der State verlassen wird und kann somit als Destruktor gesehen werden.

```
class test final : public so_5::agent_t{
public:

  void so_define_agent() override {

    off.on_exit([this]{
      cout << "Dieser Agent ist gerade dabei in den Dienst zu wechseln!!!" << endl;
    });

    on.on_exit([this]{
      cout << "Dieser Agent beendet gerade seinen Dienst!!!" << endl;
    });

  }

private:
  state_t server_state,
  on{substate_of(server_state)},
  off{initial_substate_of(server_state)};
}
```

3.1.8 event(*Funktion*)

Events bilden das Herzstück von SObjectizer-States. Wenn ein Element in der Inbox ankommt was mit dem Typ übereinstimmt, welches die übergebene Funktion als Parameter besitzt wird dieses Event aktiviert.

```
struct msg_container {
  int sender_id;
  string msg;
}

class test final : public so_5::agent_t{
public:

  void so_define_agent() override {

    server_state.event(&event_handler);

  }
```

```
private:
  state_t server_state,

  void event_handler(mhood_t<msg_container> msg_c) {
    cout << "Nachricht von " << msg_c->sender_id << ": " << msg_c->msg << endl;
  }
}
```

3.2 Typen/Klassen

3.2.1 so_5::environment_t

Stellt die Umgebung bereit in welcher die Agents agieren können.

3.2.2 so_5::coop_t

so_5::coop_t stellt den Kontext bereit, mithilfe dessen Agents erstellt werden können.

3.2.3 so_5::mbox_t

Dieser Typ ist die eigentliche Inbox/Adresse mithilfe der sich die Agents untereinander verständigen.

3.2.4 so_5::agent_t

Von dieser Klasse muss geerbt werden, wenn ein Agent erstellt werden soll.

3.2.5 so_5::signal_t

Ist ein vereinfachtes struct welches ohne Parameter abeschickt werden kann. Wie der Name bereits sagt können zwischen Agents Signale herumgeschickt werden, welche für Events als Indikator dienen können.

3.2.6 state_t

Das Actor-Modell für welches SObjectizer ein Framework ist kennt sogenannte States. Ein Agent verhält sich unterschiedlich je nach dem in welchem State er gerade ist. States können auf folgende weise erstellt werden:

```
state_t server_state{this};
```

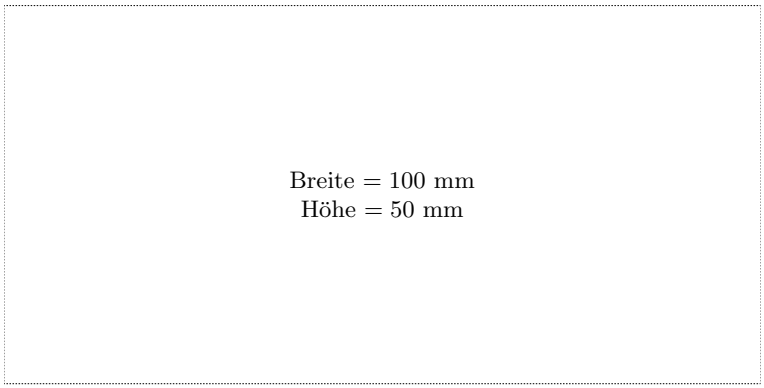
Dabei ist State eine Member-Variable der erstellten Agent-Klasse. Um das Event-Handling zu vereinfachen können States auch von States ”erben”. Das kann mit ”substate_of()” und ”initial_substate_of()” erreicht werden.

```
state_t server_state{this},
on{substate_of(server_state)},
off{initial_substate_of(server_state)};
```

Der Unterschied zwischen diesen beiden Funktionen ist, dass bei "initial_substate_of()" der erstellte State derjenige ist, in welchem sich der Agent initial befindet.

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —