GFE – Graphical Finite State Machine Editor for Parallel Execution *

David Obdrzalek and Jan Benda

Charles University, Faculty of Mathematics and Physics,
Malostranske namesti 25, 118 00 Prague 1, Czech Republic
david.obdrzalek@mff.cuni.cz, jbe@matfyz.cz

Abstract. In this paper we present GFE – the Graphical FSM (Finite State Machine) Editor based on the Grafcet SFC (Sequential Function Chart) model. The GFE takes advantage of automated code generation and provides strong tools for complex control. At the same time it gives a high-level overview of the entire robotic control architecture. A complex control system may be designed, tested and deployed using visual approach. This is particularly useful for education where the students do not have to start always from scratch, or for young robot builders who are not as experienced in low-level programming. Once a control library is implemented for a particular robot, it may be reused and the robot may be programmed using solely graphical approach, because the most complicated part of controller design - the state machine - is automatically generated. This avoids typing errors and allows fast and simple redesign.

Keywords: robot control, automatic finite state machine generation, graphical control design.

1 Introduction

Robot building is increasingly popular. Even very simple robots are used to do complex jobs in industry and at home. Robots are used in education to demonstrate theoretical algorithmic concepts in real world, or they are even used as a hobby tool to proudly show the author is able to create a movable intelligent toy. Last, but not least, robot-based toys get increasingly more interest by vast public. In all mentioned areas, the users and builders usually do not like to build everything from scratch every time, so they want to reuse whatever possible. Also, the age of robot builders increasingly lowers, and younger authors usually do not have enough experience in programming and so their work is hard. There are numerous tools for such starting roboticians, be it a child or a student, but as any other tool, they have limitations. One of such limitations is quite common:

* The work was partly supported by the project 1ET100300419 of the Program Information Society of the Thematic Program II of the National Research Program of the Czech Republic.

most graphic design tools do not offer the author the freedom to control the hardware completely and are tied to a specific hardware (e.g. Lego Mindstorms NTX Software [1], Fischertechnik Robo Pro or Lucky Logic [2] etc.), or are too complex for beginners. Some of the tools provide basic blocks, but as the complexity of the system grows, they become to be unmaintainable.

One of the pretty popular tool categories for robotic application uses Finite state machines (FSM). They provide a rich formalism for high level control of robot's processes and strategies. Unfortunately, for practical applications the number of considered states and conditions quickly exceeds the size reasonably maintainable by hand. Moreover, the transcription of a FSM to most programming languages is rarely lucid and easily extensible.

In this paper we present the Graphical Finite State Machine Editor (GFE), a tool for visual design of generalized finite state machines based on the Sequential Function Chart (SFC) model. Compared to a traditional FSM, an SFC chart allows multiple states being active at a time. It also provides means for easy synchronization of parallelly executed branches. The editor allows users to create powerful state machine while using the graphical approach and at the same time allows the user to create low-level functionality for the robotic hardware without losing control of the program because of its complexity.

From within the editor, a function implementing the designed SFC chart in a high-level programming language may be automatically generated. To complete the overall functional control program, basic sensing and control functions have to be linked with the generated code. This set of functions is immutable from the sight of the user and must be created only once for a specific robot hardware. The major advantage of this approach is that no manual edit of code is required when the logic of the chart changes, as only the generated FSM code is affected. Once the input/action library is implemented for a particular robotic platform, the robot may be programmed by a vast public using solely graphical approach. Also, the same user-created control may be used for more hardware platforms without any change, only by replacing the corresponding underlying library. And these are very important features - particularly in the domain of educational robotics.

2 The SFC Language

The SFC is a specialized graphical language developed as a tool for design of control automata. It is a subset of the Grafcet norm [3], which (among others) proposes one language for the control system structure - the SFC (Sequential Function Chart). Its intuitive graphical syntax and powerful semantic was taken as the structural basis for the GFE. This section is dedicated to the necessary SFC basics.

The SFC chart has a form of an oriented bi-partitioned graph where two different kinds of nodes are linked using oriented edges:

- *Places* (represented as rectangular blocks in the chart) correspond roughly to the states of an automaton. Each place defines a specific action to be

performed by the control program when the place is active. Multiple SFC places may be active at a time. Places active at the beginning of the code execution are called initial and are marked with a double border.
- *Transitions* (represented as thick lines over interconnections) act as conditional expressions that transmit activation within the chart given that all the transition input places are active and the transition condition (a boolean expression) is met. Several transitions may be open at a time.
- *Interconnections* (represented as lines connecting places and transitions) are oriented edges describing the activation flow within the chart.

In a classical deterministic FSM, the network would only branch from a place and only one state may be active at a time. The SFC allows also branching and synchronization for parallel activation of multiple places. This makes it useful for deterministic design of systems with multiple interdependent subsystems and/or asynchronous input events. The available control structures are:

- *Conditioning* (one place connected to multiple transitions): the activation passes from a single input place to one output place (*branching*) or two output places are connected to one input place (*merging*). The flow is determined by the first open transition (see Figure 1).
- *Parallelism* (one transition connected to multiple places): the activation passes through a single transition to all output places (*forking*) or from multiple active places into a single transition (*synchronization*). Prior to opening the transition all the input places connected to the transition must be active (see Figure 2).

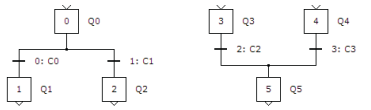

Fig. 1. Serial branching and merging.

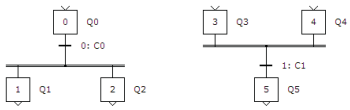

Fig. 2. Parallel forking and synchronization.

3 FSM Generation in GFE

Over the years, many formats were proposed for description of the finite state machines (FSM), for example transition graphs, transition tables, SFCs, formal definitions of a FSM as quintuple {S, X, Y, d, l} and many others. For our purposes we use the SFC because it allows the FSM designer to naturally work with parallel execution branches without having to withdraw the deterministic FSM model. This is because a well-formed SFC chart maps unambiguously to a deterministic FSM. In this section we describe the automatic creation of an SFC engine.

The state of the entire SFC chart is given by the list of activation numbers of the individual places in the chart (an *activation vector*). To build a SFC engine, the SFC activation rules are applied on the activation vector of the SFC model. The activation vector is implemented using a fixed-length array of activation counters. The SFC engine then consists of a single function that should be periodically called to update the chart state. This function is composed of two sections: the activation update and the place action blocks.

The activation update block contains one conditional block generated for each transition in the chart - the values of input places and the value of the boolean condition are checked to determine if a particular transition is open. If this test succeeds, the transition is opened transferring the activation from the input places (decreasing activation counter) to the output places (increasing the counter).

The second section of the generated function is responsible for performing the actions of the active places. One conditional block is generated for each place to execute the appropriate action if and only if the particular place is active (see the generated code example in Section 5).

4 The GFE Output

The presented subset of the SFC language is well suited for description of the control code structure. Unfortunately, the executive part of the SFC is too weak to express every desired action. Many visual control system editors choose to provide the chart designer with access to the low level features of the underlying hardware (see for example the Programming Editor for picaxe [4]). This has two big disadvantages: firstly, programming on the level of hardware control signals is too fine-grained and quickly leads to a high amount of duplicate objects and subsections in the chart, and secondly, programming using low level hardware identifiers is hardly lucid and easy to maintain. Therefore we propose the use of a high level programming language for the actions behind the SFC model and the use of the implementation library for individual low-level hardware operations. The visually edited chart provides easily maintainable structure and a high-level overview of the control actions while the code behind provides powerful, reusable and programmable operations on the hardware.

But how are the individual robot's actions triggered? As stated above, a place in the chart corresponds to an action that should be performed by the control program. In the GFE, the required actions are entered into the text field of each place. When generating the SFC engine, all the place descriptions are treated as control commands and output directly into the resulting engine code. This approach has a great advantage both in easy maintainability and great extensibility as the place actions can range from simple function calls via parameterized function calls with arguments to arbitrarily advanced operations[1].

[1] We recommend using time-simple operations here because of performance reasons, but it is not a strict dictate. Also, using library calls for low-level functionality instead of coding it in the GFE is highly recommended for maintainability reasons.

It relieves the tiring state machine coding from the programmer but still allows to use all the comfortable features of a high-level programming language.

The generated language was chosen to be the C language. This is because of its high penetration in the domain of robotic control systems, great extensibility, portability and a steep learning curve. The C language is widely supported by a broad variety of popular control architectures from small scale RISC microcontrollers to embedded PCs and other portable devices, which enables the same control chart to be used with a wide range of targets. It is also important to note that the choice of C is not binding. Virtually any language could be used, and the GFE may even be enhanced to support custom-defined syntax[2].

5 Example

This example is very simple, but even that it demonstrates the principles well. Let us assume the simplest case of a line-following robot. It is equipped with just two sensors (left / right infrared line detectors) and two actuators (left / right engine). The simplest approach directs the robot ahead only stopping individual wheels to keep the black line between the two sensors. When a sensor at one side detects black instead of white surface, the motor at that side is stopped so that the robot turns and the sensor gets off the line (and similarly the other side).

The control code in SFC (see Figure 3 left) is simple and lucid taking advantage of parallel simulation of two independent controllers. The advantage of the presented approach is obvious in comparison with the FSM (see Figure 3 right) isomorphic to the proposed SFC controller.

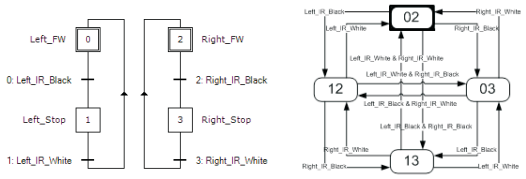

Fig. 3. A simple line-follower SFC and a corresponding FSM.

The generated SFC engine for this example is shown in Figure 4. The activation vector for the entire SFC chart is represented in a global array initialized by the values of starting activation numbers of the chart's places. The code of the SFC engine consists of two sections: the state update and action execution. In the first section the activation vector is updated following the rules of SFC signal propagation (the GFE_Activated and GFE_Enabled callbacks may be used in the encapsulating control software). The second section is responsible for execution of action code for all active places. Note how the place and transition captions

[2] The only required features of the target language are the support for function calls and the support for conditional block execution.

```
int _AV[] = {1, 0, 1, 0};
void GFE_Run() {
  // Perform state update
  if (_AV[0] && Left_IR_Black()) {
    if (!--_AV[0]) GFE_Activated(0, FALSE); // Place 0 lost activation
    if (!_AV[1]++) GFE_Activated(1, TRUE);  // Place 1 gained activation
    GFE_Enabled(1); // Transition 1 may be opened from now on
  }
  if (_AV[1] && Left_IR_White())  { /* ... similar to _AV[0] */ }
  if (_AV[2] && Right_IR_Black()) { /* ... */ }
  if (_AV[3] && Right_IR_White()) { /* ... */ }

  // Perform place actions
  if (_AV[0]) { Left_FW(); }
  if (_AV[1]) { Left_Stop(); }
  if (_AV[2]) { Right_FW(); }
  if (_AV[3]) { Right_Stop(); }
}
```

Fig. 4. Code example - the generated SFC engine for the line-follower robot.

are turned into function calls. These functions are defined by the capabilities of the target platform and have to be defined in the encapsulating control code. The important fact is that these functions must only be defined once for a particular robotic platform. Once these sensing and control functions are implemented, the robot can be programmed using purely the visual approach.

6 Conclusion

In this paper we presented the GFE (Graphical FSM Editor) as a tool for visual design of generalized Finite State Machines based on the Grafcet SFC model. The GFE is intended for use by robot builders and users who want to concern on controlling the robot and not wasting time by implementing control state machine and low-level functionality every time anew.

The editor supports automatic generation of the SFC code to be included into a control system written in the C language. This allows an inexperienced user, given with a predefined set of control and sensing functions, to design, deploy and execute a complete control system using only visual approach. The major advantage of this approach is that no manual edit of code is required when the logic of chart changes, as only the generated SFC engine is affected. The editor also has great educational potential: once the action/input library is implemented for a particular robot, it may be programmed by a vast public using solely graphical approach.

References

1. Lego, Mindstorms NTX Software, http://mindstorms.lego.com/
2. Fischertechnik, Robo Pro, Lucky Logic, http://www.fischertechnik.com/html/computing-software.html
3. International Electrotechnical Commission, IEC 1131-3: Programmable Controllers - Part3: Programming Languages, IEC 1131-3, 1993.
4. Crocodile Technology, Programming Editor, http://www.rev-ed.co.uk/picaxe/progedit.htm