Petunjuk dan Penjelasan Penggunaan State Machine

Source code state machine dapat diunduh pada link referensi yang telah dicantumkan pada bagian akhir tulisan. Klik link download yang berjudul Download StateMachine.zip - 22.5 KB.

Pada state machine ini, terdapat dua jenis state yaitu normal state dan extended state. Normal state seperti state pada umumnya, hanya mengeksekusi perintah tertentu jika machine memasuki state tersebut. Sedangkan pada extended state, terdapat empat jenis eksekusi yang mungkin terjadi ketika machine memasuki/keluar dari sebuah state. Keempat eksekusi tersebut adalah sebagai berikut.

1. Guard excecution
   Eksekusi guard akan selalu dijalankan ketika machine akan memasuki sebuah state. Guard berfungsi sebagai penjaga yang akan mengecek apakah machine memenuhi syarat/kondisi untuk memasuki state tersebut. Jika machine memenuhi syarat, maka machine secara otomatis akan masuk ke dalam state. Programmer dapat melakukan eksekusi tertentu selama machine memasuki guard.
2. Entry excecution
   Eksekusi entry hanya akan dipanggil saat machine memasuki suatu state untuk pertama kali.
3. State excecution
   Eksekusi state akan selalu dijalankan setiap kali machine memasuki state. Agar state dapat dimasuki, maka state tidak perlu diberikan guard atau jika guard diperlukan, maka kondisi guard tersebut harus terpenuhi.
4. Exit excecution
   Eksekusi exit hanya akan dipanggil saat machine keluar dari suatu state.

Catatan: setiap state tidak diharuskan memiliki keempat kondisi tersebut. (minimal state excecution)

Berikut adalah beberapa hal yang perlu diperhatikan dalam membuat state machine beserta contoh kode programnya.

1. Deklarasikan berbagai external event dan variable lainnya yang diperlukan pada class state machine yang ingin dibuat.

```
class CentrifugeTest : public SelfTest
{
public:
	CentrifugeTest();

	virtual void Start();
	void Poll();

	BOOL IsPollActive() { return m_pollActive; }

private:
	BOOL m_pollActive;
	INT m_speed;

	void StartPoll() { m_pollActive = TRUE; }
	void StopPoll() { m_pollActive = FALSE; }
. . .
```

2. Setiap state harus dideklarasikan terlebih dahulu secara berurutan dengan menggunakan enum, dengan diakhiri ST_MAX_STATES.

```
enum States
{
// Continue state numbering using the last SelfTest::States enum value
      ST_START_TEST = SelfTest::ST_MAX_STATES,
      ST_ACCELERATION,
      ST_WAIT_FOR_ACCELERATION,
      ST_DECELERATION,
      ST_WAIT_FOR_DECELERATION,
      ST_MAX_STATES
};
```

3. Setiap jenis eksekusi yang telah dijelaskan di atas, jika digunakan maka harus dideklarasikan setiap fungsinya dan aksinya (action)

```
// Contoh deklarasi dengan menggunakan macro
STATE_DECLARE(CentrifugeTest,       Idle,                   NoEventData)
STATE_DECLARE(CentrifugeTest,       StartTest,              NoEventData)
GUARD_DECLARE(CentrifugeTest,       GuardStartTest,         NoEventData)
STATE_DECLARE(CentrifugeTest,       Acceleration,           NoEventData)
STATE_DECLARE(CentrifugeTest,       WaitForAcceleration,    NoEventData)
EXIT_DECLARE(CentrifugeTest,  ExitWaitForAcceleration)
STATE_DECLARE(CentrifugeTest,       Deceleration,           NoEventData)
STATE_DECLARE(CentrifugeTest,       WaitForDeceleration,    NoEventData)
EXIT_DECLARE(CentrifugeTest,  ExitWaitForDeceleration)
```

Parameter pertama adalah class state machine yang sedang dibuat, parameter kedua adalah nama fungsi yang ingin dideklarasikan, parameter ketiga adalah data yang ingin dioper ke dalam fungsi. Deklarasi exit tidak membutuhkan parameter data.

4. Setiap jenis eksekusi harus dibuatkan state map dengan mengelompokkan setiap eksekusi sesuai dengan state terkait.

```
// Contoh deklarasi State Map dengan macro
BEGIN_STATE_MAP_EX
   STATE_MAP_ENTRY_ALL_EX(&Idle, 0, &EntryIdle, 0)
   STATE_MAP_ENTRY_EX(&Completed)
   STATE_MAP_ENTRY_EX(&Failed)
   STATE_MAP_ENTRY_ALL_EX(&StartTest, &GuardStartTest, 0, 0)
   STATE_MAP_ENTRY_EX(&Acceleration)
   STATE_MAP_ENTRY_ALL_EX(&WaitForAcceleration,0,0,&ExitWaitForAcceleration)
   STATE_MAP_ENTRY_EX(&Deceleration)
   STATE_MAP_ENTRY_ALL_EX(&WaitForDeceleration,0,0,&ExitWaitForDeceleration)
END_STATE_MAP_EX
```

State yang tidak memiliki eksekusi tambahan (guard/entry/exit) cukup dipetakan dengan fungsi STATE_MAP_ENTRY_EX. Sedangkan state yang memiliki eksekusi tambahan dipetakan dengan fungsi STATE_MAP_ENTRY_ALL_EX. Jika state yang tidak memiliki jenis eksekusi tertentu, maka eksekusi tersebut diisi dengan 0. Parameter yang diberikan harus sesuai/konsisten dengan nama fungsi yang dideklarasikan pada poin 3.

5. Pada setiap external event yang telah diklarasikan sebelumnya, berikan definisi external event yang berisi transisition map setiap state.

```
// Contoh definisi transition map dengan macro
void CentrifugeTest::Start()
{
   BEGIN_TRANSITION_MAP                    // - Current State -
      TRANSITION_MAP_ENTRY (ST_START_TEST)   // ST_IDLE
```

```
      TRANSITION_MAP_ENTRY (CANNOT_HAPPEN)   // ST_COMPLETED
      TRANSITION_MAP_ENTRY (CANNOT_HAPPEN)   // ST_FAILED
      TRANSITION_MAP_ENTRY (EVENT_IGNORED)   // ST_START_TEST
      TRANSITION_MAP_ENTRY (EVENT_IGNORED)   // ST_ACCELERATION
      TRANSITION_MAP_ENTRY (EVENT_IGNORED)   // ST_WAIT_FOR_ACCELERATION
      TRANSITION_MAP_ENTRY (EVENT_IGNORED)   // ST_DECELERATION
      TRANSITION_MAP_ENTRY (EVENT_IGNORED)   // ST_WAIT_FOR_DECELERATION
   END_TRANSITION_MAP(NULL)
}

void CentrifugeTest::Poll()
{
   BEGIN_TRANSITION_MAP                              // - Current State -
      TRANSITION_MAP_ENTRY (EVENT_IGNORED)   // ST_IDLE
      TRANSITION_MAP_ENTRY (EVENT_IGNORED)   // ST_COMPLETED
      TRANSITION_MAP_ENTRY (EVENT_IGNORED)   // ST_FAILED
      TRANSITION_MAP_ENTRY (EVENT_IGNORED)   // ST_START_TEST
      TRANSITION_MAP_ENTRY (ST_WAIT_FOR_ACCELERATION)// ST_ACCELERATION
      TRANSITION_MAP_ENTRY (ST_WAIT_FOR_ACCELERATION)// ST_WAIT_FOR_ACCELERATION
      TRANSITION_MAP_ENTRY (ST_WAIT_FOR_DECELERATION)// ST_DECELERATION
      TRANSITION_MAP_ENTRY (ST_WAIT_FOR_DECELERATION)// ST_WAIT_FOR_DECELERATION
   END_TRANSITION_MAP(NULL)
}
```

Transition map harus dideklarasikan secara berurutan sesuai dengan state yang telah dideklarasikan dengan fungsi enum (poin 2). Pada contoh di atas, jumlah state lebih banyak daripada yang dideklarasikan di poin 2 karena state pada Centrifuge Test ini merupakan lanjutan dari state SelftTest. Jika machine memasuki EVENT_IGNORED, maka tidak akan terjadi perpindahan state dan data yang dioper akan dihapus. Sedangkan jika machine memasuki CANNOT_HAPPEN, maka program akan mengalami error dan program tidak dilanjutkan. Jika external event menerima data, misalnya `void Motor::SetSpeed(MotorData* data)`, maka pada bagian akhir transition map diberikan operan data, misalnya `END_TRANSITION_MAP(data)`.

6. Buat definisi setiap fungsi untuk menjalankan setiap jenis eksekusi baik state/guard/entry/exit untuk setiap state.

```
// Contoh definisi fungsi state dengan Macro
// Idle state here overrides the SelfTest Idle state.
STATE_DEFINE(CentrifugeTest, Idle, NoEventData)
{
   cout << "CentrifugeTest::ST_Idle" << endl;

   // Call base class Idle state
   SelfTest::ST_Idle(data);
   StopPoll();
}

// Start the centrifuge test state.
STATE_DEFINE(CentrifugeTest, StartTest, NoEventData)
{
   cout << "CentrifugeTest::ST_StartTest" << endl;
   InternalEvent(ST_ACCELERATION);
}

// Guard condition to detemine whether StartTest state is executed.
GUARD_DEFINE(CentrifugeTest, GuardStartTest, NoEventData)
{
   cout << "CentrifugeTest::GD_GuardStartTest" << endl;
```

```
   if (m_speed == 0)
       return TRUE; // Centrifuge stopped. OK to start test.
   else
       return FALSE;      // Centrifuge spinning. Can't start test.
}

// Start accelerating the centrifuge.
STATE_DEFINE(CentrifugeTest, Acceleration, NoEventData)
{
   cout << "CentrifugeTest::ST_Acceleration" << endl;

   // Start polling while waiting for centrifuge to ramp up to speed
   StartPoll();
}

// Wait in this state until target centrifuge speed is reached.
STATE_DEFINE(CentrifugeTest, WaitForAcceleration, NoEventData)
{
   cout << "CentrifugeTest::ST_WaitForAcceleration : Speed is " <<m_speed<< endl;
   if (++m_speed >= 5)
       // InternalEvent(ST_DECELERATION);
       ExternalEvent(ST_DECELERATION);
}

// Exit action when WaitForAcceleration state exits.
EXIT_DEFINE(CentrifugeTest, ExitWaitForAcceleration)
{
   cout << "CentrifugeTest::EX_ExitWaitForAcceleration" << endl;

   // Acceleration over, stop polling
   StopPoll();
}

// Start decelerating the centrifuge.
STATE_DEFINE(CentrifugeTest, Deceleration, NoEventData)
{
   cout << "CentrifugeTest::ST_Deceleration" << endl;

   // Start polling while waiting for centrifuge to ramp down to 0
   StartPoll();
}

// Wait in this state until centrifuge speed is 0.
STATE_DEFINE(CentrifugeTest, WaitForDeceleration, NoEventData)
{
   cout << "CentrifugeTest::ST_WaitForDeceleration : Speed is " <<m_speed<< endl;
   if (m_speed-- == 0)
       InternalEvent(ST_COMPLETED);
}

// Exit action when WaitForDeceleration state exits.
EXIT_DEFINE(CentrifugeTest, ExitWaitForDeceleration)
{
   cout << "CentrifugeTest::EX_ExitWaitForDeceleration" << endl;

   // Deceleration over, stop polling
   StopPoll();
}
```

Parameter untuk setiap definisi fungsi sama dengan parameter untuk deklarasi fungsi (poin 3).

7. Langkah selanjutnya adalah membuat main.cpp

Catatan:

- Contoh kode program di atas diambil dari Centrifuge Test.h dan Centrifuge.cpp
- poin 1-4 dibuat di file dengan ekstensi .h, misalnya Centrifuge Test.h
- poin 5-6 dibuat di file dengan ekstensi .cpp, misalnya Centrifuge Test.cpp

Referensi:

State Machine: https://www.codeproject.com/Articles/1087619/State-Machine-Design-in-Cplusplus
UML: https://en.wikipedia.org/wiki/UML_state_machine