Test Suite Modularization: Module Containment

Modules, whose identifiers contain any of the following substrings, shall contain only certain definition types, permissible for the particular type of module. These substrings will be referred to as "module restrictions" in the following, to establish a unified terminology and simplify the descriptions. All module restrictions are case-sensitive. The configuration entries are following the schema "check{ModuleRestriction}ModuleContainmentCheck", where "ModuleRestriction" is to be substituted with the particular substring. Apart from the permissible definition types, all modules allow the presence of import and group definitions. The output for the quality checks indicates the location/scope of the definition (starting line - end line) violating the particular constraints and the type of module restriction that has been recognized and applied (based on the substring that has been found in the module name). Note that, if a module name contains multiple restrictions, all the checks will be applied individually. This means that a "TypesAndValuesAndTemplates" module will first be checked for the permissible definition types for a "TypesAndValues" module and throw a warning on any other definition (be it a definition of the permissible definitions for a "!Template" module), and then the other way around for the permissible definitions in a "!Templates" module.

Note that control part definitions are allowed in all the following module containment checks. This may be subject to changes in the future (restricted to the "TestControl" modules only).

The details for the individual module restrictions are outlined below.

TypesAndValues Module Must Contain Only Type and Constant Definitions

- **Symbolic Name in XML Configuration**: checkTypesAndValuesModuleContainmentCheck
- **Dependant Tags in XML Configuration**: -

Any module that contains the substring *TypesAndValues* in its name will be analyzed. Only type and constant definitions are permitted within such a module. The example below illustrates a module which will produce warnings if this quality check is enabled. The first three definitions are type and constant definitions. However, templates, testcases, and functions must not be present in such a module. Thus, these last three module definitions will be reported as not of the permissible types.

Example:

```
module checkTypesAndValuesModuleContainmentCheckBad {
    import from LibCommon_time all;

    // good part
    type record MyRecordType {
        integer field1 optional,
        charstring field2,
```

```
        boolean field3
    }

    type integer myInt;

    const myInt myIntValue := 2;
    // bad part

    template MyRecordType MyTemplate1 := {
        field1 := 123,
        field2 := "A string",
        field3 := true
    }

    testcase t_sendMsg() runs on myComponent {
    }

    function myfunc() {
    }
        group g_1{

                function f_1(){
                }
        }
}
```

Notes: Component and port type definitions also fall under the generic "type definitions" term. Therefore, they are also allowed in a "TypesAndValues" module. This may be subject to changes in the future.

Templates Module Must Contain Only Template Definitions

- **Symbolic Name in XML Configuration**: checkTemplatesModuleContainmentCheck
- **Dependant Tags in XML Configuration**: -

Any module that contains the substring *Templates* in its name will be analyzed. Similar to the "TypesAndValues" module restriction, only template definitions are permitted within such a module. The example below illustrates a module which will produce warnings if this quality check is enabled. The first three definitions are template definitions (one is within a group). The constant and function definitions that follow, however, are not allowed within a "Templates" module and therefore will produce a warning.

Example:

```
module checkTemplatesModuleContainmentBad {
    import from LibCommon_time all;
```

```
    // good part, only templates
    template MyRecordType t_1 := {
        field1 := 1,
        field2 := "a",
        field3 :=true
    }

    template integer t_2 := (0,1,2,3,4,5,6,7);


    group GroupedDefs{
            template integer t_3 :=(0 .. 10);
                //bad part
                const integer c_1 := 2;

    }

        //bad part
        function f_2(){
        }

}
```

Functions Module Must Contain Only Function and Altstep Definitions

- **Symbolic Name in XML Configuration**: checkFunctionsModuleContainmentCheck
- **Dependant Tags in XML Configuration**: checkFunctionsModuleContainmentCheckAllowExtFunction

Any module that contains the substring *Functions* in its name will be analyzed. Similar to the other module restrictions, only function and altstep definitions are permitted within such a module. The example below illustrates a module which will produce warnings if this quality check is enabled. Additionally, external function definitions may (by default) or may not be allowed within such modules, depending on whether **checkFunctionsModuleContainmentCheckAllowExtFunction** is enabled or not.

Example:

```
module checkFunctionsModuleContainmentBad {
    import from LibCommon_time all;
    // good part, only functions and altsteps
    function f_1 () {
    }
    altstep a_1 () {
    }
    //bad part
    type record typeA {
```

```
        integer field1,
        boolean feild2
    }
    group GroupedDefs {
        function f_2 () {
        }
        altstep a_2 () {
        }
        const integer c_1 := 1;
    }
    //configurable part
    //depending on the configuration external functions
    //may (default) or may not be allowed
    external function ef_1 ();
}
```

Testcases Module Must Contain Only Testcase and Function Definitions That Are Referenced In Start Statements

- **Symbolic Name in XML Configuration**: checkTestcasesModuleContainmentCheck
- **Dependant Tags in XML Configuration**: -

Any module that contains the substring *Testcases* in its name will be analyzed. Similar to the other module restrictions, only testcase and function definitions that are referenced in start statements are permitted within such a module. The example below illustrates a module which will produce warnings if this quality check is enabled.

Example:

```
module checkTestcasesModuleContainmentBad {
    import from LibCommon_sync all;

    // bad module
        function f_1() runs on component_1{
        }

        testcase tc_1() runs on component_1{
                        var component_1 ptc0 := component_1.create;
                        ptc0.start(f_1());
                        ptc0.start(f_0()); // defined elsewhere
                        f_3(); //does not count - not in a start statement
        }
        //not used in a start statement
        function f_3() runs on component_1{
        }

    //not a testcase or a function\
```

```
    const integer c_1 := 0;


    group GroupedDefs{
        //not used in a start statement
                function f_4() runs on component_1 return float{
                }

                function f_2() runs on component_1{
                        var component_1 ptc1 := component_1.create;
                        ptc1.start(f_1());
                        ptc1.start(f_0(f_4())); // does not count - used as a nested parameter
                        f_4(); //does not count - not in a start statement
                        timer t;
                        t.start(f_4()); //does not count - used in a timer start operation

                }

                testcase tc_2() runs on component_1{
                        var component_1 ptc2 := component_1.create;
                        ptc2.start(f_1());
                        ptc2.start(f_0()); // defined elsewhere

                }

    }
}
```

ModuleParams Module Must Contain Only Modulepar Definitions

- **Symbolic Name in XML Configuration**: checkModuleParamsModuleContainmentCheck
- **Dependant Tags in XML Configuration**: -

Any module that contains the substring *ModuleParams* in its name will be analyzed. Similar to the other module restrictions, only modulepar definitions are permitted within such a module. The example below illustrates a module which will produce warnings if this quality check is enabled.

Example:

```
module checkModuleParamsModuleContainmentBad {
    import from LibCommon_time all;

    // good part, only module parameters
        modulepar integer mp_1;

        modulepar {
```

```
            integer mp_2;
            charstring mp_3;
        }

    //bad part
    const integer c_1 := 1;

    group GroupedDefs{
            modulepar integer mp_4;
            const integer c_2 := 2;
    }
}
```

Interface Module Must Contain Only Component and Port Definitions

- **Symbolic Name in XML Configuration**: checkInterfaceModuleContainmentCheck
- **Dependant Tags in XML Configuration**: -

Any module that contains the substring *Interface* in its name will be analyzed. Similar to the other module restrictions, only component and port type definitions are permitted within such a module. The example below illustrates a module which will produce warnings if this quality check is enabled.

Example:

```
module checkInterfaceModuleContainmentBad {
    import from LibCommon_time all;

    // bad module

        type component component_1 {
        }

        type port port_1 message{
                in integer
        }
        const integer c_1 := 0;

        group g_1{
                type component component_2 {
                }
                type port port_2 message{
                        out integer
                }

                const integer c_2 := 1;
        }
```

```
        //are control parts permissible?
        control {

        }

}
```

TestSystem Module Must Contain Only Component and Port Definitions

- **Symbolic Name in XML Configuration**: checkTestSystemModuleContainmentCheck
- **Dependant Tags in XML Configuration**: -

Any module that contains the substring *TestSystem* in its name will be analyzed. Similar to the other module restrictions, only component type definitions are permitted within such a module. The example below illustrates a module which will produce warnings if this quality check is enabled.

Example:

```
module checkTestSystemModuleContainmentBad {
    import from LibCommon_time all;

    // bad module, port definitions as well

        type component component_1 {
        }

        type port port_1 message{
                in integer
        }

        group g_1{
                type component component_2 {
                }
                type port port_2 message{
                        out integer
                }


        }

        //are control parts permissible?
        control {

        }
```

```
}
```

TestControl Module Must Contain Only Control Part Definition

- **Symbolic Name in XML Configuration**: checkTestControlModuleContainmentCheck
- **Dependant Tags in XML Configuration**: -

Any module that contains the substring *TestControl* in its name will be analyzed. Similar to the other module restrictions, only control part definition is permitted within such a module. The example below illustrates a module which will produce warnings if this quality check is enabled.

Example:

```
module checkTestControlModuleContainmentBad {
    import from LibCommon_time all;

    //bad part

        modulepar integer mp_1;

        modulepar {
                integer mp_2;
                charstring mp_3;
        }


    const integer c_1 := 1;

    group GroupedDefs{
                modulepar integer mp_4;
                const integer c_2 := 2;
    }

        control {

        }

}
```