Использование конечных автоматов в KVS

Скрипты на KVS выполняются в среде реального времени, в которой события сети могут произойти в любой момент времени . Когда поведение управляемой событиями программы зависит от порядка, в котором происходят события, написание такой программы становится очень запутанным, а значит, и более сложным для отладки и внесения изменений. Инженеры по программному обеспечению давно используют конечные автоматы в качестве организующего принципа при разработке управляемых событиями программ.

Дисциплина, которая диктуется конечными автоматами, придает проекту строгость, заменяя запутанную логику простыми таблицами, в результате программы становятся проще в реализации и в тестировании.

Конечные автоматы моделируют поведение, при котором реакции на будущие события зависят от предыдущих событий. Существует огромный пласт научной литературы по этой теме, но удобное рабочее определение очень простое. Конечный автомат - это компьютерная программа, которая состоит из:

- Событий, на которые реагирует программа;
- Состояний, в которых программа пребывает между событиями;
- Переходов между состояниями при реагировании на события;
- Действий, выполняемых в процессе переходов;
- Переменных, которые содержат значения, необходимые для выполнения действий между событиями.

Конечные автоматы наиболее полезны в ситуациях, в которых поведение управляется многими различными типами событий, а реакция на определенное событие зависит от последовательности предыдущих событий. События, которые управляют конечными автоматами, могут быть внешними по отношению к компьютеру и исходить от клавиатуры, мыши, таймера или сетевой активности, или внутренними, исходящими от других компонентов прикладной программы или от других приложений.

Состояния - это метод запоминания предыдущих событий, а переходы - метод организации реагирования на будущие события. Одно из событий должно быть помечено как исходное состояние. Может существовать также конечное состояние, но это необязательно.

Существует два распространенных представления конечных автоматов:

1. Ориентированные графы
   Эллипсы представляют состояния, а стрелки между ними - переходы, над которыми указаны события и действия.
2. Двумерные таблицы
   Столбцы и строки представляют события и состояния, а ячейки содержат действия и переходы.

Эти представления эквивалентны, но делают акцент на разных аспектах проекта. Оба представления полезны и используются в данной статье.

Разработка управляемых событиями программ с помощью конечных автоматов несколько сложнее обычного процедурного программирования; такая разработка требует больше дисциплины в общем и больше проектной работы в частности. При хорошем исполнении в результате можно получить более простой программный код, меньшую продолжительность тестирования и облегченное сопровождение. Несмотря на это, сложность подхода конечных автоматов оправдана не для всех управляемых событиями программ. Если, например, диапазон событий невелик, или действия, запускаемые событиями, всегда одни и те же, дополнительные затраты ресурсов на разработку могут не оправдаться.

Рассмотрим решение прикладной задачи при помощи метода конечного автомата. А именно требуется обеспечить возможность возвращения своего ника даже, если он уже кем-то используется при заходе в сеть, проидентифицироваться и зайти на некий защищенный канал после успешной идентификации.

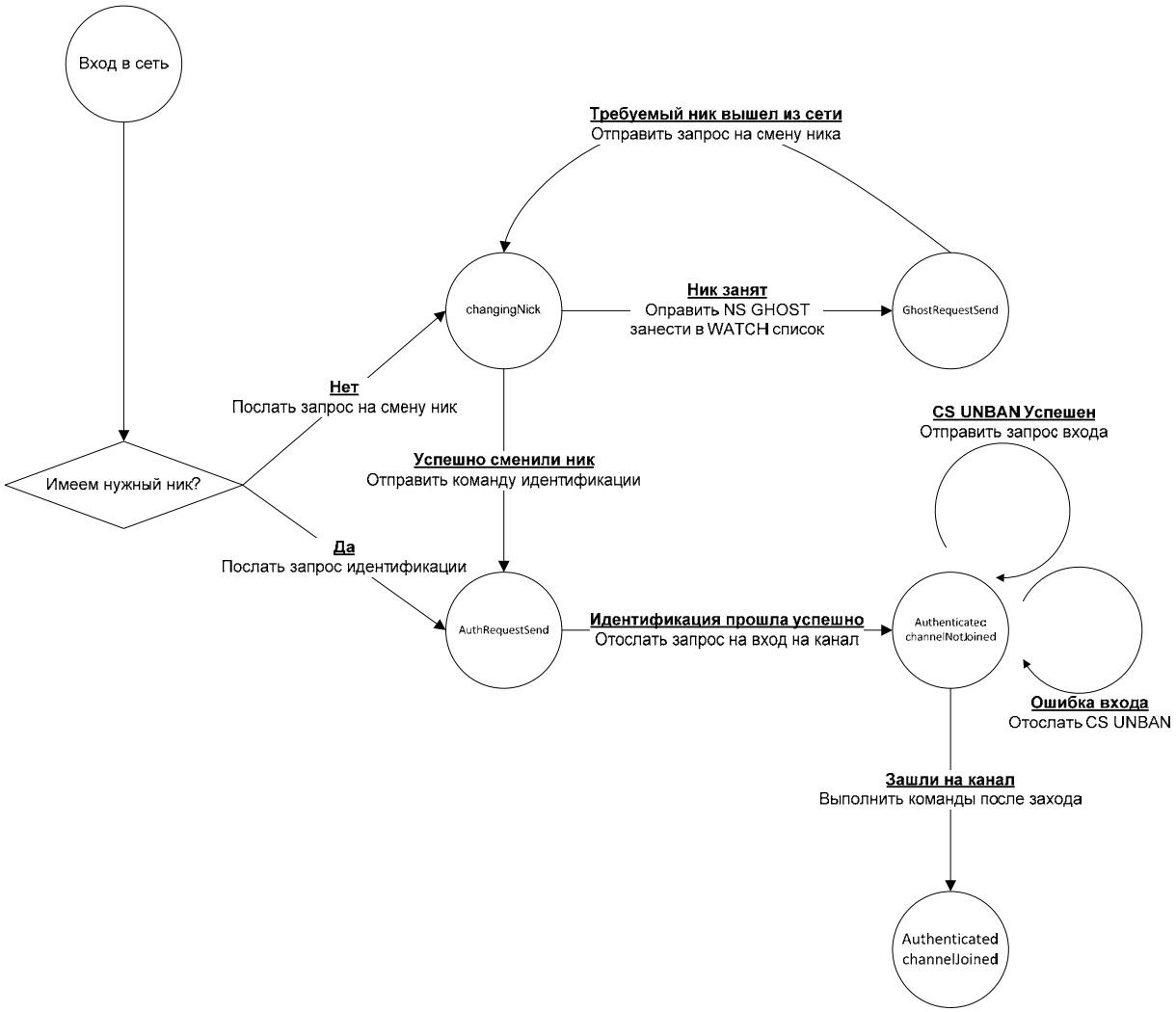
Опишем состояния конечного автомата:

- NickBusy - Требуется возврат ника (мы находимся в сети под другим ником)
- GhostRequestSend - Запрос на возврат ника отослан
- ChangingNick – меняем ник на нужный
- AuthRequestSend - Ник возвращен, отослана команда идентификации
- Authenticated_channelNotJoined -Ник возвращен, проидентифицирован, на канал еще не зашли
- Authenticated_channelJoined -Ник возвращен, проидентифицирован, зашли на канал

Опишем обрабатываемые события:

- NickAlreadyOnline- требуемый ник в онлайне
- NickOffline – ник ушел в оффлайн
- NickChange – мы сменили ник
- AuthOk – мы идентифицированы
- ChannelJoined – зашли на канал
- CannotJoinChannel – не могу зайти на канал
- UnbanCommandOk – команда CS UNBAN выполнена успешно

В случае, если все проходит так, как оно должно проходить, мы должны иметь примерно такую последовательность событий:



Опишем таблицу состояний:

	GhostRequestSend	AuthRequestSend	ChangingNick	Authenticated_channelNotJoined	Authenticated_channelJoined
NickAlreadyOnline			**SendGhost** Отправить NS GHOST, занести ник в WATCH список Перейти к GhostRequestSend		
NickOffline	**RequestNickChange** Сменить на нужный ник Перейти к ChangingNick				
NickChange			**SendAuth** Идентифицироваться Перейти к AuthRequestSend		
AuthOk		**SendJoin** Отослать запрос на вход на канал Перейти к Authenticated_channelNotJoined			
ChannelJoined				**SendOnJoin** Выполнить какие-то	

				действия после захода на канал, перейти к Authenticated_channelJoined	
CannotJoinChannel				**SendUnban** Отослать CS NBAN, состояние не менять	
UnbanCommandOk				**SendJoin** Отослать JOIN, состояние не менять	

Таким образом видим, что у нашего автомата существует 6 различных функций переходов.

Создадим пространство имен FinitiveStateMachine::Handlers и поместим туда обработчики этих событий. Назовем их также, как указано в таблице выше. Также туда поместим специальную функцию обработки события, не предусмотренного в таблице выше. Она поможет нам, если мы всеже не предусмотрели какое-либо из возможных вариантов событий. Назовем ее UnexpectedSituation.

Теперь создадим алиас **FinitiveStateMachine::Init**, в котором инициализируем наш конечный автомат.

Сделаем это примерно так:

```
%FSMData{"GhostRequestSend"}{NickOffline}{"next_state"} = ChangingNick
%FSMData{"GhostRequestSend"}{NickOffline}{"handler"} = RequestNickChange

%FSMData{"AuthRequestSend"}{AuthOk}{"next_state"} = Authenticated_channelNotJoined
%FSMData{"AuthRequestSend"}{AuthOk}{"handler"} = SendJoin

%FSMData{"ChangingNick"}{"NickAlreadyOnline"}{"next_state"} = GhostRequestSend
%FSMData{"ChangingNick"}{"NickAlreadyOnline"}{"handler"} = SendGhost
%FSMData{"ChangingNick"}{NickChange}{"next_state"} = AuthRequestSend
%FSMData{"ChangingNick"}{NickChange}{"handler"} = SendAuth

%FSMData{"Authenticated_channelNotJoined"}{ChannelJoined}{"next_state"} = Authenticated_channelJoined
%FSMData{"Authenticated_channelNotJoined"}{ChannelJoined}{"handler"} = SendOnJoin
%FSMData{"Authenticated_channelNotJoined"}{CannotJoinChannel}{"next_state"} = Authenticated_channelNotJoined
%FSMData{"Authenticated_channelNotJoined"}{CannotJoinChannel}{"handler"} = SendUnban
%FSMData{"Authenticated_channelNotJoined"}{UnbanCommandOk}{"next_state"} = Authenticated_channelNotJoined
%FSMData{"Authenticated_channelNotJoined"}{UnbanCommandOk}{"handler"} = SendJoin
```

Теперь создадим алиас, который по названию произошедшего осбытия и текущему состоянию будет вызывать функцию перехода и менять состияние. Назовем ее **FinitiveStateMachine::HandleEvent.**

```
%event=$0
%function = %FSMData{%FSMCurrentState}{%event}{"handler"}
if(%function)
{
eval "FinitiveStateMachine::Handlers::"%function
} else {
FinitiveStateMachine::Handlers::UnhandledSituation %event
}
%nextState = %FSMData{%FSMCurrentState}{%event}{"next_state"}
if(%nextState)
{
%FSMCurrentState = %nextState
}
```

Как видим, функция выбирает нужный обработчик из ассоциативного массива, созданного выше, и вызывает его. Если же обработчика не существует, то вызывается обработчик по умолчанию. Не правда ли, совсем просто? А главное, нам нет никакой нужды помнить, каким путем конечный автомат оказался в данной ситуации. Он просто выполняет команды перехода, которые, в свою очередь, не обязаны вообще знать когда и для чего они выполняются.

Это позволяет крайне эффективно организовать работу системы, и тратить минимум усилий на ее отладку. Проиллюстрируем сказанное выше примерами обработчиков.

Обработчик **RequestNickChange**:

```
nick Alexey
```

Обработчик **SendAuth**:

```
raw -q Nickserv identify MyC00lPassword
```

Обработчик **SendGhost**:

```
raw –q WATCH +Alexey

raw -q nickserv ghost Alexey MyC00lPassword
```

Обработчик **SendJoin**:

```
join #myC00lChannel
```

Обработчик **SendOnJoin**:

```
raw -q ChanServ op #myC00lChannel
```

Обработчик **SendUnban**:

```
raw -q ChanServ unban #myC00lChannel
```

Обработчик **UnhandledSituation**:

```
debug "unhandled event $0 in state %FSMCurrentState"
```

Замечу, что для поддержки различных ников и/или сетей, следует поменять только алиасы, описанные выше. На работу автомата это никак влиять не будет.

Теперь нам нужно написать обработчики реальных событий IRC сети, приводящих в действие нашу систему, и вызывающих функцию **FinitiveStateMachine::HandleEvent.**

```
event(433,FSM-handler)
{
	if($my.network == "WeNet")
	{
		FinitiveStateMachine::HandleEvent NickAlreadyOnline
	}
}


event(440,FSM-HANDLER)
{
	if($my.network == "WeNet")
	{
		FinitiveStateMachine::HandleEvent AuthOk
```

```
        }
}


event(474,FSM-HANDLER)
{
        if($my.network == "WeNet")
        {
                if($3=="#myC00lChannel")
                {
                        FinitiveStateMachine::HandleEvent CannotJoinChannel
                }
        }
}


event(601,FSM-HANDLER)
{
        if($my.network == "WeNet")
        {
                if($3=="Alexey")
                {
                        FinitiveStateMachine::HandleEvent NickOffline
                }
        }
}
event(OnChanServNotice,FSM-HANDLER)
{
        if($my.network == "WeNet")
        {
                if($str.contains($3,"все баны")
                {
                        FinitiveStateMachine::HandleEvent UnbanCommandOk
                }
        }
}
event(OnIRC,FSM-Handler)
{
        if($my.network == "WeNet")
        {
                FinitiveStateMachine::Init
                if($me == "Alexey")
                {
                        FinitiveStateMachine::Handlers::SendAuth
                        %FSMCurrentState = AuthRequestSend
                } else {
                        FinitiveStateMachine::Handlers::RequestNickChange
                        %FSMCurrentState = ChangingNick
                }
        }
}
event(OnMeJoin,FSM-HANDLER)
{
        if($my.network == "WeNet")
        {
                if($chan.name == "# myC00lChannel")
                {
                        FinitiveStateMachine::HandleEvent ChannelJoined
                }
        }
}
event(OnMeNickChange,FSM-HANDLER)
{
        if($my.network == "WeNet")
        {
                if($1 == "Alexey")
```

```
                {
                        FinitiveStateMachine::HandleEvent NickChange
                }
        }
}
event(OnNickServAuth,FSM-handler)
{
        if($my.network == "WeNet")
        {
                FinitiveStateMachine::HandleEvent AuthOk
        }
}
event(OnNotifyOnline,FSM-HANDLER)
{
        if($my.network == "WeNet")
        {
                if($0 == "Alexey")
                {
                        FinitiveStateMachine::HandleEvent NickOffline
                }
        }
}
```

Автомат готов к применению!