F. Wagner December 2007

Vfsm executor library

The idea

We provide a library vfsm.lib which contains the executor of the StateWORKS specification. The library can be integrated into an application where it is used to process the state machine(s). The VFSM library is intended for use in projects where, for any reason, it is not appropriate to use the RTDB library, and its use implies that the developer will have to generate a greater amount of code than would be the case with the RTDB.

While coding the application we use a set of methods to:

- establish the VFSM Executor (CVfsm, GetSpecInfo)
- process the state transitions and trigger the actions (Execute)
- trace the VFSM Executor (Trace)

The VFSM Executor processes the state transition tables as specified in the StateWORKS Studio, changing states and calling output functions (actions). The main task of a developer is to code the output functions whose prototypes are declared in a file generated by the StateWORKS Studio.

Accessing the executor

The StateWORKS Studio generates (at Build) several files that can be used for implementation of the run-time system. The files: xxx.str, xxx.h and xxx_out.h, (xxx stands for the state machine name) are used for building an application with VFSM Executor. Their content must not be changed.

In addition, we use a library vfsm.lib (which contains the state machine executor) and files: vfsm.h and vfsmset.h. The file vfsm.h contains declarations of methods that are used for accessing the VFSM Executor while writing the application system:

- CVfsm(const string stVfsmName, const string stConfPath, const pFunc* aActions, unsigned short iActNum, vector<pSet>* InputObjectName)
  is a constructor for the state machine, where
    - stVfsmName is the name of the state machine (for instance onoff).
    - stConfPath is the path to the directory containing the specification results (for instance "..\\Samples\\OnOff\\Conf".
    - aActions is the pointer to the output functions table.
    - iActNum is the number of output functions.
    - InputObjectName is the address of the set of pointers to sets of input names.

  The 3 last parameters are declared in the file xxx_out.h.

- bool GetSpecInfo( ) reads the information from the specification files required by the Vfsm executor.
- bool Execute(const unsigned short& iEvent) is called if an event occurs (an input changes), where
    - iEvent has to be an Input Name as specified in the StateWORKS Studio (declared in the file xxx.h).
- bool Execute(const unsigned short& iEvent, const CSet& Remove) is called if an event occurs (an input changes), where
    - iEvent has to be an Input Name as specified in the StateWORKS Studio (declared in the file xxx.h).

    - Remove has to be a set of Input Names to be removed while actualising the VI.

- bool Execute(CSet& Events) is called if an event occurs (an input changes), where
    - Event has to be a set of Input Names as specified in the StateWORKS Studio (declared in the file xxx.h).

- bool Execute(CSet& Events, const CSet& Remove) is called if an event occurs (an input changes), where
    - Event has to be a set of Input Names as specified in the StateWORKS Studio (declared in the file xxx.h).
    - Remove has to be a set of Input Names to be removed while actualising the VI.

- string DisplayState( ) delivers a string – the name of the present state.
- string DisplayVI( ) delivers a string – the content of the VI.
- string GetState( ) delivers a number representing the present state.
- void Trace(const bool bTraceEnable, string stFileName = "MyTrace.txt") is used to enable / disable tracing. The trace file with the name stFileName (default "MyTrace.txt") contains all state and VI changes. On entering the Vfsm executor an empty line and the present time is inserted: this arrangement allows to see multiple state changes caused by a single event.

The vfsm.h file contains a number of other methods that cannot be used in programming the application; they are for internal use by the VFSM Executor. Their *private* access specifier disables any attempt to use them.

Writing output functions (actions)

In contrary to an RTDB based application [1][2], for a coded application that uses the Vfsm library the use and meaning of RTDB objects is less relevant. Especially object properties do not play any role, as unless you would try to program something similar to the RTDB (and in such a case we would advise you to just use the RTDB!). Thus, any action independently of its type is just a call to an output function which has to perform an action as identified while specifying the state machine.

Prototypes of all output functions are generated by **Build**ing the state machine configuration in StateWORKS Studio in the file xxx_out.h. The implementation of these functions has to be done by a programmer, typically in a file xxx_out.cpp.

The names of the output functions are composed of the name of the state machine and the Output Name as used in the StateWORKS Studio, for instance: OnOff_Timer_ResetStart. The function may return information required to actualize the VI if applicable. For instance, the OnOff_Timer_ResetStart that starts a timer should return the value to be inserted into VI (RUN if it would be used). In the example OnOff the specification uses only the value _OVER therefore the function OnOff_Timer_ResetStart will return the value 0 (which means nothing to insert).

Signal conversions

The VFSM Executor operates in a virtual environment defined by a state machine specification in the StateWORKS Studio. Using the RTDB, conversions between the real input signals to the virtual control values as well as between the virtual output values to real output functions are done automatically by the real time data base. The RTDB holds the entire control structure of the system and handles the communication between objects which are the basic elements of the system containing both the real properties of signals and their virtual values.

The Vfsm library contains only the VFSM Executor. We could implement the conversion between the signals in the real world and the virtual environment of the Executor while writing the application. As this task is an error-prone one (changes in the specification may change the conversion rules) the StateWORKS Studio delivers (in the file xxx_out.h) a set of constants which are used in the methods Execute() for these real/virtual conversions and for actualisation of the VI. These constants are automatically adjusted to any change in the specification.

Tracing

The VFSM Executor has a trace facility which can be enabled or disabled using the method Trace(). If enabled all changes of VI and of states are written into the file xxx_trace.txt. The time on entering the executor is written also into the trace file. If the trace facility is disabled the trace content is redirected onto the operator console.

The tracing facility is an essential debugging tool displaying the control flow while processing the state machines by the VFSM Executor. An excerpt of a trace file illustrates its content:

```
Sat Dec 08 17:20:14 2007
Trace file opened **********

Sat Dec 08 17:20:14 2007
Slave executing
vi = { always (1) }
    state = Idle (2)
vi = { always (1) }

Sat Dec 08 17:20:19 2007
Master executing
vi = { always (1), MyCmd_DoSomething (2) }
    state = Done (4)
vi = { always (1) }

Sat Dec 08 17:20:19 2007
Slave executing
vi = { always (1), Cmd_Start (2) }
    state = Busy (3)
vi = { always (1), Cmd_Start (2) }

Sat Dec 08 17:20:21 2007
Slave executing
vi = { always (1), Cmd_Start (2), OK (5) }
    state = Done (4)
    vi = { always (1), Cmd_Start (2), OK (5) }
```

Limitations

The constants provided by StateWORKS Studio in the file xxx-out.h do not cover the complement value. If a developer uses the complement values in specification of the state machine he has to provide the interface between the real input/outputs and virtual environment of the VFSM Executor in his code. As this becomes a relatively difficult and error prone task we suggest use of the RTDB in such a case. Anyway, we do provide a variant of the Execute method that allows actualisation of VI by hand.

References

[1] Wagner, F. et al., *Modeling Software with Finite State Machines – A Practical Approach*, Auerbach Publications, New York, 2006.

[2] SW Software, RTDB Programmers Guide.pdf. Release 5.0.6. 2006

.