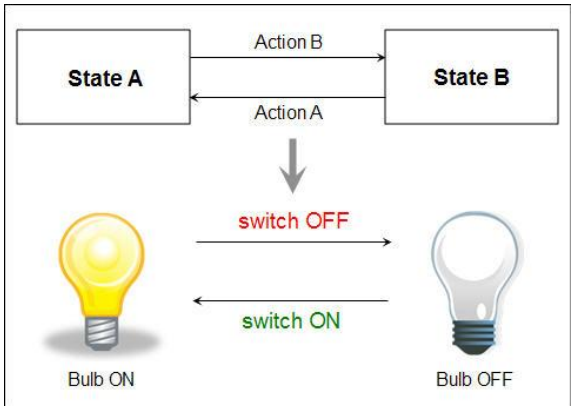
Разработка роботов

Конечные автоматы и автоматизация или программируем ПЛК

Автор: Dmitry
Опубликовано: 26 марта 2011
Метки: Software, Автоматизация, ПЛК, Промышленные компьютеры
Рубрики: Публикации



При создании АСУ ТП с аппаратной частью все технологии уже хорошо отработанны: берём комплектующие от известных мировых производителей(***Beckhoff***, ***Siemens***, ***Schneider Electric***), монтируем их в шкаф. Много трудностей это не вызывает. Небольшой обзор по аппаратной части можете посмотреть здесь. По моему опыту, больше всего проблем возникает с написанием управляющих программ и их отладкой на реальном железе.

Чтобы минимизировать временные (а значит и финансовые) затраты на программирование и отладку, разработку любой программной системы стоит начать с проектирования архитектуры. Если архитектура сделана грамотно, в процессе отладки не придётся переписывать большие объёмы кода, посторонним людям будет легко разбираться в вашем коде. Возможно, часть кода удастся использовать в следующем проекте.

В этой статье я рассмотрю возможный вариант построения программной архитектуры АСУ ТП на основе конечного автомата(state machine).

Нестрого говоря, конечный автомат – это модель устройства с конечной памятью, воспринимающего дискретные входные сигналы и выдающая некие выходные сигналы. ПЛК идеально подходят под эту модель. Автомат состоит из трёх основных элементов:

- состояние (state) – условие или ситуация, при которых автомат удовлетворяет некоторому условию, осуществляет некоторую деятельность или ожидает некоторого события;
- событие (event) – описание заслуживающего внимания происшествия, занимающего определенное положение во времени и пространстве;
- переход (transition) – переход из одного состояния в другое в ответ на событие.

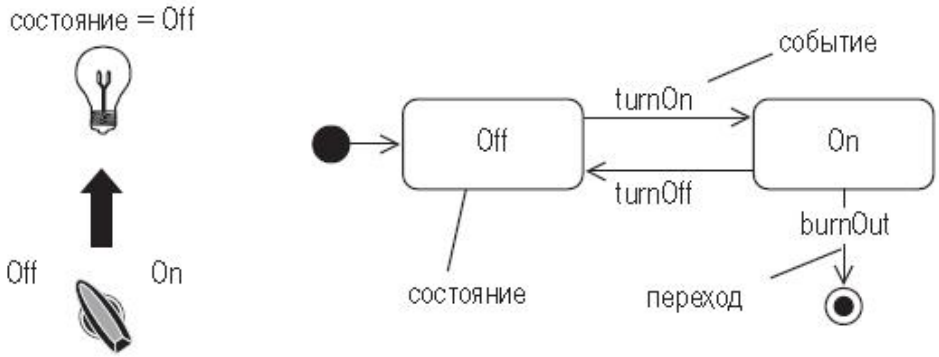
У каждого конечного автомата должно быть начальное состояние обозначающее первое состояние последовательности, его мы будем использовать для начальной инициации. Если смена состояний не бесконечна, должно присутствовать и конечное состояние, которое завершает последовательность переходов, в случае с ПЛК его, скорее всего, не будет.

Пока автомат (читай ПЛК) находится в каком-либо состоянии, он постоянно выполняет определённое действие (один и тоже участок кода). По приходу какого-нибудь внешнего события (изменение состояния дискретного входа, превышение значения с аналогового входа некоторого порогового значения, изменение внешней переменной…) он может переходить в другое состояние. Перейдёт автомат (ПЛК) или нет в другое состояние, и перейдет ли вообще, зависит от выполнения так называемых сторожевых условий(guards). Переходы могут быть безусловными. Условий для одного перехода не может быть более одного. При переходе из одного состояния в другое может выполняться какое-либо действие.

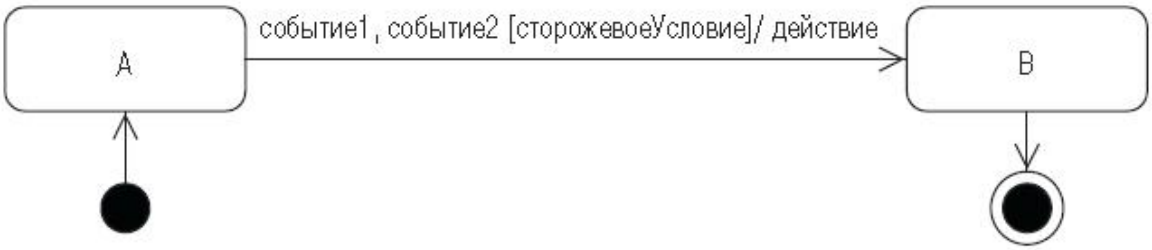
Теперь перед началом написания программы для АСУ я предлагаю представить логику её работы в виде конечного автомата. Для графического представления конечного автомата я использую UML 2.0. В этом стандарте есть диаграммы состояний (state machine diagrams), которыми я и пользуюсь. Могу посоветовать неплохой бесплатный UML редактор под Windows и Linux: **Visual Paradigm.**

Для читателей не знакомых с UML 2.0 и/или конечными автоматами рассмотрю простенький пример описания логики устройства в виде конечных автоматов. А за более детальным описанием отправляю за книгу Лармана "Применение UML 2.0 и шаблонов проектирования", глава 21.

Лампочка. 🙂 Имеет состояния: включена(on), выключена(off) и сгорела(конечное состояние). Переход между состояниями on/off - по переключению выключателя. Переход в состояние сгорела - по некоторому внешнему событию, будем считать, что из состояния off. Таким образом, у нас есть три события: turnOn(включить), turnOff(выключить), burnOut(сжечь). Все переходы безусловные. Ниже приведено графическое представление соответствующего автомата в терминах UML 2.0.



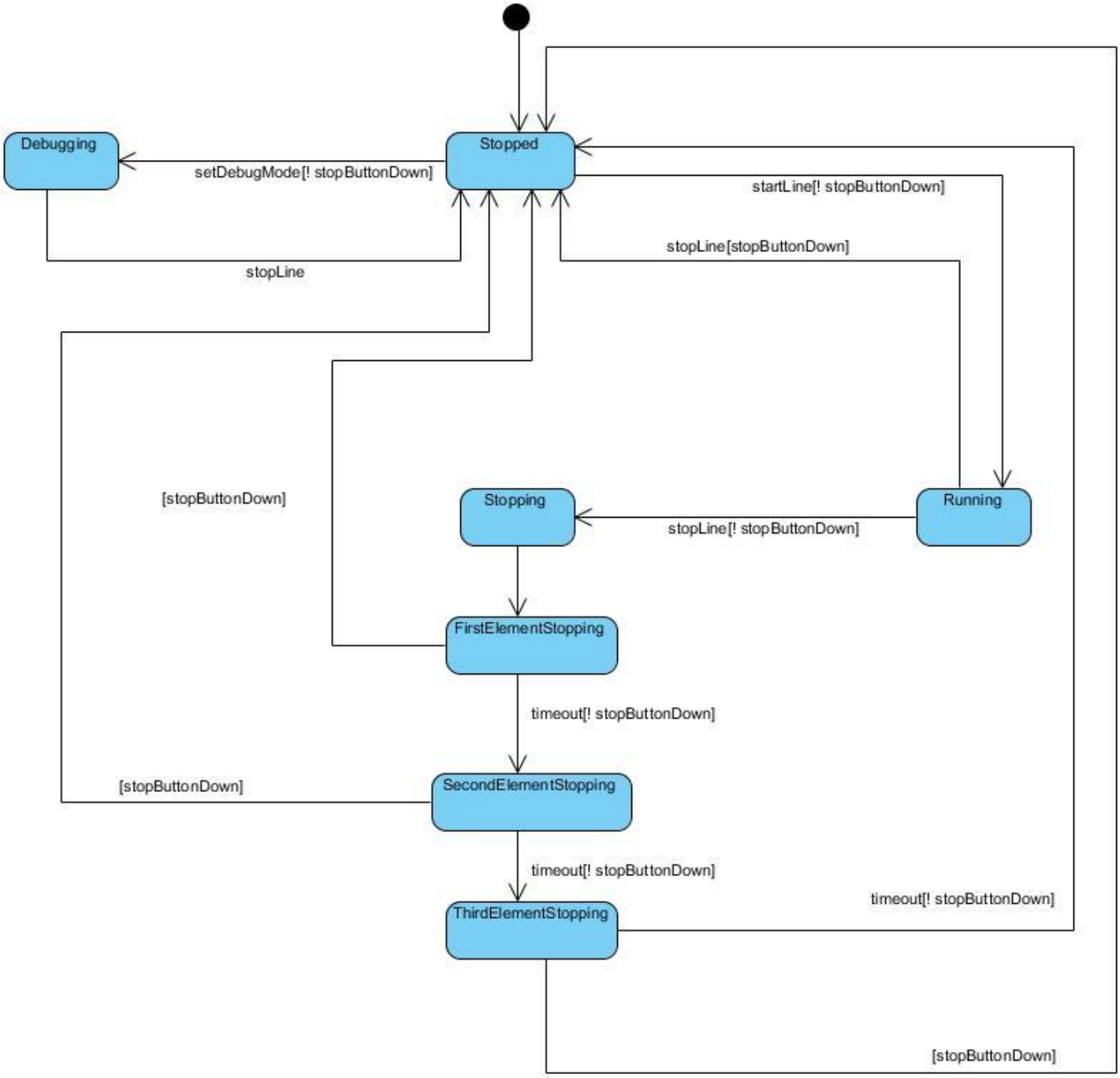
Полный синтаксис описания перехода выглядит так:



Как видно переход может инициироваться одним из группы событий.

А теперь рассмотрим небольшой пример описания АСУ ТП. У нас есть автоматическая линия, которая может находиться в одном из четырёх глобальных состояний: "остановлена", "запущенна", "останавливается", "наладка". Состояние "останавливается" переводит в состояния "останавливается первый узел", "останавливается второй узел", "останавливается третий узел". Переход между состояниями "запущенна", "останавливается" и "наладка" осуществляется оператором при помощи touch-panel. Так же имеется кнопка аварийный стоп, которая мгновенно переводит в состояние "остановлена" и, пока нажата, не даёт из него выйти.

Полученный конечный автомат:



Конечные автоматы удобно реализовывать почти на всех языках программирования. Но всё же ПЛК чаще всего программируются на языках стандарта МЭК61131-3. Поэтому приведу пример реализации на ST.

Сначала объявим несколько перечислений:

Все возможные состояния линии:

```
TYPE LineState :
(
    INITIAL_STATE,
    RUNNING_STATE,
    STOPED_STATE,
    DEBUGGING_STATE,
    STOPPING_STATE,
    FIRST_ELEMENT_STOPPING_STATE,
    SECOND_ELEMENT_STOPPING_STATE,
    THIRD_ELEMENT_STOPPING_STATE
);
END_TYPE
```

Состояния линии, в которые может попытаться перевести её оператор:

```
TYPE LineMode :
    (
        RUNNING_MODE,
        STOPED_MODE,
        DEBUG_MODE
    );
END_TYPE
```

Для взаимодействия с пользовательским интерфейсом объявим две структуры. Текущие параметры линии, которые оператор может только считывать:

```
TYPE ReadLineParameters :
    STRUCT
        lineStopped: BOOL;
        currentLineState: LineState;
    END_STRUCT
END_TYPE
```

Параметры, доступные для изменения оператором:

```
TYPE WriteLineParameters :
    STRUCT
        requiredState: LineMode;
    END_STRUCT
END_TYPE
```

Время необходимое для остановки одного узла:

```
VAR_GLOBAL CONSTANT
    elementStoppingTime: TIME := T#1m;
END_VAR
```

Переменные для взаимодействия с пользователем придётся сделать глобальными. Разместим их по известным адресам, чтобы можно было считать их, например, по протоколу ADS.

```
VAR_GLOBAL
    readParameters AT %MB0: ReadLineParameters;
    writeParameters AT %MB20: WriteLineParameters;
END_VAR
```

Функциональный блок для управления линией.

```
FUNCTION_BLOCK FB_Line
VAR_IN_OUT
    controlReadParameters: ReadLineParameters;
    controlWriteParameters: WriteLineParameters;
END_VAR
VAR
    currentLineState: LineState;

    watchDog: TON;
    watchDogOn: BOOL;

    stopButtonPressed AT %I*: BOOL;
END_VAR
```

Конечный автомат реализуем при помощи оператора выбора CASE.

```
(*Process Stop button down event.*)
IF stopButtonPressed THEN
    currentLineState := STOPED_STATE;
END_IF
controlReadParameters.lineStopped := MOVE( stopButtonPressed );
CASE currentLineState OF
    INITIAL_STATE:
```

```
        (*Required code.*)
        currentLineState := RUNNING_STATE;
    RUNNING_STATE:
        IF controlWriteParameters.requiredState = STOPED_MODE THEN
            currentLineState := STOPPING_STATE;
        ELSE
            (*Required code.*)
            ;
        END_IF
    STOPED_STATE:
        CASE controlWriteParameters.requiredState OF
            RUNNING_MODE:
                (*Required code.*)
                ;
                currentLineState := RUNNING_STATE;
            DEBUG_MODE:
                (*Required code.*)
                currentLineState := DEBUGGING_STATE;
            ELSE
                (*Required code.*)
                ;
        END_CASE
    DEBUGGING_STATE:
        (*Required code.*)
        IF controlWriteParameters.requiredState = STOPED_MODE THEN
            currentLineState := STOPED_STATE;
        END_IF
    STOPPING_STATE:
        watchDogOn := TRUE;
        currentLineState := FIRST_ELEMENT_STOPPING_STATE;
    FIRST_ELEMENT_STOPPING_STATE:
        IF watchDog.Q  THEN
            currentLineState := SECOND_ELEMENT_STOPPING_STATE;
        ELSE
            (*Required code.*)
            ;
        END_IF
    SECOND_ELEMENT_STOPPING_STATE:
        IF watchDog.Q  THEN
            currentLineState := THIRD_ELEMENT_STOPPING_STATE;
        ELSE
            (*Required code.*)
            ;
        END_IF
    THIRD_ELEMENT_STOPPING_STATE:
        IF watchDog.Q  THEN
            watchDogOn := FALSE;
            currentLineState := STOPED_STATE;
        ELSE
            (*Required code.*)
            ;
        END_IF
END_CASE
controlReadParameters.currentLineState := MOVE( currentLineState );
IF watchDogOn  THEN
    watchDog ( IN := NOT watchDog.Q, PT := elementStoppingTime );
END_IF
```

Ну и естественно приведу вид точки входа в нашу задачу. Она получилась очень простой, комментариев не требует.

```
PROGRAM MAIN
VAR
    line: FB_Line;
END_VAR
VAR_EXTERNAL
    readParameters: ReadLineParameters;
    writeParameters: WriteLineParameters;
END_VAR

line( controlReadParameters := readParameters, controlWriteParameters := writeParameters);
```

Рассмотренный мною конечный автомат называется поведенческим. Он полезен не только при проектировании АСУ ТП, но и, например, при создании роботов. Наша команда применила такой автомат при создании системы управления мобильным роботом для Eurobot 2011. Ещё существуют протокольные конечные автоматы, но о них в какой-нибудь следующей статье.

Проектирование автоматов не занимает много времени. Зато они в наглядном виде представляют требуемую логику работы системы. Такое формальное описание легко переносится в код. При этом, по моему опыту, код упрощается, становиться легко сопровождаемым и хорошо читаемым.

Проект целиком (TwinCAT 2 и Visual Paradigm).