### 21.3.10 The Call test component behaviour operation

The call operation is used start a test behaviour on a test component and wait until that behaviour has terminated.

***Syntactical Structure***

*ObjectReference* "." **call** "(" ( *FunctionInstance | AltstepInstance* )

*[*"," *SimpleExpression* ] ")"

[ "->" [**value** *Ref*] [**verdict** Ref] ]

[ **catch** "(" **timeout** ")" *StatementBlock* ]

[ catch "(" **stop** ")" *StatementBlock* ]

***Semantic Description***

Similar to the start operation on test components which is not blocking, the blocking **call** operation implicitly uses a **start** operation, but waits until either the started behaviour has terminated or some timeout has occurred.

A timeout duration in seconds can be given explicitly in the form of a *SimpleExpression* as an additional parameter to the call operation. If no timeout duration is given, an infinite timeout duration is used.

The actions taken by the call operation are dependent on whether the execution of the started behaviour is complete or incomplete. Complete execution occurs when the started function is terminated by executing a return statement or if it reaches the end of the function body. If the started behaviour is terminated for any other reason, the execution is incomplete.

If the i incomplete execution occurs because the called component was stopped or killed and a **catch stop** clause is added to the call operation, the *StatementBlock* of that clause is executed before the call operation terminates.

If the started behaviour does not terminate in the given timeout duration and a **catch timout** clause is added to the call operation, the called component is implicitly stopped and the *StatementBlock* of the catch timeout clause is executed before the call operation terminates.

In all other cases when theexecution is incomplete, the call operation ends with a test case error.

After complete execution of the started behaviour, the **out** and **inout** actual parameters given to the actual parameter list of the called function or altstep instance will be updated in the same manner as if it was a normal function/altstep invocation.

Additionally, a redirect clause can be added to the operation which allows assignment of the return result (in case that the called function has a return clause) to a variable via the **value** clause and also the assignment of the termination verdict of the called component via the **verdict** clause.

In all cases of incomplete execution, the variables referenced in the value and verdict clause or in **out** and **inout** actual parameters will stay unchanged and no assignment will be made.

If the called component is not created alive and has already been started or called once or if it has been killed, additional call operations are not allowed.

***Restrictions***

In addition to the general static rules of TTCN‑3 given in clauses 5 and 21 and shown in table 16, the following restrictions apply:

1. The *ObjectReference* shall be of a component type.
2. The function or altstep invoked in a **call** test component operation shall have a **runs on** definition referencing a component type to which the called component is compatible (see clause 6.3.3).
3. Ports, defaults and timers shall not be passed into a function or altstep invoked in a **call** test component operation. All formal parameter types of the behaviour shall neither be of a port, default or timer type nor should contain a direct or indirect element or field of a port, default or timer type.
4. The return value of the function or altstep invoked from a **call** test component operation neither be of a port, default or timer type nor should contain a direct or indirect element or field of a port, default or timer type.
5. The optional *SimpleExpression* representing the timer value shall be of a float type.
6. The optional **catch** **timeout** clause may be present only if the timeout value has been defined.
7. The variable in the value clause shall be compatible with the return value of the invoked function.
8. The variable in the verdict clause shall be of type verdicttype.

***Examples***

 **function** f\_myFirstBehaviour() **runs on** MyComponentType { … }

 **function** f\_mySecondBehaviour() **runs on** MyComponentType { … }

 **function** f\_myThirdBehaviour(out integer p\_p1, inout integer p\_p2)

 **runs** **on** MyComponentType

 return integer { … }

 **altstep** a\_myFourthBehaviour() **runs on** MyComponentType { ... }

 :

 **var** MyComponentType v\_myNewPTC;

 **var** MyComponentType v\_myAlivePTC;

 **var integer** v\_out, v\_inout := 0, v\_result;

 :

 v\_myNewPTC := MyComponentType.**create**; // Creation of a new non-alive test component.

 v\_myAlivePTC := MyComponentType.**create alive**; // Creation of a new alive-type test component

 :

 v\_myNewPTC.**call**(f\_myFirstBehaviour()); // Call to the non-alive component.

 v\_myNewPTC.**call**(f\_mySecondBehaviour()); // Test case error

 :

 v\_myAlivePTC.**call**(f\_myFirstBehaviour()); // Call to the alive-type component

 v\_myAlivePTC.**call**(f\_mySecondBehaviour()); // Another call to the same component

 :

 v\_myAlivePTC.**call**(f\_myThirdBehaviour(v\_out,v\_inout)) // v\_out/v\_inout can be changed

-> value v\_result verdict v\_verdict; // v\_result/v\_verdict are assigned on successful
 // termination

v\_myAlivePTC.**call**(a\_myFourthBehaviour()); // Direct call of an altstep behaviour