FSC Designer: A Visual FSM Design Tool for Robot Control

Okan Aşık OKAN.ASIK@BOUN.EDU.TR
H. Levent Akın AKIN@BOUN.EDU.TR
Boğaziçi University, Department of Computer Engineering, 34342, İstanbul, Turkey

Abstract

This paper describes the specification and design of a visual tool which enables finite state machine design and automatic code generation for robot control. The main aim of this study is to achieve rapid behavior development based on finite state machines. Although it is currently used only in the *RoboCup Standard Platform League*, it can be used in any robot software since the visual designer generates platform independent modified version of State Chart XML. The tool also has facilities for push-button code generation and compilation.

1. Introduction

A robot may look like a bunch of motors, sensors and some processing units. Actually, this is true, if we just consider the hardware of a robot. However, without software, a robot is no more useful than a table. Typical robot software have many components such as localization, locomotion, perception and planning. Every component tries to solve a specific robot problem. Some of the problems are highly hardware dependent and some are environment dependent. Planning is one of the core components of every robot software. If we abstract the low-level behaviors of a robot, the functionality of planning the actions of a robot does not change from one robot to another. When low-level behaviors are combined in a meaningful way, the robot completes the required task. The aim in this study is to provide a tool so that once such low-level behaviors are developed, meaningful combinations of them can be easily implemented. The *FSC Designer* models planning using a finite state machine (FSM) where one or more actions are executed at a particular state.

The organization of the rest of the paper is as follows. In Section 2, related work is presented. Then, the architecture of visual designer is explained in Section 3. The advantages of the tool are discussed in Section 4. Finally, in Section 5 conclusions and future work are presented.

2. Related Work

RoboCup (Anon., 2012) is an international robotics competition to promote robotics and AI research. There are mainly soccer, rescue, @Home, and Junior leagues. The planning architectures in the *RoboCup* domain can be categorized as follows: (1) Behavior-based (Lenser et al., 2001), (2) FSM-based (Obst, 2002; Loetzsch et al., 2006; Lausen et al., 2004), and (3) Logic-based (Ferrein & Lakemeyer, 2008). However, FSM-based architectures can be viewed as a variation of behavior-based systems since FSM provides a special arbitration mechanism for the behavior-based architectures.

The main motivation for using FSM in robot planning is a need for an easy to design graphical formalism. In (Hugel et al., 2006; Murray, 2004; Loetzsch et al., 2006), they propose Hierarchical Finite State Machines to achieve such a formalism. (Hugel et al., 2006), and (Murray, 2004) propose a visual tool to simplify the design-test cycle of robot planning. In (Loetzsch et al., 2006), they propose a more general robot planning framework in which they use the *XABSL* programming language to define agent behaviors.

3. FSC Architecture

The proposed approach is implemented in the *Cerberus RoboCup Standard Platform League* team robot software (Akın et al., 2011). The FSC architecture has two components, namely, the *FSC Planner Engine*, and the *Visual FSC Designer*. These are described below.

3.1 FSC Planner Engine

A Finite State Controller (FSC) is a modified version of the well-known finite state machine. A FSC consists of states, transitions, and actions. The states and transitions have the same semantics as the conventional finite state machines. When the robot runs a FSC planner, all actions of the active state are run, and all transitions of the active state are checked. If a transition condition holds, active state changes to the state pointed by the transition. There is a priority mechanism where the first transition overwrites other

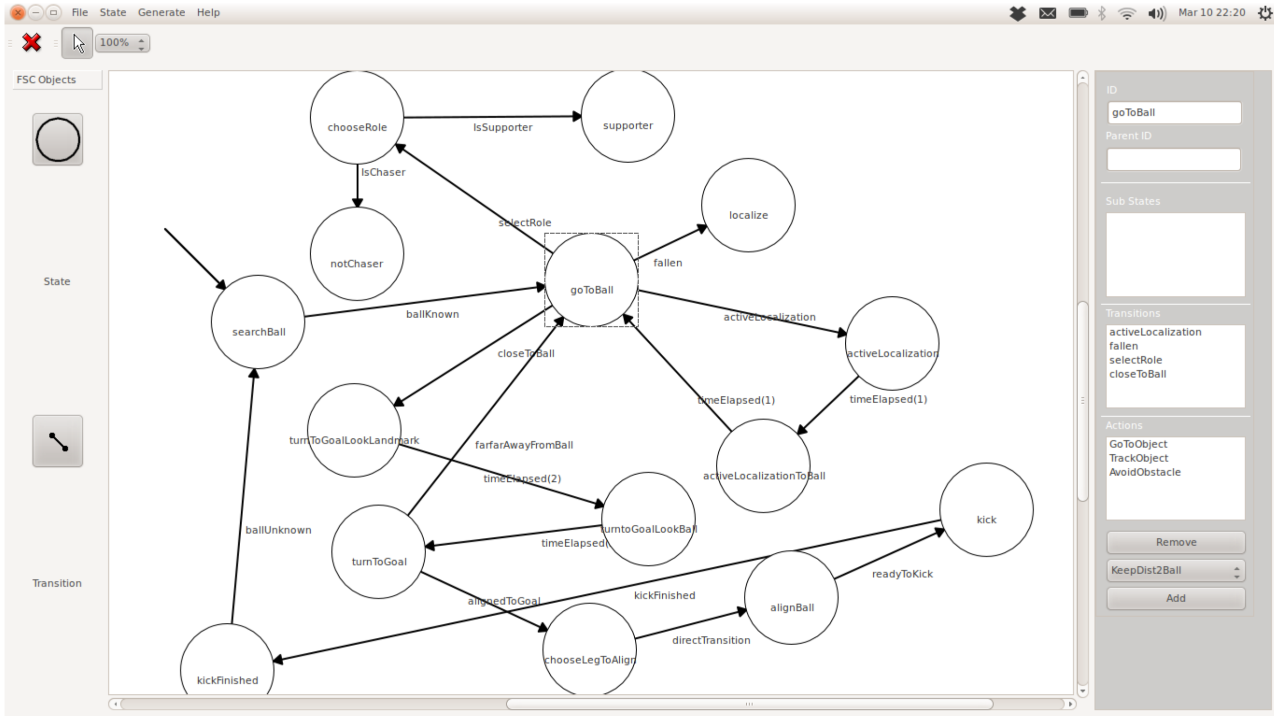

Figure 1. A screen shot of the FSC Designer

transitions if it holds any.

Actions are low-level behaviors such as walking with the given speed vector. The planner engine first runs the actions of the active state, then checks the transitions and completes the planner step. There is an implicit subsumption architecture. The action with the highest priority is executed last since it may overwrite the other actions' behaviors.

3.2 Visual FSC Designer

The Visual *FSC Designer* is the front-end of the *FSC Planner Engine*. It provides an easy-to-use graphical design tool. It has three functionalities: (1) graphical representation, and manipulation of finite state controllers, (2)c ode generation, and (3) deployment to the robot. In Figure 1, a screen shot of the *Visual FSC Designer* can be seen. The designer shows simplified version of *Cerberus*'s *Chaser* behavior. The corresponding XML code generated by the *FSC Designer* can be seen in Figure 2. Note that we omit some states and conditions for the sake of comprehensibility. On the canvas of the *FSC Designer*, we manipulate states and transitions. We add actions to the transitions from the *Actions* property of *Property Viewer*. *Actions* are filled from the special directory, but the transitions' condition is written in *Code Editor Window*.

The *Visual FSC Designer* uses a modified version of State Chart XML (*SCXML*) (Barnett et al., 2011). Therefore, by developing different engines for different robot software frameworks we can reuse a previously developed plan. The only requirement is to have the corresponding low-level behaviors.

4. Discussion

Many *RoboCup* teams need such a planning module. This is due to the importance of low-level behaviors. Teams generally work hard on low-level behaviors, and there is little time left to develop complex behaviors. By simplifying the development of behavior development, teams would have more opportunity to work on complex behaviors.

Some test scenarios or development activities require complex behaviors. A localization test should not require to hack some part of the planning module. Any module developer should be able to implement his test scenario easily. This is where the *FSC Designer* becomes handy by enabling reuse of previously developed low-level behaviors.

5. Conclusion and Future Work

The proposed system can be used for different robot platforms since the *FSC Designer* generates platform independent XML files. However, it should have its own implementation of finite state controller.

```
<?xml version="1.0" encoding="UTF-8"?>
<scxml xmlns="http://www.w3.org/2005/07/
    scxml" version="1.0" initial="
    searchBall">
  <state id="searchBall" x="976" y="991">
    <transition id="ballKnown" event=""
        cond="" target="goToBall"/>
    <action id="SearchObject"/>
  </state>
  <state id="kick" x="1756" y="1192">
    <transition id="kickFinished" event
        ="" cond="" target="kickFinished
        "/>
    <action id="TrackObject"/>
    <action id="Kick"/>
    <action id="LookKickTarget"/>
  </state>
  <state id="alignBall" x="1547" y="1286">
    <transition id="readyToKick" event=""
         cond="" target="kick"/>
    <action id="TrackObject"/>
    <action id="AlignBall"/>
  </state>
  .
  .
  .
  <state id="supporter" x="1372" y="770">
    <action id="TrackObject"/>
    <action id="KeepDist2Ball"/>
  </state>
  <state id="chooseRole" x="1082" y="772">
    <transition id="IsChaser" event=""
        cond="" target="notChaser"/>
    <transition id="IsSupporter" event=""
         cond="" target="supporter"/>
  </state>
  <state id="goToBall" x="1333" y="946">
    <transition id="activeLocalization"
        event="" cond="" target="
        activeLocalization"/>
    <transition id="fallen" event="" cond
        ="" target="localize"/>
    <transition id="selectRole" event=""
        cond="" target="chooseRole"/>
    <transition id="closeToBall" event=""
         cond="" target="
        turnToGoalLookLandmark"/>
    <action id="GoToObject"/>
    <action id="TrackObject"/>
    <action id="AvoidObstacle"/>
  </state>
</scxml>
```

Figure 2. Corresponding XML Code

In *RoboCup 2011*, the *Cerberus Team* used the *FSC Designer* and ranked in the first 16 teams.

Hierarchical implementation of the finite state controller is one of the future works since it will enable the development of even more complex behaviors on top of the possible complex behaviors. There is also limited support to write the transition conditions. As a future work, we aim to develop a syntax checker and include an auto-completion feature in the *Code Editor Window*.

References

Akın, H. L., Gökçe, B., Özkucur, E., Kavaklıoğlu, C., Sevim, M. M., Ayar, T., & Aşık, O. (2011). *Cerberus'11 team description paper* (Technical Report). Bogazici University.

Anon. (2012). RoboCup Federation http://www.robocup.org/.

Barnett, J., Akolkar, R., Auburn, R., Bodell, M., Burnett, D., Carter, J., McGlashan, S., Lager, T., Helbing, M., Hosn, R., et al. (2011). *State Chart XML (SCXML): State Machine Notation for Control Abstraction* (Technical Report). W3C.

Ferrein, A., & Lakemeyer, G. (2008). Logic-based robot control in highly dynamic domains. *Robot. Auton. Syst.*, *56*, 980–991.

Hugel, V., Amouroux, G., Costis, T., Bonnin, P., & Blazevic, P. (2006). Specifications and design of graphical interface for hierarchical finite state machines. *RoboCup 2005: Robot Soccer World Cup IX* (pp. 648–655). Springer.

Lausen, H., Nielsen, J., Nielsen, M., & Lima, P. (2004). Model and behavior-based robotic goalkeeper. *RoboCup 2003: Robot Soccer World Cup VII* (pp. 169–180). Springer.

Lenser, S., Bruce, J., & Veloso, M. (2001). A modular hierarchical behavior-based architecture. *RoboCup-2001: The Fifth RoboCup Competitions and Conferences* (pp. 423–428). Springer Verlag.

Loetzsch, M., Risler, M., & Jungel, M. (2006). XABSL-A Pragmatic Approach to Behavior Engineering. *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 5124–5129).

Murray, J. (2004). Specifying agent behaviors with UML statecharts and StatEdit. *RoboCup 2003: Robot Soccer World Cup VII* (pp. 145–156). Springer.

Obst, O. (2002). Specifying rational agents with statecharts and utility functions. *RoboCup 2001: Robot Soccer World Cup V* (pp. 173–182). Springer.