BAMBERGER BEITRÄGE

ZUR WIRTSCHAFTSINFORMATIK UND ANGEWANDTEN INFORMATIK

ISSN 0937-3349

Nr. 94

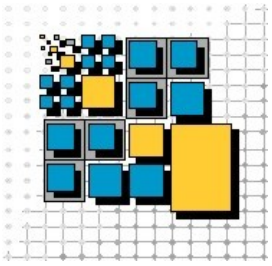
The Static Analysis Rules of the BPEL Specification: Analysis, Evaluation and Implementation

Christian R. Preißinger, Simon Harrer

August 2014

FAKULTÄT WIRTSCHAFTSINFORMATIK UND ANGEWANDTE INFORMATIK

OTTO-FRIEDRICH-UNIVERSITÄT BAMBERG



Distributed Systems Group

Otto-Friedrich Universität Bamberg

An der Weberei 5, 96052 Bamberg, GERMANY

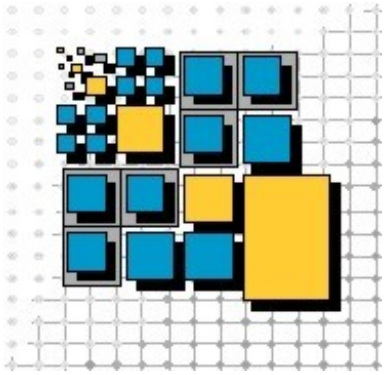
Prof. Dr. rer. nat. Guido Wirtz

Due to hardware developments, strong application needs and the overwhelming influence of the internet, distributed systems have become one of the most important topics for nowadays software industry. Owing to their ever increasing importance for everyday business, distributed systems have high requirements with respect to dependability, robustness and performance. Unfortunately, distribution adds its share to the problems of developing complex systems. Heterogeneity in both, hardware and software, frequent changes, concurrency, distribution of components and the need for inter-operability between systems complicate matters. Moreover, new technical aspects like resource management, load balancing and guaranteeing consistent operation in the presence of partial failures put an additional burden onto the developer. *Our long-term research goal is the development, implementation and evaluation of methods helpful for the realization of robust and easy-to-use software for complex systems in general while putting a focus on the problems and issues regarding distributed systems on all levels.* This includes design methods, visual languages and tools for distributed systems development as well as middleware, SOA and cloud computing issues. Our current research activities focus on different aspects centered around that theme:

- *Implementation of Business Processes and Business-to-Business-Integration (B2Bi)*: Starting from requirements for successful B2Bi development processes, languages and systems, we investigate the practicability and inter-operability of different approaches and platforms for the design and implementation of business processes.
- *Quality, esp. Robustness, Standard-conformance, Portability, Compatibility and Performance of Process-based Software and Service-oriented Systems*: In both, industry and academia, process languages have emerged, e.g. Windows Workflow (WF), Business Process Model and Notation (BPMN) and Web Services Business Process Execution Language (WS-BPEL). Although widely used in practice, current implementations of these languages and models are far from perfect. We work on metrics to compare such languages w.r.t. expressive power, conformance and portability as well as additional quality properties, such as installability, replaceability, adaptability and inter-operability. These metrics are developed and validated formally as well as evaluated practically. In the context of BPMN, we work on tools to check for and improve the standard compliance for human-centric process models on different layers of abstraction. Runtime environments for process languages with a focus on BPEL are investigated by means of a framework that eases the comparative test of different run-times and process engines.
- *Cloud Application Portability:* The hype surrounding the Cloud has lead to a variety of offerings that span the whole cloud stack. We examine important aspects of portability in cloud environments and enhance the portability of cloud applications by applying common standards between heterogeneous clouds. We make use of a holistic view of the cloud including important aspects like cloud specific restrictions, platform configurations, the deployment and life cycle of cloud applications.
- *Visual Programming- and Design-Languages*: The goal of this long-term effort is the utilization of visual metaphors and visualization techniques to make design- and programming languages more understandable and, hence, more easy-to-use. Currently, languages for designing and programming sensor networks are at the focus of this effort.

More information about our work can be found at `www.uni-bamberg.de/en/pi/`. If you have any questions or suggestions regarding this report or our work, don't hesitate to contact us.

Bamberg, June 2014 Guido Wirtz



The Static Analysis Rules of the BPEL Specification: Analysis, Evaluation and Implementation

Christian R. Preißinger, Simon Harrer

Abstract The Static Analysis Rules of the BPEL Specification: Analysis, Evaluation and Implementation

Keywords SOA, BPEL, conformance testing, static analysis, validation

Contact:

christian-roland.preissinger@uni-bamberg.de, simon.harrer@uni-bamberg.de

Contents

1 Static Analysis Rule Tests **2**

1.1 SA00001 3

1.2 SA00002 4

1.3 SA00003 5

1.4 SA00004 52

1.5 SA00005 52

1.6 SA00006 54

1.7 SA00007 55

1.8 SA00008 56

1.9 SA00009 58

1.10 SA00010 58

1.11 SA00011 61

1.12 SA00012 62

1.13 SA00013 62

1.14 SA00014 63

1.15 SA00015 64

1.16 SA00016 65

1.17 SA00017 65

1.18 SA00018 67

1.19 SA00019 67

1.20 SA00020 68

1.21 SA00021 70

1.22 SA00022 73

1.23 SA00023 73

1.24 SA00024 75

1.25 SA00025 75
1.26 SA00026 77
1.27 SA00027 78
1.28 SA00028 78
1.29 SA00029 78
1.30 SA00030 78
1.31 SA00031 78
1.32 SA00032 79
1.33 SA00033 83
1.34 SA00034 83
1.35 SA00035 84
1.36 SA00036 85
1.37 SA00037 85
1.38 SA00038 86
1.39 SA00039 86
1.40 SA00040 86
1.41 SA00041 86
1.42 SA00042 86
1.43 SA00043 87
1.44 SA00044 87
1.45 SA00045 87
1.46 SA00046 89
1.47 SA00047 89
1.48 SA00048 91
1.49 SA00050 93
1.50 SA00051 95
1.51 SA00052 95

1.52 SA00053 95
1.53 SA00054 96
1.54 SA00055 97
1.55 SA00056 98
1.56 SA00057 102
1.57 SA00058 103
1.58 SA00059 104
1.59 SA00060 105
1.60 SA00061 106
1.61 SA00062 109
1.62 SA00063 109
1.63 SA00064 110
1.64 SA00065 110
1.65 SA00066 111
1.66 SA00067 112
1.67 SA00068 113
1.68 SA00069 113
1.69 SA00070 113
1.70 SA00071 116
1.71 SA00072 117
1.72 SA00073 117
1.73 SA00074 117
1.74 SA00075 117
1.75 SA00076 119
1.76 SA00077 119
1.77 SA00078 122
1.78 SA00079 122

1.79 SA00080 124
1.80 SA00081 124
1.81 SA00082 126
1.82 SA00083 126
1.83 SA00084 127
1.84 SA00085 127
1.85 SA00086 129
1.86 SA00087 129
1.87 SA00088 131
1.88 SA00089 131
1.89 SA00090 132
1.90 SA00091 132
1.91 SA00092 133
1.92 SA00093 134
1.93 SA00094 135
1.94 SA00095 136

References **137**

2 List of previous University of Bamberg reports **138**

List of Figures

List of Tables

1 Tagged Rules excluding out of scope rules. 2

2 SA00001 test pairs 3

3 SA00002 test pairs 4

4 SA00003 test pairs 51

5 SA00005 test pairs 53

6 SA00006 test pairs 55

7 SA00007 test pairs 56

8 SA00008 test pairs 57

9 SA00010 test pairs 61

10 SA00011 test pairs 61

11 SA00012 test pairs 62

12 SA00013 test pairs 62

13 SA00014 test pairs 64

14 SA00015 test pairs 65

15 SA00016 test pairs 65

16 SA00017 test pairs 66

17 SA00018 test pairs 67

18 SA00019 test pairs 68

19 SA00020 test pairs 70

20 SA00021 test pairs 72

21 SA00022 test pairs 73

22 SA00023 test pairs 75

23 SA00024 test pairs 75

24 SA00025 test pairs 76

25 SA00032 test pairs 83

26 SA00034 test pairs 84
27 SA00035 test pairs 85
28 SA00036 test pairs 85
29 SA00037 test pairs 85
30 SA00044 test pairs 87
31 SA00045 test pairs 88
32 SA00046 test pairs 89
33 SA00047 test pairs 91
34 SA00048 test pairs 93
35 SA00050 test pairs 94
36 SA00051 test pairs 95
37 SA00052 test pairs 95
38 SA00053 test pairs 96
39 SA00054 test pairs 97
40 SA00055 test pairs 98
41 SA00056 test pairs 101
42 SA00057 test pairs 103
43 SA00058 test pairs 104
44 SA00059 test pairs 104
45 SA00060 test pairs 106
46 SA00061 test pairs 109
47 SA00062 test pairs 109
48 SA00063 test pairs 110
49 SA00064 test pairs 110
50 SA00065 test pairs 111
51 SA00066 test pairs 112
52 SA00067 test pairs 112

53 SA00068 test pairs 113
54 SA00069 test pairs 113
55 SA00070 test pairs 116
56 SA00071 test pairs 116
57 SA00072 test pairs 117
58 SA00076 test pairs 119
59 SA00077 test pairs 121
60 SA00078 test pairs 122
61 SA00079 test pairs 123
62 SA00080 test pairs 124
63 SA00081 test pairs 125
64 SA00082 test pairs 126
65 SA00083 test pairs 126
66 SA00084 test pairs 127
67 SA00085 test pairs 128
68 SA00086 test pairs 129
69 SA00087 test pairs 130
70 SA00088 test pairs 131
71 SA00089 test pairs 131
72 SA00090 test pairs 132
73 SA00091 test pairs 132
74 SA00092 test pairs 133
75 SA00093 test pairs 134
76 SA00095 test pairs 136

Abbreviations

BPEL Web Service Business Process Execution Language

QName qualified name

WSDL Web Service Description Language

1 Static Analysis Rule Tests

group	tag	Σ	rules
violation check	constrainedNode	19	1, 3, 13, 15, 17, 24, 35, 36, 45, 47, 50, 53, 54, 57, 62, 76, 78, 80, 91
	choice	18	16, 17, 19, 20, 25, 32, 34, 47, 51, 52, 55, 59, 63, 80, 81, 83, 85, 90
	unique	15	2, 14, 18, 22, 23, 44, 64, 66, 67, 68, 69, 76, 86, 92, 93
	consistentRedundancy	14	5, 11, 12, 34, 35, 36, 37, 46, 48, 57, 58, 79, 86, 87
	location	8	6, 7, 8, 61, 65, 70, 71, 79
	executionInstructions	4	84, 88, 89, 95
	resolveToDefinition	4	10, 65, 86, 95
	controlCycleDetection	2	72, 82
target activities	WSDL	22	1, 2, 5, 10, 11, 12, 13, 14, 19, 20, 22, 45, 46, 47, 48, 50, 53, 54, 58, 84, 87, 88
	messageActivities	21	5, 10, 46, 47, 48, 50, 51, 52, 53, 54, 55, 58, 59, 61, 63, 78, 84, 85, 87, 89, 90
	messageAssignment	14	47, 48, 50, 51, 52, 53, 54, 55, 58, 59, 63, 85, 87, 90
	processAndScope	14	3, 18, 23, 44, 61, 78, 79, 80, 82, 83, 88, 91, 92, 93
	fct	12	3, 6, 7, 8, 10, 70, 71, 78, 79, 80, 81, 93
	flow	10	64, 65, 66, 67, 68, 69, 70, 71, 72, 82
	partnerLink	9	5, 10, 16, 17, 18, 35, 36, 37, 84
	variable	9	10, 23, 24, 25, 34, 48, 58, 86, 90
	XSD	6	10, 11, 12, 13, 14, 45
	assignment	6	10, 32, 34, 35, 36, 37
	correlation	5	10, 44, 45, 46, 88
	eventHandler	5	83, 86, 88, 89, 95
	loops	4	62, 70, 76, 83
	startActivities	3	15, 57, 62

Table 1: Tagged Rules excluding out of scope rules.

1.1 SA00001

"A WS-BPEL processor MUST reject a WS-BPEL that refers to solicit-response or notification operations portTypes."[2, p. 194]

The SA rule #1 describes two possible error types that are excluded in the formal model by [1, p. 27]. These errors narrow the definition of the four possible message exchange patterns [4, see section 2.4] down to two, namely, one-way and request-response. In terms of implementation, these patterns differ in the order and existence of `<output>` and `<input>` messages for an `<operation>` in a WSDL definition. Hence, the permutation of both `<output>` and `<input>` in a WSDL `<operation>` definition creates four combinations. Each combination refers to a specific message exchange pattern. Consequently, two of these combinations are forbidden whereas the remaining two are valid. We solely change the locations of the Web Service Description Language (WSDL)s in the derived Web Service Business Process Execution Language (BPEL) processes and modified the WSDLs (see Table 2).

Notification includes TestInterface-Notification.wsdl which has a `<operation name="notification">` just containing an `<output>` in a separate `<portType>`.

SolicitResponse imports TestInterface-SolicitResponse.wsdl that defines an `<output>` followed by `<input>` in `<operation name="solicitResponse">` contained in a separate `<portType>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
ReceiveReply	**Notification**	1
ReceiveReply	**SolicitResponse**	1

Table 2: SA00001 test pairs

1.2 SA00002

"A WS-BPEL processor MUST reject any WSDL portType definition that includes overloaded operation names."[2, p. 194]

The SA rule #2 ensures that the `@name` of an `<operation>` is unique within its `<portType>` in a WSDL definition. The negation of the formalization of this rule in [1, p. 11] reveals that only a single test is required which contains a name duplicate.

OverloadedOperationNames solely imports a modified WSDL TestInterface-Overloaded-OperationNames.wsdl that overloaded the operation name `startProcessSync` by overwriting the name attribute in the `startProcessSyncString` operation in `<portType>` and `<binding>`. Thus, the operation `startProcessSync` exists twice, but with different messages.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
ReceiveReply	**OverloadedOperationNames**	1

Table 3: SA00002 test pairs

1.3 SA00003

"If the value of exitOnStandardFault of a <scope> or <process> is set to 'yes', then a fault handler that explicitly targets the WS-BPEL standard faults MUST NOT be used in that scope."[2, p. 194]

Due to inheritance rules of `@exitOnStandardFault` in both `<process>` and `<scope>`, we can distinguish between two different inheritance situations: 1) `<process>` and `<scope>` explicitly defining the value, and 2) a `<scope>` and `<process>` use the implicit value, namely, a `<scope>` that inherits the value `"yes"` for the attribute or a `<process>` that uses the default value. The exceptions are derived from the specification which lists all available standard faults [3, pp. 193], except for the `joinCondition` as this standard fault is explicitly excluded in the rule definition. In [1, p. 56], the same list of standard faults is used, as in their definition they exclude the standard fault `joinCondition` as well. This results in 152 combinations, of which half are valid combinations whereas the other half corresponds to invalid ones, as `exitOnStandardFault="yes"` always holds for all erroneous tests.

Combinations for rule #3 for `<process>`. This is complete as every possible state of a `<process>` as well as every possible standard fault for this case is listed. There are 19 test cases, namely, one for each standard fault with the `<process>` having `exitOnStandardFault="yes"`.

`<catch>` with `faultName="bpel:VALUE"`
`VALUE` ∈ [ambiguousReceive, completionConditionFailure, conflictingReceive, conflictingRequest, correlationViolation, invalidBranchCondition, invalidExpressionValue, invalidVariables, mismatchedAssignmentFailure, missingReply, missingRequest, scopeInitializationFailure, selectionFailure, subLanguageExecutionFault, uninitializedPartnerRole, uninitializedVariable, unsupportedReference, xsltInvalidSource, xsltStylesheetNotFound]
×
[`exitOnStandardFault="yes"`, `exitOnStandardFault="no"`, default `exitOnStandardFault="no"`]

Formalization 1: SA Rule #3 for `<process>`

Combinations for rule #3 for `<scope>`. To combat test explosion, we considered to include both explicit setting of `@exitOnStandardFault` and inheriting the value from an enclosing `<scope>` or `<process>`. But we did not include arbitrary large inheritance dependencies, e.g., `<scope>` in `<scope>` in `<scope>`, as this is not feasible. In this case, we have twelve test cases when the `<scope>` has `exitOnStandardFault="yes"` set explicitly, and three when the `<scope>` inherits `exitOnStandardFault="yes"` from the enclosing `<scope>` that has set `exitOnStandardFault="yes"` explicitly, and two when it inherits the explicitly set `exitOnStandardFault="yes"` from the enclosing `<process>` via the enclosing `<scope>` or directly from the enclosing `<process>`.

`<catch>` with `faultName="bpel:VALUE"`
`VALUE` ∈ [ambiguousReceive, completionConditionFailure, conflictingReceive, conflictingRequest, correlationViolation, invalidBranchCondition, invalidExpressionValue, invalidVariables, mismatchedAssignmentFailure, missingReply, missingRequest, scopeInitializationFailure, selectionFailure, subLanguageExecutionFault, uninitializedPartnerRole, uninitializedVariable, unsupportedReference, xsltInvalidSource, xsltStylesheetNotFound]
×
[`exitOnStandardFault="yes"`, `exitOnStandardFault="no"`, inherited]
×
enclosing `<scope>` [`exitOnStandardFault="yes"`, `exitOnStandardFault="no"`, inherited, no enclosing scope]
×
enclosing `<process>` [`exitOnStandardFault="yes"`, `exitOnStandardFault="no"`, default]

Formalization 2: SA Rule #3 for `<scope>`

ProcessExitAndCatchAR has the `exitOnStandartFault` set to `yes` in `<process>` that contains the `<faultHandlers>` directly. The fault that is caught is the `bpel:ambiguousReceive`.

ScopeInheritedScopeInhertedProcessYesCatchAR has the `exitOnStandartFault` set to `yes` in `<process>` which is inherited by a `<scope>` that contains the `<faultHandlers>` via an intermediate `<scope>` which inherits the value itself. The fault that is caught is the `bpel:ambiguousReceive`.

ScopeInheritedScopeYesProcessDefaultCatchAR has the `exitOnStandartFault` default value in `<process>` and a `<scope>` which sets the value to `yes`. A nested `<scope>` inherts the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:ambiguousReceive`.

ScopeInheritedScopeYesProcessNoCatchAR has the `exitOnStandartFault` set to `no` in `<process>` and a `<scope>` which sets the value to `yes`. A nested `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:ambiguousReceive`.

ScopeInheritedScopeYesProcessYesCatchAR has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` with the same value. A nested `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:ambiguousReceive`.

ScopeInheritProcessYesCatchAR has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:ambiguousReceive`.

ScopeYesProcessDefaultCatchAR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:ambiguousReceive`.

ScopeYesProcessNoCatchAR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:ambiguousReceive`.

ScopeYesProcessYesCatchAR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:ambiguousReceive`.

ScopeYesScopeInhertedProcessDefaultCatchAR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:ambiguousReceive`.

ScopeYesScopeInhertedProcessNoCatchAR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:ambiguousReceive`.

ScopeYesScopeInhertedProcessYesCatchAR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:ambiguousReceive`.

ScopeYesScopeNoProcessDefaultCatchAR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:ambiguousReceive`.

ScopeYesScopeNoProcessNoCatchAR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:ambiguousReceive`.

ScopeYesScopeNoProcessYesCatchAR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:ambiguousReceive`.

ScopeYesScopeYesProcessDefaultCatchAR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `yes`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:ambiguousReceive`.

ScopeYesScopeYesProcessNoCatchAR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `yes`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:ambiguousReceive`.

ScopeYesScopeYesProcessYesCatchAR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `yes`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:ambiguousReceive`.

ProcessExitAndCatchCCF has the `exitOnStandartFault` set to `yes` in `<process>` that contains the `<faultHandlers>` directly. The fault that is caught is the `bpel:completionConditionFa`

ScopeInheritedScopeInhertedProcessYesCatchCCF has the `exitOnStandartFault` set to `yes` in `<process>` which is inherited by a `<scope>` that contains the `<faultHandlers>` via an intermediate `<scope>` which inherits the value itself. The fault that is caught is the `bpel:completionConditionFailure`.

ScopeInheritedScopeYesProcessDefaultCatchCCF has the `exitOnStandartFault` default value in `<process>` and a `<scope>` which sets the value to `yes`. A nested `<scope>` inherts the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:completionConditionFailure`.

ScopeInheritedScopeYesProcessNoCatchCCF has the `exitOnStandartFault` set to `no` in `<process>` and a `<scope>` which sets the value to `yes`. A nested `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:completionConditio`

ScopeInheritedScopeYesProcessYesCatchCCF has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` with the same value. A nested `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:completionConditionFailure`.

ScopeInheritProcessYesCatchCCF has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:completionConditionFailure`.

ScopeYesProcessDefaultCatchCCF has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:completionConditionFailure`.

ScopeYesProcessNoCatchCCF has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:completionConditionFailure`.

ScopeYesProcessYesCatchCCF has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:completionConditionFailure`.

ScopeYesScopeInhertedProcessDefaultCatchCCF has a `<scope>` with the `exitOnStandart-Fault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:completionConditionFailure`.

ScopeYesScopeInhertedProcessNoCatchCCF has a `<scope>` with the `exitOnStandart-Fault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:completionConditionFailure`.

ScopeYesScopeInhertedProcessYesCatchCCF has a `<scope>` with the `exitOnStandart-Fault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:completionConditionFailure`.

ScopeYesScopeNoProcessDefaultCatchCCF has a `<scope>` with the `exitOnStandart-Fault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOn-StandartFault` default value. The fault that is caught is the `bpel:completionConditionFailure`.

ScopeYesScopeNoProcessNoCatchCCF has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>`

which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:completionConditionFailure.

ScopeYesScopeNoProcessYesCatchCCF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:completionConditionFailure.

ScopeYesScopeYesProcessDefaultCatchCCF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:completionConditionFailure.

ScopeYesScopeYesProcessNoCatchCCF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:completionConditionFailure.

ScopeYesScopeYesProcessYesCatchCCF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:completionConditionFailure.

ProcessExitAndCatchCRec has the exitOnStandartFault set to yes in <process> that contains the <faultHandlers> directly. The fault that is caught is the bpel:conflictingReceive.

ScopeInheritedScopeInhertedProcessYesCatchCRec has the exitOnStandartFault set to yes in <process> which is inherited by a <scope> that contains the <faultHandlers> via an intermediate <scope> which inherits the value itself. The fault that is caught is the bpel:conflictingReceive.

ScopeInheritedScopeYesProcessDefaultCatchCRec has the exitOnStandartFault default value in <process> and a <scope> which sets the value to yes. A nested <scope> inherts the value and contains the <faultHandlers>. The fault that is caught is the bpel:conflictingReceive.

ScopeInheritedScopeYesProcessNoCatchCRec has the exitOnStandartFault set to no in <process> and a <scope> which sets the value to yes. A nested <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:conflictingReceive.

ScopeInheritedScopeYesProcessYesCatchCRec has the exitOnStandartFault set to yes in <process> and a <scope> with the same value. A nested <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:conflictingReceive.

ScopeInheritProcessYesCatchCRec has the exitOnStandartFault set to yes in <process> and a <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:conflictingReceive.

ScopeYesProcessDefaultCatchCRec has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:conflictingReceive`.

ScopeYesProcessNoCatchCRec has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:conflictingReceive`.

ScopeYesProcessYesCatchCRec has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:conflictingReceive`.

ScopeYesScopeInhertedProcessDefaultCatchCRec has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:conflictingReceive`.

ScopeYesScopeInhertedProcessNoCatchCRec has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:conflictingReceive`.

ScopeYesScopeInhertedProcessYesCatchCRec has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:conflictingReceive`.

ScopeYesScopeNoProcessDefaultCatchCRec has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:conflictingReceive`.

ScopeYesScopeNoProcessNoCatchCRec has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:conflictingReceive`.

ScopeYesScopeNoProcessYesCatchCRec has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:conflictingReceive`.

ScopeYesScopeYesProcessDefaultCatchCRec has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `yes`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:conflictingReceive`.

ScopeYesScopeYesProcessNoCatchCRec has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `yes`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:conflictingReceive`.

ScopeYesScopeYesProcessYesCatchCRec has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `yes`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:conflictingReceive`.

ProcessExitAndCatchCReq has the `exitOnStandartFault` set to `yes` in `<process>` that contains the `<faultHandlers>` directly. The fault that is caught is the `bpel:conflictingRequest`.

ScopeInheritedScopeInhertedProcessYesCatchCReq has the `exitOnStandartFault` set to `yes` in `<process>` which is inherited by a `<scope>` that contains the `<faultHandlers>` via an intermediate `<scope>` which inherits the value itself. The fault that is caught is the `bpel:conflictingRequest`.

ScopeInheritedScopeYesProcessDefaultCatchCReq has the `exitOnStandartFault` default value in `<process>` and a `<scope>` which sets the value to `yes`. A nested `<scope>` inherts the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:conflictingRequest`.

ScopeInheritedScopeYesProcessNoCatchCReq has the `exitOnStandartFault` set to `no` in `<process>` and a `<scope>` which sets the value to `yes`. A nested `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:conflictingRequest`.

ScopeInheritedScopeYesProcessYesCatchCReq has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` with the same value. A nested `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:conflictingRequest`.

ScopeInheritProcessYesCatchCReq has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:conflictingRequest`.

ScopeYesProcessDefaultCatchCReq has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:conflictingRequest`.

ScopeYesProcessNoCatchCReq has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:conflictingRequest`.

ScopeYesProcessYesCatchCReq has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:conflictingRequest`.

ScopeYesScopeInhertedProcessDefaultCatchCReq has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another

<scope> which inherits the exitOnStandartFault value from <process>. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:conflictingRequest.

ScopeYesScopeInhertedProcessNoCatchCReq has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which inherits the exitOnStandartFault value from <process>. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:conflictingRequest.

ScopeYesScopeInhertedProcessYesCatchCReq has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which inherits the exitOnStandartFault value from <process>. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:conflictingRequest.

ScopeYesScopeNoProcessDefaultCatchCReq has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:conflictingRequest.

ScopeYesScopeNoProcessNoCatchCReq has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:conflictingRequest.

ScopeYesScopeNoProcessYesCatchCReq has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:conflictingRequest.

ScopeYesScopeYesProcessDefaultCatchCReq has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:conflictingRequest.

ScopeYesScopeYesProcessNoCatchCReq has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:conflictingRequest.

ScopeYesScopeYesProcessYesCatchCReq has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:conflictingRequest.

ProcessExitAndCatchCV has the exitOnStandartFault set to yes in <process> that contains the <faultHandlers> directly. The fault that is caught is the bpel:correlationViolation.

ScopeInheritedScopeInhertedProcessYesCatchCV has the `exitOnStandartFault` set to `yes` in `<process>` which is inherited by a `<scope>` that contains the `<faultHandlers>` via an intermediate `<scope>` which inherits the value itself. The fault that is caught is the `bpel:correlationViolation`.

ScopeInheritedScopeYesProcessDefaultCatchCV has the `exitOnStandartFault` default value in `<process>` and a `<scope>` which sets the value to `yes`. A nested `<scope>` inherts the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:correlationViolation`.

ScopeInheritedScopeYesProcessNoCatchCV has the `exitOnStandartFault` set to `no` in `<process>` and a `<scope>` which sets the value to `yes`. A nested `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:correlationViolation`.

ScopeInheritedScopeYesProcessYesCatchCV has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` with the same value. A nested `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:correlationViolation`.

ScopeInheritProcessYesCatchCV has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:correlationViolation`.

ScopeYesProcessDefaultCatchCV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:correlationViolation`.

ScopeYesProcessNoCatchCV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:correlationViolation`.

ScopeYesProcessYesCatchCV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:correlationViolation`.

ScopeYesScopeInhertedProcessDefaultCatchCV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:correlationViolation`.

ScopeYesScopeInhertedProcessNoCatchCV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:correlationViolation`.

ScopeYesScopeInhertedProcessYesCatchCV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The

<process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:correlationViolation.

ScopeYesScopeNoProcessDefaultCatchCV has a <scope> with the exitOnStandart-Fault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOn-StandartFault default value. The fault that is caught is the bpel:correlationViolation.

ScopeYesScopeNoProcessNoCatchCV has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandart-Fault set to no. The fault that is caught is the bpel:correlationViolation.

ScopeYesScopeNoProcessYesCatchCV has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandart-Fault set to yes. The fault that is caught is the bpel:correlationViolation.

ScopeYesScopeYesProcessDefaultCatchCV has a <scope> with the exitOnStandart-Fault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOn-StandartFault default value. The fault that is caught is the bpel:correlationViolation.

ScopeYesScopeYesProcessNoCatchCV has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandart-Fault set to no. The fault that is caught is the bpel:correlationViolation.

ScopeYesScopeYesProcessYesCatchCV has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandart-Fault set to yes. The fault that is caught is the bpel:correlationViolation.

ProcessExitAndCatchIBC has the exitOnStandartFault set to yes in <process> that contains the <faultHandlers> directly. The fault that is caught is the bpel:invalidBranchConditi

ScopeInheritedScopeInhertedProcessYesCatchIBC has the exitOnStandartFault set to yes in <process> which is inherited by a <scope> that contains the <faultHandlers> via an intermediate <scope> which inherits the value itself. The fault that is caught is the bpel:invalidBranchCondition.

ScopeInheritedScopeYesProcessDefaultCatchIBC has the exitOnStandartFault default value in <process> and a <scope> which sets the value to yes. A nested <scope> inherts the value and contains the <faultHandlers>. The fault that is caught is the bpel:invalidBranchCondition.

ScopeInheritedScopeYesProcessNoCatchIBC has the exitOnStandartFault set to no in <process> and a <scope> which sets the value to yes. A nested <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:invalidBranchCondition.

ScopeInheritedScopeYesProcessYesCatchIBC has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` with the same value. A nested `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:invalidBranchCondition`.

ScopeInheritProcessYesCatchIBC has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:invalidBranchCondition`.

ScopeYesProcessDefaultCatchIBC has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:invalidBranchCondition`.

ScopeYesProcessNoCatchIBC has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:invalidBranchCondition`.

ScopeYesProcessYesCatchIBC has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:invalidBranchCondition`.

ScopeYesScopeInhertedProcessDefaultCatchIBC has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:invalidBranchCondition`.

ScopeYesScopeInhertedProcessNoCatchIBC has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:invalidBranchCondition`.

ScopeYesScopeInhertedProcessYesCatchIBC has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:invalidBranchCondition`.

ScopeYesScopeNoProcessDefaultCatchIBC has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:invalidBranchCondition`.

ScopeYesScopeNoProcessNoCatchIBC has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:invalidBranchCondition`.

ScopeYesScopeNoProcessYesCatchIBC has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:invalidBranchCondition.

ScopeYesScopeYesProcessDefaultCatchIBC has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:invalidBranchCondition.

ScopeYesScopeYesProcessNoCatchIBC has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:invalidBranchCondition.

ScopeYesScopeYesProcessYesCatchIBC has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:invalidBranchCondition.

ProcessExitAndCatchIEV has the exitOnStandartFault set to yes in <process> that contains the <faultHandlers> directly. The fault that is caught is the bpel:invalidExpressionVal

ScopeInheritedScopeInhertedProcessYesCatchIEV has the exitOnStandartFault set to yes in <process> which is inherited by a <scope> that contains the <faultHandlers> via an intermediate <scope> which inherits the value itself. The fault that is caught is the bpel:invalidExpressionValue.

ScopeInheritedScopeYesProcessDefaultCatchIEV has the exitOnStandartFault default value in <process> and a <scope> which sets the value to yes. A nested <scope> inherts the value and contains the <faultHandlers>. The fault that is caught is the bpel:invalidExpressionValue.

ScopeInheritedScopeYesProcessNoCatchIEV has the exitOnStandartFault set to no in <process> and a <scope> which sets the value to yes. A nested <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:invalidExpressionValue.

ScopeInheritedScopeYesProcessYesCatchIEV has the exitOnStandartFault set to yes in <process> and a <scope> with the same value. A nested <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:invalidExpressionValue.

ScopeInheritProcessYesCatchIEV has the exitOnStandartFault set to yes in <process> and a <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:invalidExpressionValue.

ScopeYesProcessDefaultCatchIEV has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:invalidExpressionValue.

ScopeYesProcessNoCatchIEV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:invalidExpressionValue`.

ScopeYesProcessYesCatchIEV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:invalidExpressionValue`.

ScopeYesScopeInhertedProcessDefaultCatchIEV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:invalidExpressionValue`.

ScopeYesScopeInhertedProcessNoCatchIEV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:invalidExpressionValue`.

ScopeYesScopeInhertedProcessYesCatchIEV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:invalidExpressionValue`.

ScopeYesScopeNoProcessDefaultCatchIEV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:invalidExpressionValue`.

ScopeYesScopeNoProcessNoCatchIEV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:invalidExpressionValue`.

ScopeYesScopeNoProcessYesCatchIEV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:invalidExpressionValue`.

ScopeYesScopeYesProcessDefaultCatchIEV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `yes`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:invalidExpressionValue`.

ScopeYesScopeYesProcessNoCatchIEV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `yes`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:invalidExpressionValue`.

ScopeYesScopeYesProcessYesCatchIEV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `yes`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:invalidExpressionValue`.

ProcessExitAndCatchIV has the `exitOnStandartFault` set to `yes` in `<process>` that contains the `<faultHandlers>` directly. The fault that is caught is the `bpel:invalidVariables`.

ScopeInheritedScopeInhertedProcessYesCatchIV has the `exitOnStandartFault` set to `yes` in `<process>` which is inherited by a `<scope>` that contains the `<faultHandlers>` via an intermediate `<scope>` which inherits the value itself. The fault that is caught is the `bpel:invalidVariables`.

ScopeInheritedScopeYesProcessDefaultCatchIV has the `exitOnStandartFault` default value in `<process>` and a `<scope>` which sets the value to `yes`. A nested `<scope>` inherts the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:invalidVariables`.

ScopeInheritedScopeYesProcessNoCatchIV has the `exitOnStandartFault` set to `no` in `<process>` and a `<scope>` which sets the value to `yes`. A nested `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:invalidVariables`.

ScopeInheritedScopeYesProcessYesCatchIV has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` with the same value. A nested `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:invalidVariables`.

ScopeInheritProcessYesCatchIV has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:invalidVariables`.

ScopeYesProcessDefaultCatchIV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:invalidVariables`.

ScopeYesProcessNoCatchIV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:invalidVariables`.

ScopeYesProcessYesCatchIV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:invalidVariables`.

ScopeYesScopeInhertedProcessDefaultCatchIV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:invalidVariables`.

ScopeYesScopeInhertedProcessNoCatchIV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:invalidVariables`.

ScopeYesScopeInhertedProcessYesCatchIV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:invalidVariables`.

ScopeYesScopeNoProcessDefaultCatchIV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:invalidVariables`.

ScopeYesScopeNoProcessNoCatchIV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:invalidVariables`.

ScopeYesScopeNoProcessYesCatchIV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:invalidVariables`.

ScopeYesScopeYesProcessDefaultCatchIV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `yes`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:invalidVariables`.

ScopeYesScopeYesProcessNoCatchIV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `yes`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:invalidVariables`.

ScopeYesScopeYesProcessYesCatchIV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `yes`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:invalidVariables`.

ProcessExitAndCatchMAF has the `exitOnStandartFault` set to `yes` in `<process>` that contains the `<faultHandlers>` directly. The fault that is caught is the `bpel:mismatchedAssignmentFa`

ScopeInheritedScopeInhertedProcessYesCatchMAF has the `exitOnStandartFault` set to `yes` in `<process>` which is inherited by a `<scope>` that contains the `<faultHandlers>` via an intermediate `<scope>` which inherits the value itself. The fault that is caught is the `bpel:mismatchedAssignmentFailure`.

ScopeInheritedScopeYesProcessDefaultCatchMAF has the `exitOnStandartFault` default value in `<process>` and a `<scope>` which sets the value to `yes`. A nested `<scope>` inherts the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:mismatchedAssignmentFailure`.

ScopeInheritedScopeYesProcessNoCatchMAF has the `exitOnStandartFault` set to `no` in `<process>` and a `<scope>` which sets the value to `yes`. A nested `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:mismatchedAssignme`

ScopeInheritedScopeYesProcessYesCatchMAF has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` with the same value. A nested `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:mismatchedAssignmentFailure`

ScopeInheritProcessYesCatchMAF has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:mismatchedAssignmentFailure`.

ScopeYesProcessDefaultCatchMAF has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:mismatchedAssignmentFailure`.

ScopeYesProcessNoCatchMAF has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:mismatchedAssignmentFailure`.

ScopeYesProcessYesCatchMAF has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:mismatchedAssignmentFailure`.

ScopeYesScopeInhertedProcessDefaultCatchMAF has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:mismatchedAssignmentFailure`.

ScopeYesScopeInhertedProcessNoCatchMAF has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:mismatchedAssignmentFailure`.

ScopeYesScopeInhertedProcessYesCatchMAF has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:mismatchedAssignmentFailure`.

ScopeYesScopeNoProcessDefaultCatchMAF has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another

<scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:mismatchedAssignmentFailure.

ScopeYesScopeNoProcessNoCatchMAF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:mismatchedAssignmentFailure.

ScopeYesScopeNoProcessYesCatchMAF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:mismatchedAssignmentFailure.

ScopeYesScopeYesProcessDefaultCatchMAF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:mismatchedAssignmentFailure.

ScopeYesScopeYesProcessNoCatchMAF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:mismatchedAssignmentFailure.

ScopeYesScopeYesProcessYesCatchMAF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:mismatchedAssignmentFailure.

ProcessExitAndCatchMRep has the exitOnStandartFault set to yes in <process> that contains the <faultHandlers> directly. The fault that is caught is the bpel:missingReply.

ScopeInheritedScopeInhertedProcessYesCatchMRep has the exitOnStandartFault set to yes in <process> which is inherited by a <scope> that contains the <faultHandlers> via an intermediate <scope> which inherits the value itself. The fault that is caught is the bpel:missingReply.

ScopeInheritedScopeYesProcessDefaultCatchMRep has the exitOnStandartFault default value in <process> and a <scope> which sets the value to yes. A nested <scope> inherts the value and contains the <faultHandlers>. The fault that is caught is the bpel:missingReply.

ScopeInheritedScopeYesProcessNoCatchMRep has the exitOnStandartFault set to no in <process> and a <scope> which sets the value to yes. A nested <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:missingReply.

ScopeInheritedScopeYesProcessYesCatchMRep has the exitOnStandartFault set to yes in <process> and a <scope> with the same value. A nested <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:missingReply.

ScopeInheritProcessYesCatchMRep has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:missingReply`.

ScopeYesProcessDefaultCatchMRep has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:missingReply`.

ScopeYesProcessNoCatchMRep has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:missingReply`.

ScopeYesProcessYesCatchMRep has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:missingReply`.

ScopeYesScopeInhertedProcessDefaultCatchMRep has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:missingReply`.

ScopeYesScopeInhertedProcessNoCatchMRep has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:missingReply`.

ScopeYesScopeInhertedProcessYesCatchMRep has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:missingReply`.

ScopeYesScopeNoProcessDefaultCatchMRep has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:missingReply`.

ScopeYesScopeNoProcessNoCatchMRep has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:missingReply`.

ScopeYesScopeNoProcessYesCatchMRep has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:missingReply`.

ScopeYesScopeYesProcessDefaultCatchMRep has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `yes`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:missingReply`.

ScopeYesScopeYesProcessNoCatchMRep has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `yes`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:missingReply`.

ScopeYesScopeYesProcessYesCatchMRep has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `yes`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:missingReply`.

ProcessExitAndCatchMReq has the `exitOnStandartFault` set to `yes` in `<process>` that contains the `<faultHandlers>` directly. The fault that is caught is the `bpel:missingRequest`.

ScopeInheritedScopeInhertedProcessYesCatchMReq has the `exitOnStandartFault` set to `yes` in `<process>` which is inherited by a `<scope>` that contains the `<faultHandlers>` via an intermediate `<scope>` which inherits the value itself. The fault that is caught is the `bpel:missingRequest`.

ScopeInheritedScopeYesProcessDefaultCatchMReq has the `exitOnStandartFault` default value in `<process>` and a `<scope>` which sets the value to `yes`. A nested `<scope>` inherts the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:missingRequest`.

ScopeInheritedScopeYesProcessNoCatchMReq has the `exitOnStandartFault` set to `no` in `<process>` and a `<scope>` which sets the value to `yes`. A nested `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:missingRequest`.

ScopeInheritedScopeYesProcessYesCatchMReq has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` with the same value. A nested `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:missingRequest`.

ScopeInheritProcessYesCatchMReq has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:missingRequest`.

ScopeYesProcessDefaultCatchMReq has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:missingRequest`.

ScopeYesProcessNoCatchMReq has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:missingRequest`.

ScopeYesProcessYesCatchMReq has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:missingRequest.

ScopeYesScopeInhertedProcessDefaultCatchMReq has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which inherits the exitOnStandartFault value from <process>. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:missingRequest.

ScopeYesScopeInhertedProcessNoCatchMReq has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which inherits the exitOnStandartFault value from <process>. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:missingRequest.

ScopeYesScopeInhertedProcessYesCatchMReq has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which inherits the exitOnStandartFault value from <process>. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:missingRequest.

ScopeYesScopeNoProcessDefaultCatchMReq has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:missingRequest.

ScopeYesScopeNoProcessNoCatchMReq has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:missingRequest.

ScopeYesScopeNoProcessYesCatchMReq has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:missingRequest.

ScopeYesScopeYesProcessDefaultCatchMReq has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:missingRequest.

ScopeYesScopeYesProcessNoCatchMReq has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:missingRequest.

ScopeYesScopeYesProcessYesCatchMReq has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope>

which sets the exitOnStandartFault to yes. The <process> has the exitOnStandart-Fault set to yes. The fault that is caught is the bpel:missingRequest.

ProcessExitAndCatchSIF has the exitOnStandartFault set to yes in <process> that contains the <faultHandlers> directly. The fault that is caught is the bpel:scopeInitializationFailu

ScopeInheritedScopeInhertedProcessYesCatchSIF has the exitOnStandartFault set to yes in <process> which is inherited by a <scope> that contains the <faultHandlers> via an intermediate <scope> which inherits the value itself. The fault that is caught is the bpel:scopeInitializationFailure.

ScopeInheritedScopeYesProcessDefaultCatchSIF has the exitOnStandartFault default value in <process> and a <scope> which sets the value to yes. A nested <scope> inherts the value and contains the <faultHandlers>. The fault that is caught is the bpel:scopeInitializationFailure.

ScopeInheritedScopeYesProcessNoCatchSIF has the exitOnStandartFault set to no in <process> and a <scope> which sets the value to yes. A nested <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:scopeInitializationFailu

ScopeInheritedScopeYesProcessYesCatchSIF has the exitOnStandartFault set to yes in <process> and a <scope> with the same value. A nested <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:scopeInitializationFailure.

ScopeInheritProcessYesCatchSIF has the exitOnStandartFault set to yes in <process> and a <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:scopeInitializationFailure.

ScopeYesProcessDefaultCatchSIF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:scopeInitializationFailure.

ScopeYesProcessNoCatchSIF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:scopeInitializationFailure.

ScopeYesProcessYesCatchSIF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:scopeInitializationFailure.

ScopeYesScopeInhertedProcessDefaultCatchSIF has a <scope> with the exitOnStandart-Fault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which inherits the exitOnStandartFault value from <process>. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:scopeInitializationFailure.

ScopeYesScopeInhertedProcessNoCatchSIF has a <scope> with the exitOnStandart-Fault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which inherits the exitOnStandartFault value from <process>. The

<process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:scopeInitializationFailure.

ScopeYesScopeInhertedProcessYesCatchSIF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which inherits the exitOnStandartFault value from <process>. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:scopeInitializationFailure.

ScopeYesScopeNoProcessDefaultCatchSIF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:scopeInitializationFailure.

ScopeYesScopeNoProcessNoCatchSIF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:scopeInitializationFailure.

ScopeYesScopeNoProcessYesCatchSIF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:scopeInitializationFailure.

ScopeYesScopeYesProcessDefaultCatchSIF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:scopeInitializationFailure.

ScopeYesScopeYesProcessNoCatchSIF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:scopeInitializationFailure.

ScopeYesScopeYesProcessYesCatchSIF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:scopeInitializationFailure.

ProcessExitAndCatchSF has the exitOnStandartFault set to yes in <process> that contains the <faultHandlers> directly. The fault that is caught is the bpel:selectionFailure.

ScopeInheritedScopeInhertedProcessYesCatchSF has the exitOnStandartFault set to yes in <process> which is inherited by a <scope> that contains the <faultHandlers> via an intermediate <scope> which inherits the value itself. The fault that is caught is the bpel:selectionFailure.

ScopeInheritedScopeYesProcessDefaultCatchSF has the exitOnStandartFault default value in <process> and a <scope> which sets the value to yes. A nested <scope>

inherts the value and contains the <faultHandlers>. The fault that is caught is the bpel:selectionFailure.

ScopeInheritedScopeYesProcessNoCatchSF has the exitOnStandartFault set to no in <process> and a <scope> which sets the value to yes. A nested <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:selectionFailure.

ScopeInheritedScopeYesProcessYesCatchSF has the exitOnStandartFault set to yes in <process> and a <scope> with the same value. A nested <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:selectionFailure.

ScopeInheritProcessYesCatchSF has the exitOnStandartFault set to yes in <process> and a <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:selectionFailure.

ScopeYesProcessDefaultCatchSF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:selectionFailure.

ScopeYesProcessNoCatchSF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:selectionFailure.

ScopeYesProcessYesCatchSF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:selectionFailure.

ScopeYesScopeInhertedProcessDefaultCatchSF has a <scope> with the exitOnStandart-Fault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which inherits the exitOnStandartFault value from <process>. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:selectionFailure.

ScopeYesScopeInhertedProcessNoCatchSF has a <scope> with the exitOnStandart-Fault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which inherits the exitOnStandartFault value from <process>. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:selectionFailure.

ScopeYesScopeInhertedProcessYesCatchSF has a <scope> with the exitOnStandart-Fault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which inherits the exitOnStandartFault value from <process>. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:selectionFailure.

ScopeYesScopeNoProcessDefaultCatchSF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandart-Fault default value. The fault that is caught is the bpel:selectionFailure.

ScopeYesScopeNoProcessNoCatchSF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandart-Fault set to no. The fault that is caught is the bpel:selectionFailure.

ScopeYesScopeNoProcessYesCatchSF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandart-Fault set to yes. The fault that is caught is the bpel:selectionFailure.

ScopeYesScopeYesProcessDefaultCatchSF has a <scope> with the exitOnStandart-Fault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOn-StandartFault default value. The fault that is caught is the bpel:selectionFailure.

ScopeYesScopeYesProcessNoCatchSF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandart-Fault set to no. The fault that is caught is the bpel:selectionFailure.

ScopeYesScopeYesProcessYesCatchSF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandart-Fault set to yes. The fault that is caught is the bpel:selectionFailure.

ProcessExitAndCatchSLEF has the exitOnStandartFault set to yes in <process> that contains the <faultHandlers> directly. The fault that is caught is the bpel:subLanguageExecution

ScopeInheritedScopeInhertedProcessYesCatchSLEF has the exitOnStandartFault set to yes in <process> which is inherited by a <scope> that contains the <faultHandlers> via an intermediate <scope> which inherits the value itself. The fault that is caught is the bpel:subLanguageExecutionFault.

ScopeInheritedScopeYesProcessDefaultCatchSLEF has the exitOnStandartFault default value in <process> and a <scope> which sets the value to yes. A nested <scope> inherts the value and contains the <faultHandlers>. The fault that is caught is the bpel:subLanguageExecutionFault.

ScopeInheritedScopeYesProcessNoCatchSLEF has the exitOnStandartFault set to no in <process> and a <scope> which sets the value to yes. A nested <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:subLanguageExecuti

ScopeInheritedScopeYesProcessYesCatchSLEF has the exitOnStandartFault set to yes in <process> and a <scope> with the same value. A nested <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:subLanguageExecutionFault.

ScopeInheritProcessYesCatchSLEF has the exitOnStandartFault set to yes in <process> and a <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:subLanguageExecutionFault.

ScopeYesProcessDefaultCatchSLEF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:subLanguageExecutionFault.

ScopeYesProcessNoCatchSLEF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:subLanguageExecutionFault.

ScopeYesProcessYesCatchSLEF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:subLanguageExecutionFault.

ScopeYesScopeInhertedProcessDefaultCatchSLEF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which inherits the exitOnStandartFault value from <process>. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:subLanguageExecutionFault.

ScopeYesScopeInhertedProcessNoCatchSLEF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which inherits the exitOnStandartFault value from <process>. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:subLanguageExecutionFault.

ScopeYesScopeInhertedProcessYesCatchSLEF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which inherits the exitOnStandartFault value from <process>. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:subLanguageExecutionFault.

ScopeYesScopeNoProcessDefaultCatchSLEF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:subLanguageExecutionFault.

ScopeYesScopeNoProcessNoCatchSLEF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:subLanguageExecutionFault.

ScopeYesScopeNoProcessYesCatchSLEF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:subLanguageExecutionFault.

ScopeYesScopeYesProcessDefaultCatchSLEF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:subLanguageExecutionFault.

ScopeYesScopeYesProcessNoCatchSLEF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:subLanguageExecutionFault.

ScopeYesScopeYesProcessYesCatchSLEF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:subLanguageExecutionFault.

ProcessExitAndCatchUPR has the exitOnStandartFault set to yes in <process> that contains the <faultHandlers> directly. The fault that is caught is the bpel:uninitializedPartner

ScopeInheritedScopeInhertedProcessYesCatchUPR has the exitOnStandartFault set to yes in <process> which is inherited by a <scope> that contains the <faultHandlers> via an intermediate <scope> which inherits the value itself. The fault that is caught is the bpel:uninitializedPartnerRole.

ScopeInheritedScopeYesProcessDefaultCatchUPR has the exitOnStandartFault default value in <process> and a <scope> which sets the value to yes. A nested <scope> inherts the value and contains the <faultHandlers>. The fault that is caught is the bpel:uninitializedPartnerRole.

ScopeInheritedScopeYesProcessNoCatchUPR has the exitOnStandartFault set to no in <process> and a <scope> which sets the value to yes. A nested <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:uninitializedPartn

ScopeInheritedScopeYesProcessYesCatchUPR has the exitOnStandartFault set to yes in <process> and a <scope> with the same value. A nested <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:uninitializedPartnerRole.

ScopeInheritProcessYesCatchUPR has the exitOnStandartFault set to yes in <process> and a <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:uninitializedPartnerRole.

ScopeYesProcessDefaultCatchUPR has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:uninitializedPartnerRole.

ScopeYesProcessNoCatchUPR has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:uninitializedPartnerRole.

ScopeYesProcessYesCatchUPR has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:uninitializedPartnerRole.

ScopeYesScopeInhertedProcessDefaultCatchUPR has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another

<scope> which inherits the exitOnStandartFault value from <process>. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:uninitializedPartnerRole.

ScopeYesScopeInhertedProcessNoCatchUPR has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which inherits the exitOnStandartFault value from <process>. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:uninitializedPartnerRole.

ScopeYesScopeInhertedProcessYesCatchUPR has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which inherits the exitOnStandartFault value from <process>. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:uninitializedPartnerRole.

ScopeYesScopeNoProcessDefaultCatchUPR has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:uninitializedPartnerRole.

ScopeYesScopeNoProcessNoCatchUPR has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:uninitializedPartnerRole.

ScopeYesScopeNoProcessYesCatchUPR has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:uninitializedPartnerRole.

ScopeYesScopeYesProcessDefaultCatchUPR has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:uninitializedPartnerRole.

ScopeYesScopeYesProcessNoCatchUPR has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:uninitializedPartnerRole.

ScopeYesScopeYesProcessYesCatchUPR has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:uninitializedPartnerRole.

ProcessExitAndCatchUV has the exitOnStandartFault set to yes in <process> that contains the <faultHandlers> directly. The fault that is caught is the bpel:uninitializedVariable.

ScopeInheritedScopeInhertedProcessYesCatchUV has the `exitOnStandartFault` set to `yes` in `<process>` which is inherited by a `<scope>` that contains the `<faultHandlers>` via an intermediate `<scope>` which inherits the value itself. The fault that is caught is the `bpel:uninitializedVariable`.

ScopeInheritedScopeYesProcessDefaultCatchUV has the `exitOnStandartFault` default value in `<process>` and a `<scope>` which sets the value to `yes`. A nested `<scope>` inherts the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:uninitializedVariable`.

ScopeInheritedScopeYesProcessNoCatchUV has the `exitOnStandartFault` set to `no` in `<process>` and a `<scope>` which sets the value to `yes`. A nested `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:uninitializedVariable`.

ScopeInheritedScopeYesProcessYesCatchUV has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` with the same value. A nested `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:uninitializedVariable`.

ScopeInheritProcessYesCatchUV has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:uninitializedVariable`.

ScopeYesProcessDefaultCatchUV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:uninitializedVariable`.

ScopeYesProcessNoCatchUV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:uninitializedVariable`.

ScopeYesProcessYesCatchUV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:uninitializedVariable`.

ScopeYesScopeInhertedProcessDefaultCatchUV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:uninitializedVariable`.

ScopeYesScopeInhertedProcessNoCatchUV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:uninitializedVariable`.

ScopeYesScopeInhertedProcessYesCatchUV has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The

<process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:uninitializedVariable.

ScopeYesScopeNoProcessDefaultCatchUV has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:uninitializedVariable.

ScopeYesScopeNoProcessNoCatchUV has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:uninitializedVariable.

ScopeYesScopeNoProcessYesCatchUV has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:uninitializedVariable.

ScopeYesScopeYesProcessDefaultCatchUV has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:uninitializedVariable.

ScopeYesScopeYesProcessNoCatchUV has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:uninitializedVariable.

ScopeYesScopeYesProcessYesCatchUV has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:uninitializedVariable.

ProcessExitAndCatchUR has the exitOnStandartFault set to yes in <process> that contains the <faultHandlers> directly. The fault that is caught is the bpel:unsupportedReference.

ScopeInheritedScopeInhertedProcessYesCatchUR has the exitOnStandartFault set to yes in <process> which is inherited by a <scope> that contains the <faultHandlers> via an intermediate <scope> which inherits the value itself. The fault that is caught is the bpel:unsupportedReference.

ScopeInheritedScopeYesProcessDefaultCatchUR has the exitOnStandartFault default value in <process> and a <scope> which sets the value to yes. A nested <scope> inherts the value and contains the <faultHandlers>. The fault that is caught is the bpel:unsupportedReference.

ScopeInheritedScopeYesProcessNoCatchUR has the exitOnStandartFault set to no in <process> and a <scope> which sets the value to yes. A nested <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:unsupportedReference.

ScopeInheritedScopeYesProcessYesCatchUR has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` with the same value. A nested `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:unsupportedReference`.

ScopeInheritProcessYesCatchUR has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:unsupportedReference`.

ScopeYesProcessDefaultCatchUR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:unsupportedReference`.

ScopeYesProcessNoCatchUR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:unsupportedReference`.

ScopeYesProcessYesCatchUR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:unsupportedReference`.

ScopeYesScopeInhertedProcessDefaultCatchUR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:unsupportedReference`.

ScopeYesScopeInhertedProcessNoCatchUR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:unsupportedReference`.

ScopeYesScopeInhertedProcessYesCatchUR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:unsupportedReference`.

ScopeYesScopeNoProcessDefaultCatchUR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:unsupportedReference`.

ScopeYesScopeNoProcessNoCatchUR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:unsupportedReference`.

ScopeYesScopeNoProcessYesCatchUR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:unsupportedReference`.

ScopeYesScopeYesProcessDefaultCatchUR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `yes`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:unsupportedReference`.

ScopeYesScopeYesProcessNoCatchUR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `yes`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:unsupportedReference`.

ScopeYesScopeYesProcessYesCatchUR has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `yes`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:unsupportedReference`.

ProcessExitAndCatchXIS has the `exitOnStandartFault` set to `yes` in `<process>` that contains the `<faultHandlers>` directly. The fault that is caught is the `bpel:xsltInvalidSource`.

ScopeInheritedScopeInhertedProcessYesCatchXIS has the `exitOnStandartFault` set to `yes` in `<process>` which is inherited by a `<scope>` that contains the `<faultHandlers>` via an intermediate `<scope>` which inherits the value itself. The fault that is caught is the `bpel:xsltInvalidSource`.

ScopeInheritedScopeYesProcessDefaultCatchXIS has the `exitOnStandartFault` default value in `<process>` and a `<scope>` which sets the value to `yes`. A nested `<scope>` inherts the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:xsltInvalidSource`.

ScopeInheritedScopeYesProcessNoCatchXIS has the `exitOnStandartFault` set to `no` in `<process>` and a `<scope>` which sets the value to `yes`. A nested `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:xsltInvalidSource`.

ScopeInheritedScopeYesProcessYesCatchXIS has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` with the same value. A nested `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:xsltInvalidSource`.

ScopeInheritProcessYesCatchXIS has the `exitOnStandartFault` set to `yes` in `<process>` and a `<scope>` inherits the value and contains the `<faultHandlers>`. The fault that is caught is the `bpel:xsltInvalidSource`.

ScopeYesProcessDefaultCatchXIS has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:xsltInvalidSource`.

ScopeYesProcessNoCatchXIS has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:xsltInvalidSource.

ScopeYesProcessYesCatchXIS has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:xsltInvalidSource.

ScopeYesScopeInhertedProcessDefaultCatchXIS has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which inherits the exitOnStandartFault value from <process>. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:xsltInvalidSource.

ScopeYesScopeInhertedProcessNoCatchXIS has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which inherits the exitOnStandartFault value from <process>. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:xsltInvalidSource.

ScopeYesScopeInhertedProcessYesCatchXIS has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which inherits the exitOnStandartFault value from <process>. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:xsltInvalidSource.

ScopeYesScopeNoProcessDefaultCatchXIS has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:xsltInvalidSource.

ScopeYesScopeNoProcessNoCatchXIS has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:xsltInvalidSource.

ScopeYesScopeNoProcessYesCatchXIS has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to no. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:xsltInvalidSource.

ScopeYesScopeYesProcessDefaultCatchXIS has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:xsltInvalidSource.

ScopeYesScopeYesProcessNoCatchXIS has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:xsltInvalidSource.

ScopeYesScopeYesProcessYesCatchXIS has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which sets the exitOnStandartFault to yes. The <process> has the exitOnStandart-Fault set to yes. The fault that is caught is the bpel:xsltInvalidSource.

ProcessExitAndCatchXSNF has the exitOnStandartFault set to yes in <process> that contains the <faultHandlers> directly. The fault that is caught is the bpel:xsltStylesheetNotFound

ScopeInheritedScopeInhertedProcessYesCatchXSNF has the exitOnStandartFault set to yes in <process> which is inherited by a <scope> that contains the <faultHandlers> via an intermediate <scope> which inherits the value itself. The fault that is caught is the bpel:xsltStylesheetNotFound.

ScopeInheritedScopeYesProcessDefaultCatchXSNF has the exitOnStandartFault default value in <process> and a <scope> which sets the value to yes. A nested <scope> inherts the value and contains the <faultHandlers>. The fault that is caught is the bpel:xsltStylesheetNotFound.

ScopeInheritedScopeYesProcessNoCatchXSNF has the exitOnStandartFault set to no in <process> and a <scope> which sets the value to yes. A nested <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:xsltStylesheetNotFou

ScopeInheritedScopeYesProcessYesCatchXSNF has the exitOnStandartFault set to yes in <process> and a <scope> with the same value. A nested <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:xsltStylesheetNotFound.

ScopeInheritProcessYesCatchXSNF has the exitOnStandartFault set to yes in <process> and a <scope> inherits the value and contains the <faultHandlers>. The fault that is caught is the bpel:xsltStylesheetNotFound.

ScopeYesProcessDefaultCatchXSNF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:xsltStylesheetNotFound.

ScopeYesProcessNoCatchXSNF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. The <process> has the exitOnStandartFault set to no. The fault that is caught is the bpel:xsltStylesheetNotFound.

ScopeYesProcessYesCatchXSNF has a <scope> with the exitOnStandartFault set to yes and with the <faultHandlers>. The <process> has the exitOnStandartFault set to yes. The fault that is caught is the bpel:xsltStylesheetNotFound.

ScopeYesScopeInhertedProcessDefaultCatchXSNF has a <scope> with the exitOn-StandartFault set to yes and with the <faultHandlers>. This <scope> is enclosed in another <scope> which inherits the exitOnStandartFault value from <process>. The <process> has the exitOnStandartFault default value. The fault that is caught is the bpel:xsltStylesheetNotFound.

ScopeYesScopeInhertedProcessNoCatchXSNF has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:xsltStylesheetNotFound`.

ScopeYesScopeInhertedProcessYesCatchXSNF has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which inherits the `exitOnStandartFault` value from `<process>`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:xsltStylesheetNotFound`.

ScopeYesScopeNoProcessDefaultCatchXSNF has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:xsltStylesheetNotFound`.

ScopeYesScopeNoProcessNoCatchXSNF has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:xsltStylesheetNotFound`.

ScopeYesScopeNoProcessYesCatchXSNF has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `no`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:xsltStylesheetNotFound`.

ScopeYesScopeYesProcessDefaultCatchXSNF has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `yes`. The `<process>` has the `exitOnStandartFault` default value. The fault that is caught is the `bpel:xsltStylesheetNotFound`.

ScopeYesScopeYesProcessNoCatchXSNF has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `yes`. The `<process>` has the `exitOnStandartFault` set to `no`. The fault that is caught is the `bpel:xsltStylesheetNotFound`.

ScopeYesScopeYesProcessYesCatchXSNF has a `<scope>` with the `exitOnStandartFault` set to `yes` and with the `<faultHandlers>`. This `<scope>` is enclosed in another `<scope>` which sets the `exitOnStandartFault` to `yes`. The `<process>` has the `exitOnStandartFault` set to `yes`. The fault that is caught is the `bpel:xsltStylesheetNotFound`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-Compensate	**ProcessExitAndCatchXIS**	-
Scope-TerminationHandlers	**ScopeInheritedScopeInherted-ProcessYesCatchXIS**	- -

Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessDefaultCatchXIS**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessNoCatchXIS**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessYesCatchXIS**	- -
Scope-FaultHandlers	**ScopeInheritProcessYesCatchXIS**	-
Scope-FaultHandlers	**ScopeYesProcessDefaultCatchXIS**	-
Scope-FaultHandlers	**ScopeYesProcessNoCatchXIS**	-
Scope-FaultHandlers	**ScopeYesProcessYesCatchXIS**	-
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessDefaultCatchXIS**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessNoCatchXIS**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessYesCatchXIS**	- -
Scope-TerminationHandlers	**ScopeYesScopeNo-ProcessDefaultCatchXIS**	- -
Scope-TerminationHandlers	**ScopeYesScopeNoProcessNoCatchXIS**	-
Scope-TerminationHandlers	**ScopeYesScopeNoProcessYesCatchXIS**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcess-DefaultCatchXIS**	- -
Scope-TerminationHandlers	**ScopeYesScopeYesProcessNoCatchXIS**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcessYesCatchXIS**	-
Scope-Compensate	**ProcessExitAndCatchUR**	-
Scope-TerminationHandlers	**ScopeInheritedScopeInherted-ProcessYesCatchUR**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessDefaultCatchUR**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessNoCatchUR**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessYesCatchUR**	- -
Scope-FaultHandlers	**ScopeInheritProcessYesCatchUR**	-
Scope-FaultHandlers	**ScopeYesProcessDefaultCatchUR**	-
Scope-FaultHandlers	**ScopeYesProcessNoCatchUR**	-
Scope-FaultHandlers	**ScopeYesProcessYesCatchUR**	-
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessDefaultCatchUR**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessNoCatchUR**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessYesCatchUR**	- -
Scope-TerminationHandlers	**ScopeYesScopeNo-**	-

	ProcessDefaultCatchUR	-
Scope-TerminationHandlers	**ScopeYesScopeNoProcessNoCatchUR**	-
Scope-TerminationHandlers	**ScopeYesScopeNoProcessYesCatchUR**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcess-DefaultCatchUR**	- -
Scope-TerminationHandlers	**ScopeYesScopeYesProcessNoCatchUR**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcessYesCatchUR**	-
Scope-Compensate	**ProcessExitAndCatchUV**	-
Scope-TerminationHandlers	**ScopeInheritedScopeInherted-ProcessYesCatchUV**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessDefaultCatchUV**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessNoCatchUV**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessYesCatchUV**	- -
Scope-FaultHandlers	**ScopeInheritProcessYesCatchUV**	-
Scope-FaultHandlers	**ScopeYesProcessDefaultCatchUV**	-
Scope-FaultHandlers	**ScopeYesProcessNoCatchUV**	-
Scope-FaultHandlers	**ScopeYesProcessYesCatchUV**	-
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessDefaultCatchUV**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessNoCatchUV**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessYesCatchUV**	- -
Scope-TerminationHandlers	**ScopeYesScopeNo-ProcessDefaultCatchUV**	- -
Scope-TerminationHandlers	**ScopeYesScopeNoProcessNoCatchUV**	-
Scope-TerminationHandlers	**ScopeYesScopeNoProcessYesCatchUV**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcess-DefaultCatchUV**	- -
Scope-TerminationHandlers	**ScopeYesScopeYesProcessNoCatchUV**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcessYesCatchUV**	-
Scope-Compensate	**ProcessExitAndCatchUPR**	-
Scope-TerminationHandlers	**ScopeInheritedScopeInherted-ProcessYesCatchUPR**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessDefaultCatchUPR**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessNoCatchUPR**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessYesCatchUPR**	- -
Scope-FaultHandlers	**ScopeInheritProcessYesCatchUPR**	-
Scope-FaultHandlers	**ScopeYesProcessDefaultCatchUPR**	-

Scope-FaultHandlers	**ScopeYesProcessNoCatchUPR**	-
Scope-FaultHandlers	**ScopeYesProcessYesCatchUPR**	-
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessDefaultCatchUPR**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessNoCatchUPR**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessYesCatchUPR**	- -
Scope-TerminationHandlers	**ScopeYesScopeNo-ProcessDefaultCatchUPR**	- -
Scope-TerminationHandlers	**ScopeYesScopeNoProcessNoCatchUPR**	-
Scope-TerminationHandlers	**ScopeYesScopeNoProcessYesCatchUPR**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcess-DefaultCatchUPR**	- -
Scope-TerminationHandlers	**ScopeYesScopeYesProcessNoCatchUPR**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcessYesCatchUPR**	-
Scope-Compensate	**ProcessExitAndCatchSLEF**	-
Scope-TerminationHandlers	**ScopeInheritedScopeInherted-ProcessYesCatchSLEF**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessDefaultCatchSLEF**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessNoCatchSLEF**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessYesCatchSLEF**	- -
Scope-FaultHandlers	**ScopeInheritProcessYesCatchSLEF**	-
Scope-FaultHandlers	**ScopeYesProcessDefaultCatchSLEF**	-
Scope-FaultHandlers	**ScopeYesProcessNoCatchSLEF**	-
Scope-FaultHandlers	**ScopeYesProcessYesCatchSLEF**	-
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessDefaultCatchSLEF**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessNoCatchSLEF**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessYesCatchSLEF**	- -
Scope-TerminationHandlers	**ScopeYesScopeNo-ProcessDefaultCatchSLEF**	- -
Scope-TerminationHandlers	**ScopeYesScopeNoProcessNoCatchSLEF**	-
Scope-TerminationHandlers	**ScopeYesScopeNoProcessYesCatchSLEF**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcess-DefaultCatchSLEF**	- -
Scope-TerminationHandlers	**ScopeYesScopeYesProcessNoCatchSLEF**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcessYesCatchSLEF**	-
Scope-Compensate	**ProcessExitAndCatchSF**	-

Scope-TerminationHandlers	**ScopeInheritedScopeInherted-ProcessYesCatchSF**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessDefaultCatchSF**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessNoCatchSF**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessYesCatchSF**	- -
Scope-FaultHandlers	**ScopeInheritProcessYesCatchSF**	-
Scope-FaultHandlers	**ScopeYesProcessDefaultCatchSF**	-
Scope-FaultHandlers	**ScopeYesProcessNoCatchSF**	-
Scope-FaultHandlers	**ScopeYesProcessYesCatchSF**	-
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessDefaultCatchSF**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessNoCatchSF**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessYesCatchSF**	- -
Scope-TerminationHandlers	**ScopeYesScopeNo-ProcessDefaultCatchSF**	- -
Scope-TerminationHandlers	**ScopeYesScopeNoProcessNoCatchSF**	-
Scope-TerminationHandlers	**ScopeYesScopeNoProcessYesCatchSF**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcess-DefaultCatchSF**	- -
Scope-TerminationHandlers	**ScopeYesScopeYesProcessNoCatchSF**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcessYesCatchSF**	-
Scope-Compensate	**ProcessExitAndCatchSIF**	-
Scope-TerminationHandlers	**ScopeInheritedScopeInherted-ProcessYesCatchSIF**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessDefaultCatchSIF**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessNoCatchSIF**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessYesCatchSIF**	- -
Scope-FaultHandlers	**ScopeInheritProcessYesCatchSIF**	-
Scope-FaultHandlers	**ScopeYesProcessDefaultCatchSIF**	-
Scope-FaultHandlers	**ScopeYesProcessNoCatchSIF**	-
Scope-FaultHandlers	**ScopeYesProcessYesCatchSIF**	-
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessDefaultCatchSIF**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessNoCatchSIF**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-**	-

	ProcessYesCatchSIF	-
Scope-TerminationHandlers	**ScopeYesScopeNo-** **ProcessDefaultCatchSIF**	- -
Scope-TerminationHandlers	**ScopeYesScopeNoProcessNoCatchSIF**	-
Scope-TerminationHandlers	**ScopeYesScopeNoProcessYesCatchSIF**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcess-** **DefaultCatchSIF**	- -
Scope-TerminationHandlers	**ScopeYesScopeYesProcessNoCatchSIF**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcessYesCatchSIF**	-
Scope-Compensate	**ProcessExitAndCatchMReq**	-
Scope-TerminationHandlers	**ScopeInheritedScopeInherted-** **ProcessYesCatchMReq**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-** **ProcessDefaultCatchMReq**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-** **ProcessNoCatchMReq**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-** **ProcessYesCatchMReq**	- -
Scope-FaultHandlers	**ScopeInheritProcessYesCatchMReq**	-
Scope-FaultHandlers	**ScopeYesProcessDefaultCatchMReq**	-
Scope-FaultHandlers	**ScopeYesProcessNoCatchMReq**	-
Scope-FaultHandlers	**ScopeYesProcessYesCatchMReq**	-
Scope-TerminationHandlers	**ScopeYesScopeInherted-** **ProcessDefaultCatchMReq**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-** **ProcessNoCatchMReq**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-** **ProcessYesCatchMReq**	- -
Scope-TerminationHandlers	**ScopeYesScopeNo-** **ProcessDefaultCatchMReq**	- -
Scope-TerminationHandlers	**ScopeYesScopeNoProcessNoCatchMReq**	-
Scope-TerminationHandlers	**ScopeYesScopeNoProcessYesCatchMReq**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcess-** **DefaultCatchMReq**	- -
Scope-TerminationHandlers	**ScopeYesScopeYesProcessNoCatchMReq**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcessYesCatchMReq**	-
Scope-Compensate	**ProcessExitAndCatchMRep**	-
Scope-TerminationHandlers	**ScopeInheritedScopeInherted-** **ProcessYesCatchMRep**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-** **ProcessDefaultCatchMRep**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-** **ProcessNoCatchMRep**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-** **ProcessYesCatchMRep**	- -

Scope-FaultHandlers	**ScopeInheritProcessYesCatchMRep**	-
Scope-FaultHandlers	**ScopeYesProcessDefaultCatchMRep**	-
Scope-FaultHandlers	**ScopeYesProcessNoCatchMRep**	-
Scope-FaultHandlers	**ScopeYesProcessYesCatchMRep**	-
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessDefaultCatchMRep**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessNoCatchMRep**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessYesCatchMRep**	- -
Scope-TerminationHandlers	**ScopeYesScopeNo-ProcessDefaultCatchMRep**	- -
Scope-TerminationHandlers	**ScopeYesScopeNoProcessNoCatchMRep**	-
Scope-TerminationHandlers	**ScopeYesScopeNoProcessYesCatchMRep**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcess-DefaultCatchMRep**	- -
Scope-TerminationHandlers	**ScopeYesScopeYesProcessNoCatchMRep**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcessYesCatchMRep**	-
Scope-Compensate	**ProcessExitAndCatchMAF**	-
Scope-TerminationHandlers	**ScopeInheritedScopeInherted-ProcessYesCatchMAF**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessDefaultCatchMAF**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessNoCatchMAF**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessYesCatchMAF**	- -
Scope-FaultHandlers	**ScopeInheritProcessYesCatchMAF**	-
Scope-FaultHandlers	**ScopeYesProcessDefaultCatchMAF**	-
Scope-FaultHandlers	**ScopeYesProcessNoCatchMAF**	-
Scope-FaultHandlers	**ScopeYesProcessYesCatchMAF**	-
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessDefaultCatchMAF**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessNoCatchMAF**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessYesCatchMAF**	- -
Scope-TerminationHandlers	**ScopeYesScopeNo-ProcessDefaultCatchMAF**	- -
Scope-TerminationHandlers	**ScopeYesScopeNoProcessNoCatchMAF**	-
Scope-TerminationHandlers	**ScopeYesScopeNoProcessYesCatchMAF**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcess-DefaultCatchMAF**	- -
Scope-TerminationHandlers	**ScopeYesScopeYesProcessNoCatchMAF**	-

Scope-TerminationHandlers	**ScopeYesScopeYesProcessYesCatchMAF**	-
Scope-Compensate	**ProcessExitAndCatchIV**	-
Scope-TerminationHandlers	**ScopeInheritedScopeInherted-ProcessYesCatchIV**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessDefaultCatchIV**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessNoCatchIV**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessYesCatchIV**	- -
Scope-FaultHandlers	**ScopeInheritProcessYesCatchIV**	-
Scope-FaultHandlers	**ScopeYesProcessDefaultCatchIV**	-
Scope-FaultHandlers	**ScopeYesProcessNoCatchIV**	-
Scope-FaultHandlers	**ScopeYesProcessYesCatchIV**	-
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessDefaultCatchIV**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessNoCatchIV**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessYesCatchIV**	- -
Scope-TerminationHandlers	**ScopeYesScopeNo-ProcessDefaultCatchIV**	- -
Scope-TerminationHandlers	**ScopeYesScopeNoProcessNoCatchIV**	-
Scope-TerminationHandlers	**ScopeYesScopeNoProcessYesCatchIV**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcess-DefaultCatchIV**	- -
Scope-TerminationHandlers	**ScopeYesScopeYesProcessNoCatchIV**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcessYesCatchIV**	-
Scope-Compensate	**ProcessExitAndCatchCV**	-
Scope-TerminationHandlers	**ScopeInheritedScopeInherted-ProcessYesCatchCV**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessDefaultCatchCV**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessNoCatchCV**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessYesCatchCV**	- -
Scope-FaultHandlers	**ScopeInheritProcessYesCatchCV**	-
Scope-FaultHandlers	**ScopeYesProcessDefaultCatchCV**	-
Scope-FaultHandlers	**ScopeYesProcessNoCatchCV**	-
Scope-FaultHandlers	**ScopeYesProcessYesCatchCV**	-
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessDefaultCatchCV**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-**	-

	ProcessNoCatchCV	-
Scope-TerminationHandlers	**ScopeYesScopeInherited-ProcessYesCatchCV**	- -
Scope-TerminationHandlers	**ScopeYesScopeNo-ProcessDefaultCatchCV**	- -
Scope-TerminationHandlers	**ScopeYesScopeNoProcessNoCatchCV**	-
Scope-TerminationHandlers	**ScopeYesScopeNoProcessYesCatchCV**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcess-DefaultCatchCV**	- -
Scope-TerminationHandlers	**ScopeYesScopeYesProcessNoCatchCV**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcessYesCatchCV**	-
Scope-Compensate	**ProcessExitAndCatchIEV**	-
Scope-TerminationHandlers	**ScopeInheritedScopeInherted-ProcessYesCatchIEV**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessDefaultCatchIEV**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessNoCatchIEV**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessYesCatchIEV**	- -
Scope-FaultHandlers	**ScopeInheritProcessYesCatchIEV**	-
Scope-FaultHandlers	**ScopeYesProcessDefaultCatchIEV**	-
Scope-FaultHandlers	**ScopeYesProcessNoCatchIEV**	-
Scope-FaultHandlers	**ScopeYesProcessYesCatchIEV**	-
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessDefaultCatchIEV**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessNoCatchIEV**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessYesCatchIEV**	- -
Scope-TerminationHandlers	**ScopeYesScopeNo-ProcessDefaultCatchIEV**	- -
Scope-TerminationHandlers	**ScopeYesScopeNoProcessNoCatchIEV**	-
Scope-TerminationHandlers	**ScopeYesScopeNoProcessYesCatchIEV**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcess-DefaultCatchIEV**	- -
Scope-TerminationHandlers	**ScopeYesScopeYesProcessNoCatchIEV**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcessYesCatchIEV**	-
Scope-Compensate	**ProcessExitAndCatchIBC**	-
Scope-TerminationHandlers	**ScopeInheritedScopeInherted-ProcessYesCatchIBC**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessDefaultCatchIBC**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessNoCatchIBC**	- -

Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessYesCatchIBC**	- -
Scope-FaultHandlers	**ScopeInheritProcessYesCatchIBC**	-
Scope-FaultHandlers	**ScopeYesProcessDefaultCatchIBC**	-
Scope-FaultHandlers	**ScopeYesProcessNoCatchIBC**	-
Scope-FaultHandlers	**ScopeYesProcessYesCatchIBC**	-
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessDefaultCatchIBC**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessNoCatchIBC**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessYesCatchIBC**	- -
Scope-TerminationHandlers	**ScopeYesScopeNo-ProcessDefaultCatchIBC**	- -
Scope-TerminationHandlers	**ScopeYesScopeNoProcessNoCatchIBC**	-
Scope-TerminationHandlers	**ScopeYesScopeNoProcessYesCatchIBC**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcess-DefaultCatchIBC**	- -
Scope-TerminationHandlers	**ScopeYesScopeYesProcessNoCatchIBC**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcessYesCatchIBC**	-
Scope-Compensate	**ProcessExitAndCatchCReq**	-
Scope-TerminationHandlers	**ScopeInheritedScopeInherted-ProcessYesCatchCReq**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessDefaultCatchCReq**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessNoCatchCReq**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessYesCatchCReq**	- -
Scope-FaultHandlers	**ScopeInheritProcessYesCatchCReq**	-
Scope-FaultHandlers	**ScopeYesProcessDefaultCatchCReq**	-
Scope-FaultHandlers	**ScopeYesProcessNoCatchCReq**	-
Scope-FaultHandlers	**ScopeYesProcessYesCatchCReq**	-
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessDefaultCatchCReq**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessNoCatchCReq**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessYesCatchCReq**	- -
Scope-TerminationHandlers	**ScopeYesScopeNo-ProcessDefaultCatchCReq**	- -
Scope-TerminationHandlers	**ScopeYesScopeNoProcessNoCatchCReq**	-
Scope-TerminationHandlers	**ScopeYesScopeNoProcessYesCatchCReq**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcess-**	-

	DefaultCatchCReq	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcessNoCatchCReq**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcessYesCatchCReq**	-
Scope-Compensate	**ProcessExitAndCatchCRec**	-
Scope-TerminationHandlers	**ScopeInheritedScopeInherted-ProcessYesCatchCRec**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessDefaultCatchCRec**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessNoCatchCRec**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessYesCatchCRec**	- -
Scope-FaultHandlers	**ScopeInheritProcessYesCatchCRec**	-
Scope-FaultHandlers	**ScopeYesProcessDefaultCatchCRec**	-
Scope-FaultHandlers	**ScopeYesProcessNoCatchCRec**	-
Scope-FaultHandlers	**ScopeYesProcessYesCatchCRec**	-
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessDefaultCatchCRec**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessNoCatchCRec**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessYesCatchCRec**	- -
Scope-TerminationHandlers	**ScopeYesScopeNo-ProcessDefaultCatchCRec**	- -
Scope-TerminationHandlers	**ScopeYesScopeNoProcessNoCatchCRec**	-
Scope-TerminationHandlers	**ScopeYesScopeNoProcessYesCatchCRec**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcess-DefaultCatchCRec**	- -
Scope-TerminationHandlers	**ScopeYesScopeYesProcessNoCatchCRec**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcessYesCatchCRec**	-
Scope-Compensate	**ProcessExitAndCatchCCF**	-
Scope-TerminationHandlers	**ScopeInheritedScopeInherted-ProcessYesCatchCCF**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessDefaultCatchCCF**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessNoCatchCCF**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessYesCatchCCF**	- -
Scope-FaultHandlers	**ScopeInheritProcessYesCatchCCF**	-
Scope-FaultHandlers	**ScopeYesProcessDefaultCatchCCF**	-
Scope-FaultHandlers	**ScopeYesProcessNoCatchCCF**	-
Scope-FaultHandlers	**ScopeYesProcessYesCatchCCF**	-
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessDefaultCatchCCF**	- -

Scope-TerminationHandlers	**ScopeYesScopeInherited-ProcessNoCatchCCF**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherited-ProcessYesCatchCCF**	- -
Scope-TerminationHandlers	**ScopeYesScopeNo-ProcessDefaultCatchCCF**	- -
Scope-TerminationHandlers	**ScopeYesScopeNoProcessNoCatchCCF**	-
Scope-TerminationHandlers	**ScopeYesScopeNoProcessYesCatchCCF**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcess-DefaultCatchCCF**	- -
Scope-TerminationHandlers	**ScopeYesScopeYesProcessNoCatchCCF**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcessYesCatchCCF**	-
Scope-Compensate	**ProcessExitAndCatchAR**	-
Scope-TerminationHandlers	**ScopeInheritedScopeInherted-ProcessYesCatchAR**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessDefaultCatchAR**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessNoCatchAR**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessYesCatchAR**	- -
Scope-FaultHandlers	**ScopeInheritProcessYesCatchAR**	-
Scope-FaultHandlers	**ScopeYesProcessDefaultCatchAR**	-
Scope-FaultHandlers	**ScopeYesProcessNoCatchAR**	-
Scope-FaultHandlers	**ScopeYesProcessYesCatchAR**	-
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessDefaultCatchAR**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessNoCatchAR**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessYesCatchAR**	- -
Scope-TerminationHandlers	**ScopeYesScopeNo-ProcessDefaultCatchAR**	- -
Scope-TerminationHandlers	**ScopeYesScopeNoProcessNoCatchAR**	-
Scope-TerminationHandlers	**ScopeYesScopeNoProcessYesCatchAR**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcess-DefaultCatchAR**	- -
Scope-TerminationHandlers	**ScopeYesScopeYesProcessNoCatchAR**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcessYesCatchAR**	-
Scope-Compensate	**ProcessExitAndCatchXSNF**	-
Scope-TerminationHandlers	**ScopeInheritedScopeInherted-ProcessYesCatchXSNF**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessDefaultCatchXSNF**	- -

Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessNoCatchXSNF**	- -
Scope-TerminationHandlers	**ScopeInheritedScopeYes-ProcessYesCatchXSNF**	- -
Scope-FaultHandlers	**ScopeInheritProcessYesCatchXSNF**	-
Scope-FaultHandlers	**ScopeYesProcessDefaultCatchXSNF**	-
Scope-FaultHandlers	**ScopeYesProcessNoCatchXSNF**	-
Scope-FaultHandlers	**ScopeYesProcessYesCatchXSNF**	-
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessDefaultCatchXSNF**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessNoCatchXSNF**	- -
Scope-TerminationHandlers	**ScopeYesScopeInherted-ProcessYesCatchXSNF**	- -
Scope-TerminationHandlers	**ScopeYesScopeNo-ProcessDefaultCatchXSNF**	- -
Scope-TerminationHandlers	**ScopeYesScopeNoProcessNoCatchXSNF**	-
Scope-TerminationHandlers	**ScopeYesScopeNoProcessYesCatchXSNF**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcess-DefaultCatchXSNF**	- -
Scope-TerminationHandlers	**ScopeYesScopeYesProcessNoCatchXSNF**	-
Scope-TerminationHandlers	**ScopeYesScopeYesProcessYesCatchXSNF**	-

Table 4: SA00003 test pairs

1.4 SA00004

"If any referenced queryLanguage or expressionLanguage is unsupported by the WS-BPEL processor then the processor MUST reject the submitted WS-BPEL process definition."[2, p. 194]

This rule depends on engine implementations, therefore, it is postponed to future work.

1.5 SA00005

"If the portType attribute is included for readability, in a <receive>, <reply>, <invoke>, <onEvent> or <onMessage> element, the value of the portType attribute MUST match the portType value implied by the combination of the specified partnerLink and the role implicitly specified by the activity."[2, p. 194]

For each of the five different message activities, namely, `<invoke>`, `<receive>`, `<reply>`, `<onMessage>` and `<onEvent>`, there is one error if the `@portType` that has not the already implied value. Additionally, [1, p. 26] distinguish between receiving and sending message activities because the implication is resolved via the communication role of the `<partnerLink>`, being `@partnerRole` and `@myRole`. However, this is not important for the tests, as we still require five tests, one for each message activity, in which the `@portType` is wrong.

The tests for each error type include the same modified WSDL TestInterface-SecondPortType.wsdl that has a additional empty `<portType>` with `@name=SecondTestInterfacePortType`. In Table 5 this `<portType>` is used within the `@portType` attribute of the message activity and differs from the implicit `<portType>` that defines the corresponding `<operation>`.

ReceiveWithNonExistantPortType references the `SecondTestInterfacePortType` of the modified WSDL in the `<receive>`.

ReplyWithNonExistantPortType references the `SecondTestInterfacePortType` of the modified WSDL in the `<reply>`.

InvokeWithNonExistantPortType references the `SecondTestInterfacePortType` of the modified WSDL in the `<invoke>`.

OnEventWithNonExistantPortType references the `SecondTestInterfacePortType` of the modified WSDL in the `<onEvent>`.

OnMessageWithNonExistantPortType references the `SecondTestInterfacePortType` of the modified WSDL in the `<onMessage>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Receive	**ReceiveWithNonExistantPortType**	1
ReceiveReply	**ReplyWithNonExistantPortType**	1

Invoke-Async	**InvokeWithNonExistantPortType**	1
Scope-EventHandlers-Parts	**OnEventWithNonExistantPortType**	1
Pick-CreateInstance	**OnMessageWithNonExistantPortType**	1

Table 5: SA00005 test pairs

1.6 SA00006

"The <rethrow> activity MUST only be used within a faultHandler (i.e. <catch> and <catchAll> elements)."[2, p. 194]

The correctness of the implementation of rule #6 can be determined by detecting `<rethrow>` in every wrong place. As this is unfeasible, we only test this condition in every activity that can contain other activities, namely, `<if>`, `<else>`, `<elseIf>`, `<flow>`, `<onAlarm>`, `<onMessage>`, `<repeatUntil>`, `<scope>`, `<sequence>`, `<while>`, `<compensationHandler>`, and `<terminationHandler>`. This list of activities does not include `<catch>` and `<catchAll>` due to the rule logic. We do, however, not test any nesting of various activities, as this would result in test explosion. The accepted opposite is expressed as existential quantification in [1, p. 42]. Hence, we get twelve test cases, one for each containing activity in which we place `<rethrow>`.

RethrowInCompensationHandler contains `<rethrow>` as activity in `<compensationHandler>`.

RethrowInElse contains `<rethrow>` as activity in `<else>`.

RethrowInElseIf contains `<rethrow>` as activity in `<elseIf>`.

RethrowInFlow contains `<rethrow>` as last activity in `<flow>`.

RethrowInIf contains `<rethrow>` as activity in `<if>`.

RethrowInOnAlarm contains `<rethrow>` as activity in `<onAlarm>`.

RethrowInOnMessage contains `<rethrow>` as activity in `<else>`.

RethrowInRepeatUntil contains `<rethrow>` as activity in `<repeatUntil>`.

RethrowInScope contains `<rethrow>` as activity in `<sope name="Scope1">`.

RethrowInTerminationHandlers contains `<rethrow>` as activity in `<terminationHandlers>`.

RethrowInWhile contains `<rethrow>` as activity in `<while>`.

RethrowOutsideFaultHandlers with a `<rethrow>` at the end of the `<sequence>`, thus, it is outside of a `<faultHandlers>`. (Therefore outside a `<catch>` or `<catchAll>`.)

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-Compensate	**RethrowInCompensationHandler**	-
If-Else	**RethrowInElse**	-
If-ElseIf	**RethrowInElseIf**	-
Flow	**RethrowInFlow**	-
If	**RethrowInIf**	-
Pick-OnAlarm-For	**RethrowInOnAlarm**	-

Pick-OnAlarm-For	**RethrowInOnMessage**	-
RepeatUntil	**RethrowInRepeatUntil**	-
Scope-Isolated	**RethrowInScope**	-
Scope-TerminationHandlers	**RethrowInTerminationHandlers**	-
While	**RethrowInWhile**	-
Rethrow	**RethrowOutsideFaultHandlers**	-

Table 6: SA00006 test pairs

1.7 SA00007

"The <compensateScope> activity MUST only be used from within a faultHandler, another compensationHandler, or a terminationHandler."[2, p. 194]

The correctness of the implementation of rule #7 can be determined by detecting `<compensateScope>` in every wrong place. As this is unfeasible, we test this condition in every activity that can contain other activities (analogously to #6), namely, `<if>`, `<else>`, `<elseIf>`, `<flow>`, `<onAlarm>`, `<onMessage>`, `<repeatUntil>`, `<scope>`, `<sequence>` and `<while>`. This list of activities does not include `<catch>`, `<catchAll>`, `<compensationHandler>`, and `<terminationHandler>` due to the rule logic. Thus, we require ten tests and the accepted opposite is modeled with an existential quantification by [1, p. 59] as well.

CompensateScopeInElse contains `<compensateScope>` as activity in `<else>` and an additional `<scope>` in the same enclosing `<scope>` which is the target of the `<compensateScope>`.

CompensateScopeInElseIf contains `<compensateScope>` as activity in `<elseIf>` and an additional `<scope>` in the same enclosing `<scope>` which is the target of the `<compensateScope>`.

CompensateScopeInFlow contains `<compensateScope>` as last activity in `<flow>` and an additional `<scope>` in the same enclosing `<scope>` which is the target of the `<compensateScope>`.

CompensateScopeInIf contains `<compensateScope>` as activity in `<if>` and an additional `<scope>` in the same enclosing `<scope>` which is the target of the `<compensateScope>`.

CompensateScopeInOnAlarm contains `<compensateScope>` as activity in `<onAlarm>` and an additional `<scope>` in the same enclosing `<scope>` which is the target of the `<compensateScope>`.

CompensateScopeInOnMessage contains `<compensateScope>` as activity in `<else>` and an additional `<scope>` in the same enclosing `<scope>` which is the target of the `<compensateScope>`.

CompensateScopeInRepeatUntil contains `<compensateScope>` as activity in `<repeatUntil>` and an additional `<scope>` in the same enclosing `<scope>` which is the target of the `<compensateScope>`.

CompensateScopeInScope contains `<compensateScope>` as activity in `<sope name="Scope1">` and an additional `<scope>` in the same enclosing `<scope>` which is the target of the `<compensateScope>`.

CompensateScopeInWhile contains `<compensateScope>` as activity in `<while>` and an additional `<scope>` in the same enclosing `<scope>` which is the target of the `<compensateScope>`.

CompensateScopeOutsideFaultHandlers contains a `<compensateScope>` as last child of `<sequence>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
If-Else	**CompensateScopeInElse**	-
If-ElseIf	**CompensateScopeInElseIf**	-
Flow	**CompensateScopeInFlow**	-
If	**CompensateScopeInIf**	-
Pick-OnAlarm-For	**CompensateScopeInOnAlarm**	-
Pick-OnAlarm-For	**CompensateScopeInOnMessage**	-
RepeatUntil	**CompensateScopeInRepeatUntil**	-
Scope-Isolated	**CompensateScopeInScope**	-
While	**CompensateScopeInWhile**	-
Invoke-CompensateScope-CompensationHandler	**CompensateScopeOutsideFaultHandlers**	-

Table 7: SA00007 test pairs

1.8 SA00008

"The <compensate> activity MUST only be used from within a faultHandler, another compensationHandler, or a terminationHandler."[2, p. 194]

Analogous to #7, the rule #8 requires the same amount of test cases and the accepted opposite is modeled with an existential quantification by [1, p. 59] as well.

CompensateInElse contains `<compensate>` as activity in `<else>`.

CompensateInElseIf contains `<compensate>` as activity in `<elseIf>`.

CompensateInFlow contains `<compensate>` as last activity in `<flow>`.

CompensateInIf contains `<compensate>` as activity in `<if>`.

CompensateInOnAlarm contains `<compensate>` as activity in `<onAlarm>`.

CompensateInOnMessage contains `<compensate>` as activity in `<else>`.

CompensateInRepeatUntil contains `<compensate>` as activity in `<repeatUntil>`.

CompensateInScope contains `<compensate>` as activity in `<sope name="Scope1">`.

CompensateInWhile contains `<compensate>` as activity in `<while>`.

CompensateOutsideFaultHandlers contains a `<compensate>` as last child of `<sequence>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
If-Else	**CompensateInElse**	-
If-ElseIf	**CompensateInElseIf**	-
Flow	**CompensateInFlow**	-
If	**CompensateInIf**	-
Pick-OnAlarm-For	**CompensateInOnAlarm**	-
Pick-OnAlarm-For	**CompensateInOnMessage**	-
RepeatUntil	**CompensateInRepeatUntil**	-
Scope-Isolated	**CompensateInScope**	-
While	**CompensateInWhile**	-
Scope-Compensate	**CompensateOutsideFaultHandlers**	-

Table 8: SA00008 test pairs

1.9 SA00009

"In the case of mandatory extensions declared in the <extensions> element not supported by a WS-BPEL implementation, the process definition MUST be rejected."[2, p. 194]

This rule depends on engine implementations, therefore, it is postponed to future work.

1.10 SA00010

"A WS-BPEL process definition MUST import all XML Schema and WSDL definitions it uses. This includes all XML Schema type and element definitions, all WSDL port types and message types as well as property and property alias definitions used by the process."[2, p. 195]

All elements introducing constructs (depending on XML schema or WSDL definition) must refer an imported file, e.g. a `<variable>` requires an *include check* for these defining documents. Rule #10 is comprehensive and is not modeled by [1, p. 65] because it requires a formalization of WSDL and XML schema. Eleven elements need to be checked with WSDL definitions. Additionally, attributes of `<variable>` and `<correlationSet>` can reference constructs defined in XSDs directly and transitively. While the rule text states the `@propertyAlias` explicitly, it is implicitly tested every time a `@property` is resolved. Because of this, we did not include `@propertyAlias` in our test set.

[`<reply>`, `<receive>`, `<invoke>`, `<onMessage>`, `<onEvent>`]
$\times$
[`@operation`, `@partnerLink`]

Formalization 3: SA Rule #10, for message activities

In addition to the rules from the previous listing, `<onEvent>` implicitly defines its own scope with its own variable, hence, the `@messageType` or `@element` can be set optionally.

[`<onEvent>`]
$\times$
[`@messageType`, `@element`]

Formalization 4: SA Rule #10, for `<onEvent>`

[`<catch>`]
$\times$
[`@faultMessageType`, `@faultElement`]

Formalization 5: SA Rule #10, for the `<catch>` activity

[<variable>]
×
[@messageType, @type, @element]

Formalization 6: SA Rule #10, for <variable>

[<partnerLink>]
×
[@partnerLinkType]

Formalization 7: SA Rule #10, for <partnerLink>

[<to>, <from>]
×
[@property]

Formalization 8: SA Rule #10, for <to> and <from>

[<correlationSet>]
×
[@properties]

Formalization 9: SA Rule #10, for <correlationSet>

UndefinedType-Catch-FaultElement has a changed `@faultElement` in the `<catch>` which has no definition in corresponding schema (http://www.w3.org/2001/XMLSchema).

UndefinedType-Catch-FaultMessageType has a changed `@faultMessageType` in the `<catch>` with the TestInterface.wsdl namespace to a message, which is undefined. A BPEL-file with a `<catch>@faultMessageType` which has no definition in corresponding TestInterface.wsdl.

UndefinedType-CorrelationSet contains a modified `@properties` in the `<correlationSet>` which has no definition in corresponding TestInterface.wsdl.

UndefinedType-From copies from a modified `@property` which has no definition in corresponding TestInterface.wsdl.

UndefinedType-Invoke has a modified `@operation` in the `<invoke>` which has no definition in corresponding TestInterface.wsdl.

UndefinedType-OnEvent has a modified `@operation` in the `<onEvent>` which has no definition in corresponding TestInterface.wsdl.

UndefinedType-OnMessage has a modified `@operation` in the `<onMessage>` which has no definition in corresponding TestInterface.wsdl.

UndefinedType-PartnerLink is a BPEL file with a `<partnerLink>@partnerLinkType` which has no definition in corresponding TestInterface.wsdl.

UndefinedType-Receive has a modified `@operation` in the `<receive>` that has no definition in corresponding TestInterface.wsdl.

UndefinedType-Reply has a modified `@operation` in the `<reply>` which has no definition in corresponding TestInterface.wsdl.

UndefinedType-To copies to a modified `@property` which has no definition in corresponding TestInterface.wsdl.

UndefinedType-Variable-Element adds a `<variable>` to the origin test with a `@element` that has no definition in corresponding schema (http://www.w3.org/2001/XMLSchema).

UndefinedType-Variable-MessageType has a changed `@messageType` in the `<variable>` ReplyData with the TestInterface.wsdl namespace to a message, which is undefined.

UndefinedType-Variable-Type adds a `<variable>` to the origin test with a `@type` that has no definition in corresponding schema (http://www.w3.org/2001/XMLSchema).

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-FaultHandlers-FaultElement	**UndefinedType-Catch-FaultElement**	-
Rethrow-FaultData	**UndefinedType-Catch-FaultMessageType**	-
Pick-Correlations-InitAsync	**UndefinedType-CorrelationSet**	-

Assign-PartnerLink	**UndefinedType-From**	-
Invoke-Async	**UndefinedType-Invoke**	-
Scope-EventHandlers-InitSync	**UndefinedType-OnEvent**	-
Pick-Correlations-InitAsync	**UndefinedType-OnMessage**	-
Sequence	**UndefinedType-PartnerLink**	-
Assign-PartnerLink	**UndefinedType-Receive**	-
Pick-Correlations-InitAsync	**UndefinedType-Reply**	-
Assign-PartnerLink	**UndefinedType-To**	-
Assign-PartnerLink	**UndefinedType-Variable-Element**	-
Assign-PartnerLink	**UndefinedType-Variable-MessageType**	-
Assign-PartnerLink	**UndefinedType-Variable-Type**	-

Table 9: SA00010 test pairs

1.11 SA00011

”If a namespace attribute is specified on an <import> then the imported definitions MUST be in that namespace.”[2, p. 195]

To violate rule #11 the document that is imported via `<import>` in the BPEL process definition, the `@namespace` of the `<import>` has to differ from the `@targetNamespace` of the imported document. Hence, only a single test case is required. This rule is explicitly excluded from the model of [1, p. 65].

Import-WrongNameSpace has a modified `@namespace` in the `<import>` of TestInterface.wsdl, thus, differing from `targetNamespace` of the WSDL.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Empty	**Import-WrongNameSpace**	-

Table 10: SA00011 test pairs

1.12 SA00012

"If no namespace is specified then the imported definitions MUST NOT contain a targetNamespace specification."[2, p. 195]

Rule #12 is similar to rule #11, as to violate it, we require an `<import>` with a `namespace=""` and a non-empty `@targetNamespace` in the imported document. Thus, a single test is sufficient. [1, p. 65] exclude this rule as well.

Import-NoNameSpace contains no `@namespace` in the `<import>` of TestInterface.wsdl that has a `targetNamespace`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Empty	**Import-NoNameSpace**	-

Table 11: SA00012 test pairs

1.13 SA00013

"The value of the importType attribute of element <import> MUST be set to `http://www.w3.org/2001/XMLSchema` when importing XML Schema 1.0 documents, and to `http://schemas.xmlsoap.org/wsdl/` when importing WSDL 1.1 documents. "[2, p. 195]

The import of both XSD and WSDL files using `<import>` requires specifying the correct `@importType`, i.e., the namespace of the imported document, as stated in rule #13. Therefore, two tests are required: a) one for XSD `<import>` and b) one for WSDL `<import>`. [1, p. 65] exclude this rule from their model.

Import-WrongImportType imports TestInterface.wsdl with a modified `importType` differing from the default for WSDL.

WrongXsdImportType imports the months.xsd with a modified `importType` differing from the default for XML Schema.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Empty	**Import-WrongImportType**	-
Assign-Validate	**WrongXsdImportType**	-

Table 12: SA00013 test pairs

1.14 SA00014

”A WS-BPEL process definition MUST be rejected if the imported documents contain conflicting definitions of a component used by the importing process definition (as could be caused, for example, when the XSD redefinition mechanism is used).”[2, p. 195]

This rule #14 ensures that the qualified name of a XSD or WSDL element is unique. In general, this is required for all different elements in XSD and WSDL, however, in practice, we resort to test only

[1, p. 65] do not model rule #14. We, however, determine the necessary tests by permuting the combinations of doubled definitions in WSDL and schema files.

Regarding the redefinition of an element, the original version has to be defined within the XSD, whereas the redefinition may occur in the WSDL under `<types>`, or in a separate XSD. For that purpose, we have created an additional XSD that contains also `<simpleType>`, `<group>`, and `<attributeGroup>` in addition to `<complexType>` and `<element>` definitions. Please note that `<element>`s cannot be redefined.

XSD with [`<simpleType>`, `<complexType>`, `<group>`, `<attributeGroup>`]
×
redefinition in [WSDL, XSD, none]

Formalization 10: SA Rule #14 for XSD redefinition

[`<simpleType>`, `<complexType>`, `<element>`, `<group>`, `<attributeGroup>`]
×
definition in [WSDL, XSD]
×
in [WSDL, XSD, none]

Formalization 11: SA Rule #14 for XSD elements, types, etc.

WSDL and XSD files combined have the same effects in any import order.

[`<operation>`, `<message>`, `<portType>`, `<partnerLinkType>`, `<property>`]
×
definition in [WSDL]
×
in [WSDL, none]

Formalization 12: SA Rule #14 for WSDL messages, operations, etc.

ImportRedefine imports RedefinedCalculatorSchema.xsd that has a `<redefine>` element. The base schema is CalculatorSchema-Copy.xsd, and another XML simple type is added to the XML complex type `calculationInputType`. Schema elements are conflicting, because the XSD redefinition mechanism is used.

ImportWsdlAndWsdlCopy imports a second WSDL file (TestInterface-CopyWithoutPropertyAliases.wsdl), which is a copy of the TestInterface.wsdl without `<propertyAlias>` definitions.

ImportXsdAndDefineInside imports TestInterface-DefineInside.wsdl that defines the same elements (with the same qualified name (QName)s) as in CalculatorSchema.xsd. The BPEL process additionally imports CalculatorSchema-Copy.xsd, which is a copy of CalculatorSchema.xsd Schema elements are conflicting, because they are defined twice.

ImportXsdAndWsdlXsdCopy imports the same schema twice with distinct names (CalculatorSchema.xsd directly imported, and TestInterface-ImportXsdCopy.wsdl imports the CalculatorSchema-Copy.xsd). Schema elements are conflicting, because they are defined twice.

ImportXsdAndXsdCopy directly imports the same schema twice with distinct names (CalculatorSchema.xsd and CalculatorSchema-Copy.xsd). Schema elements are conflicting because they are defined twice.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Sequence	**ImportRedefine**	-
Sequence	**ImportWsdlAndWsdlCopy**	1
Sequence	**ImportXsdAndDefineInside**	1
Sequence	**ImportXsdAndWsdlXsdCopy**	1
Sequence	**ImportXsdAndXsdCopy**	-

Table 13: SA00014 test pairs

1.15 SA00015

”To be instantiated, an executable business process MUST contain at least one <receive> or <pick> activity annotated with a createInstance=’yes’ attribute.”[2, p. 195]

In line with the rule #15 model by [1, p. 61], we test `<receive>`s and `<pick>`s. In erroneous processes, `createInstance="no"` must hold in all `<receive>` and `<pick>` activities. This can be achieved in two ways: 1) explicitly set the attribute or 2) use the default value.

If `createInstance="yes"` the process is valid. Four tests are required to cover the violations of rule #15, two for either `<receive>` and `<pick>` activities.

[only `<receive>`, only `<pick>`] in process
×
with [`createInstance="no"`, default value of `@createInstance` (=no),
`createInstance="yes"`]

Formalization 13: SA Rule #15

NoActivityWithCreateInstanceSetToYes has no `createInstance` attribute in the `<receive>`.

OnlyActivityWithCreateInstanceSetToNo has the `createInstance` value of the `<receive>` set to `no`.

PickCreateInstanceMissing has no `createInstance` attribute in the `<pick>`.

PickCreateInstanceNo has the `createInstance` value of the `<pick>` set to `no`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
ReceiveReply	**NoActivityWithCreateInstanceSetToYes**	-
ReceiveReply	**OnlyActivityWithCreateInstanceSetToNo**	-
Pick-CreateInstance	**PickCreateInstanceMissing**	-
Pick-CreateInstance	**PickCreateInstanceNo**	-

Table 14: SA00015 test pairs

1.16 SA00016

"A partnerLink MUST specify the myRole or the partnerRole, or both."[2, p. 195]

By negating the rule #16 definition of [1, p. 25], we identify the single test case for this rule, which is a `<partnerLink>` with neither `@myRole` nor `@partnerRole`.

PartnerLinkWithoutMyRoleAndPartnerRole has no `myRole` or `partnerRole` attribute in the `<patnerLink>` element.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
ReceiveReply	**PartnerLinkWithoutMyRoleAndPartnerRole**	-

Table 15: SA00016 test pairs

1.17 SA00017

"The initializePartnerRole attribute MUST NOT be used on a partnerLink that does not have a partner role."[2, p. 195]

Analog to #16, rule #17 has a single test that is also specified by the negation of the rule definition by [1, p. 25], in this case, a `<partnerLink>` with `initializePartnerRole="yes"`

InitializePartnerRoleUsedOnPartnerLinkWithoutPartnerRole contains an additional `initializePartnerRole="yes"` in the `<partnerLink>` with the `myRole` attribute. There is no `partnerRole` within this `<partnerLink>`, thus, it violates the rule.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Invoke-Sync	**InitializePartnerRoleUsedOn-PartnerLinkWithoutPartnerRole**	-

Table 16: SA00017 test pairs

1.18 SA00018

"The name of a partnerLink MUST be unique among the names of all partnerLinks defined within the same immediately enclosing scope."[2, p. 195]

A `<partnerLink>` can be declared in a `<scope>` or in `<process>`, so these are the locations where the violations of rule #18 can occur. Hence, we have created two tests, one for a duplicated `<partnerLink>` name on the `<process>` level and one on the `<scope>` level. In [1, p. 25], there is a more complex definition than is required for executable BPEL processes, as they include abstract BPEL processes as well.

TwoPartnerLinksWithSameName contains a duplication of the `<partnerLink>` element, therefore, the name is not unique in `<process>`.

ScopeSamePartnerLinkTwice contains a duplication of the `<partnerLink>` element in the `<scope>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
ReceiveReply	**TwoPartnerLinksWithSameName**	-
Scope-PartnerLinks	**ScopeSamePartnerLinkTwice**	-

Table 17: SA00018 test pairs

1.19 SA00019

"Either the type or element attributes MUST be present in a <vprop:property> element but not both."[2, p. 195]

The violation case of rule #19 is to have an `@element` and a `@type` in a `<property>`, or neither an `@element` nor a `@type`. [1, p. 12] mention this rule, but they do not model it because the rule concerns only WSDL.

[`@element`, none]
×
[`@type`, none]

Formalization 14: SA Rule #19

PropertyWithoutTypeOrElement imports TestInterface-PropertyWithoutTypeOrElement.wsdl, which defines a `<property>` without an `element` or `type` attribute.

PropertyWithTypeAndElement imports TestInterface-PropertyWithTypeAndElement.wsdl, which defines a `<property>` including an `element` and `type` attribute.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
ReceiveReply	**PropertyWithoutTypeOrElement**	1
ReceiveReply	**PropertyWithTypeAndElement**	1

Table 18: SA00019 test pairs

1.20 SA00020

”A <vprop:propertyAlias> element MUST use one of the three following combinations of attributes: messageType and part, type or element”[2, p. 195]

Even though rule #20 is about WSDL constraints similarly to #19, [1, p. 13] model it. We look directly at the attributes of a `<propertyAlias>` and identify following possible combinations.

[`@messageType`, none]
×
[`@part`, none]
×
[`@type`, none]
×
[`@element`, none]

Formalization 15: SA Rule #20

The single `@element`, `@type` or the pair `@messageType` and `@part` are valid attributes, i.e., three out of the 16 combinations are valid. Hence, 13 combinations are erroneous, requiring 13 tests.

The derived test cases import all a modified WSDL that have all combinations of `@messageType`, `@part`, `@type`, and `@element` except the three allowed combinations.

PropertyAlias-AllOptionalAttributes imports TestInterface-PropertyAlias-AllOptionalAttributes.ws that contains modified first `<propertyAlias>` with `@messageType`, `@part`, `@type` and `@element`.

PropertyAlias-MessageTypeAttribute imports TestInterface-PropertyAlias-MessageType-Attribute.wsdl that contains modified first `<propertyAlias>` with `@messageType` but no attribute out of `@part`, `@type`, `@element`.

PropertyAlias-MessageTypeElementAttributes imports TestInterface-PropertyAlias-Message-TypeElementAttributes.wsdl that contains modified first `<propertyAlias>` with `@mes-sageType` and `@element` but no attribute out of `@part`, `@type`.

PropertyAlias-MessageTypePartElementAttributes imports TestInterface-PropertyAlias-MessageTypePartElementAttributes.wsdl that contains modified first `<propertyAlias>` with `@messageType`, `@part` and `@element` but no attribute `@type`.

PropertyAlias-MessageTypePartTypeAttributes imports TestInterface-PropertyAlias-Message-TypePartTypeAttributes.wsdl that contains modified first `<propertyAlias>` with `@part`, `@messageType` and `@type` but no attribute `@element`.

PropertyAlias-MessageTypeTypeAttributes imports TestInterface-PropertyAlias-Message-TypeTypeAttributes.wsdl that contains modified first `<propertyAlias>` with `@messageType` and `@type` but no attribute out of `@part`, `@element`.

PropertyAlias-MessageTypeTypeElementAttributes imports TestInterface-PropertyAlias-MessageTypeTypeElementAttributes.wsdl that contains modified first `<propertyAlias>` with `@messageType`, `@element` and `@type` but no attribute `@part`.

PropertyAlias-NoOptionalAttributes imports TestInterface-PropertyAlias-NoOptionalAttributes.ws that contains modified first `<propertyAlias>` with no attribute out of `@messageType`, `@part`, `@type`, `@element`.

PropertyAlias-PartAttribute imports TestInterface-PropertyAlias-PartAttribute.wsdl that contains modified first `<propertyAlias>` with `@part` but no attribute out of `@mes-sageType`, `@type`, `@element`.

PropertyAlias-PartElementAttributes imports TestInterface-PropertyAlias-PartElement-Attributes.wsdl that contains modified first `<propertyAlias>` with `@part` and `@element` but no attribute out of `@messageType`, `@type`.

PropertyAlias-PartTypeAttributes imports TestInterface-PropertyAlias-PartTypeAttributes.wsdl that contains modified first `<propertyAlias>` with `@part` and `@type` but no attribute out of `@messageType`, `@element`.

PropertyAlias-PartTypeElementAttributes imports TestInterface-PropertyAlias-PartType-ElementAttributes.wsdl that contains modified first `<propertyAlias>` with `@part`, `@type` and `@element` but no attribute `@messageType`.

PropertyAlias-TypeElementAttributes.bpel imports TestInterface-PropertyAlias-Type-ElementAttributes.wsdl that contains modified first `<propertyAlias>` with `@type` and `@element` but no attribute out of `@messageType`, `@part`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Empty	**PropertyAlias-AllOptionalAttributes**	1
Empty	**PropertyAlias-MessageTypeAttribute**	1
Empty	**PropertyAlias-MessageTypeElementAttributes**	1
Empty	**PropertyAlias-MessageTypePartElementAttributes**	1
Empty	**PropertyAlias-MessageTypePartTypeAttributes**	1
Empty	**PropertyAlias-MessageTypeTypeAttributes**	1
Empty	**PropertyAlias-MessageTypeTypeElementAttributes**	1
Empty	**PropertyAlias-NoOptionalAttributes**	1
Empty	**PropertyAlias-PartAttribute**	1
Empty	**PropertyAlias-PartElementAttributes**	1
Empty	**PropertyAlias-PartTypeAttributes**	1
Empty	**PropertyAlias-PartTypeElementAttributes**	1
Empty	**PropertyAlias-TypeElementAttributes**	1

Table 19: SA00020 test pairs

1.21 SA00021

"Static analysis MUST detect property usages where propertyAliases for the associated variable's type are not found in any WSDL definitions directly imported by the WS-BPEL process."[2, p. 196]

Rule #21 is marked experimental, as we do not handle detecting the correct property usage via the `getVariableProperty` function.

Rule #21 is not mentioned in the model of Kopp et al. [1]. According to the rule, we have to identify the property usages in the message activities with `<correlation>`, and in the data flow activity `<assign>` in `<from>` or `<to>` directives. As the `<propertyAlias>` links a `<property>` to either a `<message>`, an XSD type or XSD `<element>` and vice versa, we assume that both are available for our various combinations because there are other rules covering their absence.

[`<receive>`, `<reply>`, `<onMessage>`, `<invoke>`, `<onEvent>`, `<from>`, `<to>`]
$\times$
[correct `<propertyAlias>` exists, correct `<propertyAlias>` is missing]

Formalization 16: SA Rule #21

Seven test cases are required that have no `<propertyAlias>` for the `<property>` that is used in the `<correlation>` or assignment.

From-Property-AlienAlias imports TestInterface-AlienAlias.wsdl that adds an additional part to the first `<message>`. This part has an additional type definition, which does not

comply with the type of the <property>, but it is used as <propertyAlias> of the third <message>. For the type definition a XML schema is imported.

OnEvent-Variable-AlienAlias imports TestInterface-AlienAlias.wsdl that adds an additional part to the first <message>. This part has an additional type definition, which does not comply with the type of the <property>, but it is used as <propertyAlias> of the third <message>. For the type definition a XML schema is imported. The <process> has no part attribute in the <from>, but a property attribute.

Assign-To-Property imports TestInterface-To-AlienAlias.wsdl that adds an additional part to the third <message>. This part has an additional type definition, which does not comply with the type of the <property>, but it is used as <propertyAlias> of the third <message>. For the type definition a XML schema is imported.

InvokeCorrelationAlienAlias imports TestPartner-AlienAlias.wsdl that adds an additional part to the first <message>. This part has an additional type definition, which does not comply with the type of the <property>, but it is used as <propertyAlias> of the third <message>. For the type definition a XML schema is imported.

InvokeToCorrelationAlienAlias imports TestPartner-To-AlienAlias.wsdl that adds an additional part to the third <message>. This part has an additional type definition, which does not comply with the type of the <property>, but it is used as <propertyAlias> of the third <message>. For the type definition a XML schema is imported.

OnEventCorrelationAlienAlias imports TestInterface-AlienAlias.wsdl that adds an additional part to the first <message>. This part has an additional type definition, which does not comply with the type of the <property>, but it is used as <propertyAlias> of the third <message>. For the type definition a XML schema is imported.

OnMessageCorrelationAlienAlias imports TestInterface-AlienAlias.wsdl that adds an additional part to the first <message>. This part has an additional type definition, which does not comply with the type of the <property>, but it is used as <propertyAlias> of the third <message>. For the type definition a XML schema is imported.

ReceiveCorrelationAlienAlias imports TestInterface-AlienAlias.wsdl that adds an additional part to the first <message>. This part has an additional type definition, which does not comply with the type of the <property>, but it is used as <propertyAlias> of the third <message>. For the type definition a XML schema is imported.

ReplyCorrelationAlienAlias imports TestInterface-To-AlienAlias.wsdl that adds an additional part to the third <message>. This part has an additional type definition, which does not comply with the type of the <property>, but it is used as <propertyAlias> of the third <message>. For the type definition a XML schema is imported.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Assign-Property	**From-Property-AlienAlias**	1
Scope-EventHandlers-InitAsync	**OnEvent-Variable-AlienAlias**	1

Assign-To-Property	**To-Property-AlienAlias**	1
Invoke-Correlation-Pattern-InitSync	**InvokeCorrelationAlienAlias**	1
Invoke-Correlation-Pattern-InitSync	**InvokeToCorrelationAlienAlias**	1
Scope-EventHandlers-InitAsync	**OnEventCorrelationAlienAlias**	1
Pick-Correlations-InitAsync	**OnMessageCorrelationAlienAlias**	1
ReceiveReply-Correlation-InitSync	**ReceiveCorrelationAlienAlias**	1
ReceiveReply-Correlation-InitSync	**ReplyCorrelationAlienAlias**	1

Table 20: SA00021 test pairs

1.22 SA00022

”A WS-BPEL process definition MUST NOT be accepted for processing if it defines two or more propertyAliases for the same property name and WS-BPEL variable type.”[2, p. 196]

Each `<propertyAlias>` element refers to either an XSD `<element>` via `@element`, an XSD type via `@type`, or a WSDL `<message>` and its `<part>` via `@messageType` and `@part`. For modeling this rule, we can omit the part, as a `<propertyAlias>` links a specific `<message>` to a specific `<property>` regardless of their part because correlation works on the message level and not on the part level. To test violations against rule #22, we require duplicate `<propertyAlias>` elements for each of the three possibilities. The rule model of [1, p. 13] can be negated to define the error cases.

Duplicate-propertyAliasElement imports TestInterface-Duplicate-propertyAliasElement.wsdl, which has two `<propertyAlias>` entries, where `@propertyName='tns:correlationId'` and `@element='tns:executeProcessSyncRequest'` are the same.

Duplicate-propertyAliasMessageType imports TestInterface-Duplicate-propertyAliasMessageType.w which duplicates the first `<propertyAlias>`.

Duplicate-propertyAliasType imports TestInterface-Duplicate-propertyAliasType.wsdl, which has two `<propertyAlias>` entries, where `@propertyName='tns:correlationId'` and `@type='xsd:string'` are the same.

DoubleImportedPropertyAlias imports TestInterface-DoubleImportedPropertyAlias.wsdl, which has renamed major elements except a single `<propertyAlias>` points to the original `<message>`. The modified WSDL imports the base WSDL and does not have a own `<property>`, but share the one of the base file.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Empty	**Duplicate-propertyAliasElement**	1
Empty	**Duplicate-propertyAliasMessageType**	1
Empty	**Duplicate-propertyAliasType**	1
Sequence	**DoubleImportedPropertyAlias**	1

Table 21: SA00022 test pairs

1.23 SA00023

”The name of a variable MUST be unique among the names of all variables defined within the same immediately enclosing scope.”[2, p. 196]

Like all uniqueness checks, rule #23 can be tested by means of duplicated names. The `@name`

of a variable has to be unique within its variable container (`<variables>`). This container can be within a `<scope>` or `<process>` element, hence, we require two test cases, one for each of the different locations of the container with duplicate `<variable>`s. The negation of this rule model by [1, p. 22] reveals the two necessary tests that we mentioned above.

Process-Duplicated-Variables contains a `<variable>` duplicate with `@name='ReplyData'` within `<process>`.

Scope-Duplicated-Variables contains a `<variable>` duplicate with `@name='ReplyData'` within a `<scope>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Variables-DeafaultInitialization	**Process-Duplicated-Variables**	-
Scope-Variables	**Scope-Duplicated-Variables**	-

Table 22: SA00023 test pairs

1.24 SA00024

"Variable names are BPELVariableNames, that is, NCNames (as defined in XML Schema specification) but in addition they MUST NOT contain the '.' character."[2, p. 196]

The `@name` is possible for almost every element in a BPEL process. Each test for rule #24 has a BPEL element with a `@name` containing a ".", which can be identified by negating the rule model by [1, p. 22]. This rule can be checked by means of a schema validation with the official XSD of the BPEL specification. This is the only rule that is already covered as part of the schema validation, which we assume the engines already use extensively. Because of this, instead of creating a plethora of tests, we have created a single test case that reveals whether this rule is checked or not. The test case uses bad variable names because they are crucial for any BPEL process.

Variable-containing-dot has a `<variable>` where the value of `name` has a dot at the end of the string. All usages of the variable name are replaced with the new name.

OnEvent-containing-dot has `<onEvent>` where the value of `variable` has dots within the string. All usages of the variable name are replaced with the new name.

CatchContainingDot has `<catch>` where the value of `faultVariable` has a dot within the string.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Variables-DefaultInitialization	**Variable-containing-dot**	-
Scope-EventHandlers-InitSync	**OnEvent-containing-dot**	-
Scope-FaultHandlers-CatchOrder	**CatchContainingDot**	-

Table 23: SA00024 test pairs

1.25 SA00025

"The messageType, type or element attributes are used to specify the type of a variable. Exactly one of these attributes MUST be used."[2, p. 196]

Rule #25 is not defined precisely in [1, p. 11], but the required tests can be easily identified by permuting the three attributes.

[`@element`, none]
×
[`@type`, none]
×
[`@messageType`, none]

Formalization 17: SA Rule #25

Three combinations are valid; each has just one attribute. Therefore, we require five tests in total.

Variable-havingMessageTypeAndElement has the first `<variable>` modified, so an additional `element` attribute is specified.

Variable-havingTypeAndElement lacks a `messageType` attribute in the first `<variable>`, but specifies `type` and `element` attributes instead. In the assignment, that uses the variable, no `part` attribute can be used because the `messageType` is missing.

Variable-havingTypeAndMessageType has the first `<variable>` modified, so an additional `type` attribute is specified.

Variable-havingTypeAndMessageTypeAndElement has the first `<variable>` modified, so additional `type` and `element` attributes are specified.

Variable-missingMessageTypeAndTypeAndElement has a additional `<variable>` without any type defining attributes.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Variables-DefaultInitialization	**Variable-havingMessageTypeAndElement**	-
Variables-DefaultInitialization	**Variable-havingTypeAndElement**	-
Variables-DefaultInitialization	**Variable-havingTypeAndMessageType**	-
Variables-DefaultInitialization	**Variable-havingTypeAndMessageTypeAndElement**	-
Variables-DefaultInitialization	**Variable-missingMessageTypeAndTypeAndElement**	-

Table 24: SA00025 test pairs

1.26 SA00026

”Variable initialization logic contained in scopes that contain or whose children contain a start activity MUST only use idempotent functions in the from-spec.”[2, p. 196] Because the rule deals with general expression parsing, it is postponed to future work.

1.27 SA00027

"When XPath 1.0 is used as an expression language in WS-BPEL there is no context node available. Therefore the legal values of the XPath Expr (http://www.w3.org/TR/xpath#NT-Expr) production must be restricted in order to prevent access to the context node. Specifically, the 'LocationPath' (http://www.w3.org/TR/xpath#NT-LocationPath) production rule of 'PathExpr' (http://www.w3.org/TR/xpath#NT-PathExpr) production rule MUST NOT be used when XPath is used as an expression language."[2, p. 196] The rule deals with the parsing of XPath expressions and is postponed to future work.

1.28 SA00028

"WS-BPEL functions MUST NOT be used in joinConditions."[2, p. 196] Because the rule deals with general expression parsing, it is postponed to future work.

1.29 SA00029

"WS-BPEL variables and WS-BPEL functions MUST NOT be used in query expressions of propertyAlias definitions."[2, p. 196] Because the rule deals with general expression parsing, it is postponed to future work.

1.30 SA00030

"The arguments to bpel:getVariableProperty MUST be given as quoted strings. It is therefore illegal to pass into a WS-BPEL XPath function any XPath variables, the output of XPath functions, a XPath location path or any other value that is not a quoted string."[2, p. 196] The rule deals with the parsing of XPath expressions and is postponed to future work.

1.31 SA00031

"The second argument of the XPath 1.0 extension function bpel:getVariableProperty(string, string) MUST be a string literal conforming to the definition of QName in [XML Namespaces] section 3."[2, p. 197] The parsing of BPEL functions, as required for this rule, is postponed to future work.

1.32 SA00032

”For <assign>, the <from> and <to> element MUST be one of the specified variants. The <assign> activity copies a type-compatible value from the source (’from-spec’) to the destination (’to-spec’), using the <copy> element. Except in Abstract Processes, the fromspec MUST be one of the following variants:

```
<from variable='BPELVariableName' part='NCName'?>
    <query queryLanguage='anyURI'?>?queryContent </query>
</from>

<from partnerLink='NCName'  endpointReference='myRole|partnerRole' />

<from variable='BPELVariableName'  property='QName' />

<from expressionLanguage='anyURI'?> expression </from>

<from>
    <literal>literal value</literal>
</from>

<from/>
```

In Abstract Processes, the from-spec MUST be either one of the above or the opaque variant described in section 13.1.3. Hiding Syntactic Elements The to-spec MUST be one of the following variants:

```
<to variable='BPELVariableName' part='NCName'?>
    <query queryLanguage='anyURI'?>?queryContent </query>
</to>

<to partnerLink='NCName' />

<to variable='BPELVariableName'  property='QName' />

<to expressionLanguage='anyURI'?> expression </to>
```

<to/>”[2, p. 197]

The rule #32 model by [1, p. 37] ignores the <from>, and the definition is vague and more textual than the original rule. To get the tests we modified each of the six variants of <from> (and five forms of <to>) by adding elements or attributes.

The empty version of both <from> and <to> is ignored, as it is meaningless.

[`@property`, `@part`, `<query>`, `@part` and `<query>`, `@property` and `@part`, `@property` and `@query`,]
×
one additional one out of [none, `@partnerLink`, `@endpointReference`, `expression`, `@expressionLanguage`, `<literal>`]

Formalization 18: SA Rule #32 for `<from>` with `@variable`

Out of the 36 combinations, only four are valid, hence, we have 32 faulty combinations. However, we ignore ten of them because `@property` and `@part` as well as `@property` and `@query` already violates this rule.

one additional one out of [none, `@variable`, `expression`, `@expressionLanguage`, `<literal>`, `@property`, `@part`, `<query>`]

Formalization 19: SA Rule #32 for `<from>` with `@partnerLink` and `@endpointReference`

Hence, we have seven test cases.

[`expression`, `@expressionLanguage`, `expression` and `@expressionLanguage`,]
×
one additional one out of [none, `@partnerLink`, `@variable`, `@part`, `<query>`, `@property`, `@endpointReference`, `<literal>`]

Formalization 20: SA Rule #32 for `<from>` with `expression`

Hence, we have 15 test cases.

Hence, we have eight test cases, with one valid case.

Hence, we have 14 test cases.

Hence, we have six test cases.

Hence, we have eleven test cases.

In addition, we created six additional cases for `<from>` with only a single element, and four additional cases for `<to>` for a subset of these conditions.

The rule is very specific, so we create tests for combinations that are not allowed. No WSDLs are modified from the `betsy` test cases. Because of the long names the indication row for such WSDLs is left out.

FromExpressionLanguageSuperfliciousAttributes has an additional `part` attribute in the `<from>` element.

FromExpressionLanguageSuperfliciousChild has an additional `<query>` child in the `<from>` element.

FromLiteralSuperfliciousAttribute has an additional `part` attribute in the `<from>` element.

one additional one out of [none, `@partnerLink`, `@variable`, `expression`, `@expressionLanguage`, `@endpointReference`, `@property`, `@part`, `<query>`]

Formalization 21: SA Rule #32 for `<from>` with `<literal>`

[`@property`, `@part`, `<query>`, `@part` and `<query>`, `@property` and `@part`, `@property` and `@query`,]
×
one additional one out of [none, `@partnerLink`, `expression`, `@expressionLanguage`]

Formalization 22: SA Rule #32 for `<to>` with `@variable`

FromLiteralSuperfliciousChild has an additional `<documentation>` child in the `<from>` element.

FromMessageTypeVariableSuperfliciousAttribute has an additional `expressionLanguage` attribute in the `<from>` element.

FromPartnerLinkMissingEndpointReferenceAttribute has no `partnerLink` attribute in the `<from>` element.

FromPartnerLinkSuperfliciousAttribute has an additional `part` attribute in the `<from>` element.

FromPartnerLinkSuperfliciousChild has an additional `<query>` child in the `<from>` element.

FromVariablePropertySuperfliciousAttribute has an additional `part` attribute in the `<from>` element.

FromVariablePropertySuperfliciousChild has an additional `<query>` child in the `<from>` element.

FromVariableQueryAdditionalAttribute has an additional attribute from a different namespace in the `<query>` element.

FromVariableQueryAdditionalChild has an additional `<documentation>` child in the `<from>` element.

FromVariableQuerySuperfliciousAttribute has an additional attribute from a different namespace in the `<query>` element.

FromVariableSuperfliciousChild has an additional `<literal>` child in the `<from>` element.

ToExpressionLanguageSuperfliciousAttributes has an additional `part` attribute in the `<to>` element.

ToExpressionLanguageSuperfliciousChild has an additional `<query>` child in the `<to>` element.

ToMessageTypeVariableSuperfluousAttribute has an additional `expressionLanguage` attribute in the `<to>` element.

one additional one out of [none, `@variable`, `expression`, `@expressionLanguage`, `@property`, `@part`, `<query>`]

Formalization 23: SA Rule #32 for `<to>` with `@partnerLink`

[`expression`, `@expressionLanguage`, `expression` and `@expressionLanguage`,] × one additional one out of [none, `@partnerLink`, `@variable`, `@part`, `<query>`, `@property`]

Formalization 24: SA Rule #32 for `<to>` with `expression`

ToPartnerLinkSuperfliciousAttribute has an additional `part` attribute in the `<to>` element.

ToPartnerLinkSuperfliciousChild has an additional attribute from a different namespace in the `<query>` element.

ToVariablePropertySuperfliciousAttribute has an additional `part` attribute in the `<to>` element.

ToVariablePropertySuperfliciousChild has an additional `<query>` child in the `<to>` element.

ToVariableQueryAdditionalAttribute has an additional attribute from a different namespace in the `<query>` element.

ToVariableQueryAdditionalChild has an additional `<documentation>` child in the `<to>` element.

ToVariableQuerySuperfliciousAttribute has an additional attribute from a different namespace in the `<to>` element.

ToVariableSuperfliciousChild has an additional `<documentation>` child in the `<to>` element.

betsy Test Case Origin	Derived Test Case
Assign-ExpressionLanguage-From	**FromExpressionLanguageSuperfliciousAttributes**
Assign-ExpressionLanguage-From	**FromExpressionLanguageSuperfliciousChild**
Assign-Literal	**FromLiteralSuperfliciousAttribute**
Assign-Literal	**FromLiteralSuperfliciousChild**
Assign-Expression-To	**FromMessageTypeVariableSuperfliciousAttribute**
Assign-PartnerLink-PartnerLink	**FromPartnerLinkMissingEndpointReferenceAttribute**
Assign-PartnerLink-PartnerLink	**FromPartnerLinkSuperfliciousAttribute**
Assign-PartnerLink-PartnerLink	**FromPartnerLinkSuperfliciousChild**
Assign-Property	**FromVariablePropertySuperfliciousAttribute**

Assign-Property	**FromVariablePropertySuperficiousChild**
Assign-Copy-QueryLanguage	**FromVariableQueryAdditionalAttribute**
Assign-Copy-QueryLanguage	**FromVariableQueryAdditionalChild**
Assign-Copy-Query	**FromVariableQuerySuperficiousAttribute**
Assign-Expression-To	**FromVariableSuperficiousChild**
Assign-ExpressionLanguage-To	**ToExpressionLanguageSuperficiousAttributes**
Assign-ExpressionLanguage-To	**ToExpressionLanguageSuperficiousChild**
Assign-Expression-From	**ToMessageTypeVariableSuperfluousAttribute**
Assign-PartnerLink	**ToPartnerLinkSuperficiousAttribute**
Assign-PartnerLink	**ToPartnerLinkSuperficiousChild**
Assign-To-Property	**ToVariablePropertySuperficiousAttribute**
Assign-To-Property	**ToVariablePropertySuperficiousChild**
Assign-To-QueryLanguage	**ToVariableQueryAdditionalAttribute**
Assign-To-QueryLanguage	**ToVariableQueryAdditionalChild**
Assign-To-Query	**ToVariableQuerySuperficiousAttribute**
Assign-Expression-From	**ToVariableSuperficiousChild**

Table 25: SA00032 test pairs

1.33 SA00033

”The XPath expression in <to> MUST begin with an XPath VariableReference.”[2, p. 198] The rule deals with the parsing of XPath expressions and is postponed to future work.

1.34 SA00034

”When the variable used in <from> or <to> is defined using XML Schema types (simple or complex) or element, the part attribute MUST NOT be used.”[2, p. 198]

[1, p. 35] define the positive model of rule #34 for `<from>` but not for `<to>` elements. The other attributes combined with the `@part` are the tests for this rule.

[`<from>`, `<to>`]
×
[`<variable>` `@element`, `<variable>` `@type`, `<variable>` `@messageType`, `<onEvent>` `@variable` `@element`, `<onEvent>` `@variable` `@messageType`, `<forEach>` `@counterName`]
×
[`@part`, none]

Formalization 25: SA Rule #34

All tests with `@messageType` and `@part` are valid, as well as all variants without `<part>`. Thus, we need four tests for each `<from>` and `<to>`, leading to eight tests in total to test all aspects of rule #34.

FromElementVariablePartAttribute has an additional `part` attribute in the `<from>` element referring a definition from months.xsd.

FromOnEventElementVariablePartAttribute has an additional `part` attribute in the `<from>` element.

FromTypeVariablePartAttribute has an additional `part` attribute in the `<from>` of the assignment.

ToOnEventElementVariablePartAttribute contains an additional literal assignment that uses the `part` attribute in `<to>` referencing to the `<onEvent>` variable.

ToElementVariablePartAttribute has an additional `part` attribute in the first `<to>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Assign-Element-Variable	**FromElementVariablePartAttribute**	-
Scope-EventHandlers-Element-InitAsync	**FromOnEventElementVariablePartAttribute**	-
Variables-DefaultInitialization	**FromTypeVariablePartAttribute**	-
Assign-Element-Variable	**ToElementVariablePartAttribute**	-
Scope-EventHandlers-Element-InitAsync	**ToOnEventElementVariablePartAttribute**	-
Assign-Validate	**ToTypeVariablePartAttribute**	-

Table 26: SA00034 test pairs

1.35 SA00035

”In the from-spec of the partnerLink variant of <assign> the value ”myRole” for attribute endpointReference is only permitted when the partnerLink specifies the attribute myRole.”[2, p. 198]

If there is no `@myRole` in the referenced `<partnerLink>` used in an `<assign>`, but the `endpointReference="` then rule #35 is violated, which can be evaluated in a single test. In the model of [1, p. 36], the concrete usage of the `<partnerLink>` is unmentioned, however, the negation of the model indicates the test case.

FromLinkTypeMyRolePartnerLinkWithoutMyRole has a substituted `endpointReference` attribute (`myRole` instead of `partnerRole`).

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Assign-PartnerLink-PartnerRole	**FromLinkTypeMyRolePartnerLinkWithoutMyRole**	-

Table 27: SA00035 test pairs

1.36 SA00036

”In the from-spec of the partnerLink variant of <assign> the value ”partnerRole” for attribute endpointReference is only permitted when the partnerLink specifies the attribute partnerRole.”[2, p. 198]

Rule #36 resembles #35, so does the model by [1, p. 36]. They differ in the role only, as rule #36 refers to the `partnerRole` instead of the `myRole`.

FromPartnerRoleWithoutPartnerRolePartnerLink has a `myRole` attribute in the third `<partnerLink>` instead of a `partnerRole` attribute.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Assign-PartnerLink-PartnerRole	**FromPartnerRoleWithoutPartnerRolePartnerLink**	-

Table 28: SA00036 test pairs

1.37 SA00037

”In the to-spec of the partnerLink variant of assign only partnerLinks are permitted which specify the attribute partnerRole.”[2, p. 198]

Negating the rule #37 model by [1, p. 38] shows the single test required: The `<partnerLink>` shall have no `@partnerRole` when referenced from a `<to>`.

ToLinkTypeWithoutPartnerRolePartnerLink has a `myRole` attribute in the second `<partnerLink>` instead of a `partnerRole` attribute.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Assign-PartnerLink-PartnerRole	**ToLinkTypeWithoutPartnerRolePartnerLink**	-

Table 29: SA00037 test pairs

1.38 SA00038

”The literal from-spec variant returns values as if it were a from-spec that selects the children of the <literal> element in the WS-BPEL source code. The return value MUST be a single EII or Text Information Item (TII) only.”[2, p. 198]

The rule is postponed to future work because it deals with expression parsing of arbitrary literal expressions.

1.39 SA00039

”The first parameter of the XPath 1.0 extension function bpel:doXslTransform(string, node-set, (string, object)*) is an XPath string providing a URI naming the style sheet to be used by the WS-BPEL processor. This MUST take the form of a string literal.”[2, p. 198]

The parsing of BPEL functions, as required for this rule, is postponed to future work.

1.40 SA00040

”In the XPath 1.0 extension function bpel:doXslTransform(string, node-set, (string, object)*) the optional parameters after the second parameter MUST appear in pairs. An odd number of parameters is not valid.”[2, p. 198]

The parsing of BPEL functions, as required for this rule, is postponed to future work.

1.41 SA00041

”For the third and subsequent parameters of the XPath 1.0 extension function bpel:doXsl-Transform(string, node-set, (string, object)*) the global parameter names MUST be string literals conforming to the definition of QName in section 3 of [Namespaces in XML].”[2, p. 198]

The parsing of BPEL functions, as required for this rule, is postponed to future work.

1.42 SA00042

”For <copy> the optional keepSrcElementName attribute is provided to further refine the behavior. It is only applicable when the results of both from-spec and to-spec are EIIs, and MUST NOT be explicitly set in other cases.”[2, p. 198] To identify the types of all `<from>` and `<to>` expression parsing is required.

Because the rule deals with general expression parsing, it is postponed to future work.

1.43 SA00043

”For a copy operation to be valid, the data referred to by the from-spec and the to-spec MUST be of compatible types. The following situations are considered type incompatible: • the selection results of both the from-spec and the to-spec are variables of a WSDL message type, and the two variables are not of the same WSDL message type (two WSDL message types are the same if their QNames are equal). • the selection result of the from-spec is a variable of a WSDL message type and that of the to-spec is not, or vice versa (parts of variables, selections of variable parts, or endpoint references cannot be assigned to/from variables of WSDL message types directly).”[2, p. 199]

To identify the types of all `<from>` and `<to>` expression parsing is required. Because the rule deals with general expression parsing, it is postponed to future work.

1.44 SA00044

”The name of a <correlationSet> MUST be unique among the names of all <correlationSet> defined within the same immediately enclosing scope.”[2, p. 199]

Rule #44 requires both `<scope>`s and `<process>`s to have unique `@name`s for `<correlationSet>` elements in their `<correlationSets>` container. We test this with a duplication test for both `<scope>` and `<process>`, hence, we have two test cases. These tests are also modeled by negating the formalization in [1, p. 32].

Process-CorrelationSet-Ambiguous contains a `<correlationSet>` duplicate in `<process>`.

Scope-CorrelationSets-Ambiguous contains a `<correlationSet>` duplicate in the `<scope>` element.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
ReceiveReply-Multiple-Message-Exchanges	**Process-CorrelationSet-Ambiguous**	-
Scope-CorrelationSets-InitAsync	**Scope-CorrelationSets-Ambiguous**	-

Table 30: SA00044 test pairs

1.45 SA00045

”Properties used in a <correlationSet> MUST be defined using XML Schema simple types.”[2, p. 199]

`<property>` elements referenced in a `<correlationSet>` with either complex types of the XML

schema or no type violate rule #45. Such `<correlationSet>` are forbidden in the rule model of [1, p. 31], thus, this results in two test cases, one for each variant of the `<correlationSet>`.

Property-TypeMissing imports TestInterface-Property-TypeMissing.wsdl, where the `type` attribute of the `<property>` definition is omitted.

Property-TypeComplexType imports TestInterface-Property-TypeComplexType.wsdl, where the `type` attribute of the `<property>` definition is a `<complexType>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
ReceiveReply-Correlation-InitAsync	**Property-TypeMissing**	-
ReceiveReply-Correlation-InitAsync	**Property-TypeComplexType**	-

Table 31: SA00045 test pairs

1.46 SA00046

”The pattern attribute used in <correlation> within <invoke> is required for request-response operations, and disallowed when a one-way operation is invoked.”[2, p. 199]

The rule #46 model by [1, p. 32] has an equivalence so we can switch sides of the parameters and negate the original and the switched statement. The two subsequent tests have either no `@pattern` in conjunction with a request-response communication or a `@pattern` in conjunction with a one-way communication.

Invoke-RequestResponse-Correlation-PatternMissing contains no `request-response pattern` in the <invoke>.

Invoke-OneWay-Correlation-Pattern contains a modified `<invoke>` that uses an one-way operation. The last `<assign>` copies from a different variable, because the former variable is not initialized any more.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Invoke-Correlation-Pattern-InitAsync	**Invoke-RequestResponse-Correlation-PatternMissing**	-
Invoke-Empty	**Invoke-OneWay-Correlation-Pattern**	-

Table 32: SA00046 test pairs

1.47 SA00047

”One-way invocation requires (<invoke>) only the inputVariable (or its equivalent <toPart> elements) since a response is not expected as part of the operation. Request-response invocation requires both an inputVariable (or its equivalent <toPart> elements) and an outputVariable (or its equivalent <fromPart> elements). If a WSDL message definition does not contain any parts, then the associated attributes variable, inputVariable or outputVariable, MAY be omitted,and the <fromParts> or <toParts> construct MUST be omitted.”[2, p. 199]

[1, p. 29] model rule #47 for the assignment of parts using `<fromPart>` or `<toPart>` for the empty message aspect, but not for non-empty messages. To violate the rule, however, you need to consider both cases.

The valid combinations are either an empty `<message>` without any `<fromParts>` or `<toParts>`, or a `<message>` with `<part>` and a variable or part assignment. Having a part assignment implies the rule is violated if the `<message>` is empty, but the variable assignment has no effect then. If we have both variable and part assignment, one of the rules #51, #52, #55, #59, #63, or #85 is violated, but not rule #47. Thus, we require twelve tests in total, two for each of the five message activities and two additional tests for `<invoke>` as this can handle sending

[empty <message>, <message> with <part>(s)]
×
[<invoke>]
×
[@inputVariable, @outputVariable, none]
×
[<fromParts>, <toParts>, none]

Formalization 26: SA Rule #47 for <invoke>

[empty <message>, <message> with <part>(s)]
×
[<receive>, <onMessage>, <onEvent>]
×
[@variable, none]
×
[<fromParts>, none]

Formalization 27: SA Rule #47 for receiving message activities

and receiving messages.

EmptyMessage-Invoke-FromParts imports TestPartner-MessageWithoutParts.wsdl, which has an additional empty <output>. This empty <message> is used by an additional <variable> as messageType attribute. The inputVariable of the <invoke> references this <variable>.

EmptyMessage-Invoke-ToParts has a modified operation attribute in the <invoke>.

EmptyMessage-OnEvent-FromParts imports TestInterface-MessageWithoutParts.wsdl, which defines an empty <message> and uses it in a synchronous <operation> as request and response. The <onEvent> operation attribute uses the new <operation>.

EmptyMessage-OnMessage-FromParts imports TestInterface-MessageWithoutParts.wsdl, which defines an empty <message> and uses it in a synchronous <operation> as request and response. The <onMessage> operation attribute uses the new <operation>.

EmptyMessage-Receive-FromParts imports TestInterface-MessageWithoutParts.wsdl, which defines an empty <message> and uses it in a synchronous <operation> as request and response. The <receive> uses the new <operation> in the corresponding operation attribute.

EmptyMessage-Reply-ToParts imports TestInterface-MessageWithoutParts.wsdl, which defines an empty <message> and uses it in a synchronous <operation> as request and response. The <reply> uses the new <operation> in the corresponding operation attribute.

Invoke-OneWay-NoInputVariable-NoToParts has no inputVariable attribute in <invoke>.

[empty <message>, <message> with <part>(s)]
×
[<reply>]
×
[@variable, none]
×
[<toParts>, none]

Formalization 28: SA Rule #47 for <reply>

Invoke-RequestResponse-NoOutputVariable-NoFromParts has no <fromParts>.

NoVariable-NoFromPart-OnEvent has no <fromParts>.

NoVariable-NoFromPart-OnMessage has no <fromParts>.

NoVariable-FromPart-Receive has no <fromParts>.

NoVariable-NoToPart-Reply has no <toParts>.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Invoke-FromParts	**EmptyMessage-Invoke-FromParts**	1
Invoke-ToParts	**EmptyMessage-Invoke-ToParts-From-Parts**	1
Invoke-ToParts	**EmptyMessage-Invoke-ToParts**	-
Scope-EventHandlers-Parts	**EmptyMessage-OnEvent-FromParts**	1
Pick-CreateInstance-FromParts	**EmptyMessage-OnMessage-FromParts**	1
ReceiveReply-FromParts	**EmptyMessage-Receive-FromParts**	1
ReceiveReply-ToParts	**EmptyMessage-Reply-ToParts**	1
Invoke-InitAsync	**Invoke-OneWay-NoInputVariable-No-ToParts**	-
Invoke-Sync	**Invoke-RequestResponse-NoInput-OutputVariables-NoToFromParts**	-
Invoke-ToParts	**Invoke-RequestResponse-NoInput-Variables-NoToParts**	-
Invoke-FromParts	**Invoke-RequestResponse-NoOutput-Variable-NoFromParts**	-
Scope-EventHandlers-Parts	**NoVariable-NoFromPart-OnEvent**	-
Pick-CreateInstance-FromParts	**NoVariable-NoFromPart-OnMessage**	-
Receive-FromParts	**NoVariable-NoFromPart-Receive**	-
ReceiveReply-ToParts	**NoVariable-NoToPart-Reply**	-
ReceiveReply-FromParts	**NoVariable-NoToPart-NoFromPart-ReceiveReply**	-

Table 33: SA00047 test pairs

1.48 SA00048

”When the optional inputVariable and outputVariable attributes are being used in an <invoke> activity, the variables referenced by inputVariable and outputVariable MUST be messageType

inputVariable and outputVariable attributes respectively."[2, p. 200]

The rules #48, #58, #87 and #90 are alike as they restrict different message activities with the same constraint: to have either `@messageType` or `@element` if there is exactly one `<part>` in a `<message>` as defined in the model of [1, p. 27] (on the same page #47 is stated with the same definition, but this is a typo and means #48). The following combinations violate rule #48, which refers to the message activity `<invoke>`.

`<invoke>` with [`@inputVariable`, `@outputVariable`]
×
[`<message>` with one `<part>`, `<message>` with two `<part>`s, `<message>` with no `<part>`s]
×
[`<variable>` with same `@element`, `<variable>` with different `@element`, `<variable>` with type `@type`, `<variable>` with same `@messageType`]

Formalization 29: SA Rule #48

If we have one `<part>` in a `<message>`, it is valid to use the `@element` instead of `@messageType` for the variable definition. Using the `@messageType` is always valid. A different `@messageType` check is already part of rule #10. Thus, we get 16 tests as a result of the permutation.

InputVariable-MessageType-Message-NotFound has a modified `messageType` attribute of input `<variable>` of the `<invoke>`.

InputVariable-Type-MessageOnePart-NotFound has a `type` instead of a `messageType` attribute for the input `<variable>` of the `<invoke>`. The `type` is set to `xsd:boolean` and the namespace abbreviation is defined in `<process>`. Also no `part` attribute is used during the initial assignment of the `<variable>`.

InputVariable-Type-MessageManyParts imports TestPartner-InputVariable-Type-MessageManyPart which has an additional `<operation>` using a message with two `<part>` elements. The `<variable>` for the `<invoke>` has a `element` instead of a `messageType` attribute and during the initial assignment of the `<variable>` no `part` attribute is used. The `<invoke>` uses the new `<operation>`.

OutputVariable-MessageType-Message-NotFound has a modified `messageType` attribute of output `<variable>` of the `<invoke>`.

OutputVariable-Type-MessageOnePart-NotFound has a `type` instead of a `messageType` attribute for the output `<variable>` of the `<invoke>`. The `type` is set to `xsd:boolean` and the namespace abbreviation is defined in `<process>`. Also no `part` attribute is used during the last assignment from the `<variable>`.

OutputVariable-Type-MessageManyParts imports TestPartner-MessageTwoParts.wsdl, which has an additional operation using a message with two `<part>` elements as a response. The `<variable>` for the `<invoke>` has an `element` instead of a `messageType` attribute and during the last assignment from the `<variable>` no `part` attribute is used. The `<invoke>` uses the new `<operation>`.

InputOutputVariable-Message-NotFound has switched the `messageType` attributes of input and and output `<variable>`s of the `<invoke>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Invoke-Async	**InputVariable-MessageType-Message-NotFound**	-
Invoke-Async	**InputVariable-Type-MessageOnePart-NotFound**	-
Invoke-Async	**InputVariable-Type-MessageManyParts**	-
Invoke-Sync	**OutputVariable-MessageType-Message-NotFound**	-
Invoke-Sync	**OutputVariable-Type-MessageOnePart-NotFound]**	-
Invoke-Sync	**OutputVariable-Type-MessageManyParts**	-
Invoke-Sync	**InputOutputVariable-Message-NotFound**	-

Table 34: SA00048 test pairs

1.49 SA00050

"When <toParts> is present, it is required to have a <toPart> for every part in the WSDL message definition; the order in which parts are specified is irrelevant. Parts not explicitly represented by <toPart> elements would result in uninitialized parts in the target anonymous WSDL variable used by the <invoke> or <reply> activity. Such processes with missing <toPart> elements MUST be rejected during static analysis. "[2, p. 200]

The two sending activities `<invoke>` and `<reply>` violate rule #50 with at least one unmentioned `<part>` in `<toParts>`. We need two tests, one for each sending activity. [1, p. 28] define this rule along with #54 by bijectively mapping the parts of the message with the toParts. But for this rule, we are solely interested in the direction $Set_{\texttt{<part>}} \hookrightarrow Set_{\texttt{<toPart>}}$, as every `<part>` has to be used in the `<toPart>`s, but not vice versa.

Concerning message activities that leaves the process for each `<part>` of the message a `<toPart>` is required.

Invoke-MissingToPart imports TestPartner-Invoke-MissingToPart.wsdl, which has an additional `<part>` in the synchronous request message, thus, a `<toPart>` in the `<invoke>` is missing.

Receive-MissingToPart imports TestInterface-Receive-MissingToPart.wsdl, which has an additional `<part>` in the synchronous response message, thus, a `<toPart>` in the `<reply>` is missing.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Invoke-ToParts	**Invoke-MissingToPart**	1

ReceiveReply-ToParts	**Receive-MissingToPart**	1

Table 35: SA00050 test pairs

1.50 SA00051

"The inputVariable attribute MUST NOT be used on an invoke activity that contains <toPart> elements."[2, p. 200]

The rule #51 model by [1, p. 29] is defined along with the one of #59. In these two rules, it is forbidden to use both a `<toPart>` and a variable. As this rule refers to `<invoke>`, the test has an `@inputVariable` and a `<toPart>`.

Invoke-ToPartsAndInputVariable contains an additional `inputVariable` attribute in the <invoke>.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Invoke-ToParts	**Invoke-ToPartsAndInputVariable**	-

Table 36: SA00051 test pairs

1.51 SA00052

"The outputVariable attribute MUST NOT be used on an <invoke> activity that contains a <fromPart> element."[2, p. 200]

[1, p. 29] model the rules #52, #55, #63, and #85 together. All rules deal with the exclusive use of a variable or `<fromParts>`, similar to the definitions of `<toParts>` in rules #51 and #59. Rule #52 has a BPEL process with `<toParts>` and an `@outputVariable` in an `<invoke>` as test.

Invoke-FromPartsAndOutputVariable contains an additional `outputVariable` attribute in the <invoke>.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Invoke-FromParts	**Invoke-FromPartsAndOutputVariable**	-

Table 37: SA00052 test pairs

1.52 SA00053

"For all <fromPart> elements the part attribute MUST reference a valid message part in the WSDL message for the operation."[2, p. 200]

The rule #53 deals with <fromPart> elements and is not as strict as for <toPart> elements, because only one implication is checked in contrast to both directions (#50 and #54). It is mentioned by [1, p. 27], but is excluded later on [1, p. 65]. There are four tests having at least one more <fromPart> in either <receive>, <onMessage>, <invoke> or <onEvent>, and thereby violating rule #53.

Invoke-FromPartDifferingFromMessageDefinition contains a <fromPart> more in <invoke> as the required by the WSDL file.

OnEvent-FromPartDifferingFromMessageDefinition contains an additional <fromPart> in <onEvent>.

OnMessage-FromPartDifferingFromMessageDefinition contains a <fromParts> in <onMessage> with one superfluous <fromPart>.

Receive-FromPartDifferingFromMessageDefinition contains an additional <fromPart> in <receive>.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Invoke-FromParts	**Invoke-FromPartDifferingFromMessageDefinition**	-
Scope-EventHandlers-Parts	**OnEvent-FromPartDifferingFromMessageDefinition**	-
Pick-Correlations-InitSync	**OnMessage-FromPartDifferingFromMessageDefinition**	-
ReceiveReply-FromParts	**Receive-FromPartDifferingFromMessageDefinition**	-

Table 38: SA00053 test pairs

1.53 SA00054

"For all <toPart> elements the part attribute MUST reference a valid message part in the WSDL message for the operation. "[2, p. 200]

Rule #54 is the converse direction of #50 $Set_{\texttt{<part>}} \hookleftarrow Set_{\texttt{<toPart>}}$, so an erroneous test has at least one more <toPart> than <message> <part> elements. Therefore, two tests are required one for <reply>, another for <receive>.

Invoke-ToPartDifferingFromMessageDefinition contains an additional <toPart> in <invoke>.

Reply-ToPartDifferingFromMessageDefinition contains an additional <toPart> in <reply>.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
InvokeToParts	**Invoke-ToPartDifferingFromMessageDefinition**	-
ReceiveReply-ToParts	**Reply-ToPartDifferingFromMessage-Definition**	-

Table 39: SA00054 test pairs

1.54 SA00055

”For <receive>, if <fromPart> elements are used on a <receive> activity then the variable attribute MUST NOT be used on the same activity.”[2, p. 200]

A process containing a `<receive>` with a `@variable` and `<fromParts>` violates rule #55. This is similar to #52 which is defined accordingly by [1, p. 29].

Receive-WithFromPartElementAndVariableAttribute contains an additional `variable` attribute in <receive> and a suitable variable.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
ReceiveReply-FromParts	**Receive-WithFromPartElementAnd-VariableAttribute**	-

Table 40: SA00055 test pairs

1.55 SA00056

"A 'start activity' is a <receive> or <pick> activity that is annotated with a createInstance='yes' attribute. Activities other than the following: start activities, <scope>, <flow> and <sequence> MUST NOT be performed prior to or simultaneously with start activities."[2, p. 200]

A starting `<receive>` or `<onMessage>` (in an `<pick>`) in another structured element than the permitted ones of #56 violates this rule. The same applies to activities executed before or in parallel to the start activities which are not start activities by themselves. We cannot identify these activities from the rule model by [1, p. 50]. In the formalizations, we ignore multiple nestings, focusing on a single nesting relation only.

[`<onMessage>` in a `<pick>` with `createInstance="yes"`, `<receive>` with `createInstance="yes"`]
×
in [`<process>`, `<sequence>`, `<flow>`, `<scope>`, `<catch>`, `<catchAll>`, `<else>`, `<elseIf>`, `<forEach>`, `<if>`, `<onMessage>` in a `<pick>` with `createInstance="no"`, `<onAlarm>`, `<onEvent>`, `<repeatUntil>`, `<terminationHandler>`, `<compensationHandler>`, `<while>`]

Formalization 30: SA Rule #56 with start activities within invalid encompassing activities

For the first formalization, we get 13 test cases, as four of the 17 activities (namely, `<process>`, `<sequence>`, `<flow>`, `<scope>`) are allowed, for each start activity. Hence, there are 26 test cases in total.

[`<onMessage>` in a `<pick>` with `createInstance="yes"`, `<receive>` with `createInstance="yes"`]
×
activity before [`<onMessage>` in a `<pick>` with `createInstance="yes"`, `<receive>` with `createInstance="yes"`, none, `<assign>`, `<exit>`, `<empty>`, `<invoke>`, `<receive>`, `<throw>`, `<reply>`, `<wait>`, `<else>`, `<elseIf>`, `<forEach>`, `<if>`, `<onMessage>` in a `<pick>` with `createInstance="no"`, `<onAlarm>` in a `<pick>`, `<repeatUntil>`, `<while>`]

Formalization 31: SA Rule #56 with start activities in sequence

For the second formalization, we have 16 test cases for `<sequence>` for each start activity, totaling to 32 test cases, as valid start activities or no activities can happen before.

For the third formalization, we have 16 test cases for either no links or links to start activities for each start activity, totaling in 64 test cases.

[<onMessage> in a <pick> with createInstance="yes", <receive> with createInstance="yes"]
×
activities in parallel [<onMessage> in a <pick> with createInstance="yes", <receive> with createInstance="yes", none, <assign>, <exit>, <empty>, <invoke>, <receive>, <throw>, <reply>, <wait>, <else>, <elseIf>, <forEach>, <if>, <onMessage> in a <pick> with createInstance="no", <onAlarm> in a <pick>, <repeatUntil>, <while>]
×
[link from start activity to other activity, link to start activity from other activity, none]

Formalization 32: SA Rule #56 with start activities in flow

StartPickInCatch contains an additional start activity <pick> in <catch>.

StartPickInCatchAll contains an additional start activity <pick> in <catchAll>.

StartPickInElse contains an additional start activity <pick> in <else>.

StartPickInElseIf contains an additional start activity <pick> in <elseIf>.

StartPickInForEach contains an additional start activity <pick> in <forEach>.

StartPickInIf contains an additional start activity <pick> in <if>.

StartPickInNonStartPick contains an additional start activity <pick> in <onMessage>. The <correlation> of the other start activity has "join" as initiate attribute.

StartPickInOnAlarm contains an additional start activity <pick> in <onAlarm>. The <correlation> of the other start activity has "join" as initiate attribute.

StartPickInOnEvent contains an additional start activity <pick> in <onEvent>. The <correlation> of the other start activity has "join" as initiate attribute.

StartPickInRepeatUntil contains an additional start activity <pick> in <repeatUntil>.

StartPickInTerminationHandlers contains an additional start activity <pick> in <terminationHandler>.

StartPickInWhile contains an additional start activity <pick> in <while>.

StartPickParalellNonStartPick has the createInstance of the <pick> modified to no;

StartPickParalellNonStartReceive has the createInstance of the <receive> modified to no;

StartPickPreviousNonStartPick has the createInstance of the first <pick> modified to no;

StartPickPreviousNonStartReceive has the createInstance of the initial <receive> modified to no;

StartReceiveInCatch contains an additional start activity <receive> in <catch>.

StartReceiveInCatchAll contains an additional start activity `<receive>` in `<catchAll>`.

StartReceiveInElse contains an additional start activity `<receive>` in `<else>`.

StartReceiveInElseIf contains an additional start activity `<receive>` in `<elseIf>`.

StartReceiveInForEach contains an additional start activity `<receive>` in `<forEach>`.

StartReceiveInIf contains an additional start activity `<receive>` in `<if>`.

StartReceiveInNonStartPick contains an additional start activity `<receive>` in `<onMessage>`. The `<correlation>` of the other start activity has `"join"` as `initiate` attribute.

StartReceiveInOnAlarm contains an additional start activity `<receive>` in `<onAlarm>`. The `<correlation>` of the other start activity has `"join"` as `initiate` attribute.

StartReceiveInOnEvent contains an additional start activity `<receive>` in `<onEvent>`. The `<correlation>` of the other start activity has `"join"` as `initiate` attribute.

StartReceiveInRepeatUntil contains an additional start activity `<receive>` in `<repeatUntil>`.

StartReceiveInTerminationHandlers contains an additional start activity `<receive>` in `<terminationHandler>`.

StartReceiveInWhile contains an additional start activity `<receive>` in `<while>`.

StartReceiveParalellNonStartPick has the `createInstance` of the `<pick>` modified to `no`.

StartReceiveParalellNonStartReceive has the `createInstance` of the `<receive>` modified to `no`.

StartReceivePreviousNonStartPick has the `createInstance` of the `<pick>` modified to `no`.

StartReceivePreviousNonStartReceive has the `createInstance` of the initial `<receive>` modified to `no`.

FlowLateStartReceive has an `createInstance` attribute added to the fourth `<receive>` with value `yes`. The previous start activity and the new one get a `initiate` attribute of the `<correlation>` set to `join`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-FaultHandlers-CatchOrder	**StartPickInCatch**	-
Scope-FaultHandlers-CatchAll	**StartPickInCatchAll**	-
If-Else	**StartPickInElse**	-
If-ElseIf	**StartPickInElseIf**	-
ForEach	**StartPickInForEach**	-
If	**StartPickInIf**	-
Pick-Correlations-InitSync	**StartPickInNonStartPick**	-

Pick-OnAlarm-For	**StartPickInOnAlarm**	-
Scope-EventHandlers-InitSync	**StartPickInOnEvent**	-
RepeatUntil	**StartPickInRepeatUntil**	-
Scope-TerminationHandlers	**StartPickInTermination-Handlers**	-
While	**StartPickInWhile**	-
Flow-Two-Starting-OnMessage-Cor-Correlation	**StartPickParalellNonStart-Pick**	-
Flow-Starting-Receive-OnMessage-Correlation	**StartPickParalellNonStart-Receive**	-
Pick-Multiple-MessageExchanges	**StartPickPreviousNonStart-Pick**	-
Receive-Pick-FIFO-MessageExchanges	**StartPickPreviousNonStart-Receive**	-
Scope-FaultHandlers-CatchOrder	**StartReceiveInCatch**	-
Scope-FaultHandlers-CatchAll	**StartReceiveInCatchAll**	-
If-Else	**StartReceiveInElse**	-
If-ElseIf	**StartReceiveInElseIf**	-
ForEach	**StartReceiveInForEach**	-
If	**StartReceiveInIf**	-
Pick-Correlations-InitSync	**StartReceiveInNonStartPick**	-
Pick-OnAlarm-For	**StartReceiveInOnAlarm**	-
Scope-EventHandlers-Element-InitSync	**StartReceiveInOnEvent**	-
RepeatUntil	**StartReceiveInRepeatUntil**	-
Scope-TerminationHandlers	**StartReceiveInTermination-Handlers**	-
While	**StartReceiveInWhile**	-
Flow-Starting-Receive-OnMessage-Correlation	**StartReceiveParalellNon-StartPick**	-
Flow-Two-Starting-Receive-Cor-relation	**StartReceiveParalellNon-StartReceive**	-
Pick-Receive-FIFO-MessageExchanges	**StartReceivePreviousNon-StartPick**	-
ReceiveReply-FIFO-MessageExchanges	**StartReceivePreviousNon-StartReceive**	-
Flow-GraphExample	**FlowLateStartReceive**	-

Table 41: SA00056 test pairs

1.56 SA00057

"If a process has multiple start activities with correlation sets then all such activities MUST share at least one common correlationSet and all common correlationSets defined on all the activities MUST have the value of the initiate attribute be set to 'join'."[2, p. 200]

In a process with multiple start activities, starting `<pick>` or `<receive>` activities with at least one `<correlation>` can violate rule #57 that is modeled by [1, p. 33]. In the negation of the model we may receive an empty intersection of `<correlation>` or another `initiate="join"`, as every start activity requires at least one shared `<correlation>` with `initiate="join"`.

starting activity [`<pick>`, `<receive>`]
×
with [no `<correlation>`, `<correlation>` with `initiate="join"`]
×
additional starting activity [`<pick>`, `<receive>`]
×
with [no `<correlation>`, `<correlation>` with `initiate="join"` different `<correlationSet>`, `<correlation>` with `initiate="yes"` different `<correlationSet>`, `<correlation>` with `initiate="join"` same `<correlationSet>`, `<correlation>` with `initiate="yes"` same `<correlationSet>`]

Formalization 33: SA Rule #57

From the total 40 combinations, we have only eight valid ones for two start activities with either no `<correlation>` at all or both using the same `<correlationSet>` with `initiate="join"`. Thus, we have 32 test cases which violate this rule. But because some test cases can be seen as duplicated as the order of the activities is irrelevant, hence, we have twelve test cases only.

OnMessageCorrelationYesAndJoin has a modified `initiate` attribute in the `<correlation>` of the first `<onMessage>`.

OnMessageReceiveCorrelationYesAndJoin has a modified `initiate` attribute in the `<correlation>` of the `<onMessage>`.

ReceiveCorrelationYesAndJoin has a modified `initiate` attribute in the `<correlation>` of the first `<receive>`.

OnMessageSingleCorrelation has no `<correlation>` in the first `<onMessage>`.

OnMessageReceiveSingleCorrelation has no `<correlation>` in the `<onMessage>`.

ReceiveSingleCorrelation has no `<correlation>` in the first `<receive>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Flow-Two-Starting-On Message-Correlation	**OnMessageCorrelationYesAndJoin**	-

Flow-Starting-Receive-OnMessage-Correlation	**OnMessageReceiveCorrelationYesAndJoin**	-
Flow-Two-Starting-Receive-Correlation	**ReceiveCorrelationYesAndJoin**	-
Flow-Two-Starting-On Message-Correlation	**OnMessageSingleCorrelation**	-
Flow-Starting-Receive-OnMessage-Correlation	**OnMessageReceiveSingleCorrelation**	-
Flow-Two-Starting-Receive-Correlation	**ReceiveSingleCorrelation**	-

Table 42: SA00057 test pairs

1.57 SA00058

"In a <receive> or <reply> activity, the variable referenced by the variable attribute MUST be a messageType variable whose QName matches the QName of the input (for <receive>) or output (for <reply>) message type used in the operation, except as follows: if the WSDL operation uses a message containing exactly one part which itself is defined using an element, then a WS-BPEL variable of the same element type as used to define the part MAY be referenced by the variable attribute of the <receive> or <reply>activity."[2, p. 201]

Rule #58 resembles #48, but deals with `<receive>` and `<reply>`. Because they are modeled alike, see #48 for more information on the model of Kopp et al. [1].

[`<receive>`, `<reply>`]
×
[`<message>` with one `<part>`, `<message>` with two `<part>`s, `<message>` with no `<part>`s]
×
[`<variable>` with same `@element`, `<variable>` with different `@element`, `<variable>` with type `@type`, `<variable>` with same `@messageType`]

Formalization 34: SA Rule #58

If we have one `<part>` in a `<message>`, it is valid to use the `@element` instead of `@messageType` for the variable definition. Using the `@messageType` is always valid. A different `@messageType` check is already part of rule #10. Thus, we get 16 tests as a result of the permutation.

ReceiveDeviantMessageType has a modified `messageType` attribute in the request `<variable>`.

ReceiveDeviantType has the attribute wrong `element` instead of `messageType` in the request `<variable>`. In the `<from>` no `part` attribute is used.

ReceiveReplyDeviantTypes has the attribute wrong `element` instead of `messageType` in the request and response `<variable>`. In the `<to>` and `<from>` no `part` attribute is used.

ReceiveTwoPartsPartType imports TestInterface-ReceiveTwoPartsPartType.wsdl that contains an operation with a request message containing two `<part>` elements. This operation is called instead of `startProcessSync` and the request `<variable>` has the wrong attribute `element` instead of `messageType`. In the `<from>` no `part` attribute is used.

ReplyDeviantMessageType has a modified `messageType` attribute in the response `<variable>`.

ReplyDeviantType has the attribute wrong `element` instead of `messageType` in the response `<variable>`. In the `<to>` no `part` attribute is used.

ReplyTwoPartsPartType imports TestInterface-ReplyTwoPartsPartType.wsdl that contains an operation with a response message containing two `<part>` elements. This operation is called instead of `startProcessSync` and the response `<variable>` has the attribute `element` instead of `messageType`. In the `<to>` no `part` wrong attribute is used.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
ReceiveReply	**ReceiveDeviantMessageType**	-
ReceiveReply	**ReceiveDeviantType**	-
ReceiveReply	**ReceiveReplyDeviantTypes**	-
ReceiveReply	**ReceiveTwoPartsPartType**	1
ReceiveReply	**ReplyDeviantMessageType**	-
ReceiveReply	**ReplyDeviantType**	-
ReceiveReply	**ReplyTwoPartsPartType**	1

Table 43: SA00058 test pairs

1.58 SA00059

”For <reply>, if <toPart> elements are used on a <reply> activity then the variable attribute MUST NOT be used on the same activity.”[2, p. 201]

The test of rule #59 has a `@variable` for `<reply>` and a `<toPart>`. According to [1, p. 29] rule #59 is similar to #51, hence, only a single test case is required.

Reply-WithToPartElementAndVariableAttribute contains an additional `<variable>` for the `<reply>` that has the attribute `variable`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
ReceiveReply-ToParts	**Reply-WithToPartElementAndVariable-Attribute**	-

Table 44: SA00059 test pairs

1.59 SA00060

"The explicit use of messageExchange is needed only where the execution can result in multiple IMA-<reply> pairs (e.g. <receive>-<reply> pair) on the same partnerLink and operation being executed simultaneously. In these cases, the process definition MUST explicitly mark the pairing-up relationship."[2, p. 201]

The rule #60 model by [1, p. 31] constraints the possible `@messageExchange` equalities. The negation indicates a test set. However, this would require a change of a single attributes which

start1Mex = start2Mex, start1Mex != stop1Mex and start1Mex != stop2Mex; start1Mex != stop1Mex and start2Mex != stop1Mex; start2Mex != stop2Mex and start1Mex != stop2Mex; start2Mex != stop2Mex and start2Mex != stop1Mex; startMex1 = /; stopMex1 = /; startMex2 = /; stopMex2 = /;

Formalization 35: The negation of SA Rule #60 from Kopp et al. which is not used

results in a violation of rule #61. To isolate the tests we have to use proper processes and remove one of the `<messageExchange>` and corresponding attributes in the IMA and the `<reply>`.

first IMA [`<receive>`, `<pick>`]
×
second IMA [`<receive>`, `<pick>`, `<onEvent>`]
×
marked message Exchange [none, both, first IMA, second IMA]

Formalization 36: SA Rule #60

Therefore, we have 18 tests in total.

OnEventFiloFirstMexMissing has no `messageExchange` attribute neither in the `<receive>` nor in the second `<reply>`.

OnEventFiloSecondMexMissing has no `messageExchange` attribute neither in the `<onEvent>` nor in the first `<reply>`.

OnMessageFifoFirstMexMissing has no `messageExchange` attribute neither in the first `<onMessage>` nor in the first `<reply>`.

OnMessageFifoSecondMexMissing has no `messageExchange` attribute neither in the second `<onMessage>` nor in the second `<reply>`.

OnMessageFiloFirstMexMissing has no `messageExchange` attribute neither in the first `<onMessage>` nor in the second `<reply>`.

OnMessageFiloSecondMexMissing has no `messageExchange` attribute neither in the second `<onMessage>` nor in the first `<reply>`.

ReceiveFifoFirstMexMissing has no `messageExchange` attribute neither in the first `<receive>` nor in the first `<reply>`.

ReceiveFifoSecondMexMissing has no `messageExchange` attribute neither in the second `<receive>` nor in the second `<reply>`.

ReceiveFiloFirstMexMissing has no `messageExchange` attribute neither in the first `<receive>` nor in the second `<reply>`.

ReceiveFiloSecondMexMissing has no `messageExchange` attribute neither in the second `<receive>` nor in the fist `<reply>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-FILO-MessageExchanges	**OnEventFiloFirstMexMissing**	-
Scope-FILO-MessageExchanges	**OnEventFiloSecondMexMissing**	-
Pick-FIFO-MessageExchanges	**OnMessageFifoFirstMexMissing**	-
Pick-FIFO-MessageExchanges	**OnMessageFifoSecondMexMissing**	-
Pick-FILO-MessageExchanges	**OnMessageFiloFirstMexMissing**	-
Pick-FILO-MessageExchanges	**OnMessageFiloSecondMexMissing**	-
ReceiveReply-FIFO-Message-Exchanges	**ReceiveFifoFirstMexMissing**	-
ReceiveReply-FIFO-Message-Exchanges	**ReceiveFifoSecondMexMissing**	-
ReceiveReply-FILO-Message-Exchanges	**ReceiveFiloFirstMexMissing**	-
ReceiveReply-FILO-Message-Exchanges	**ReceiveFiloSecondMexMissing**	-

Table 45: SA00060 test pairs

1.60 SA00061

”The name used in the optional messageExchange attribute MUST resolve to a messageExchange declared in a scope (where the process is considered the root scope) which encloses the <reply> activity and its corresponding IMA.”[2, p. 201]

For all rule #61 tests the message exchange definition is not in a valid location. The rule model by [1, p. 30] is not precise as they only check the uniqueness of the `@name`s of `<messageExchange>`s, but the negation of the model indicates the tests.

IMA is [`<receive>`, `<pick>`, `<onEvent>`]
×
location of `<messageExchange>` [no, `<process>`]
×
set `@messageExchange` for [IMA, `<reply>`, both, none]

Formalization 37: SA Rule #61

When no `@messageExchange` is set, every case is valid. The same holds when the `<messageExchange>` is located in the `<process>` and both message activities have set `@messageExchange`. Hence, we require 15 test cases.

IMA is [`<receive>`, `<pick>`, `<onEvent>`]
×
set `@messageExchange` for [IMA, `<reply>`, both, none]
×
is `<messageExchange>` reachable for [none, IMA, `<reply>`, both]

Formalization 38: SA Rule #61 for `<scope>`

When no `@messageExchange` is set, every case is valid. The same holds when both message activities have set `@messageExchange` and both can reach to `<messageExchange>`. Hence, we require 33 test cases.

OnEventMessageExchangeNotInProcess contains no `messageExchange` attribute in the `<onEvent>`.

OnEventMessageExchangeNotInScope contains no `messageExchange` attribute in the `<onEvent>`.

OnEventNoMessageExchangeInProcess has no `<messageExchanges>` element nor its child.

OnEventNoMessageExchangeInScope has a modified `name` attribute in the `<messageExchange>`.

OnEventReplyMessageExchangeNotInProcess has no `messageExchange` attribute in the last `<reply>`.

OnEventReplyMessageExchangeNotInScope has no `messageExchange` attribute in the last `<reply>`.

OnMessageMessageExchangeNotInProcess contains no `messageExchange` attribute in the `<onMessage>`.

OnMessageMessageExchangeNotInScope contains no `messageExchange` attribute in the `<onMessage>`.

OnMessageMessageExchangeOutOfScope has the `<reply>` moved outside of the `<scope>`.

OnMessageNoMessageExchangeInProcess has a modified `name` attribute in the `<messageExchange>`.

OnMessageNoMessageExchangeInScope has a modified `name` attribute in the `<messageExchange>`.

OnMessageReplyMessageExchangeNotInProcess has no `messageExchange` attribute in the `<reply>`.

OnMessageReplyMessageExchangeNotInScope has no `messageExchange` attribute in the `<reply>`.

ReceiveMessageExchangeNotInProcess contains no `messageExchange` attribute in the `<receive>`.

ReceiveMessageExchangeNotInScope contains no `messageExchange` attribute in the `<receive>`.

ReceiveMessageExchangeOutOfScope has a `<sequence>` enclosing all other activities and the `<reply>` is moved outside of the `<scope>`.

ReceiveNoMessageExchangeInProcess has a modified `name` attribute in the `<messageExchange>`.

ReceiveNoMessageExchangeInScope has a modified `name` attribute in the `<messageExchange>`.

ReceiveReplyMessageExchangeNotInProcess has no `messageExchange` attribute in the last `<reply>`.

ReceiveReplyMessageExchangeNotInScope has no `messageExchange` attribute in the last `<reply>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-EventHandlers-MessageExchange-InitSync	**OnEventMessageExchangeNotInProcess**	-
Scope-EventHandlers-ScopeMessageExchange-InitSync	**OnEventMessageExchangeNotInScope**	-
Scope-EventHandlers-MessageExchange-InitSync	**OnEventNoMessageExchangeInProcess**	-
Scope-EventHandlers-ScopeMessageExchange-InitSync	**OnEventNoMessageExchangeInScope**	-
Scope-EventHandlers-MessageExchange-InitSync	**OnEventReplyMessageExchangeNotInProcess**	-
Scope-EventHandlers-ScopeMessageExchange-InitSync	**OnEventReplyMessageExchangeNotInScope**	-
Pick-MessageExchange	**OnMessageMessageExchangeNotInProcess**	-
Pick-MessageExchange-Scope	**OnMessageMessageExchangeNotInScope**	-
Pick-MessageExchange-Scope	**OnMessageMessageExchangeOutOfScope**	-
Pick-MessageExchange	**OnMessageNoMessageExchangeInProcess**	-
Pick-MessageExchange-Scope	**OnMessageNoMessageExchangeInScope**	-
Pick-MessageExchange	**OnMessageReplyMessageExchangeNotInProcess**	-
Pick-MessageExchange-Scope	**OnMessageReplyMessageExchangeNotInScope**	-

ReceiveReply-MessageExchange	**ReceiveMessageExchangeNotInProcess**	-
Scope-MessageExchange	**ReceiveMessageExchangeNotInScope**	-
Scope-MessageExchange	**ReceiveMessageExchangeOutOfScope**	-
ReceiveReply-MessageExchange	**ReceiveNoMessageExchangeInProcess**	-
Scope-MessageExchange	**ReceiveNoMessageExchangeInScope**	-
ReceiveReply-MessageExchange	**ReceiveReplyMessageExchangeNotIn-Process**	-
Scope-MessageExchange	**ReceiveReplyMessageExchangeNotIn-Scope**	-

Table 46: SA00061 test pairs

1.61 SA00062

"If <pick> has a createInstance attribute with a value of yes, the events in the <pick> MUST all be <onMessage> events."[2, p. 201]

To violate rule #62 a sole test is sufficient that has an `<onAlarm>` in a starting `<pick>`. This constraint of #62 is also modeled by [1, p. 46].

Pick-CreateInstanceWithOnAlarm contains an additional `<onAlarm>` in the `<pick>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Pick-CreatInstance	**Pick-CreateInstanceWithOnAlarm**	-

Table 47: SA00062 test pairs

1.62 SA00063

"The semantics of the <onMessage> event are identical to a <receive> activity regarding the optional nature of the variable attribute or <fromPart> elements, if <fromPart>elements on an activity then the variable attribute MUST NOT be used on the same activity (see SA00055)."[2, p. 201]

A process containing an `<onMessage>` with a `@variable` and `<fromParts>` violates rule #63. This is similar to #55 which is defined accordingly by [1, p. 29].

OnMessage-With-FromPartAndAttributeVariable has an additional `<variable>` that is used in `<onMessage>` in the `variable` attribute.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Pick-CreateInstance-FromParts	**OnMessage-With-FromPartAnd-AttributeVariable**	-

Table 48: SA00063 test pairs

1.63 SA00064

"For <flow>, a declared link's name MUST be unique among all <link> names defined within the same immediately enclosing <flow>."[2, p. 201]

Rule #64 is violated with a doubled `<link> @name` in the same `<flow>`. [1, p. 47] model this rule that it can be negated to identify the test, but the definition contains a typo in second declareLink: the l has to be a m.

LinkNameDuplicate contains a `<link>` duplicate.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Flow-Links	**LinkNameDuplicate**	-

Table 49: SA00064 test pairs

1.64 SA00065

"The value of the linkName attribute of <source> or <target> MUST be the name of a <link> declared in an enclosing <flow> activity."[2, p. 201]

Negating the rule #65 model of [1, p. 50] we can identify two tests: a `<source>` that is not in the same `<flow>` as the `<link>`, and a `<target>` also not within the `<flow>`. In addition, two tests can be derived for the case if the `<link>` element is missing.

[`<source>`, `<target>`]
×
[`<link>` undefined, `<link>` out of `<flow>`, `<link>` in `<flow>`]

Formalization 39: SA Rule #65

A process is valid if the `<link>` is in the `<flow>`.

NoLink contains no `<link>` definition.

SourceLinkIsMissing has a modified `linkName` in the `<source>`.

SourceLinkOutOfFlow contains a second `<flow>` with a `<link>` definition. This `<link>` is used in the previous `<flow>` in the `<source>`, but the `<target>` and the original `<link>` definition are deleted.

TargetLinkIsMissing has a modified `linkName` in the `<target>`.

TargetLinkOutOfFlow contains a second `<flow>` with a `<link>` definition. This `<link>` is used in the previous `<flow>` in the `<target>`, but the `<source>` and the original `<link>` definition are deleted.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Flow-Links	**NoLink**	-
Flow-Links	**SourceLinkIsMissing**	-
Flow-Links	**SourceLinkOutOfFlow**	-
Flow-Links	**TargetLinkIsMissing**	-
Flow-Links	**TargetLinkOutOfFlow**	-

Table 50: SA00065 test pairs

1.65 SA00066

”Every link declared within a <flow> activity MUST have exactly one activity within the <flow> as its source and exactly one activity within the <flow> as its target.”[2, p. 201]

If there is not exactly one `<source>` with a single `<target>` for a `<link>`, rule #66 is violated. In the rule model by [1, p. 48], the constraint of the `<link>` tuples is given by the definition af a `<link>` relation.

any activity in `<flow>` [`<source>`, `<target>`]
×
any other activity in `<flow>` [`<source>`, none]
×
yet another activity in `<flow>` [`<target>`, none]

Formalization 40: SA Rule #66

It is valid to have a single activity as a `<source>` with another single element as a `<target>`. The remaining six combinations are invalid, resulting in six test cases.

LinkNoSource has no `<source>` element.

LinkNoTarget has no `<target>` element.

LinkTwoSources has deleted the second `<link>` and the corresponting `<target>` is also removed. The associated `<source>` has a modified `linkName` attribute. A `<joinCondition>` that contained the `<link>` is cleared of it.

LinkTwoTargets has deleted the last `<link>` and the corresponting `<source>` is also removed. The associated `<target>` has a modified `linkName` attribute.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Flow-Links	**LinkNoSource**	-
Flow-Links	**LinkNoTarget**	-
Flow-GraphExample	**LinkTwoSources**	-
Flow-GraphExample	**LinkTwoTargets**	-

Table 51: SA00066 test pairs

1.66 SA00067

"Two different links MUST NOT share the same source and target activities; that is, at most one link may be used to connect two activities."[2, p. 202]

A single test is required for rule #67. The negation of the rule model by [1, p. 48] points to have two `<link>` elements connecting the same activities.

DoubleLink contains an additional `<link>` that is used in a additional `<target>` and a additional `<source>` both located in the existent containers.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Flow-Links	**DoubleLink**	-

Table 52: SA00067 test pairs

1.67 SA00068

"An activity MAY declare itself to be the source of one or more links by including one or more <source> elements. Each <source> element MUST use a distinct link name."[2, p. 202]

Violating rule #68 can be tested by a `@linkName` duplicate in the `<source>` elements of an activity. [1, p. 48] provide a rule model that shows the test if it is negated.

LinkSourceDuplicate contains an additional `<source>` with the same `linkName` attribute as the previous `<source>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Flow-Links	**LinkSourceDuplicate**	-

Table 53: SA00068 test pairs

1.68 SA00069

"An activity MAY declare itself to be the target of one or more links by including one or more <target> elements. Each <target> element associated with a given activity MUST use a link name distinct from all other <target> elements at that activity."[2, p. 202]

Similar to rule #68, rule #69 is violated by a `@linkName` duplicate in the `<target>` elements of an activity, and the model (by [1, p. 48]) negation provide a rule model that shows the test.

LinkTargetDuplicate contains an additional `<target>` with the same `linkName` attribute as the previous `<target>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Flow-Links	**LinkTargetDuplicate**	-

Table 54: SA00069 test pairs

1.69 SA00070

"A link MUST NOT cross the boundary of a repeatable construct or the <compensationHandler> element. This means, a link used within a repeatable construct (<while>, <repeatUntil>, <forEach>, <eventHandlers>) or a <compensationHandler> MUST be declared in a <flow>that is itself nested inside the repeatable construct or <compensationHandler>."[2,

p. 202]

Rule #70 can be violated by crossing a) `<compensationHandler>`, b) repeatable constructs (`<while>`, `<repeatUntil>`, `<forEach>`) and c) `<eventHandlers>`. The rule model for (a) by [1, p. 55] can be negated to receive tests, a `<target>`/ `<source>` out of the handler. For (b), the rule model by [1, p. 51] can be negated as well and the tests are the same for each repeatable construct. There is no model for (c) and the `<link>` could be also outside of the construct.

[`<compensationHandler>`, `<while>`, `<repeatUntil>`, `<forEach>`, `<eventHandlers>`]
×
[`<source>` outside boundary, `<target>` outside boundary, `<link>` outside boundary, all elements in the boundary]

Formalization 41: SA Rule #70

If all elements are in the boundary, the process is valid. Thus, we have 15 tests in total.

LinkOutOfCompensationHandler contains an additional `<flow>` enclosing the `<scope>` containing the `<comensationHandler>` directly and the `<link>` from the internal `<flow>` is now defined outside of the `<scope>`.

LinkOutOfEventHandlers contains an additional `<flow>` enclosing the `<scope>` containing the `<onEvent>` directly and a `<link>` is defined in this `<flow>`. The internal `<sequence>` is replaced by a `<flow>`. Internal activities `<reply>` (target) and `<assign>` (source) are linked with the `<link>`.

LinkOutOfForEach contains an additional `<flow>` enclosing the `<forEach>` and the `<link>` from the internal `<flow>` is now defined outside of the `<forEach>`.

LinkOutOfRepeatUntil contains an additional `<flow>` enclosing the `<repeatUntil>` and the `<link>` from the internal `<flow>` is now defined outside of the `<repeatUntil>`.

LinkOutOfWhile contains an additional `<flow>` enclosing the `<while>` and the `<link>` from the internal `<flow>` is now defined outside of the `<while>`.

SourceOutOfCompensationHandler contains an additional `<flow>` enclosing the `<scope>` containing the `<comensationHandler>` directly and the `<link>` from the internal `<flow>` is now defined outside of the `<scope>`. The internal `<assign>` of the `<comensationHandler>` is relocated outside of the `<scope>`.

SourceOutOfEventHandlers contains an additional `<flow>` enclosing the `<scope>` containing the `<onEvent>` directly and a `<link>` is defined in this `<flow>`. The internal `<sequence>` is replaced by a `<flow>`. Internal activities `<reply>` (target) and `<assign>` (source) are linked with the `<link>`. The internal `<assign>` of the `<onEvent>` is relocated outside of the `<scope>`.

SourceOutOfForEach contains an additional `<flow>` enclosing the `<forEach>` and the `<link>` from the internal `<flow>` is now defined outside of the `<forEach>`. The second assignment is relocated outside of the `<forEach>`.

SourceOutOfRepeatUntil contains an additional `<flow>` enclosing the `<repeatUntil>` and the `<link>` from the internal `<flow>` is now defined outside of the `<repeatUntil>`. The second assignment is relocated outside of the `<repeatUntil>`.

SourceOutOfWhile contains an additional `<flow>` enclosing the `<while>` and the `<link>` from the internal `<flow>` is now defined outside of the `<while>`. The second assignment is relocated outside of the `<while>`.

TargetOutOfCompensationHandler contains an additional `<flow>` enclosing the `<scope>` containing the `<comensationHandler>` directly and the `<link>` from the internal `<flow>` is now defined outside of the `<scope>`. The internal `<reply>` of the `<comensationHandler>` is relocated outside of the `<scope>`.

TargetOutOfEventHandlers contains an additional `<flow>` enclosing the `<scope>` containing the `<onEvent>` directly and a `<link>` is defined in this `<flow>`. The internal `<sequence>` is replaced by a `<flow>`. Internal activities `<reply>` (target) and `<assign>` (source) are linked with the `<link>`. The internal `<reply>` of the `<onEvent>` is relocated outside of the `<scope>`.

TargetOutOfForEach contains an additional `<flow>` enclosing the `<forEach>` and the `<link>` from the internal `<flow>` is now defined outside of the `<forEach>`. The first assignment is relocated outside of the `<forEach>`.

TargetOutOfRepeatUntil contains an additional `<flow>` enclosing the `<repeatUntil>` and the `<link>` from the internal `<flow>`. The first assignment is relocated outside of the `<repeatUntil>`.

TargetOutOfWhile contains an additional `<flow>` enclosing the `<while>` and the `<link>` from the internal `<flow>` is now defined outside of the `<while>`. The first assignment is relocated outside of the `<while>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-Compensate-Flow	**LinkOutOfCompensationHandler**	-
Scope-EventHandlers-InitSync	**LinkOutOfEventHandlers**	-
ForEach-Flow	**LinkOutOfForEach**	-
RepeatUntil-Flow	**LinkOutOfRepeatUntil**	-
While-Flow	**LinkOutOfWhile**	-
Scope-Compensate-Flow	**SourceOutOfCompensationHandler**	-
Scope-EventHandlers-InitSync	**SourceOutOfEventHandlers**	-
ForEach-Flow	**SourceOutOfForEach**	-
RepeatUntil-Flow	**SourceOutOfRepeatUntil**	-
While-Flow	**SourceOutOfWhile**	-
Scope-Compensate-Flow	**TargetOutOfCompensationHandler**	-
Scope-EventHandlers-InitSync	**TargetOutOfEventHandlers**	-
ForEach-Flow	**TargetOutOfForEach**	-
RepeatUntil-Flow	**TargetOutOfRepeatUntil**	-

While-Flow	**TargetOutOfWhile**	-

Table 55: SA00070 test pairs

1.70 SA00071

"A link that crosses a <catch>, <catchAll> or <terminationHandler> element boundary MUST be outbound only, that is, it MUST have its source activity within the <faultHandlers> or <terminationHandler>, and its target activity outside of the scope associated with the handler."[2, p. 202]

Rule #71 resembles #70, but is more lax. For 1) `<catch>`, 2) `<catchAll>`, and 3) `<terminationHandler>`, incoming links violate the rule, resulting in three test cases. The rule model by [1, p. 55] can be negated to show these tests, yet, there is a typo a' has to be b.

CatchAllIncommingLink has switched `<target>` and `<source>`.

CatchIncommingLink has switched `<target>` and `<source>`.

TerminationHandlerIncommingLink has switched `<target>` and `<source>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-FaultHandlers-OutboundLink-CatchAll	**CatchAllIncomming-Link**	-
Scope-FaultHandlers-OutboundLink	**CatchIncommingLink**	-
Scope-TerminationHandlers-Outbound-Link	**TerminationHandler-IncommingLink**	-

Table 56: SA00071 test pairs

1.71 SA00072

"A <link> declared in a <flow> MUST NOT create a control cycle, that is, the source activity must not have the target activity as a logically preceding activity."[2, p. 202]

To develop the tests for rule #72 we negate the rule model by [1, p. 50]. In the set of clan defined in [1, p. 50], we find the action itself and all descendants. Thus, a self linked activity and two cross linked activities are sufficient to test rule #72, totaling in two test cases.

FlowCyclic contains an additional `<link>` that has the corresponding `<target>` within the activity that contained the existing `<source>` and the new `<source>` within the activity that contained the existing `<target>`.

FlowSelfLinked contains the `<target>` in the same element that has the `<source>` as a child.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Flow-Links	**FlowCyclic**	-
Flow-Links	**FlowSelfLinked**	-

Table 57: SA00072 test pairs

1.72 SA00073

"The expression for a join condition MUST be constructed using only Boolean operators and the activity's incoming links' status values."[2, p. 202]

The rule deals with the parsing of XPath expressions and is postponed to future work.

1.73 SA00074

"The expressions in <startCounterValue> and <finalCounterValue> MUST return a TII (meaning they contain at least one character) that can be validated as a xsd:unsignedInt. Static analysis MAY be used to detect this erroneous situation at design time when possible (for example, when the expression is a constant)."[2, p. 202]

Because the rule deals with general expression parsing, it is postponed to future work.

1.74 SA00075

"For the <forEach> activity, <branches> is an integer value expression. Static analysis MAY be used to detect if the integer value is larger than the number of directly enclosed activities

of <forEach> at design time when possible (for example, when the branches expression is a constant)."[2, p. 202]

Because the rule deals with general expression parsing, it is postponed to future work.

1.75 SA00076

”For <forEach> the enclosed scope MUST NOT declare a variable with the same name as specified in the counterName attribute of <forEach>.”[2, p. 203]

The rule #76 model by [1, p. 52] can be negated to show the single test for the rule which is a process with `<variable>` defined with the `@name` of the `<forEach>` `@counterName`.

ForEach-DuplicateCounterVariable contains a `<variable>` definition in the `<scope>` of the `<forEach>` with a `counterName` attribute that is equal to the `name` of the `<variable>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
ForEach	**ForEach-DuplicateCounterVariable**	-

Table 58: SA00076 test pairs

1.76 SA00077

”The value of the target attribute on a <compensateScope> activity MUST refer to the name of an immediately enclosed scope of the scope containing the FCT-handler with the <compensateScope> activity. This includes immediately enclosed scopes of an event handler (<onEvent> or <onAlarm>) associated with the same scope.”[2, p. 203]

[1, p. 59] model rule #77 and one test type can be directly derived. If the `<compensateScope>` targets a `<scope>` or `<invoke>` outside its own enclosing `<scope>` than rule #77 is violated. Other violations target `<scope>` elements inside other constructs. Therefore, they are not direct children of the enclosing `<scope>`.

targets [`<invoke>`, `<scope>`]
×
location of `<compensateScope>` [nested in `<catch>`, nested in `<catchAll>`, nested in `<compensationHandler>`, nested in `<terminationHandler>`,]
×
location of target [`<if>`, `<else>`, `<elseIf>`, `<flow>`, `<onAlarm>`, `<onEvent>`, `<onMessage>`, `<repeatUntil>`, `<sequence>`, `<while>`, `<catch>`, `<catchAll>`, `<compensationHandler>`, `<terminationHandler>`,
`<scope>` in enclosing `<scope>`, sibling of enclosing `<scope>`, enclosing `<scope>`]

Formalization 42: SA Rule #77

If the target is within the enclosing `<scope>`, the process is valid. Thus, we have 128 ($2 * 4 * 16$) tests for rule #77.

CompensateTargetInvokeNestedInCatch contains an additional `<catch>` that encloses

the <invoke>.

CompensateTargetInvokeNestedInCatchAll contains an additional <catch> that encloses the <compensationScope>. The <invoke> is relocated in the <catchAll>.

CompensateTargetInvokeNestedInCompensationHandler contains an additional <invoke> with a <compensationHandler> inside the existing <compensationHandler>.

CompensateTargetInvokeNestedInScope contains an additional <scope> enclosing the <invoke>.

CompensateTargetInvokeNestedInTerminationHandler contains a additional <scope> containing all other activities and the <faultHandlers>. A <terminationHandler> is also located in this <scope> and the <invoke> is relocated inside of it.

CompensateTargetInvokeOutOfScope contains an additional <sequence> that encloses all other activities. The first child is a <scope> that contains the <faultHandlers> and all activities to the <throw>, except the <invoke>. The <invoke> is located in a new <scope>.

CompensateTargetScopeNestedInCatch contains an additional <scope> with a <compensationHandler> inside a <catch>. This <scope> is the target of the <compensateScope>.

CompensateTargetScopeNestedInCatchAll contains an additional <scope> with a <compensationHan
inside the <catchAll>. This <scope> is the target of the <compensateScope>. The <compensateScope> is relocated in a new <catch>

CompensateTargetScopeNestedInCompensationHandler contains an additional <scope> with a <compensationHandler> inside the existing <compensationHandler>. This <scope> is the target of the <compensateScope>.

CompensateTargetScopeNestedInScope contains an additional <scope> that encloses the existing <scope>.

CompensateTargetScopeNestedInTerminationHandler contains a scope enclosing all other activities and the <faultHandlers>. An additional <terminationHandler> contain a <scope> with a <compensationHandler>. This <scope> is the target of the <compensateScope>.

CompensateTargetScopeOutOfScope has the <faultHandlers> inside the scope. The target of the <compensateScope> is a additional <scope> with a <compensationHandler>. The new <scope> is a peer scope of the existing scope.

NoCompensateTarget has a modified target in the <compensateScope>.

betsy Test Case Origin	Derived Test Case	Modified WSDLs

Invoke-CompensateScope-CompensationHandler	**CompensateTargetInvokeNestedInCatch**	-
Invoke-CompensateScope-CompensationHandler	**CompensateTargetInvokeNestedInCatchAll**	-
Invoke-CompensateScope-CompensationHandler	**CompensateTargetInvokeNestedInCompensationHandler**	-
Invoke-CompensateScope-CompensationHandler	**CompensateTargetInvokeNestedInScope**	-
Invoke-CompensateScope-CompensationHandler	**CompensateTargetInvokeNestedInTerminationHandler**	-
Invoke-CompensateScope-CompensationHandler	**CompensateTargetInvokeOutOfScope**	-
Scope-CompensateScope	**CompensateTargetScopeNestedInCatch**	-
Scope-CompensateScope	**CompensateTargetScopeNestedInCatchAll**	-
Scope-CompensateScope	**CompensateTargetScopeNestedInCompensationHandler**	-
Scope-CompensateScope	**CompensateTargetScopeNestedInScope**	-
Scope-CompensateScope	**CompensateTargetScopeNestedInTerminationHandler**	-
Scope-CompensateScope	**CompensateTargetScopeOutOfScope**	-
Scope-CompensateScope	**NoCompensateTarget**	-

Table 59: SA00077 test pairs

1.77 SA00078

"The target attribute of a <compensateScope> activity MUST refer to a scope or an invoke activity with a fault handler or compensation handler."[2, p. 203]

The rule #78 model by [1, p. 58] does not model the rule. It solely checks unique name of the children, but the rule #78 requires `<faultHandlers>` or `<compensationHandler>` in a target `<scope>` that is referenced by a `<compensateScope>`.

targets [`<invoke>`, `<scope>`]
×
elements of targets [none, `<compensationHandler>`, `<faultHandler>`, both]

Formalization 43: SA Rule #78

Thus, we require two tests, one for `<invoke>` and one for `<scope>` with no handler at all.

InvokeMissingFCHandler has no <compensationHandler>.

ScopeMissingFCHandler has no <compensationHandler>.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Invoke-CompensateScope-CompensationHandler	**InvokeMissingFCHandler**	- -
Scope-CompensateScope	**ScopeMissingFCHandler**	-

Table 60: SA00078 test pairs

1.78 SA00079

"The root scope inside a FCT-handler MUST not have a compensation handler."[2, p. 203]

To violate rule #79, the `<scope>` inside a 1) `<catch>`, 2) `<catchAll>`, 3) `<compensationHandler>`, or 4) `<terminationHandler>` has to carry a `<compensationHandler>`. The rule model by [1, p. 60] can be negated to show the structure of those tests, but it has a typo: the last } must be a |. In total, there are four test cases.

CompensationHandlerInCatchAllRootScope contains an additional `<scope>` with a `<compensationH` in the `<catchAll>`.

CompensationHandlerInCatchRootScope contains an additional `<scope>` with a `<compensationHand` in the first `<catch>`.

HandlersCompensationInCompensationRootScope contains an additional `<scope>` with a `<compensationHandler>` in the `<compensationHandler>`.

HandlersCompensationInTerminationRootScope contains an additional `<scope>` with a `<compensationHandler>` in the `<teminationHandler>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-FaultHandlers-CatchAll	**CompensationHandlerInCatchAllRoot-Scope**	-
Scope-FaultHandlers-CatchOrder	**CompensationHandlerInCatchRootScope**	-
Scope-Compensate	**HandlersCompensationInCompensation-RootScope**	-
Scope-Termination-Handlers	**HandlersCompensationInTermination-RootScope**	-

Table 61: SA00079 test pairs

1.79 SA00080

"There MUST be at least one <catch> or <catchAll> element within a <faultHandlers> element."[2, p. 203]

[1, p. 20] mention rule #80 but do not provide a model. The test is a `<process>` or `<scope>` with an empty `<faultHandlers>`, requiring two test cases in total.

EmptyFaultHandlersInProcess contains no `<scope>` element and no `<catch>`.

EmptyFaultHandlersInScope contains no and no `<catch>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-FaultHandlers	**EmptyFaultHandlersInProcess**	-
Scope-FaultHandlers	**EmptyFaultHandlersInScope**	-

Table 62: SA00080 test pairs

1.80 SA00081

"For the <catch> construct; to have a defined type associated with the fault variable, the faultVariable attribute MUST only be used if either the faultMessageType or faultElement attributes, but not both, accompany it. The faultMessageType and faultElement attributes MUST NOT be used unless accompanied by faultVariable attribute."[2, p. 203]

Rule #81 tests have a combination out of three attributes. The negation of the rule model by [1, p. 58] also shows the tests.

[`@faultVariable`, none]
×
[`@faultMessageType`, none]
×
[`@faultElement`, none]

Formalization 44: SA Rule #81

Three combinations are valid, namely, no attribute selected, and `@faultVariable` with either `@faultMessageType` or `@faultElement`, resulting in five test cases in total.

CatchElement contains additional `variable` attribute in the `<catch>`.

CatchType contains no `variable` attribute in the `<catch>`.

CatchTypeElement contains additional `faultMessageType` attribute in the `<catch>`.

CatchVariable contains no `faultMessageType` attribute in the `<catch>`.

CatchVariableTypeElement contains additional `faultMessageType` attribute in the `<catch>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-FaultHandlers-FaultElement	**CatchElement**	-
Scope-FaultHandlers-CatchOrder	**CatchType**	-
Scope-FaultHandlers-FaultElement	**CatchTypeElement**	-
Scope-FaultHandlers-CatchOrder	**CatchVariable**	-
Scope-FaultHandlers-FaultElement	**CatchVariableTypeElement**	-

Table 63: SA00081 test pairs

1.81 SA00082

"The peer-scope dependency relation MUST NOT include cycles. In other words, WS-BPEL forbids a process in which there are peer scopes S1 and S2 such that S1 has a peer-scope dependency on S2 and S2 has a peer-scope dependency on S1."[2, p. 203]

The negation of the rule #82 model by [1, p. 51] models the single test. Two `<scope>` elements with activities "linking" them in a cycle.

CyclicLinkedPeerScopes contains two additional `<scope>` elements containing each a `<sequence>` that devide the `<flow>` into three parts. The third receive is outside of the two `<scope>` elements. The activities above are contained in the first `<scope>` and additional the last receive is placed here. The other activities are in the second `<scope>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Flow-GraphExample	**CyclicLinkedPeerScopes**	-

Table 64: SA00082 test pairs

1.82 SA00083

"An event handler MUST contain at least one <onEvent> or <onAlarm> element."[2, p. 203]

The two test cases for rule #83 consist of either a `<process>` or `<scope>` with an empty `<eventHandlers>`. The rule #83 is not modeled by [1, p. 56] as the model does not contain the wrapper element.

EmptyEventHandlersInProcess contains an additional empty `<eventHandlers>`.

EmptyEventHandlersInScope contains an additional `<scope>` with a empty `<eventHandlers>` and an `<empty>` element.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
ReceiveReply	**EmptyEventHandlersInProcess**	-
ReceiveReply	**EmptyEventHandlersInScope**	-

Table 65: SA00083 test pairs

1.83 SA00084

"The partnerLink reference of <onEvent> MUST resolve to a partner link declared in the process in the following order: the associated scope first and then the ancestor scopes."[2, p. 203]

This rule is explicitly excluded from the model of [1, p. 65] because it is about runtime behavior. The only way to test such rules during static analysis is violating other rules, in particular, by defining a wrong `<partnerLink>` inside with the same `@name` as the correct `<partnerLink>` outside the `<onEvent>`. Hence, we have one test case.

OnEventScopeDifferingPartnerLinkRole has no `portType` attribute in `<onEvent>` and redefines the `<partnerLink>` in the associated `<scope>` of the `<onEvent>` with a `partnerRole` instead of a `myRole` attribute.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-EventHandlers-InitSync	**OnEventScopeDifferingPartnerLinkRole**	-

Table 66: SA00084 test pairs

1.84 SA00085

"The syntax and semantics of the <fromPart> elements as used on the <onEvent> element are the same as specified for the receive activity. This includes the restriction that if <fromPart> elements are used on an onEvent element then the variable, element and messageType attributes MUST NOT be used on the same element."[2, p. 203]

A process containing an `<onEvent>` with a `@variable` and `<fromParts>` violates rule #85. This is similar to #63 which is defined accordingly by [1, p. 29], but differs as it considers the `<onEvent>` specific `@element` and `@messageType` as well.

[`@variable`, none]
×
[`@element`, none]
×
[`@messageType`, none]

Formalization 45: SA Rule #85

Out of the eight total combinations, only the absence of all three attributes is the valid case, resulting in seven test cases. As `@variable` can only exist with either `@element` or `@messageType` according to rule #90, we can ignore these two test cases as they are already covered, totaling in five test cases.

OnEventFormPartsElement contains an additional `element` attribute in `<onEvent>`.

OnEventFormPartsElementMessageType contains additional attributes `element` and `messageType` in `<onEvent>`.

OnEventFormPartsMessageType contains an additional `messageType` attribute in `<onEvent>`.

OnEventFormPartsVariableElement contains additional attributes `variable` and `element` in `<onEvent>`.

OnEventFormPartsVariableMessageType contains additional attributes `variable` and `messageType` in `<onEvent>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-EventHandlers-Parts	**OnEventFormPartsElement**	-
Scope-EventHandlers-Parts	**OnEventFormPartsElementMessageType**	-
Scope-EventHandlers-Parts	**OnEventFormPartsMessageType**	-
Scope-EventHandlers-Parts	**OnEventFormPartsVariableElement**	-
Scope-EventHandlers-Parts	**OnEventFormPartsVariableMessageType**	-

Table 67: SA00085 test pairs

1.85 SA00086

"For <onEvent>, variables referenced by the variable attribute of <fromPart> elements or the variable attribute of an <onEvent> element are implicitly declared in the associated scope of the event handler. Variables of the same names MUST NOT be explicitly declared in the associated scope."[2, p. 204]

Rule #86 has two tests that are also shown in the negation of the rule model by [1, p. 55]. Both have a `<variable>` definition and an additional `@name` occurrence in the same `<onEvent>` 1) in the `@variable` of `<onEvent>` or 2) as `@toVariable` of a `<fromPart>`.

OnEventExplicitFromPartToVaribaleDoublicate has a redefined `<variable>` in the `<onEvent>`.

OnEventExplicitVaribaleDoublicate has a redefined `<variable>` in the `<onEvent>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-EventHandlers-Parts	**OnEventExplicitFromPartToVaribale-Doublicate**	-
Scope-EventHandlers-InitSync	**OnEventExplicitVaribaleDoublicate**	-

Table 68: SA00086 test pairs

1.86 SA00087

"For <onEvent>, the type of the variable (as specified by the messageType attribute) MUST be the same as the type of the input message defined by operation referenced by the operation attribute. Optionally the messageType attribute may be omitted and instead the element attribute substituted if the message to be received has a single part and that part is defined with an element type. That element type MUST be an exact match of the element type referenced by the element attribute."[2, p. 204]

Rule #87 resembles #48 for `<onEvent>`. See rule #48 for more information on the model of Kopp et al. [1].

[`<message>` with one `<part>`, `<message>` with two `<part>`s, `<message>` with no `<part>`s]
×
[`@variable` with same `@element`, `@variable` with different `@element`, `@variable` with same `@messageType`]

Formalization 46: SA Rule #87

If we have one `<part>` in a `<message>`, it is valid to use the `@element` instead of `@messageType` for the variable definition. Using the `@messageType` is always valid. A different `@messageType` check is already part of rule #10. An empty `<message>` cannot have a correct element. Thus,

we get four tests as a result of the permutation.

OnEventElementTwoParts imports TestInterface-OnEventElementTwoParts.wsdl that have an additional operation, which request message consist of two `<part>` elements.

OnEventUnresolvedElement has a modified `element` attribute in `<onEvent>`.

OnEventUnresolvedMessageType has a modified `messageType` attribute in `<onEvent>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-EventHandlers-Element-InitSync	**OnEventElementTwoParts**	1
Scope-EventHandlers-Element-InitSync	**OnEventUnresolvedElement**	-
Scope-EventHandlers-InitSync	**OnEventUnresolvedMessageType**	-

Table 69: SA00087 test pairs

1.87 SA00088

”For <onEvent>, the resolution order of the correlation set(s) referenced by <correlation> MUST be first the associated scope and then the ancestor scopes.”[2, p. 204]

Because rule #88 describes engine behavior, it is not part of the model of [1, p. 65]. The sole test is a `<correlation>` of the wrong type inside of an `<onEvent>` and a correct `<correlation>` outside.

OnEventCorrelationWrongType imports `TestInterface-StringCorrelation.wsdl` that defines an additional `<property>`. The latter is used to define a `<correlationSet>` in the `<onEvent>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-EventHandlers-InitSync	**OnEventCorrelationWrongType**	1

Table 70: SA00088 test pairs

1.88 SA00089

”For <onEvent>, when the messageExchange attribute is explicitly specified, the resolution order of the message exchange referenced by messageExchange attribute MUST be first the associated scope and then the ancestor scopes.”[2, p. 204]

Because rule #89 relates to an element that just carries a string, the rule is totally covered by #61. The difference is in positive cases where the resolution detects the `<messageExchange>` inside as well, but this cannot modeled in a negative scenario. Thus, there is a single test case without any `<messageExchange>` that can be resolved from `<onEvent>`. [1, p. 65] exclude this rule from their model.

OnEventNoMessageExchange contains no `<messageExchange>` and there is also no `messageExchange` attribute in the last `<reply>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-EventHandlers-Internal-MessageExchange-InitSync	**OnEventNoMessageExchange**	-

Table 71: SA00089 test pairs

1.89 SA00090

”If the variable attribute is used in the <onEvent> element, either the messageType or the element attribute MUST be provided in the <onEvent> element.”[2, p. 204]

The BPEL standard does not state in the related section of rule #90 if the element variable can be applied on single `<part>` `<messages>` exclusively. Yet, Kopp et al. [1] used the same restriction as modeled for rule #48. The two tests derived for #90, as defined in the standard, have all two additional attributes with `@variable` or just the sole attribute.

OnEventVariable has no `messageType` attribute in `<onEvent>`.

OnEventVariableElementMessageType contains an additional `element` attribute in `<onEvent>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-EventHandlers-InitSync	**OnEventVariable**	-
Scope-EventHandlers-InitSync	**OnEventVariableElementMessageType**	-

Table 72: SA00090 test pairs

1.90 SA00091

”A scope with the isolated attribute set to ’yes’ is called an isolated scope. Isolated scopes MUST NOT contain other isolated scopes.”[2, p. 204]

The rule #91 has a single test and is modeled by [1, p. 57] such that the negation of the rule model shows the test. But the model has a typo: Before the ⇒ the “)” is false.

IsolatedScopeInIsolatedSope contains an additional `<scope>` with a positive `isolated` attribute surrounding the `<flow>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-Isolated	**IsolatedScopeInIsolatedSope**	-

Table 73: SA00091 test pairs

1.91 SA00092

”Within a scope, the name of all named immediately enclosed scopes MUST be unique.”[2, p. 204]

The negation of the rule #92 model by [1, p. 57] shows the two test cases that have two `<scope>` elements with equal `@name` directly enclosed in the same `<scope>` or `<process>`.

ScopeNameDuplicate has a modified `name` attribute in the second `<scope>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-Isolated	**ScopeNameDuplicate**	-

Table 74: SA00092 test pairs

1.92 SA00093

"Identical <catch> constructs MUST NOT exist within a <faultHandlers> element."[2, p. 204]

Each test of rule #93 has a `<catch>` duplicate in the same `<faultHandlers>` element. This can be seen in the negation of the rule model by [1, p. 59]. However, a `<catch>` can be distinguished by three attributes, thus we need one test per combination.

[`<scope>`, `<process>`]
×
[`@faultElement`, none]
×
[`@faultMessageType`, none]
×
[`@faultName`, none]

Formalization 47: SA Rule #93

There shall always be at least one attribute in a `<catch>` and having all attributes together adds no new information. In addition the combination of `@faultMessageType` and `@faultElements` violate rule #81, so we require ten tests in total.

SameCatchFaultElement has no `faultName` in the `<catch>`, but the `<catch>` is present a second time.

SameCatchFaultElementFaultName contains a duplicate of the `<catch>`.

SameCatchFaultMessageType contains addtional `faultMessageType` and no `faultVariable`, but no `faultName` attribute in the first `<catch>`.

SameCatchFaultMessageTypeFaultName contains an additional `faultName` attribute in the second `<catch>`.

SameCatchFaultName contains no `faultMessageType` and no `faultVariable`, but an additional `faultName` attribute in the second `<catch>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-FaultHandlers-FaultElement	**SameCatchFaultElement**	-
Scope-FaultHandlers-FaultElement	**SameCatchFaultElementFaultName**	-
Scope-FaultHandlers-CatchOrder	**SameCatchFaultMessageType**	-
Scope-FaultHandlers-CatchOrder	**SameCatchFaultMessageTypeFaultName**	-
Scope-FaultHandlers-CatchOrder	**SameCatchFaultName**	-

Table 75: SA00093 test pairs

1.93 SA00094

“For <copy>, when the keepSrcElementName attribute is set to “yes” and the destination element is the Document EII of an element-based variable or an element-based part of a WSDL message-type-based variable, the name of the source element MUST belong to the substitutionGroup of the destination element. This checking MAY be enforced through static analysis of the expression/query language.”[2, p. 204]

Because the rule deals with general expression parsing, it is postponed to future work.

1.94 SA00095

”For <onEvent>, the variable references are resolved to the associated scope only and MUST NOT be resolved to the ancestor scopes.”[2, p. 205]

Rule #95 is mentioned by [1, p. 26] but is not modeled properly. A single test is required that resolves a `<variable>` usage to the ancestor `<scope>`, only.

OnEventVariableOutboundUseAssign contains an additional `<sequence>` surrounding the `<wait>` and an additional `<assign>` that uses the same `variable` as the `<onEvent>`.

betsy Test Case Origin	Derived Test Case	Modified WSDLs
Scope-EventHandlers-InitSync	**OnEventVariableOutboundUseAssign**	-

Table 76: SA00095 test pairs

References

[1] O. Kopp, R. Mietzner, and F. Leymann. Abstract Syntax of WS-BPEL 2.0. Technical Report 2008/06, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, September 2008.

[2] Web Services Business Process Execution Language. Standard Version 2.0, OASIS Open, April 2007.

[3] OASIS. *Web Services Business Process Execution Language*, April 2007. v2.0.

[4] W3C. *Web Services Description Language (WSDL) 1.1*, March 2001.

2 List of previous University of Bamberg reports

Bamberger Beiträge zur Wirtschaftsinformatik

Nr. 1 (1989) Augsburger W., Bartmann D., Sinz E.J.: Das Bamberger Modell: Der Diplom-Studiengang Wirtschaftsinformatik an der Universität Bamberg (Nachdruck Dez. 1990)

Nr. 2 (1990) Esswein W.: Definition, Implementierung und Einsatz einer kompatiblen Datenbankschnittstelle für PROLOG

Nr. 3 (1990) Augsburger W., Rieder H., Schwab J.: Endbenutzerorientierte Informationsgewinnung aus numerischen Daten am Beispiel von Unternehmenskennzahlen

Nr. 4 (1990) Ferstl O.K., Sinz E.J.: Objektmodellierung betrieblicher Informationsmodelle im Semantischen Objektmodell (SOM) (Nachdruck Nov. 1990)

Nr. 5 (1990) Ferstl O.K., Sinz E.J.: Ein Vorgehensmodell zur Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM)

Nr. 6 (1991) Augsburger W., Rieder H., Schwab J.: Systemtheoretische Repräsentation von Strukturen und Bewertungsfunktionen über zeitabhängigen betrieblichen numerischen Daten

Nr. 7 (1991) Augsburger W., Rieder H., Schwab J.: Wissensbasiertes, inhaltsorientiertes Retrieval statistischer Daten mit EISREVU / Ein Verarbeitungsmodell für eine modulare Bewertung von Kennzahlenwerten für den Endanwender

Nr. 8 (1991) Schwab J.: Ein computergestütztes Modellierungssystem zur Kennzahlenbewertung

Nr. 9 (1992) Gross H.-P.: Eine semantiktreue Transformation vom Entity-Relationship-Modell in das Strukturierte Entity-Relationship-Modell

Nr. 10 (1992) Sinz E.J.: Datenmodellierung im Strukturierten Entity-Relationship-Modell (SERM)

Nr. 11 (1992) Ferstl O.K., Sinz E. J.: Glossar zum Begriffsystem des Semantischen Objektmodells

Nr. 12 (1992) Sinz E. J., Popp K.M.: Zur Ableitung der Grobstruktur des konzeptuellen Schemas aus dem Modell der betrieblichen Diskurswelt

Nr. 13 (1992) Esswein W., Locarek H.: Objektorientierte Programmierung mit dem Objekt-Rollenmodell

Nr. 14 (1992) Esswein W.: Das Rollenmodell der Organsiation: Die Berücksichtigung aufbauorganisatorische Regelungen in Unternehmensmodellen

Nr. 15 (1992) Schwab H. J.: EISREVU-Modellierungssystem. Benutzerhandbuch

Nr. 16 (1992) Schwab K.: Die Implementierung eines relationalen DBMS nach dem Client/Server-Prinzip

Nr. 17 (1993) Schwab K.: Konzeption, Entwicklung und Implementierung eines computergestützten Bürovorgangssystems zur Modellierung von Vorgangsklassen und Abwicklung und Überwachung von Vorgängen. Dissertation

Nr. 18 (1993) Ferstl O.K., Sinz E.J.: Der Modellierungsansatz des Semantischen Objektmodells

Nr. 19 (1994) Ferstl O.K., Sinz E.J., Amberg M., Hagemann U., Malischewski C.: Tool-Based Business Process Modeling Using the SOM Approach

Nr. 20 (1994) Ferstl O.K., Sinz E.J.: From Business Process Modeling to the Specification of Distributed Business Application Systems - An Object-Oriented Approach -. 1st edition, June 1994

Ferstl O.K., Sinz E.J. : Multi-Layered Development of Business Process Models and Distributed Business Application Systems - An Object-Oriented Approach -. 2nd edition, November 1994

Nr. 21 (1994) Ferstl O.K., Sinz E.J.: Der Ansatz des Semantischen Objektmodells zur Modellierung von Geschäftsprozessen

Nr. 22 (1994) Augsburger W., Schwab K.: Using Formalism and Semi-Formal Constructs for Modeling Information Systems

Nr. 23 (1994) Ferstl O.K., Hagemann U.: Simulation hierarischer objekt- und transaktionsorientierter Modelle

Nr. 24 (1994) Sinz E.J.: Das Informationssystem der Universität als Instrument zur zielgerichteten Lenkung von Universitätsprozessen

Nr. 25 (1994) Wittke M., Mekinic, G.: Kooperierende Informationsräume. Ein Ansatz für verteilte Führungsinformationssysteme

Nr. 26 (1995) Ferstl O.K., Sinz E.J.: Re-Engineering von Geschäftsprozessen auf der Grundlage des SOM-Ansatzes

Nr. 27 (1995) Ferstl, O.K., Mannmeusel, Th.: Dezentrale Produktionslenkung. Erscheint in CIM-Management 3/1995

Nr. 28 (1995) Ludwig, H., Schwab, K.: Integrating cooperation systems: an event-based approach

Nr. 30 (1995) Augsburger W., Ludwig H., Schwab K.: Koordinationsmethoden und -werkzeuge bei der computergestützten kooperativen Arbeit

Nr. 31 (1995) Ferstl O.K., Mannmeusel T.: Gestaltung industrieller Geschäftsprozesse

Nr. 32 (1995) Gunzenhäuser R., Duske A., Ferstl O.K., Ludwig H., Mekinic G., Rieder H., Schwab H.-J., Schwab K., Sinz E.J., Wittke M: Festschrift zum 60. Geburtstag von Walter Augsburger

Nr. 33 (1995) Sinz, E.J.: Kann das Geschäftsprozeßmodell der Unternehmung das unternehmensweite Datenschema ablösen?

Nr. 34 (1995) Sinz E.J.: Ansätze zur fachlichen Modellierung betrieblicher Informationssysteme - Entwicklung, aktueller Stand und Trends -

Nr. 35 (1995) Sinz E.J.: Serviceorientierung der Hochschulverwaltung und ihre Unterstützung durch workflow-orientierte Anwendungssysteme

Nr. 36 (1996) Ferstl O.K., Sinz, E.J., Amberg M.: Stichwörter zum Fachgebiet Wirtschaftsinformatik. Erscheint in: Broy M., Spaniol O. (Hrsg.): Lexikon Informatik und Kommunikationstechnik, 2. Auflage, VDI-Verlag, Düsseldorf 1996

Nr. 37 (1996) Ferstl O.K., Sinz E.J.: Flexible Organizations Through Object-oriented and Transaction-oriented Information Systems, July 1996

Nr. 38 (1996) Ferstl O.K., Schäfer R.: Eine Lernumgebung für die betriebliche Aus- und Weiterbildung on demand, Juli 1996

Nr. 39 (1996) Hazebrouck J.-P.: Einsatzpotentiale von Fuzzy-Logic im Strategischen Management dargestellt an Fuzzy-System-Konzepten für Portfolio-Ansätze

Nr. 40 (1997) Sinz E.J.: Architektur betrieblicher Informationssysteme. In: Rechenberg P., Pomberger G. (Hrsg.): Handbuch der Informatik, Hanser-Verlag, München 1997

Nr. 41 (1997) Sinz E.J.: Analyse und Gestaltung universitärer Geschäftsprozesse und Anwendungssysteme. Angenommen für: Informatik '97. Informatik als Innovationsmotor. 27. Jahrestagung der Gesellschaft für Informatik, Aachen 24.-26.9.1997

Nr. 42 (1997) Ferstl O.K., Sinz E.J., Hammel C., Schlitt M., Wolf S.: Application Objects – fachliche Bausteine für die Entwicklung komponentenbasierter Anwendungssysteme. Angenommen für: HMD – Theorie und Praxis der Wirtschaftsinformatik. Schwerpunkheft ComponentWare, 1997

Nr. 43 (1997): Ferstl O.K., Sinz E.J.: Modeling of Business Systems Using the Semantic Object Model (SOM) – A Methodological Framework - . Accepted for: P. Bernus, K. Mertins, and G. Schmidt (ed.): Handbook on Architectures of Information Systems. International Handbook on Information Systems, edited by Bernus P., Blazewicz J., Schmidt G., and Shaw M., Volume I, Springer 1997

Ferstl O.K., Sinz E.J.: Modeling of Business Systems Using (SOM), 2nd Edition. Appears in: P. Bernus, K. Mertins, and G. Schmidt (ed.): Handbook on Architectures of Information Systems. International Handbook on Information Systems, edited by Bernus P., Blazewicz J., Schmidt G., and Shaw M., Volume I, Springer 1998

Nr. 44 (1997) Ferstl O.K., Schmitz K.: Zur Nutzung von Hypertextkonzepten in Lernumgebungen. In: Conradi H., Kreutz R., Spitzer K. (Hrsg.): CBT in der Medizin – Methoden, Techniken, Anwendungen -. Proceedings zum Workshop in Aachen 6. – 7. Juni 1997. 1. Auflage Aachen: Verlag der Augustinus Buchhandlung

Nr. 45 (1998) Ferstl O.K.: Datenkommunikation. In. Schulte Ch. (Hrsg.): Lexikon der Logistik, Oldenbourg-Verlag, München 1998

Nr. 46 (1998) Sinz E.J.: Prozeßgestaltung und Prozeßunterstützung im Prüfungswesen. Erschienen in: Proceedings Workshop „Informationssysteme für das Hochschulmanagement". Aachen, September 1997

Nr. 47 (1998) Sinz, E.J.:, Wismans B.: Das „Elektronische Prüfungsamt". Erscheint in: Wirtschaftswissenschaftliches Studium WiSt, 1998

Nr. 48 (1998) Haase, O., Henrich, A.: A Hybrid Respresentation of Vague Collections for Distributed Object Management Systems. Erscheint in: IEEE Transactions on Knowledge and Data Engineering

Nr. 49 (1998) Henrich, A.: Applying Document Retrieval Techniques in Software Engineering Environments. In: Proc. International Conference on Database and Expert Systems

Applications. (DEXA 98), Vienna, Austria, Aug. 98, pp. 240-249, Springer, Lecture Notes in Computer Sciences, No. 1460

Nr. 50 (1999) Henrich, A., Jamin, S.: On the Optimization of Queries containing Regular Path Expressions. Erscheint in: Proceedings of the Fourth Workshop on Next Generation Information Technologies and Systems (NGITS'99), Zikhron-Yaakov, Israel, July, 1999 (Springer, Lecture Notes)

Nr. 51 (1999) Haase O., Henrich, A.: A Closed Approach to Vague Collections in Partly Inaccessible Distributed Databases. Erscheint in: Proceedings of the Third East-European Conference on Advances in Databases and Information Systems – ADBIS'99, Maribor, Slovenia, September 1999 (Springer, Lecture Notes in Computer Science)

Nr. 52 (1999) Sinz E.J., Böhnlein M., Ulbrich-vom Ende A.: Konzeption eines Data Warehouse-Systems für Hochschulen. Angenommen für: Workshop „Unternehmen Hochschule" im Rahmen der 29. Jahrestagung der Gesellschaft für Informatik, Paderborn, 6. Oktober 1999

Nr. 53 (1999) Sinz E.J.: Konstruktion von Informationssystemen. Der Beitrag wurde in geringfügig modifizierter Fassung angenommen für: Rechenberg P., Pomberger G. (Hrsg.): Informatik-Handbuch. 2., aktualisierte und erweiterte Auflage, Hanser, München 1999

Nr. 54 (1999) Herda N., Janson A., Reif M., Schindler T., Augsburger W.: Entwicklung des Intranets SPICE: Erfahrungsbericht einer Praxiskooperation.

Nr. 55 (2000) Böhnlein M., Ulbrich-vom Ende A.: Grundlagen des Data Warehousing. Modellierung und Architektur

Nr. 56 (2000) Freitag B, Sinz E.J., Wismans B.: Die informationstechnische Infrastruktur der Virtuellen Hochschule Bayern (vhb). Angenommen für Workshop "Unternehmen Hochschule 2000" im Rahmen der Jahrestagung der Gesellschaft f. Informatik, Berlin 19. - 22. September 2000

Nr. 57 (2000) Böhnlein M., Ulbrich-vom Ende A.: Developing Data Warehouse Structures from Business Process Models.

Nr. 58 (2000) Knobloch B.: Der Data-Mining-Ansatz zur Analyse betriebswirtschaftlicher Daten.

Nr. 59 (2001) Sinz E.J., Böhnlein M., Plaha M., Ulbrich-vom Ende A.: Architekturkonzept eines verteilten Data-Warehouse-Systems für das Hochschulwesen. Angenommen für: WI-IF 2001, Augsburg, 19.-21. September 2001

Nr. 60 (2001) Sinz E.J., Wismans B.: Anforderungen an die IV-Infrastruktur von Hochschulen. Angenommen für: Workshop „Unternehmen Hochschule 2001" im Rahmen der Jahrestagung der Gesellschaft für Informatik, Wien 25. – 28. September 2001

Änderung des Titels der Schriftenreihe *Bamberger Beiträge zur Wirtschaftsinformatik* in *Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik* ab Nr. 61

Note: The title of our technical report series has been changed from *Bamberger Beiträge zur Wirtschaftsinformatik* to *Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik* starting with TR No. 61

Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik

Nr. 61 (2002) Goré R., Mendler M., de Paiva V. (Hrsg.): Proceedings of the International Workshop on Intuitionistic Modal Logic and Applications (IMLA 2002), Copenhagen, July 2002.

Nr. 62 (2002) Sinz E.J., Plaha M., Ulbrich-vom Ende A.: Datenschutz und Datensicherheit in einem landesweiten Data-Warehouse-System für das Hochschulwesen. Erscheint in: Beiträge zur Hochschulforschung, Heft 4-2002, Bayerisches Staatsinstitut für Hochschulforschung und Hochschulplanung, München 2002

Nr. 63 (2005) Aguado, J., Mendler, M.: Constructive Semantics for Instantaneous Reactions

Nr. 64 (2005) Ferstl, O.K.: Lebenslanges Lernen und virtuelle Lehre: globale und lokale Verbesserungspotenziale. Erschienen in: Kerres, Michael; Keil-Slawik, Reinhard (Hrsg.); Hochschulen im digitalen Zeitalter: Innovationspotenziale und Strukturwandel, S. 247 – 263; Reihe education quality forum, herausgegeben durch das Centrum für eCompetence in Hochschulen NRW, Band 2, Münster/New York/München/Berlin: Waxmann 2005

Nr. 65 (2006) Schönberger, Andreas: Modelling and Validating Business Collaborations: A Case Study on RosettaNet

Nr. 66 (2006) Markus Dorsch, Martin Grote, Knut Hildebrandt, Maximilian Röglinger, Matthias Sehr, Christian Wilms, Karsten Loesing, and Guido Wirtz: Concealing Presence Information in Instant Messaging Systems, April 2006

Nr. 67 (2006) Marco Fischer, Andreas Grünert, Sebastian Hudert, Stefan König, Kira Lenskaya, Gregor Scheithauer, Sven Kaffille, and Guido Wirtz: Decentralized Reputation Management for Cooperating Software Agents in Open Multi-Agent Systems, April 2006

Nr. 68 (2006) Michael Mendler, Thomas R. Shiple, Gérard Berry: Constructive Circuits and the Exactness of Ternary Simulation

Nr. 69 (2007) Sebastian Hudert: A Proposal for a Web Services Agreement Negotiation Protocol Framework . February 2007

Nr. 70 (2007) Thomas Meins: Integration eines allgemeinen Service-Centers für PC-und Medientechnik an der Universität Bamberg – Analyse und Realisierungs-Szenarien. February 2007 (out of print)

Nr. 71 (2007) Andreas Grünert: Life-cycle assistance capabilities of cooperating Software Agents for Virtual Enterprises. März 2007

Nr. 72 (2007) Michael Mendler, Gerald Lüttgen: Is Observational Congruence on μ-Expressions Axiomatisable in Equational Horn Logic?

Nr. 73 (2007) Martin Schissler: out of print

Nr. 74 (2007) Sven Kaffille, Karsten Loesing: Open chord version 1.0.4 User's Manual. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 74, Bamberg University, October 2007. ISSN 0937-3349.

Nr. 75 (2008) Karsten Loesing (Hrsg.): Extended Abstracts of the Second *Privacy Enhancing Technologies Convention* (PET-CON 2008.1). Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 75, Bamberg University, April 2008. ISSN 0937-3349.

Nr. 76 (2008) Gregor Scheithauer, Guido Wirtz: Applying Business Process Management Systems – A Case Study. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 76, Bamberg University, May 2008. ISSN 0937-3349.

Nr. 77 (2008) Michael Mendler, Stephan Scheele: Towards Constructive Description Logics for Abstraction and Refinement. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 77, Bamberg University, September 2008. ISSN 0937-3349.

Nr. 78 (2008) Gregor Scheithauer, Matthias Winkler: A Service Description Framework for Service Ecosystems. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 78, Bamberg University, October 2008. ISSN 0937-3349.

Nr. 79 (2008) Christian Wilms: Improving the Tor Hidden Service Protocol Aiming at Better Performances. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 79, Bamberg University, November 2008. ISSN 0937-3349.

Nr. 80 (2009) Thomas Benker, Stefan Fritzemeier, Matthias Geiger, Simon Harrer, Tristan Kessner, Johannes Schwalb, Andreas Schönberger, Guido Wirtz: QoS Enabled B2B Integration. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 80, Bamberg University, May 2009. ISSN 0937-3349.

Nr. 81 (2009) Ute Schmid, Emanuel Kitzelmann, Rinus Plasmeijer (Eds.): Proceedings of the ACM SIGPLAN Workshop on *Approaches and Applications of Inductive Programming* (AAIP'09), affiliated with ICFP 2009, Edinburgh, Scotland, September 2009. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 81, Bamberg University, September 2009. ISSN 0937-3349.

Nr. 82 (2009) Ute Schmid, Marco Ragni, Markus Knauff (Eds.): Proceedings of the KI 2009 Workshop *Complex Cognition*, Paderborn, Germany, September 15, 2009. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 82, Bamberg University, October 2009. ISSN 0937-3349.

Nr. 83 (2009) Andreas Schönberger, Christian Wilms and Guido Wirtz: A Requirements Analysis of Business-to-Business Integration. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 83, Bamberg University, December 2009. ISSN 0937-3349.

Nr. 84 (2010) Werner Zirkel, Guido Wirtz: A Process for Identifying Predictive Correlation Patterns in Service Management Systems. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 84, Bamberg University, February 2010. ISSN 0937-3349.

Nr. 85 (2010) Jan Tobias Mühlberg und Gerald Lüttgen: Symbolic Object Code Analysis. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 85, Bamberg University, February 2010. ISSN 0937-3349.

Nr. 86 (2010) Werner Zirkel, Guido Wirtz: Proaktives Problem Management durch Eventkorrelation – ein *Best Practice* Ansatz. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 86, Bamberg University, August 2010. ISSN 0937-3349.

Nr. 87 (2010) Johannes Schwalb, Andreas Schönberger: Analyzing the Interoperability of WS-Security and WS-ReliableMessaging Implementations. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 87, Bamberg University, September 2010. ISSN 0937-3349.

Nr. 88 (2011) Jörg Lenhard: A Pattern-based Analysis of WS-BPEL and Windows Workflow. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 88, Bamberg University, March 2011. ISSN 0937-3349.

Nr. 89 (2011) Andreas Henrich, Christoph Schlieder, Ute Schmid [eds.]: Visibility in Information Spaces and in Geographic Environments – Post-Proceedings of the KI'11 Workshop. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 89, Bamberg University, December 2011. ISSN 0937-3349.

Nr. 90 (2012) Simon Harrer, Jörg Lenhard: Betsy - A BPEL Engine Test System. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 90, Bamberg University, July 2012. ISSN 0937-3349.

Nr. 91 (2013) Michael Mendler, Stephan Scheele: On the Computational Interpretation of CKn for Contextual Information Processing - Ancillary Material. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 91, Bamberg University, May 2013. ISSN 0937-3349.

Nr. 92 (2013) Matthias Geiger: BPMN 2.0 Process Model Serialization Constraints. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 92, Bamberg University, May 2013. ISSN 0937-3349.

Nr. 93 (2014) Cedric Röck, Simon Harrer: Literature Survey of Performance Benchmarking Approaches of BPEL Engines. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 93, Bamberg University, May 2014. ISSN 0937-3349.