Pós-Graduação em Ciência da Computação

"TDevC Specification and Tools"

Por

Rafael Melo Macieira

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

Recife, Setembro/2011

Universidade Federal de Pernambuco
Centro de Informática
Pós-graduação em Ciência da Computação

Rafael Melo Macieira

"TDevC Specification and Tools"

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientadora: *Edna Natividade da Silva Barros*

Recife, Setembro/2011

Eu dedico esta dissertaï¿½ï¿½o ï¿½ minha famï¿½lia,
amigos, professores e todos aqueles que me apoiaram nesta
etapa da minha vida.

Agradecimentos

A rule is a rule and let's face it! Without rules there's chaos!

—KRAMER, C.

Resumo

Abstract

Sumário

Lista de Figuras

Lista de Tabelas

1 TDevC Language

1.1 Informal View of TDevC HSM

All service declaration in TDevC permits to specify links between others services of the device. These links determine if another service can or cannot be executed after a certain service execution. This means that the device execution maintains a main flow of service calls that can be represented as a FSM. A main service flow (device flow) is import to know the current execution state of the device, infered from this state machine.

Regarding each service, they are formed by a sequence of statements and service calls that represents their behavior. As the device flow FSM, this sequence inside a service can be represented as another FSM. So, each state of the main FSM contains another FSM inside.

Besides the hierarchy between the device flow FSM and service behavior FSM, there is another hierarchical dependency between services. Every service call inside a service represents a transition to a state which is indeed another service FSM.

These hierarchies are specified here as a hierarchical state machine called TDevC HSM. The TDevC HSM is defined as a set of customized DFAs hierarchically distributed. These customized DFAs are classified in three types:

- Device-DFA (DDFA): This is the highest level DFA of the TDevC HSM. This DFA represents the device's state, i.e., the relationship between the device's services. Each state of it represents another customized DFA called SDFA. This DFA will be better explained in section 2.2.1.

- Service-DFA (SDFA): This DFA represents the behavior of the services. For a clearly understanding, their flow are splited in two (*regular* and *non-regular*) and depending on the flow type, thier states can be others SDFAs, flow execution points

and other customized DFA called NEDFA. This DFA will be better explained in section 2.2.2.

- Non-regular Event-DFA (NEDFA): This DFA represents states of the non-regular flow of the SDFAs. This DFA will be better explained in section 2.2.3.

In an overview, this approach provides, for each device, one DDFA representing the device's execution flow, which each state of this DFA is a SDFAs. This SDFAs, representing the services' execution flow, contain transitions between their states, equivalent to statement executions (linking flow execution points) and service calls.

When a transition in a SDFA is a service call, the target state must be a SDFA or a NEDFA. For SDFA target states, the transition must be in the regular flow and for NEDFA target states, non-regular flow.

A regular flow in a service flow means all sequence of statements and service calls with a fixed and predefined moment to happen. In the other hand, a non-regular flow is all sequence of statements and service calls inside a service that might occurs any moment in the service execution. Every non-regular flow must ends with a empty transition to the regular flow state that triggered the non-regular flow transition.

1.2 Formal Definition of TDevC HSM

Let's start the formal definition with some delcarations:

- K is the set of all specification's DFAs;
- Q is the set of all SDFAs, where $Q \subset K$;
- M is the set of all NEDFAs, where $M \subset K$;
- Z is the set of all "service calls";
- R is the set of the reject states.

1.2.1 DDFA

A DDFA can be defined as a "six-tuple": $D = (Q \cup \{s_f\}, \Sigma_d, \delta_d, q_0, F_d, r_0)$, onde:

- $Q \cup \{s_f\}$ is the set of all DDFA's states, where s_f is the *standby* state.
- Σ_s is the alphabet (input elements), where $\Sigma_s \subset Z$.

- δ_s is the transition function, where $\delta_s(q_i, e) \rightarrow q_j$, onde q_i e $q_j \in Q$ e $e \in \Sigma_s$
- q_0 is the initial state, where $q_0 \in Q$
- F_s is the set of the final states of the DDFA. For all DDFA, $F_s = \{s_f\}$.
- r_0 is the reject state, where $r_0 \in R$

1.2.2 SDFA

Assim, um SDFA $Q_i \in Q$ pode ser representado por $S = (Q_i \cup U_i \cup M_i, \Sigma_s, \delta_s, u_0, F_s, r_0)$, onde:

- $Q_i \cup U_i \cup M_i$ is the set of all SDFA's states, where:
  - $Q_i \subset Q$,
  - $U_i \subset U$, where U is the set of intermediate states, i. e., states that represent the flow execution points;
  - $M_i \subset Q \cup U$, where M_i is the set of all NEDFAs of the current SDFA's non-regular flow.
- Σ_q is the alphabet (input elements), where $\Sigma_q \subset T \cup Z$, and:
  - T is the set of flow elements (statements), where $T = A \cup C \cup D$;
  - A is the set of "assignment";
  - C is the set of "conditions";
  - Y is the set of "delays";
- δ_q is the transition function, where $\delta_q(qu_i, e) \rightarrow qu_j$, onde:
  - if $qu_i \in M_i$ and $qu_j \in Q_i \cup U_i$ then $e = \varepsilon$.
- u_0 is the initial state, where $u_0 \in U$ is the set of the final states of the SDFA. For all SDFA,
- F_q is the set of the final states of the SDFA, where $F_q \subset F$ and F is the set of all final states of the device.
- r_0 is the reject state, where $r_0 \in R$

The Z elements are composed by a number representing the phisical address of the function call or a index identifing the service.

On the other hand, the T elements can have the following formats:

- $e_a \in A$, where e_a =(value, address, access_type);
- $e_c \in C$, where e_c =(logic_expression);
- $e_d \in D$, where $e_d = tick|scale$;

1.2.3 NEDFA

A NEDFA $N_i \in Q$ can be defined as $N = (Q_i \cup U_i, \Sigma_n, \delta_n, u_0, F_n, r_0)$, where:

- $Q_i \cup U_i$ is the set of all NEDFA's states, where:
  - $Q_i \subset Q$;
  - $U_i \subset U$.
- Σ_n is the alphabet (input elements), where $\Sigma_q \subset T \cup Z$, and:
- δ_n is the transition function, where $\delta_q(qu_i, e) \rightarrow qu_j$, onde:
- u_0 is the initial state, where $u_0 \in U$ is the set of the final states of the SDFA. For all SDFA,
- F_q is the set of the final states of the SDFA, where $F_q \subset F$ and F is the set of all final states of the device.
- r_0 is the reject state, where $r_0 \in R$

1.2.4 Relationship Between DDFA, SDFA and NEDFA

2
TDevC HSM

2.1 Informal View of TDevC HSM

All service declaration in TDevC permits to specify links between others services of the device. These links determine if another service can or cannot be executed after a certain service execution. This means that the device execution maintains a main flow of service calls that can be represented as a FSM. A main service flow (device flow) is import to know the current execution state of the device, infered from this state machine.

Regarding each service, they are formed by a sequence of statements and service calls that represents their behavior. As the device flow FSM, this sequence inside a service can be represented as another FSM. So, each state of the main FSM contains another FSM inside.

Besides the hierarchy between the device flow FSM and service behavior FSM, there is another hierarchical dependency between services. Every service call inside a service represents a transition to a state which is indeed another service FSM.

These hierarchies are specified here as a hierarchical state machine called TDevC HSM. The TDevC HSM is defined as a set of customized DFAs hierarchically distributed. These customized DFAs are classified in three types:

- Device-DFA (DDFA): This is the highest level DFA of the TDevC HSM. This DFA represents the device's state, i.e., the relationship between the device's services. Each state of it represents another customized DFA called SDFA. This DFA will be better explained in section 2.2.1.

- Service-DFA (SDFA): This DFA represents the behavior of the services. For a clearly understanding, their flow are splited in two (*regular* and *non-regular*) and depending on the flow type, thier states can be others SDFAs, flow execution points

and other customized DFA called NEDFA. This DFA will be better explained in section 2.2.2.

- Non-regular Event-DFA (NEDFA): This DFA represents states of the non-regular flow of the SDFAs. This DFA will be better explained in section 2.2.3.

In an overview, this approach provides, for each device, one DDFA representing the device's execution flow, which each state of this DFA is a SDFAs. This SDFAs, representing the services' execution flow, contain transitions between their states, equivalent to statement executions (linking flow execution points) and service calls.

When a transition in a SDFA is a service call, the target state must be a SDFA or a NEDFA. For SDFA target states, the transition must be in the regular flow and for NEDFA target states, non-regular flow.

A regular flow in a service flow means all sequence of statements and service calls with a fixed and predefined moment to happen. In the other hand, a non-regular flow is all sequence of statements and service calls inside a service that might occurs any moment in the service execution. Every non-regular flow must ends with a empty transition to the regular flow state that triggered the non-regular flow transition.

2.2 Formal Definition of TDevC HSM

Let's start the formal definition with some delcarations:

- K is the set of all specification's DFAs;
- Q is the set of all SDFAs, where $Q \subset K$;
- M is the set of all NEDFAs, where $M \subset K$;
- Z is the set of all "service calls";
- R is the set of the reject states.

2.2.1 DDFA

A DDFA can be defined as a "six-tuple": $D = (Q \cup \{s_f\}, \Sigma_d, \delta_d, q_0, F_d, r_0)$, onde:

- $Q \cup \{s_f\}$ is the set of all DDFA's states, where s_f is the *standby* state.
- Σ_s is the alphabet (input elements), where $\Sigma_s \subset Z$.

- δ_s is the transition function, where $\delta_s(q_i, e) \rightarrow q_j$, onde q_i e $q_j \in Q$ e $e \in \Sigma_s$
- q_0 is the initial state, where $q_0 \in Q$
- F_s is the set of the final states of the DDFA. For all DDFA, $F_s = \{s_f\}$.
- r_0 is the reject state, where $r_0 \in R$

2.2.2 SDFA

Assim, um SDFA $Q_i \in Q$ pode ser representado por $S = (Q_i \cup U_i \cup M_i, \Sigma_s, \delta_s, u_0, F_s, r_0)$, onde:

- $Q_i \cup U_i \cup M_i$ is the set of all SDFA's states, where:
  - $Q_i \subset Q$,
  - $U_i \subset U$, where U is the set of intermediate states, i. e., states that represent the flow execution points;
  - $M_i \subset Q \cup U$, where M_i is the set of all NEDFAs of the current SDFA's non-regular flow.
- Σ_q is the alphabet (input elements), where $\Sigma_q \subset T \cup Z$, and:
  - T is the set of flow elements (statements), where $T = A \cup C \cup D$;
  - A is the set of "assignment";
  - C is the set of "conditions";
  - Y is the set of "delays";
- δ_q is the transition function, where $\delta_q(qu_i, e) \rightarrow qu_j$, onde:
  - if $qu_i \in M_i$ and $qu_j \in Q_i \cup U_i$ then $e = \varepsilon$.
- u_0 is the initial state, where $u_0 \in U$ is the set of the final states of the SDFA. For all SDFA,
- F_q is the set of the final states of the SDFA, where $F_q \subset F$ and F is the set of all final states of the device.
- r_0 is the reject state, where $r_0 \in R$

The Z elements are composed by a number representing the phisical address of the function call or a index identifing the service.

On the other hand, the T elements can have the following formats:

- $e_a \in A$, where e_a =(value, address, access_type);
- $e_c \in C$, where e_c =(logic_expression);
- $e_d \in D$, where $e_d = tick|scale$;

2.2.3 NEDFA

A NEDFA $N_i \in Q$ can be defined as $N = (Q_i \cup U_i, \Sigma_n, \delta_n, u_0, F_n, r_0)$, where:

- $Q_i \cup U_i$ is the set of all NEDFA's states, where:
  - $Q_i \subset Q$;
  - $U_i \subset U$.
- Σ_n is the alphabet (input elements), where $\Sigma_q \subset T \cup Z$, and:
- δ_n is the transition function, where $\delta_q(qu_i, e) \rightarrow qu_j$, onde:
- u_0 is the initial state, where $u_0 \in U$ is the set of the final states of the SDFA. For all SDFA,
- F_q is the set of the final states of the SDFA, where $F_q \subset F$ and F is the set of all final states of the device.
- r_0 is the reject state, where $r_0 \in R$

2.2.4 Relationship Between DDFA, SDFA and NEDFA

Appendices

A

Descrição BNF da Temporal DevC

<input> ::= <tdevcimport> <tdevcspec>

<tdevcimport> ::= **import** *id:string* **;**
| **EMPTY**

<tdevcspec> ::= **device(** *id:string* **) {** <object_definition> **}**

<object_definition> ::= <port_definition> **;** <object_definition>
| <pattern_definition> **;** <object_definition>
| <service_definition> **;** <object_definition>
| ***** <service_definition> **;** <object_definition>
| <requirement_definition> **;** <object_definition>
| <exception_definiton> **;** <object_definition>
| <format_definition> **;** <object_definition>
| <storage_definition> **;** <object_definition>
| <behaviors_definition> **;** <bindings_definition> **;**

<pattern_definition> ::= **pattern** *id:string* **=** *val:int*
| **pattern** *id:string* **= mask (** <bits> **)**

<bits> ::= *mask:mask_rep* <bits>
| **.** <bits>
| *mask:mask_rep*
| **.**

<format_definition> ::= **format** *id:string* **{** <fields_decl> **}**

<service_definition> ::= **service** *id:string*

<storage_definition> ::= <visibility> <reg_decl>
| <var_decl>

<reg_decl> ::= **register** *id:string* **(** *val:int* **)** **{** <fields_decl> **}**
| **register** *id:string* **(** *val:int* **)** **alias** *id:string* **{** <fields_decl> **}**
| **register** *id:string* **(** *val:int* **)** **=** *id:string*
| **register** *id:string* **(** *val:int* **)** **alias** *id:string* **=** *id:string*

<fields_decl> ::= <field_decl> **;**
| <field_decl> **;** <fields_decl>

<field_decl> ::= <rw> *id:string* **[** *val:int* **]**
| <rw> *id:string* **[** *start:int* **:** *end:int* **]**
| **reserved** **{** <fields_decl> **}**
| <rw> **reserved** **[** *start:int* **:** *end:int* **]**
| <rw> **reserved** **[** *val:int* **]**
| *id:string* **{** <fields_decl> **}**

<rw_ex> ::= **READ**
| **WRITE**

<rw> ::= **READ**
| **WRITE**
| **RW**

<behaviors_definition> ::= behaviors **{** <behaviors_body> **}**

<behaviors_body> ::= <action_def> <behaviors_body>
| <action_def>

<bindings_definition> ::= bindings **{** <bindings_body> **}**

<bindings_body> ::= *id:string* **requires** <id_list> <bind_order> ;
| *id:string* **requires_and** <id_list> <bind_order> ;
| *id:string* **always_requires** <id_list> ;
| *id:string* **requires** <id_list> <bind_order> ; <bindings_body>
| *id:string* **requires_and** <id_list> <bind_order> ;
<bindings_body>
| *id:string* **always_requires** <id_list> ; <bindings_body>
| *id:string* **. addconstraints (** <id_list> **)** ;
| *id:string* **. addconstraints (** <id_list> **)** ; <bindings_body>
| *id:string* **. addtriggers (** <id_list> **)** ;
| *id:string* **. addtriggers (** <id_list> **)** ; <bindings_body>

<expr> ::= **<l_expr>**
| **<assign>** ;
| **<random_assign>**
| $<$ *sinc_decl* $>$;
|**<l_expr><expr>**
|**<assign>**;**<expr>**
|**<random_assign>**
|**<sinc_decl>**;**<expr>**

<random_assign> ::= **<assign>**;| $*$ **<assign>**;
|**<random_assign><assign>**;
|**<random_assign>** $*$ **<assign>**;

<l_expr> ::= **<expr_cond><expr>**

<expr_cond> ::= **<cond><logic_oper><expr_cond>**
|**<cond>**
|**<expr_cond_block><logic_oper><expr_cond>**
|**<expr_cond_block>**

<expr_cond_block> ::= (**<expr_cond>**)

<assign> ::= **<string>** = **<int>**
|**<string>**.**<string>** = **<int>**
|**<string>**.**<string>**.**<string>** = **<int>**

<sinc_decl> ::= **<wait_decl>**
|**<intr_decl>**
|**<poll_decl>**

<wait_decl> ::= *wait_state*(**<cond>** >

<intr_decl> ::= *interrupt*(**<cond>**)

<poll_decl> ::= *polling*(**<cond>**)

<cond> ::= **<string><oper><int>**
|**<string>**.**<string><oper><int>**
|**<string>**.**<string>**.**<string><oper><int>**
|**<string>** > **<int>**
|**<string>**.**<string>** > **<int>**
|**<string>**.**<string>**.**<string>** > **<int>**
|**<string>** < **<int>**
|**<string>**.**<string>** < **<int>**
|**<string>**.**<string>**.**<string>** < **<int>**

<mask_rep> ::= [01.]
<int> ::= $[0-9]+$
<string> ::= $[\_a-zA-Z][\_a-zA-Z0-9]^*$
<logic_oper> ::= && | ||
<oper> ::=== |! = | <= | >=