I
STATICALLY
ANALYZE
BPEL
FILES
FLAWLESSLY

Analysis and implementation of 30 BPEL SA Rules

David Bimamisa, Christian Preissinger, Simon Harrer, Stephan Schuberth

Lehrstuhl fuer Praktische Informatik, Fakultät WIAI

Abstract The Business Process Execution Language standard defines 94 static analysis rules, which any executable BPEL process must satisfy. Since there is no open source implementation of these rules that we know of, this is an attempt to implement it.

Hence we analysed 30 of the rules. We distilled the essence of them into a pseudo code providing a step by step analysis pattern. To provide correct implementation we designed BPEL processes, for each possible rule interpretation, that could just fail one rule. The implementation of these 30 static analysis rules was done in Java.

This technical report documents the results of our work, including the pseudo code and its notation, descriptions for all test cases and annotations to the implementation.

Keywords Web Services, BPEL, static analysis, WSDL, XSD, XPath

Contents

List of Abbreviations

API	Application Programming Interface
BPEL	Business Process Execution Language
CLI	Commandline Interface
JAXB	Java Architecture for XML Binding
JAXP	Java Architecture for XML Processing
OASIS	Organization for the Advancement of Structured Information Standards
SOA	service-oriented architecture
SOAP	Simple Object Access Protocol
URL	Uniform Resource Locator
UUID	Universal Unique Identifier
WS	Web service
WSDL	Webservice Description Language
XML	eXtended Markup Language
XPath	XML Path Language
XSD	XML Schema Document
XSL	eXtensible Stylesheet Language

Chapter 1

Motivation

The *Business Process Execution Language (BPEL)* is a XML based language used for modeling and combining business processes as a Web Services. BPEL is mainly used in combination with the following specifications: *WSDL 1.1*, *XML Schema 1.0*, *XPath 1.0* and *XSLT 1.0* (see section 2). The functionality (i.e operations, ports and data types) provided by a BPEL process is defined by using the *Web Services Description Language (WSDL)*. Like WSDL messages, *XML Schema definitions (XSD)* define specific data types used in the WS-BPEL process. *XML Path Language (XPath)* is the default query language for data manipulation of BPEL processes.

However, the detection of an incorrect process definition by validating against the WS-BPEL Schema definitions is not sufficient. For that reason, the BPEL Standard provides a number of requirements for static analysis (SA). BPEL is a widespread Standard in the Web Service community. Hence, it is necessary to provide a tool that validates a BPEL process against the Standard requirements. This should reduce the efforts needed for process modeling.

Against this background, our main goal is to translate those SA requirements systematically into algorithms so that they can be easily integrated in static analysis tools for BPEL processes. Our second goal is to test those algorithms by building a tool called ISABEL. ISABEL is a validation tool that detects violations within a BPEL process file. This includes the loading of all reachable dependencies (i.e WSDL and XSD files) of the given BPEL process file and a sufficient violation description. Note that according to the WS-BPEL Standard some process definitions could be executed even thought they would be rejected by the static analysis tool, in case the invalid code in question is practically never reached.

In this paper, we present the algorithms to validate a BPEL process description according to 30 SA requirements, the architecture and the functionality of ISABEL. The next chapter Fundamentals indicates, which tools and specifications we used to build ISABEL. After that, we specify the the outline how we proceeded in general, how the specification is structured. 30 BPEL static analysis (SA) requirements descriptions by means of algorithms and test cases follow. The next chapters approach ISABEL's architecture, its API, as well as essential changes made to one of the libraries we used and a user manual.
The document closes with a reflection for future work to be done on ISABEL from our point of view and a conclusion.

Chapter 2

Fundamentals

We used the following technologies:

- BPEL 2.0 `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf`
- XPath 1.0 `http://www.w3.org/TR/xpath/`
- SOAP 1.1 `http://www.w3.org/TR/2000/NOTE-SOAP-20000508/`
- WSDL 1.1 `http://www.w3.org/TR/wsdl`
- XSD `http://www.w3.org/XML/Schema/`
- modified XOM 1.2.7 Libary `https://github.com/uniba-dsg/XOM`
- Java Code is written in Java 7 `http://docs.oracle.com/javase/7/docs/api/`
- Gradle `http://www.gradle.org/`

Chapter 3

Rules

3.1 Outline

To guarantee for comparable analysis results, we developed following structure for our specification to be used when we characterized every SA rule:

1. short description(s)
2. SA specification
3. algorithm
4. algorithm description
5. test case descriptions

The **short descriptions** are the actual error messages that will be shown when using ISABEL. In the **SA specification** section the actual SA rule description is copied and pasted, exactly as it is provided in the BPEL standard. The **algorithm** listing shows a high-level procedure how to check to concerned rule. Section **algorithm description** tries to further elaborate the preceeding algorithm, to smooth out rough edges that may exist and to prevent misunderstandings. Afterwards the **test case descriptions** shows what test cases we designed to ensure the validator in question is working properly.

3.2 Notation of the pseudo code

During specification preparation we faced the problem, how to properly record the results of our analysis of the SA rules. It was decided write down the algorithms in form of pseudo code, instead of simple plaintext. Working text-based only was not an option, because the specifications would have become too verbose in the process. Since ISABEL was a team effort, we had to agree on a pattern, which led to the pseudo code notation which will be covered in this section.

The **pseudocode structure** used for the algorithm specification of the .bpel file validation is divided into **three parts**:

- Definition of global variables to shorten things later or to mark edge cases.
- The algorithm specification itself, not containing newlines.
- Definition of subroutines used in the algorithm specification.
  - Several subroutines each separated by single newline.

All three main parts are separated by three newlines.

The keywords in use are based on the regular control structures being present in imperative programming languages and operations needed when working with tree data structures.

```
####   START OF PSEUDOCODE NOTATION SPEC   ####

 "KEYWORDS"

        written with CAPS LOCK.
        represent the control structures and logical operators in the algorithm description.

            _control structures_:
                IF
                ELSE
                UNLESS (same as IF NOT)
                FOREACH
                RETURN
                FILE ... (load the file via the reference written right of the keyword)
            _terminators_:
                FAIL (indicates definite SA rule violation)
                ERROR (variable reference across several files could not be resolved)

            _general operators_:
                EQUALS
                GREATER_THAN
                LESS_THAN
            _dom tree operators_:
                ... ROOT (gets root element of file currently working in)
                ... EXISTS (indicates element to the left being present)
                PARENT (get <parentblock> of the <block> being currently in)
                ... ANCESTOR_OF ... (checks if an element is the ancestor (a parent or a
                    parent of a parent and so on) of another element)
                ... IS_MISSING (same as NOT EXISTS)
                DUPLICATE ... (returns true if trailing element/attribute is found at least
                    twice)
            _list operations_:
                FOREACH iterator IN list (to operate on list)
                ADD element TO list
                REMOVE element FROM list
                COUNT (returns count of elements present left of it, wildcards are possible)
                EMPTY_LIST (represents empty list, i.e. used like 'SET $variable TO
                    EMPTY_LIST')
                CONTAINS
                ANY list _general operator_ ... (basically the same as FOREACH)

            _logical operators_:
                NOT
                AND
                OR
                XOR

            _ordering_: (of <blocks> ocurring)
                BEFORE
                AFTER



"$variables"

        used within the algorithm specification declare variables, that are substituted.
        this is to save space and to slim down the final spec body.
        $variables stand for a block.
        there are two use cases:
            - define 'global' variables at the head of the specification, describing element
                sets
            - to save local results of methods(...) via SET $result TO ... for later use
        they exist solely for the purpose of shortening the algorithm specification.
        i.e.
            $global = <receive> | <response>
        i.e.
            SET $local TO correspondingPartnerlink(<receive>@partnerLink)
            IF $partnerLink EXISTS ...



"<blocks>"

        represent the elements used, are written with <tag> notation.
        special case: '<*>'
            represents any element of of the set of all possible elements.
        special case: '<ROOT>'
            represents root element of the current filetype
            i.e. for .bpel files:
                (FILE ...):<ROOT> gets the element '<process>'. The file loaded through '(
                    FILE ...):' is equal to 'bpel:'
        i.e.
            <*>
            <process>
            <part>



"@attributes"

        specify an attribute present in a <block>, appended to it.
        special case: '@*'
            represents any @attribute present.
        i.e.
```

```
            <block>@attribute



"filetype:"

        specifies a file.
        can be used alone or to show where a <block> is located in.
        prepended to a <block>.
        available are:
            file: general designator representing any filetype
            wsdl:
            xsd:
            bpel: (default value, if no filetype is specified)
        i.e.
            wsdl:<block>@attribute



"subroutine(...)"

        represent subroutines, used within the algorithm body.
        outsource smaller operations or operations needed often.
        help making a simpler and higher level specification possible.
        i.e.
            correspondingPorttype: <partnerLink> -> wsdl:<portType>



"'value'"

        any actual value representation of any primitive filetype, like boolean, integers or strings
        i.e.
            'this represents a string'
            '6654'
            'true'


"methodDefinitions: <block>@attribute -> <block>@attribute"
    .
    .
    .

        "... -> ..."
            defines inputs and outputs in method signature spec in first line.
            can take <block>@attribute parts as well as input or output.
        '|=>'
            for navigation step specification.
            classifies first step.
        '=>'
            for navigation step specification.
            classifies intermediate steps.
        '=>|'
            for step navigation specification.
            classifies last step, prepended to the return statement on the last line.
        methodDefinitions: ... -> ...  are separated with newlines from algorithm spec above.
            .
            .
            .
        i.e.
        correspondingPartnerlink: <activity>@partnerLink -> <partnerLink> ?:
        FOREACH <partnerLinks>
```

```
            IF <partnerLinks><partnerLink>@name EQUALS <activity>@partnerLink
            RETURN <partnerLinks><partnerLink>
        i.e.
        correspondingPorttype: <partnerLink> -> wsdl:<portType>
            <partnerLink>   |=>
            => wsdl:<partnerLinkType>
                IF wsdl:<partnerLinkType>@name EQUALS <partnerLink>@partnerLinkType
                AND
                IF wsdl:<partnerLinkType>@name EQUALS <partnerLink>@partnerLinkType
            => wsdl:<partnerLinkType><role>
                IF wsdl:<partnerLinkType><role> EQUALS <partnerLink>@partnerRole
            => wsdl:<portType>
                IF wsdl:<partnerLinkType><role>@portType EQUALS wsdl:<portType>@name
            =>| RETURN wsdl:<portType>

####   END OF PSEUDOCODE NOTATION SPEC   ####
```

3.3 SA00001

- *Short description(s):*
  - notification operation forbidden
  - solicit-response operation forbidden
- *SA specification:*
  A WS-BPEL processor MUST reject a WS-BPEL that refers to solicit-response or notification operations portTypes.
- *Algorithm:*

```
FOREACH wsdl:
    FOREACH <operation>
        FAIL_Notification IF isNotification(<operation>)
        FAIL_solicitResponse IF isSolicitResponse(<operation>)



isNotification: <operation> -> boolean
    RETURN (<output> EXISTS AND <input> IS_MISSING)

isSolicitResponse: <operation> -> boolean
    RETURN ((<output> EXISTS AND <input> EXISTS) AND (<output> BEFORE <input>))
```

- *Algorithm description:*
  The rule SA00001 forbids the message exchange patterns solicit-response and notification for WSDL `<operation>`'s. These patterns differ from the allowed one-way and request-response pattern by having the `<output>` element stated first.

  The algorithm detects the message exchange pattern of a WSDL `<operation>` by looking at the children elements `<input>` and `<output>`.

- *Test case description(s):*
  Notification.bpel, Notification.wsdl:
  One .wsdl with a `<operation name='notification'>` just containing an `<output>` in a separate PortType.
  SolicitResponse.bpel, SolicitResponse.wsdl:
  A .wsdl with a `<operation name='solicitResponse'>` containing an `<output>` followed by `<input>` in a separate PortType.

3.4 SA00002

- *Short description(s):*
  - overloaded operation name in <portType>

- *SA specification:*
  A WS-BPEL processor MUST reject any WSDL portType definition that includes overloaded operation names.
- *Algorithm:*

```
FOREACH wsdl:<portType>
    FAIL IF DUPLICATE <operation>@name
```

- *Algorithm description:*
  The `name` attribute of all `<operation>`'s' in a `<portType>` are inspected. The validation fails, if any name occurs twice or more often.
- *Test case description(s):*
  OverloadedOperationNames.bpel, OverloadedOperationNames.wsdl:
  `<portType>` has two `<operation>` named the same with different messages.

3.5 SA00003

- *Short description(s):*
  - exitOnStandardFault='yes' in <scope> or <process>, but catches standard fault
- *SA specification:*
  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.
- *Algorithm:*

```
$enclosingScope = <process> | <scope>
$standartFault =    {'ambiguousReceive',
                    'completionConditionFailure',
                    'conflictingReceive',
                    'conflictingRequest',
                    'correlationViolation',
                    'invalidBranchCondition ',
                    'invalidExpressionValue',
                    'invalidVariables',
                    'mismatchedAssignmentFailure',
                    'missingReply',
                    'missingRequest ',
                    'scopeInitializationFailure',
                    'selectionFailure ',
                    'subLanguageExecutionFault',
                    'uninitializedPartnerRole',
                    'uninitializedVariable',
                    'unsupportedReference',
                    'xsltInvalidSource',
                    'xsltStylesheetNotFound' }



FOREACH $enclosingScope
```

```
    IF $enclosingScope@exitOnStandardFault EQUALS 'yes'
        FAIL IF catchesStandardFault($enclosingScope)



catchesStandardFault: $currentScope -> boolean
    IF catchesStandardFaultDireckly($currentScope)
        RETURN 'true'
    FOREACH $currentScope<scope>
        UNLESS $currentScope<scope>@exitOnStandardFault EQUALS 'no'
            RETURN catchesStandardFault($currentScope<scope>)
    RETURN 'false'

catchesStandardFaultDirectly: $currentScope -> boolean
FOREACH $currentScope<faultHandlers><catch>
    RETURN (<catch>@faultName EQUALS ANY $standartFault)
```

- *Algorithm description:*
  The algorithm searches in `<process>` and every `<scope>` for the value of the `exitOnStandartFault` attribute. If it equals 'yes', then the validation fails on every standard fault except 'joinFailure' as attribute `faultName` of a `<catch>`. It is important to check every descendant `<scope>`, unless one contains the `exitOnStandartFault` attribute with value 'no', because this attribute is inherited.

- *Test case description(s):*
  ExitOnStandartFaultButCatchingStandardFaultInDirectFaultHandlers.bpel:
  `<scope>` with `@exitOnStandartFault='yes'` catches the BPEL standard-fault 'completionConditionFailure'.
  ExitOnStandartFaultButCatchingStandardFaultInDirectFaultHandlersInProcess.bpel:
  `<process>` with `@exitOnStandartFault='yes'` catches the BPEL standard-fault 'completionConditionFailure'.
  ExitOnStandartFaultButCatchingStandardFaultInIndirectFaultHandlers.bpel:
  A `<scope>`, inheriting @exitOnStandartFault='yes' from its parent <process>, catches the BPEL standard-fault 'completionConditionFailure'.
  ExitOnStandartFaultButCatchingStandardFaultInNestedFaultHandlers.bpel:
  A `<scope>`, with `@exitOnStandartFault='yes'` and a parent `<scope>` with `@exitOnStandartFault='no'`, catches the BPEL standard-fault 'completionConditionFailure'.
  ExitOnStandartFaultButCatchingStandardFaultInIndirectNestedFaultHandlers.bpel:
  A `<scope>`, inheriting `@exitOnStandartFault='yes'` from its parent `<scope>` catches the BPEL standard-fault 'completionConditionFailure'.

3.6 SA00005

- *Short description(s):*
  - @portType differs from implicit portType defined in <operation>

- *SA specification:*
  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.

- *Algorithm:*

```
$messageActivity = <receive> | <reply> | <invoke> | <onEvent> | <onMessage>



FOREACH $messageActivity@portType
    SET $partnerLink TO correspondingPartnerlink($messageActivity@partnerLink)
    FAIL UNLESS correspondingPorttype($partnerLink)@name EQUALS
        $messageActivity@portType



correspondingPartnerlink: $messageActivity@partnerLink -> <partnerLink>
    FOREACH <partnerLinks>
        IF <partnerLink>@name EQUALS $messageActivity@partnerLink
             RETURN <partnerLink>

correspondingPorttype: <partnerLink> -> wsdl:<portType>
    <partnerLink> |=>
        IF wsdl:<partnerLinkType>@name EQUALS <partnerLink>@partnerLinkType
    => wsdl:<partnerLinkType>
        IF wsdl:<partnerLinkType><role> EQUALS <partnerLink>@partnerRole
    => wsdl:<partnerLinkType><role>
        IF wsdl:<partnerLinkType><role>@portType EQUALS wsdl:<portType>@name
    =>| RETURN wsdl:<portType>
```

- *Algorithm description:*
SA00005 conveys:
If an optional `portType` is actually specified in a message activity (`<receive>`, `<reply>`, `<invoke>`, `<onEvent>` or `<onMessage>`) for convenience, it must be specified in the WSDL as well.

This is ensured by checking the corresponding `<partnerLink>` through its `name` attribute in the BPEL.
With the help of the freshly located `<partnerLink>` the corresponding `<portType>` in the linked WSDL can be found, which is slightly more complex:

  1. match the `partnerRole` of the `<partnerLink>` with the `<partnerLinkType>` in the WSDL by its `name`
  2. get the `<role>` from the found `<partnerLinkType>`
  3. from the `portType` of `<partnerLinkType><role>` find the `<portType>` in the WSDL through matching its `name`

- *Test case description(s):*
ReceiveWithNonExistantPortType.bpel:
A BPEL-file with a `<receive>` which `@portType` attribute is 'nonExistantPortTypeName' which is not in the WSDL and differs from the one specified in the corresponding `<operation>`.
ReplyWithNonExistantPortType.bpel:

A BPEL-file with a `<reply>` which `@portType` attribute is 'nonExistantPortTypeName' which is not in the WSDL and differs from the one specified in the corresponding `<operation>`.
InvokeWithNonExistantPortType.bpel:
A BPEL-file with a `<invoke>` which `@portType` attribute is 'nonExistantPortTypeName' which is not in the WSDL and differs from the one specified in the corresponding `<operation>`.
OnEventWithNonExistantPortType.bpel:
A BPEL-file with a `<onEvent>` which `@portType` attribute is 'nonExistantPortTypeName' which is not in the WSDL and differs from the one specified in the corresponding `<operation>`.
OnMessageWithNonExistantPortType.bpel:
A BPEL-file with a `<onMessage>` which `@portType` attribute is 'nonExistantPortTypeName' which is not in the WSDL and differs from the one specified in the corresponding `<operation>`.

3.7 SA00006

- *Short description(s):*
  - <rethrow> used outside of <catch> or <catchAll>
- *SA specification:*
  The <rethrow> activity MUST only be used within a faultHandler (i.e. <catch> and <catchAll> elements).
- *Algorithm:*

```
FOREACH <rethrow>
    FAIL UNLESS <faultHandlers> ANCESTOR_OF <rethrow>
```

- *Algorithm description:*
  Each faultHandler is within a `<faultHandlers>` block. Therefore the algorithm checks if any `<rethrow>` element has no `<faultHandlers>` element in its line of parents.
- *Test case description(s):*
  RethrowOutsideFaultHandlers.bpel:
  BPEL-file with a `<rethrow>` placed in a `<sequence>` as a direct child of `<process>`, outside of a `<faultHandlers>`. (Therefore outside a `<catch>` or `<catchAll>`.)

3.8 SA00007

- *Short description(s):*
  - <compensateScope> not within FCT-Handler

- *SA specification:*
  The <compensateScope> activity MUST only be used from within a faultHandler, another compensationHandler, or a terminationHandler.

- *Algorithm:*

```
FOREACH <compensateScope>
    SET $isFaultParent TO <faultHandlers> EQUALS PARENT <compensateScope>
    SET $isCompensationParent TO <compensationHandler> EQUALS PARENT <compensateScope>
    SET $isTerminationParent TO <terminationHandler> EQUALS PARENT <compensateScope>
    FAIL UNLESS ($isFaultParent OR $isCompensationParent OR $isTerminationParent)
```

- *Algorithm description:*
  A `<compensateScope>` needs to have a parent being either a `<faultHandlers>`, a `<compensationHandler>` or a `<terminationHandler>`.

  Otherwise the validation fails.

- *Test case description(s):*
  CompensateScopeOutsideFaultHandlers.bpel:
  The activity `<compensateScope>` is a direct child of `<process>`.

3.9 SA00008

- *Short description(s):*

  – <compensate> used outside of FCT-Handler

- *SA specification:*
  The <compensate> activity MUST only be used from within a faultHandler, another compensationHandler, or a terminationHandler.

- *Algorithm:*

```
FOREACH <compensate>
    SET $isFaultParent TO <faultHandlers> EQUALS PARENT <compensate>
    SET $isCompensationParent TO <compensationHandler> EQUALS PARENT <compensate>
    SET $isTerminationParent TO <terminationHandler> EQUALS PARENT <compensate>
    FAIL UNLESS ($isFaultParent OR $isCompensationParent OR $isTerminationParent)
```

- *Algorithm description:*
  A `<compensate>` needs to have a parent being either a `<faultHandlers>`, a `<compensationHandler>` or a `<terminationHandler>`.

  Otherwise the validation fails.

- *Test case description(s):*
  CompensateOutsideFaultHandlers.bpel:
  The activity `<compensate>` is a direct child of `<process>`.

3.10 SA00010

- *Short description(s):*
  - XML or WSDL not imported
- *SA specification:*
  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.
- *Algorithm:*

```
FOREACH <*>
    SET $elementType TO getTypeOf(<*>)
    FAIL UNLESS typeDefinitionExists($elementType)



getTypeOf: <*> -> QName
    IF <*> EQUALS <partnerLink>
        RETURN <partnerLink>@partnerLinkType
    IF <*> EQUALS <variable>
        IF @messageType EXISTS
            RETURN <variable>@messageType
        IF @type EXISTS
            RETURN <variable>@type
        IF @element EXISTS
            RETURN <variable>@element
    IF <*> EQUALS <correlationSet>
        RETURN <correlationSet>@properties
    IF <*> EQUALS <reply>
        RETURN <reply>@portType
    IF <*> EQUALS <catch>
        IF <catch>@faultMessageType EXISTS
            RETURN <catch>@faultMessageType
        IF <catch>@faultElement EXISTS
            RETURN <catch>@faultElement
    IF <*> EQUALS <receive>
        RETURN <receive>@portType
    IF <*> EQUALS <invoke>
        RETURN <invoke>@portType
    IF <*> EQUALS <onMessage>
        RETURN <onMessage>@portType
    IF <*> EQUALS <onEvent>
        RETURN <onEvent>@portType
    IF <*> EQUALS <to>
        RETURN <to>@property
    IF <*> EQUALS <from>
        RETURN <from>@property

typeDefinitionExists: $elementType -> boolean
    IF $elementType EQUALS <partnerLink>@partnerLinkType
        RETURN searchAllWsdl($elementType, <partnerLinkType>)
    IF $elementType EQUALS <variable>@messageType
        RETURN searchAllWsdl($elementType, <message>)
    IF $elementType EQUALS <variable>@type
        RETURN (searchAllXsd($elementType, <simpleType>) OR searchAllXsd($elementType,
            <complexType>))
    IF $elementType EQUALS <variable>@element
```

```
        RETURN searchAllXsd($elementType, <element>)
    IF $elementType EQUALS <correlationSet>@properties
        RETURN searchAllWsdl($elementType, <property>)
    IF $elementType EQUALS <reply>@portType
        RETURN searchAllWsdl($elementType, <portType>)
    IF $elementType EQUALS <catch>@faultMessageType
        RETURN searchAllWsdl($elementType, <message>)
    IF $elementType EQUALS <catch>@faultElement
        RETURN searchAllXsd($elementType, <element>)
    IF $elementType EQUALS <receive>@portType
        RETURN searchAllWsdl($elementType, <portType>)
    IF $elementType EQUALS <invoke>@portType
        RETURN searchAllWsdl($elementType, <portType>)
    IF $elementType EQUALS <onMessage>@portType
        RETURN searchAllWsdl($elementType, <portType>)
    IF $elementType EQUALS <onEvent>@portType
        RETURN searchAllWsdl($elementType, <portType>)
    IF $elementType EQUALS <to>@property
        RETURN searchAllWsdl($elementType, <property>)
    IF $elementType EQUALS <from>@property
        RETURN searchAllWsdl($elementType, <property>)

searchAllWsdl: $elementType, $node -> boolean
    FOREACH wsdl:
        FOREACH $node
            IF $node@name EQUALS $elementType
                RETURN 'true'
    RETURN 'false'

searchAllXsd: $elementType, $node -> boolean
    FOREACH xsd:
        FOREACH $node
            IF $node@name EQUALS $elementType
                RETURN 'true'
    RETURN 'false'
```

- *Algorithm description:*
  For all following element-attribute pairs on the left, defined in the BPEL, there must be a corresponding definition in the WSDL.
  If there are several possible attributes specified, only one must be present.

  The WSDL's definitions shown on the right are matched via their `name` attribute:

  – `<partnerLink>`, `partnerLinkType` $\longrightarrow$ `<partnerLinkType>`
  – `<variable>`, `messageType` $\longrightarrow$ `<message>`
  – `<variable>`, `type` $\longrightarrow$ `<complexType>` or `<simpleType>`
  – `<variable>`, `element` $\longrightarrow$ `<element>`
  – `<correlationSet>`, `properties` $\longrightarrow$ `<property>`
  – `<reply>`, `portType` $\longrightarrow$ `<portType>`
  – `<catch>`, `faultMessageType` $\longrightarrow$ `<messageType>`
  – `<catch>`, `faultElement` $\longrightarrow$ `<element>`
  – `<receive>`, `portType` $\longrightarrow$ `<portType>`
  – `<invoke>`, `portType` $\longrightarrow$ `<portType>`

  – `<onMessage>`, `portType` ⟶ `<portType>`
  – `<onEvent>`, `portType` ⟶ `<portType>`
  – `<to>`, `property` ⟶ `<property>`
  – `<from>`, `property` ⟶ `<property>`

All corresponding elements are found in the WSDL, except for when a `<variable>` has a `type` or a `element` attribute.

- *Test case description(s):*
  UndefinedType-Catch-FaultElement.bpel:
  A BPEL-file with a `<catch>@faultElement` which has no definition in corresponding Schema (http://www.w3.org/2001/XMLSchema).
  UndefinedType-Catch-FaultMessageType.bpel:
  A BPEL-file with a `<catch>@faultMessageType` which has no definition in corresponding TestInterface.wsdl.
  UndefinedType-CorrelationSet.bpel:
  A BPEL-file with a `<correlationSet>@properties` which has no definition in corresponding TestInterface.wsdl.
  UndefinedType-From.bpel:
  A BPEL-file with a `<from>@property` which has no definition in corresponding TestInterface.wsdl.
  UndefinedType-Invoke.bpel:
  A BPEL-file with a `<invoke>@portType` which has no definition in corresponding TestInterface.wsdl.
  UndefinedType-OnEvent.bpel:
  A BPEL-file with a `<onEvent>@portType` which has no definition in corresponding TestInterface.wsdl.
  UndefinedType-OnMessage.bpel:
  A BPEL-file with a `<onMessage>@portType` which has no definition in corresponding TestInterface.wsdl.
  UndefinedType-PartnerLink.bpel:
  A BPEL-file with a `<partnerLink>@partnerLinkType` which has no definition in corresponding TestInterface.wsdl.
  UndefinedType-Receive.bpel:
  A BPEL-file with a `<receive>@portType` which has no definition in corresponding TestInterface.wsdl.
  UndefinedType-Reply.bpel:
  A BPEL-file with a `<reply>@portType` which has no definition in corresponding TestInterface.wsdl.
  UndefinedType-To.bpel:
  A BPEL-file with a `<to>@property` which has no definition in corresponding TestInterface.wsdl.
  UndefinedType-Variable-Element.bpel:
  A BPEL-file with a `<variable>@element` which has no definition in corresponding Schema (http://www.w3.org/2001/XMLSchema).
  UndefinedType-Variable-MessageType.bpel:

A BPEL-file with a `<variable>@messageType` which has no definition in corresponding TestInterface.wsdl.
UndefinedType-Variable-Type.bpel:
A BPEL-file with a `<variable>@type` which has no definition in corresponding Schema (http://www.w3.org/2001/XMLSchema).

3.11 SA00011

- *Short description(s):*
  - imported targetNamespace differs from expected namespace
- *SA specification:*
  If a namespace attribute is specified on an <import> then the imported definitions MUST be in that namespace.
- *Algorithm:*

```
FOREACH <import>@namespace
    FAIL UNLESS @namespace EQUALS <import>@location:ROOT@targetNamespace
```

- *Algorithm description:*
  The `namespace` attribute of each `<import>` must equal the `targetNamespace` of the imported file.

  Otherwise the validation fails.
- *Test case description(s):*
  Import-WrongNameSpace.bpel:
  `@namespace='NamespaceDifferingFromOriginTargetNameSpace/testinterface'` of the TestInterface.wsdl `<import>` is differing from `targetNamespace` being
  `'http://dsg.wiai.uniba.de/bpel-engine-comparison/activities/wsdl/testinterface'`.

3.12 SA00012

- *Short description(s):*
  - imported targetNamespace was not expected
- *SA specification:*
  If no namespace is specified then the imported definitions MUST NOT contain a targetNamespace specification.
- *Algorithm:*

```
FOREACH <import>
    FAIL IF @namespace IS_MISSING AND <import>@location:ROOT@targetNamespace EXISTS
```

- *Algorithm description:*
  This SA is included by SA00010. Nonetheless the algorithm checks through each `<import>` without a `namespace` attribute, that the `targetNamespace` is not set as well.

- *Test case description(s):*
  Import-NoNameSpace.bpel:
  `<import>` has no namespace-attribute, but imported TestInterface.wsdl has
  `targetNamespace='http://dsg.wiai.uniba.de/bpel-engine-comparison/activities/wsdl/tes`

3.13 SA00013

- *Short description(s):*
  - imported type differs from expected type

- *SA specification:*
  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.

- *Algorithm:*

```
FOREACH <import>
    FAIL UNLESS <import>@importType EQUALS <import>@location:ROOT@namespace
```

- *Algorithm description:*
  We were not able to get the type of an `<import>`, for each document, the same way. Only files with default `namespace` as type definition can pass validation. Apart from this the validator is more abstract, than required.

  It compares the `importType` with the default `namespace` of the root element, for each `<import>`. The requirement for XML Schema 1.0 and WSDL 1.1 Documents are complied, because the algorithm includes them.

- *Test case description(s):*
  Import-WrongImportType.bpel:
  TestInterface.wsdl `<import>` has `importType='wrongImportType'` differing from the default `namespace='http://schemas.xmlsoap.org/wsdl/'`.

3.14 SA00020

- *Short description(s):*
  - <propertyAlias> uses wrong combination of attributes

- *SA specification:*
  A <vprop:propertyAlias> element MUST use one of the three following combinations of attributes: messageType and part, type or element

- *Algorithm:*

```
FOREACH wsdl:
    FOREACH <propertyAlias>
        SET $messageTypeAndPart TO (@messageType EXISTS AND @part EXISTS AND @type='xsd
            :string' NOT EXISTS AND @element='tns:testElementSyncRequest' NOT EXISTS)
        SET $type TO (@messageType NOT EXISTS AND @part NOT EXISTS AND @type='xsd:
            string' EXISTS AND @element='tns:testElementSyncRequest' NOT EXISTS)
        SET $element TO (@messageType NOT EXISTS AND @part NOT EXISTS AND @type='xsd:
            string' NOT EXISTS AND @element='tns:testElementSyncRequest' EXISTS)
        FAIL UNLESS ($messageTypeAndPart OR $type OR $element)
```

- *Algorithm description:*
  SA00020 defines a BPEL file only to be valid if each `<vprop:propertyAlias>` element uses either a combination of a `messageType` and a `part` attribute, or only a `type` attribute, or only a `element` attribute.

  Each `<propertyAlias>` element is simply checked to have valid occurences of these attributes.

- *Test case description(s):*
  PropertyAlias-AllOptionalAttributes.bpel,
  PropertyAlias-AllOptionalAttributes.wsdl:
  `<propertyAlias>` with `@messageType='tns:executeProcessSyncRequest'`, `@part='inputPart'`, `@type='xsd:string'` and `@element='tns:testElementSyncRequest'`.
  PropertyAlias-MessageTypeAttribute.bpel,
  PropertyAlias-MessageTypeAttribute.wsdl:
  `<propertyAlias>` with `@messageType='tns:executeProcessSyncRequest'` but no attribute out of `@part='inputPart'`, `@type`, `@element`.
  PropertyAlias-MessageTypeElementAttributes.bpel,
  PropertyAlias-MessageTypeElementAttributes.wsdl:
  `<propertyAlias>` with `@messageType='tns:executeProcessSyncRequest'` and `@element='tns:testElementSyncRequest'` but no attribute out of `@part`, `@type`.
  PropertyAlias-MessageTypePartElementAttributes.bpel,
  PropertyAlias-MessageTypePartElementAttributes.wsdl:
  `<propertyAlias>` with `@messageType='tns:executeProcessSyncRequest'`, `@part='inputPart'` and `@element='tns:testElementSyncRequest'` but no attribute `@type`.
  PropertyAlias-MessageTypePartTypeAttributes.bpel,
  PropertyAlias-MessageTypePartTypeAttributes.wsdl:
  `<propertyAlias>` with `@part='inputPart'`, `@messageType='tns:executeProcessSyncRequest'` and `@type='xsd:string'` but no attribute `@element`.
  PropertyAlias-MessageTypeTypeAttributes.bpel,
  PropertyAlias-MessageTypeTypeAttributes.wsdl:
  `<propertyAlias>` with `@messageType='tns:executeProcessSyncRequest'` and

`@type='xsd:string'` but no attribute out of `@part`, `@element`.
PropertyAlias-MessageTypeTypeElementAttributes.bpel,
PropertyAlias-MessageTypeTypeElementAttributes.wsdl:
`<propertyAlias>` with `@messageType='tns:executeProcessSyncRequest'`, `@element='tns:testElementSyncRequest'` and `@type='xsd:string'` but no attribute `@part`.
PropertyAlias-NoOptionalAttributes.bpel,
PropertyAlias-NoOptionalAttributes.wsdl:
`<propertyAlias>` with no attribute out of `@messageType`, `@part`, `@type`, `@element`.
PropertyAlias-PartAttribute.bpel,
PropertyAlias-PartAttribute.wsdl:
`<propertyAlias>` with `@part='inputPart'` but no attribute out of `@messageType='tns:executeProcessSyncRequest'`, `@type='xsd:string'`, `@element`.
PropertyAlias-PartElementAttributes.bpel,
PropertyAlias-PartElementAttributes.wsdl:
`<propertyAlias>` with `@part='inputPart'` and `@element='tns:testElementSyncRequest'` but no attribute out of `@messageType`, `@type`.
PropertyAlias-PartTypeAttributes.bpel,
PropertyAlias-PartTypeAttributes.wsdl:
`<propertyAlias>` with `@part='inputPart'` and `@type='xsd:string'` but no attribute out of `@messageType='tns:executeProcessSyncRequest'`, `@element`.
PropertyAlias-PartTypeElementAttributes.bpel,
PropertyAlias-PartTypeElementAttributes.wsdl:
`<propertyAlias>` with `@part='inputPart'`, `@type='xsd:string'` and `@element='tns:testElementSyncRequest'` but no attribute `@messageType`.
PropertyAlias-TypeElementAttributes.bpel,
PropertyAlias-TypeElementAttributes.wsdl:
`<propertyAlias>` with `@type='xsd:string'` and `@element='tns:testElementSyncRequest'` but no attribute out of `@messageType`, `@part`.

3.15 SA00021

- *Short description(s):*
  - <propertyAlias> not defined in any *.wsdl
- *SA specification:*
  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.
- *Algorithm:*

```
$fromTo = <from> | <to>
$scopes = <process> | <scope>



FOREACH <correlationSet>@properties
    FAIL IF hasNoCorrespondingProperty(<correlationSet>@properties)
FOREACH $fromTo@property
    SET $variableType TO correspondingVariableType($fromTo)
    FAIL IF hasNoCorrespondingPropertyAlias($variableType, $fromTo@property)



hasNoCorrespondingProperty: <correlationSet>@properties -> boolean
    FOREACH wsdl:
        FOREACH wsdl:<property>
            IF wsdl:<property>@name EQUALS <correlationSet>@properties
                RETURN 'false'
    RETURN 'true'

correspondingVariableType: $fromToVar -> $variableType
    SET $scopeVariable to getScopeVariable($fromToVar, $fromToVar@variable)
    IF ($scopeVariable EQUALS <onEvent>)
        RETURN getOnEventVariableType($scopeVariable)
    ELSE
        RETURN getEnclosingScopeVariableType($scopeVariable)

getScopeVariable: $node, $variableName -> $scope
    IF (PARENT $node EQUALS <onEvent>)
        IF (<onEvent>@variable EQUALS $variableName)
            RETURN <onEvent>
    IF (PARENT $node EQUALS $scopes)
        FOREACH $node<variable>
            IF $node<variable>@name EQUALS $fromTo@variable
                RETURN $node<variable>
    ELSE getScopeVariable(PARENT $node, $variableName)

getOnEventVariableType: <onEvent> -> $type
    IF (<onEvent>@messageType EXISTS)
        RETURN <onEvent>@messageType
    IF (<onEvent>@element EXISTS)
        RETURN  <onEvent>@element
    ELSE getScopeVariable(PARENT <onEvent>

getEnclosingScopeVariableType: <variable> -> $type
    IF (<variable>@messageType EXISTS)
        RETURN <variable>@messageType
    IF (<variable>@type EXISTS)
        RETURN <variable>@type
    IF (<variable>@element EXISTS)
        RETURN <variable>@element
    fixme: onEvent (?) implicit variable definitions not checked...

hasNoCorrespondingPropertyAlias: $variableType, $fromToProperty  -> boolean
    FOREACH wsdl:<propertyAlias>
        IF (wsdl:<propertyAlias>@name EQUALS $fromToProperty@name)
            IF (<propertyAlias>@messageType EQUALS $variableType OR
                <propertyAlias>@type EQUALS $variableType OR
                <propertyAlias>@element EQUALS $variableType)
                    RETURN 'false'
    RETURN 'true'
```

- *Algorithm description:*
  SA00021 basically consists of two contentual parts:

  1. First, all `<correlationSet>`'s are checked to have corresponding `<property>` definitions in the WSDL, by matching the `properties` attribute with the `name`.

  2. Second, all present `<from>`'s and `<to>`'s are checked to have corresponding `<propertyAlias>`'es.

  The second part needs an intermediate step, matching the declared variable in use.

  The variable declaration can be found through three different ways, depending on the type of the enclosing scope:

  – If the enclosing scope is `<onEvent>` and has either a `messageType` or `element` attribute, the declaration is implicit, the matching works via the `variable`.
  – If the enclosing scope is `<onEvent>` and has neither a `messageType` nor a `element` attribute, the declaration is explicit, and the corresponding `<variable>@name` has to be found. This is done through recursive searching further up the tree for the next scope containing variable declaration, being implicit or explicit.
  – If the enclosing scope is either a `<scope>` or the `<process>` itself, the corresponding `<variable>` with matching `name` has to be found.

  When the corresponding variable type is found, from the corresponding WSDL file the matching `<propertyAlias>` is searched by its `name`. Comparison is made to the `name` of the `<to>` or `<from>`.

  If either the `type`, `messageType` or `element` attribute of the `<propertyAlias>` matches the found variable type, the BPEL process is valid according to SA00021.

- *Test case description(s):*
  CorrelationSet-Properties-Undefined.bpel:
  Contains `<correlationSet>@properties` which is not defined in any WSDL definitions.
  From-Properties-Undefined.bpel:
  Contains `<from>@property` which is not defined in any WSDL definitions.
  OnEvent-Properties-Undefined.bpel, OnEvent-Properties-Undefined.wsdl:

Contains `<eventHandlers><onEvent>@variable` which is not defined in any WSDL definitions
To-Properties-Undefined.bpel:
Contains `<to>@property` which is not defined in any WSDL definitions.

3.16 SA00022

- *Short description(s):*

    - two <propertyAlias> with same @propertyName and @type
    - two <propertyAlias> with same @propertyName and @element
    - two <propertyAlias> with same @propertyName and @messageType

- *SA specification:*
  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.

- *Algorithm:*

```
FOREACH <propertyAlias>
    FAIL IF DUPLICATE (@propertyName, @type)
    FAIL IF DUPLICATE (@propertyName, @element)
    FAIL IF DUPLICATE (@propertyName, @messageType)
```

- *Algorithm description:*
  According to SA00022, BPEL files containing at least one pair of `<propertyAlias>` elements containing the same `propertyName` and one of these three other attributes: `type`, `element` or `messageType`.

  Rule validation is ensured by a simple duplicate check.

- *Test case description(s):*
  Duplicate-propertyAliasElement.bpel, Duplicate-propertyAliasElement.wsdl:
  Duplicate-propertyAliasElement.wsdl file has two `<propertyAlias>` entries, where `@propertyName='tns:correlationId'` and `@element='tns:executeProcessSyncRequest'` are the same.
  Duplicate-propertyAliasMessageType.bpel,
  Duplicate-propertyAliasMessageType.wsdl:
  Duplicate-propertyAliasMessageType.wsdl file has two `<propertyAlias>` entries, where `@propertyName='tns:correlationId'` and `@messageType='tns:executeProcessSyncRequest'` are the same.
  Duplicate-propertyAliasType.bpel, Duplicate-propertyAliasType.wsdl:
  Duplicate-propertyAliasType.wsdl file has two `<propertyAlias>` entries, where `@propertyName='tns:correlationId'` and `@type='xsd:string'` are the same.

3.17 SA00023

- *Short description(s):*
  - variable@name must be unique in <process>
  - variable@name must be unique in <scope>
- *SA specification:*
  The name of a variable MUST be unique among the names of all variables defined within the same immediately enclosing scope.
- *Algorithm:*

```
FOREACH <process><variables><variable>
    FAIL IF DUPLICATE <variable>@name
FOREACH <scope>
    FOREACH <variables><variable>
        FAIL IF DUPLICATE <variable>@name
```

- *Algorithm description:*
  SA Rule 00023 implies that every scope (being either `<process>` itself or a sub `<scope>`) containing `<variable>`'s must have a set of them.

  This is ensured through checking for duplicate `name` attributes withing each `<variable>` set.
- *Test case description(s):*
  Process-Duplicated-Variables.bpel:
  Contains two `<variable>`'s with duplicated `@name='ReplyData'` within the `<process>`.
  Scope-Duplicated-Variables.bpel:
  Contains two `<variable>`'s with duplicated `@name='ReplyData'` within a `<scope>`.
  Scope-Scope-Duplicated-Variables.bpel:
  Contains two `<variable>`'s with duplicated `@name='replyData'` within an inner `<scope>`.

3.18 SA00024

- *Short description(s):*
  - <variable>@name or <onEvent>@variable contains illegal character '.'
- *SA specification:*
  Variable names are BPELVariableNames, that is, NCNames (as defined in XML Schema specification) but in addition they MUST NOT contain the '.' character.
- *Algorithm:*

```
FOREACH <onEvent>
    FAIL IF @variable CONTAINS '.'
FOREACH <variable>
    FAIL IF @name CONTAINS '.'
```

- *Algorithm description:*
  NCNames must not contain dots according to SA00024. This is checked by getting the values of each `variable` attribute from all present `<onEvent>` elements as well as all `name` attributes from all `<variable>` elements.

  These are tested for occurences of at least one dot.

- *Test case description(s):*
  Variable-containing-dot.bpel:
  Has `<variable>` where the value of `@name` has a dot within the string.
  OnEvent-containing-dot.bpel:
  Has `<onEvent>` where the value of `@name` has a dot within the string.

3.19 SA00025

- *Short description(s):*

  – @messageType or @type or @element in <variable> missing

  – @messageType and @type in <variable>

  – @messageType and @element in <variable>

  – @type and @element in <variable>

  – @messageType and @type and @element in <variable>

- *SA specification:*
  The messageType, type or element attributes are used to specify the type of a variable. Exactly one of these attributes MUST be used.

- *Algorithm:*

```
FOREACH <variable>
    IF (@messageType NOT EXISTS AND @type NOT EXISTS AND @element NOT EXISTS)
        FAIL TYPE 1
    IF (@messageType EXISTS AND @type EXISTS AND @element NOT EXISTS)
        FAIL TYPE 2
    IF (@messageType EXISTS AND @type NOT EXISTS AND @element EXISTS)
        FAIL TYPE 3
    IF (@messageType NOT EXISTS AND @type EXISTS AND @element EXISTS)
        FAIL TYPE 4
    IF (@messageType EXISTS AND @type EXISTS AND @element EXISTS)
        FAIL TYPE 5
```

- *Algorithm description:*
  SA00025 says that each `<variable>` must contain exactly one of these:
  `messageType`, `type`, or `element`.

  This is made sure through checks for occurences of two of the afore mentioned attributes all three, or the lack of all three.

- *Test case description(s):*
  Variable-havingMessageTypeAndElement.bpel:
  `<variable name='ReplyData'>` entry has `@messageType='ti:executeProcessSyncResponse'` and `@element='xs:attribute'`
  Variable-havingTypeAndElement.bpel:
  `<variable name='ReplyData'>` entry has `@type='xs:int'` and `@element='xs:attribute'`
  Variable-havingTypeAndMessageType.bpel:
  `<variable name='ReplyData'>` entry has `@messageType='ti:executeProcessSyncResponse'` and `@type='xs:int'`
  Variable-havingTypeAndMessageTypeAndElement.bpel:
  `<variable name='ReplyData'>` entry has `@messageType='ti:executeProcessSyncResponse'`, `@type='xs:int'` and `@element='xs:attribute'`
  Variable-missingMessageTypeAndTypeAndElement.bpel:
  `<variable name='ReplyData'/>` missing `@messageType`, `@type` and `@element`

3.20 SA00044

- *Short description(s):*

  - CorrelationSet name must be unique per scope

- *SA specification:*
  The name of a <correlationSet> MUST be unique among the names of all <correlation-Set> defined within the same immediately enclosing scope.

- *Algorithm:*

```
$enclosingScope = <process> | <scope>



FOREACH $enclosingScope
    FAIL IF DUPLICATE <correlationSet>@name
```

- *Algorithm description:*
  The name attribute of all `correlationSet` in a `scope` or `process` are inspected. The validation fails, if any name occurs twice or more often.

- *Test case description(s):*
  Process-CorrelationSet-Ambiguous.bpel:
  A BPEL-file which <process> is containing two CorrelationSet with same @name='CorrelationSet'
  Scope-CorrelationSets-Ambiguous.bpel
  A BPEL-file with a <scope> containing two CorrelationSet with same @name='CorrelationSet'

3.21 SA00045

- *Short description(s):*
  - <correlationSet>@properties is not a XML <simpleType>
- *SA specification:*
  Properties used in a <correlationSet> MUST be defined using XML Schema simple types.
- *Algorithm:*

```
FOREACH <correlationSet>
    SET $property TO getPropertyFromCorrelationSet(<correlationSet>)
    FAIL UNLESS $property@type EQUALS xsd:<simpleType>@name



getPropertyFromCorrelationSet: <correlationSet> -> wsdl:<property>
    <correlationSet> |=>
        IF <correlationSet>@properties EQUALS wsdl:<property>@name
    =>| RETURN wsdl:<property>
```

- *Algorithm description:*
  SA00045 states that all `@properties` from a `<correlationSet>` must have a corresponding definition in the referenced XSD through a `<simpleType>`.

  To navigate into the XML a detour through the WSDL over the corresponding `<property>` by its `type` attribute is needed.
- *Test case description(s):*
  Property-TypeMissing.wsdl, Property-TypeMissing.bpel:
  The `type` attribute in the WSDL `<property>` definition, which is used in the BPEL `<correlationSet>` is omitted.
  Property-TypeComplexType.wsdl, Property-TypeComplexType.bpel:
  The `type` attribute in the WSDL `<property>` definition, which is used in BPEL `<correlationSet>` is defined using `<complexType>`.

3.22 SA00046

- *Short description(s):*
  - There is no attribute ’pattern’ in <correlation> within <invoke>
  - Attribute ’pattern’ in <correlation> within <invoke> is not allowed
- *SA specification:*
  The pattern attribute used in <correlation> within <invoke> is required for request-response operations, and disallowed when a one-way operation is invoked.
- *Algorithm:*

```
FOREACH <invoke><correlation>
    SET $operation TO correspondingOperation(<invoke>)
    FAIL_noPatternInCorrelation IF (isRequestResponse($operation) AND (@pattern
        IS_MISSING))
    FAIL_patternDisallowed IF (isOneWay($operation) AND (@pattern EXISTS))

isRequestResponse: wsdl:<operation> -> boolean
    RETURN ((wsdl:<operation><input> EXISTS) AND (wsdl:<operation><input> BEFORE wsdl:<
        operation><output>))

isOneWay: wsdl:<operation> -> boolean
    RETURN ((<input> EXISTS) AND (<output> EXISTS NOT))

correspondingOperation: <invoke> -> wsdl:<portType><operation>
    SET $partnerLink TO correspondingPartnerlink(<invoke>)
    RETURN correspondingPortTypeOperation($partnerLink, <invoke>@operation)

correspondingPartnerlink: <invoke> -> <partnerLinks><partnerLink>
    FOREACH <partnerLinks>
        IF <partnerLinks><partnerLink>@name EQUALS <invoke>@partnerLink
            RETURN <partnerLinks><partnerLink>

correspondingPortTypeOperation(<partnerLink>, <invoke>@operation) -> wsdl:<portType><
    operation>
    <partnerLink>   |=>
        IF wsdl:<partnerLinkType>@name EQUALS <partnerLink>@partnerLinkType
    => wsdl:<partnerLinkType>
        IF wsdl:<partnerLinkType><role> EQUALS <partnerLink>@partnerRole
    => wsdl:<partnerLinkType><role>
        IF wsdl:<partnerLinkType><role>@portType EQUALS wsdl:<portType>@name
    => wsdl:<portType>
        IF wsdl:<portType><operation> EQUALS <invoke>@operation
    =>| RETURN wsdl:<portType><operation>
```

- *Algorithm description:*
  The rule SA00046 ensures that the `<correlation>` attribute pattern is used correctly depending on the WSDL *message exchange pattern* being used in the invoke `<activity>`.

  The *request-response message exchange pattern* is specified by a WSDL `<operation>` that contains first an `<input>` and then an `<output>` element. A WSDL `<operation>` with only an `<input>` element is stated as an *one-way message exchange pattern*.

  The algorithm searches for all `<correlation>` elements within an `<invoke>` activity and detects whether the corresponding WSDL `<operation>` of `<invoke>` uses a *request-response* or an *one-way message pattern*.

  In case of the *request-response pattern* the algorithm ensures that the `<correlation>` `pattern` attribute is not omitted and in the other case the algorithm ensures that the `<correlation>` `pattern` is omitted.

  Finding the corresponding WSDL `<operation>` used in the `<invoke>` activity is done by navigating to the `<partnerLink>` used in the <invoke> activity, then to the corresponding WSDL `<partnerLinkType><role>`, and to the `<portType>` and finally to the searched `<operation>`.

- *Test case description(s):*
  Invoke-RequestResponse-Correlation-PatternMissing.bpel:
  contains an <invoke> that uses a request-response operation but has no pattern attribute

in the <correlation> specification
Invoke-OneWay-Correlation-Pattern.bpel:
contains an <invoke> that uses an one-way operation and a request-response pattern in the <correlation> specification

3.23 SA00047

- *Short description(s):*
  - <toParts> or <fromParts> are not allowed
  - @variable or <fromParts> is missing
  - @variable or <toParts> is missing
  - @inputVariable or <toParts> for <invoke> is missing
  - @inputVariable/<toPart> and @outputVariable/<fromPart> for <invoke> is missing
- *SA specification:*
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.
- *Algorithm:*

```
$messageActivity = <receive>|<reply>|<invoke>|<onMessage>|<onEvent>



FOREACH $messageActivity
    IF hasNoCorrespondingMessagePart($messageActivity)
        FAIL_toPartsOrfromPartsNotAllowed IF ($messageActivity CONTAINS <toParts>) OR (
            $messageActivity CONTAINS <fromParts>)
    ELSE
        IF $messageActivity EQUALS <reply>
            FAIL_missingVariableToPart IF
                $messageActivity@variable IS_MISSING AND $messageActivity<toParts>
                    IS_MISSING

        IF $messageActivity EQUALS <receive>|<onMessage>|<onEvent>
            FAIL_missingVariableFromPart IF
                $messageActivity@variable IS_MISSING AND $messageActivity<fromParts>
                    IS_MISSING

        IF $messageActivity EQUALS <invoke>
            SET $operation TO correspondingOperation($messageActivity)
            IF isOneWay($operation)
                FAIL_inputVarOrToPartMissing IF
```

```
                    $messageActivity@inputVariable IS_MISSING AND $messageActivity<toParts> IS_MISSING
            IF isRequestResponse($operation)
                FAIL_toPartAndFromPartMissing IF (
                    $messageActivity@inputVariable IS_MISSING AND $messageActivity<toPart> IS_MISSING)
                    OR
                    ($messageActivity@outputVariable IS_MISSING AND $messageActivity<fromPart> IS_MISSING)



hasNoCorrespondingMessagePart($messageActivity) -> boolean
    SET $operation TO correspondingOperation($messageActivity)

    IF $messageActivity EQUALS <reply>
        SET $outputMessage TO correspondingMessage($operation, ’output’)
    IF $messageActivity EQUALS <receive>|<onMessage>|<onEvent>
        SET $inputMessage TO correspondingMessage($operation, ’input’)
    IF $messageActivity EQUALS <invoke>
        IF isOneWay($operation)
            SET $inputMessage TO correspondingMessage($operation, ’input’)
        IF isRequestResponse($operation)
            SET $inputMessage TO correspondingMessage($operation, ’input’)
            SET $outputMessage TO correspondingMessage($operation, ’output’)

    IF $inputMessage CONTAINS <part> OR $outputMessage CONTAINS <part>
        RETURN ’false’
    ELSE
        RETURN ’true’

correspondingMessage: $operation, $messageForm -> wsdl:<message>
    $operation |=>
        IF $operation$messageForm@message EQUALS wsdl:<message>@name
    RETURN wsdl:<message>

correspondingOperation: $messageActivity -> wsdl:<portType><operation>
    SET $partnerLink TO correspondingPartnerlink($messageActivity)
    =>| RETURN correspondingPortTypeOperation($partnerLink, $messageActivity@operation)

correspondingPartnerLink: $messageActivity -> <partnerLink>
    FOREACH <partnerLinks>
        IF <partnerLink>@name EQUALS $messageActivity@partnerLink
            RETURN <partnerLink>

correspondingPortTypeOperation(<partnerLink>, $messageActivity@operation) -> wsdl:<portType><operation>
    <partnerLink>   |=>
        IF wsdl:<partnerLinkType>@name EQUALS <partnerLink>@partnerLinkType
    => wsdl:<partnerLinkType>
        IF wsdl:<partnerLinkType><role> EQUALS <partnerLink>@partnerRole
    => wsdl:<partnerLinkType><role>
        IF wsdl:<partnerLinkType><role>@portType EQUALS wsdl:<portType>@name
    => wsdl:<portType>
        IF wsdl:<portType><operation> EQUALS $messageActivity@operation
    =>| RETURN wsdl:<portType><operation>

isRequestResponse: wsdl:<operation> -> boolean
    RETURN wsdl:<operation><input> EXISTS AND (wsdl:<operation><input> BEFORE wsdl:<operation><output>)

isOneWay: wsdl:<operation> -> boolean
    RETURN <input> EXISTS AND <output> EXISTS NOT
```

- *Algorithm description:*
  The rule *SA00047* ensures that variables associated with the WSDL message are used correctly within the following message activities: `<receive>`, `<reply>`, `<invoke>`, `<onMessage>`, `<onEvent>`.

  The `<invoke>` activity requires an `inputVariable` attribute or an `<toParts>` element, if the corresponding `<operation>` uses an *one-way message exchange pattern.* In case of a *request-response message exchange pattern* both `inputVariable` or `<toParts>` and `outputVariable` or `<fromParts>` are required. For more details about the *message exchange pattern*, see section 3.22.

  The `<receive>`, `<onMessage>` and `<onEvent>` activities require either a `variable` attribute or a `<fromParts>` element. The `<reply>` activity requires either a `variable` attribute or a `<toParts>` element.

  If the corresponding WSDL `<message>` contains only one `<part>` element, then both `<fromParts>` and `<toParts>` are disallowed within any of the above-mentioned activities. But there is no usage restriction to the `variable`, `inputVariable` and `outputVariable` attributes.

  First of all, the algorithm searches for the corresponding WSDL *input* and/or *output* `<message>` in each message activity. The algorithm looks for a WSDL input message if the current message activity is an `one-way` `<invoke>`, `<receive>`, `<onMessage>` or `<onEvent>` activity and for an *output message* if the message activity is a `<reply>` activity. In case of having a *request-response* *`<invoke>`* activity the algorithm searches for both input and output message.

  If the WSDL `<message>` found has only one part, then the algorithm failed if there is a `<toParts>` and/or a `<fromParts>` element within the message activity.

  Otherwise, the algorithm checks if the message activity is:

  – `<reply>` activity: It failed if both the variable attribute and the `<toParts>` element are missing.
  – `<receive>`, `<onMessage>` or `<onEvent>` activity: It failed if both the variable attribute and the `<fromParts>` element are missing.
  – *one-way* *`<invoke>`* activity: It failed if both the inputVariable and the `<toParts>` element are missing.
  – *response-request* *`<invoke>`* activity: It failed if both the inputVariable (or `<toParts>`) and the `outputVariable` (or `<fromParts>`) are missing.

- *Test case description(s):*
  EmptyMessage-Invoke-FromParts.bpel,
  TestPartner-MessageWithoutParts.wsdl:
  BPEL has `<invoke><fromParts>`, but the corresponding WSDL `<message>` definition does not contain any parts.
  EmptyMessage-Invoke-ToParts-FromParts.bpel,
  TestPartner-MessageWithoutParts.wsdl:
  BPEL has `<invoke><toParts>` and `<fromParts>`, but the corresponding WSDL `<message>` definition does not contain any parts.

EmptyMessage-Invoke-ToParts.bpel,
TestPartner-MessageWithoutParts.wsdl:
BPEL has `<invoke><toParts>`, but the corresponding WSDL `<message>` definition does not contain any parts.
EmptyMessage-OnEvent-FromParts.bpel,
TestInterface-MessageWithoutParts.wsdl:
Contains a `<onEvent>` with a `<fromParts>`, but corresponding WSDL `<message>` definition does not contain any parts.
EmptyMessage-OnMessage-FromParts.bpel,
TestInterface-MessageWithoutParts.wsdl:
Contains an `<onMessage>` with a `<fromParts>`, but corresponding WSDL `<message>` definition does not contain any parts.
EmptyMessage-Receive-FromParts.bpel,
TestInterface-MessageWithoutParts.wsdl:
Contains a `<reply>` with a `<fromParts>`, but corresponding WSDL `<message>` definition does not contain any parts.
EmptyMessage-Reply-ToParts.bpel,
TestInterface-MessageWithoutParts.wsdl:
Contains a `<reply>` with a `<toParts>`, but corresponding WSDL `<message>` definition does not contain any parts.
Invoke-OneWay-NoInputVariable-NoToParts.bpel:
One-way `<invoke>` operation with neither `@inputVariable` nor `<toPart>` defined.
Invoke-RequestResponse-NoInputOutputVariables-NoToFromParts.bpel:
Request-response `<invoke>` operation with neither `@inputVariable` and `@outputVariable` nor `<toParts>` or `<fromParts>` defined.
Invoke-RequestResponse-NoInputVariable-NoToParts.bpel:
Request-response `<invoke>` operation with neither `@inputVariable` nor `<toParts>` defined.
Invoke-RequestResponse-NoOutputVariable-NoFromParts.bpel:
Request-response `<invoke>` operation with neither `@outputVariable` nor `<fromParts>` defined.
NoVariable-NoFromPart-OnEvent.bpel:
The `<onMessage>` activity does contain neither `@variable` attribute nor `<fromParts>`.
NoVariable-NoFromPart-OnMessage.bpel:
The `<onMessage>` activity does contain neither `@variable` attribute nor `<fromParts>`.
NoVariable-FromPart-Receive.bpel:
The `<receive>` activity does contain neither `@variable` attribute nor `<fromParts>`.
NoVariable-NoToPart-Reply.bpel:
The `<reply>` activity does contain neither `@variable` attribute nor `<toParts>`.
NoVariable-NoToPart-NoFromPart-ReceiveReply.bpel:
The `<receive>` activity does contain neither `@variable` attribute nor `<fromParts>` and the `<reply>` neither `@variable` attribute nor `<toParts>`.

3.24 SA00048

- *Short description(s):*
  - @inputVariable messageType or type QName differs from the <operation>'s input <message> QName used in <invoke>
  - @outputVariable messageType or type QName differs from the <operation>'s output <message> QName used in <invoke>
- *SA specification:*
  When the optional inputVariable and outputVariable attributes are being used in an <invoke> activity, the variables referenced by inputVariable and outputVariable MUST be messageType variables whose QName matches the QName of the input and output message type used in the operation, respectively, except as follows: if the WSDL operation used in an <invoke> activity uses a message containing exactly one part which itself is defined using an element, then a variable of the same element type as used to define the part MAY be referenced by the inputVariable and outputVariable attributes respectively.
- *Algorithm:*

```



FOREACH <invoke>
    IF @inputVariable EXISTS
        SET $message TO correspondingMessages(<invoke>, 'input')
        SET $variable TO correspondingVariable(<invoke>, <invoke>@inputVariable)
        FAIL_inputVar UNLESS hasEqualMessageType($variable, $message)

    IF @outputVariable EXISTS
        SET $message TO correspondingMessages(<invoke>, 'output')
        SET $variable TO correspondingVariable(<invoke>, <invoke>@outputVariable)
        FAIL_outputVar UNLESS hasEqualMessageType($variable, $message)



correspondingMessage: $messageActivity, $messageForm -> wsdl:<message>
    correspondingOperation($messageActivity) |=>
        IF correspondingOperation($messageActivity)$messageForm@message EQUALS wsdl:<
            message>@name
    RETURN wsdl:<message>

correspondingVariable: <invoke>, $variableName -> <variable>@messageType
    FOR EARCH NEAREST <variables> OF <invoke>
        IF $variableName EQUALS $variables<variable>@name
            RETURN <variable>

hasEqualMessageType: $variable, $message -> 'boolean'
    IF $variable@messageType EXISTS
        SET variableMessage TO getVariableMessage($variable@messageType)
        IF equalsMessage(variableMessage, $message)
            RETRUN 'true'

    IF $variable@type EXISTS AND  $message<part> COUNT EQUALS '1' AND $message<part>
        @element EXISTS
        SET $messagePartType TO xsdElementType($message<part>@element)
        SET $variableType TO xsdType($variable@type)
        IF $messagePartType EQUALS $varibaleType AND equalsTargetNamespace(
            $messagePartType, $varibaleType)
            RETURN 'true'

    RETURN 'false'

equalsMessage: $variableMessage, $message -> 'boolean'
    RETURN equalsTargetNamespace($variableMessage, $message) AND ($message@name EQUALS
        variableMessage@name)

getVariableMessage: $QName
    IF wsdl:<message>@name EQUALS $QName QNAME_LOCALPART
        RETURN <message>

equalsTargetNamespace: $node1, $node2 -> 'boolean'
    IF $node1 ROOT@targetNamespace EUQALS $node2 ROOT@targetNamespace

correspondingOperation: $messageActivity -> wsdl:<portType><operation>
    SET $partnerLink TO correspondingPartnerlink($messageActivity)
    =>| RETURN correspondingPortTypeOperation($partnerLink, $messageActivity@operation)

correspondingPartnerlink: $messageActivity -> <partnerLinks><partnerLink>
    FOREACH <partnerLinks>
        IF <partnerLinks><partnerLink>@name EQUALS $messageActivity@partnerLink
             RETURN <partnerLinks><partnerLink>

correspondingPortTypeOperation(<partnerLink>, $messageActivity@operation) -> wsdl:<
    portType><operation>
    <partnerLink>   |=>
        IF wsdl:<partnerLinkType>@name EQUALS <partnerLink>@partnerLinkType
    => wsdl:<partnerLinkType>
        IF wsdl:<partnerLinkType><role> EQUALS <partnerLink>@partnerRole
    => wsdl:<partnerLinkType><role>
        IF wsdl:<partnerLinkType><role>@portType EQUALS wsdl:<portType>@name
    => wsdl:<portType>
        IF wsdl:<portType><operation> EQUALS $messageActivity@operation
    =>| RETURN wsdl:<portType><operation>

xsdType: $typeQName -> <simpleType>|<complexType>
  FOREACH xsd FILE
    IF <simpleType>@name EQUALS $typeQName QNAME_LOCALPART
        RETURN <simpleType>
    IF <complexType>@name EQUALS $typeQName QNAME_LOCALPART
         REURN <complexType>

xsdElementType: $elementQName -> <simpleType>|<complexType>
  FOREACH xsd FILE
    IF <element>@name AND <element>@type EXISTS
        RETURN xsdType(<element>@type)
```

- *Algorithm description:*
  The rule SA00048 ensures that the `inputVariable` and `outputVariable` within an `<invoke>` activity correctly correspond to their WSDL messages, i.e. the `<variable>` `messageType` *QName* matches the *QName* of the WSDL `<message>`.

  In case of having only one WSDL `<message>` `<part>`, the `<variable>`, that is used by the `inputVariable` or `outputVariable`, can have a type attribute that must correspond to the element type associated with the `<message>` `<part>`.

  If the `inputVariable` or `outputVariable` are being used, the regarding WSDL `<message>` must be an *input message* or an *output message*, respectively. The message used in the WSDL `<operation>` `<input>` is called *input message* and accordingly the *output message* is used in the `<operation>` `<output>`.

First of all, the algorithm searches for the corresponding WSDL *input* and/or `output` <*message*> in each `invoke` activity with an `inputVariable` and/or an `outputVariable` by finding the corresponding WSDL `<operation>` used in the `<invoke>` activity. This is done by navigating to the `<partnerLink>` used in the `<invoke>` activity, and to the corresponding WSDL `<partnerLinkType><role>`, and to the `<portType>` and finally to the searched `<operation>`. (Line 55 and 73)

Next, the algorithm searches for the `<variable>`. that is used in the `inputVariable` or `outputVariable` and that is nearest to the `<invoke>`, i.e is within the same *scope*.

If the `<variable>` has a `messageType` attribute then the algorithm ensures that one of the corresponding WSDL `<message>` elements found previously equal the `<message>` element used by this `messageType`. This is done by matching the `<message>` name and `messageType` *QName LocalPart* and the *targetNamespace* of both root elements.

Otherwise, if the `<variable>` uses a type attribute then the algorithm ensures that one of the corresponding WSDL messages has only one `<part>`, that has an attribute element which refers to an XML Schema Definition (XSD) `<element>`. The type of this XSD `<element>` equals the `<variable>` type attribute. Similar to the previous case the comparison is done by matching the `targetNamespace` of both element types.

- *Test case description(s):*
  InputVariable-MessageType-Message-NotFound.bpel:
  The messageType `variable` used in `invoke @inputVariable` does not correspond to the input `message` that is used in the `operation` specified in `invoke`
  InputVariable-Type-MessageOnePart-NotFound.bpel:
  The <invoke> @inputVariable references a type <variable> that has not the same element type as the corresponding single <part> <message> that is used in the <operation> specified in <invoke>.
  InputVariable-Type-MessageManyParts.bpel:
  The <invoke> @inputVariable references a type <variable> corresponds to an input <message> used in the <operation> specified in <invoke>. But the <message> has more than one <part>.
  OutputVariable-MessageType-Message-NotFound.bpel:
  The messageType `variable` used in `invoke @outputVariable` does not correspond to the output `message` that is used in the `operation` specified in `invoke`.
  OutputVariable-Type-MessageOnePart-NotFound.bpel:
  The <invoke> @outputVariable references a type <variable> that has not the same element type as the corresponding single <part> <message> that is used in the <operation> specified in <invoke>.
  OutputVariable-Type-MessageManyParts.bpel:
  The <invoke> @outputVariable references a type <variable> corresponds to an output <message> used in the <operation> specified in <invoke>. But the <message> has more than one <part>.
  InputOutputVariable-Message-NotFound.bpel:
  The messageType `variable` used in `invoke @outputVariable` and @outputVariable does

not correspond to the input `message` that is used in the `operation` specified in `invoke`.

3.25 SA00050

- *Short description(s):*
  - <toPart> element for an <invoke> / <reply> process is missing
- *SA specification:*
  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.
- *Algorithm:*

```
$outgoingOperation = <invoke> | <reply>



FOREACH $outgoingOperation
    IF $outgoingOperation<toParts> EXISTS
        SET $operation TO correspondingOperation($messageActivity)
        SET $message TO correspondingMessage($outgoingOperation, 'output')
        FOREACH $message<part>
            FAIL IF correspondingToPart($outgoingOperation, $message<part>) IS_MISSING



correspondingMessage: $operation, $messageForm -> wsdl:<message>
    $operation |=>
        IF $operation$messageForm@message EQUALS wsdl:<message>@name
    RETURN wsdl:<message>

correspondingOperation: $messageActivity -> wsdl:<portType><operation>
    SET $partnerLink TO correspondingPartnerlink($messageActivity)
    RETURN correspondingPortTypeOperation($partnerLink, $messageActivity@operation)

correspondingPartnerlink: $messageActivity -> <partnerLinks><partnerLink>
    FOREACH <partnerLinks>
        IF <partnerLinks><partnerLink>@name EQUALS $messageActivity@partnerLink
             RETURN <partnerLinks><partnerLink>

correspondingPortTypeOperation(<partnerLink>, $messageActivity@operation) -> wsdl:<
    portType><operation>
    <partnerLink>   |=>
        IF wsdl:<partnerLinkType>@name EQUALS <partnerLink>@partnerLinkType
    => wsdl:<partnerLinkType>
        IF wsdl:<partnerLinkType><role> EQUALS <partnerLink>@partnerRole
    => wsdl:<partnerLinkType><role>
        IF wsdl:<partnerLinkType><role>@portType EQUALS wsdl:<portType>@name
    => wsdl:<portType>
        IF wsdl:<portType><operation> EQUALS $messageActivity@operation
    =>| RETURN wsdl:<portType><operation>

correspondingToPart: $outgoingOperation, wsdl:<message><part> ->
```

```
40    wsdl:<message>  |=>
41        IF $outgoingOperation<toParts><toPart>@part EQUALS wsdl:<message><part>@name
42    =>| $outgoingOperation<toParts><toPart>
```

- *Algorithm description:*
  This algorithm is like an injective function from all `<part>`'s of a WSDL `<message>` to all the `<toPart>`'s of a outgoing message activity's `<toParts>` element. This SA, along with SA00054, states the amount of `<part>` and `<toPart>` are equal.

  To ensure this, the algorithm tries to match the `name` of each `<part>` element of the corresponding WSDL `<message>` with any `name` of the `<toParts><toPart>` of the `<reply>` or `<invoke>`. (Line 7-8)

  The `<message>` element in the WSDL is found through matching its `name` with the corresponding `message` of `<portType><operation>` in the same file. (Line 12 to 19)
  This is achieved utilizing the `<partnerLink>`.

  The `<partnerLink>` is found via matching the `partnerLink` of `<invoke>` or `<reply>` with each existing `name` of `<partnerLinks><partnerLink>`. (Line 21 to 24)

  The `<portType><operation>` itself in the WSDL is found through several steps:

  1. match `<partnerLink>` from the BPEL by `partnerLinkType` attribute with `<partnerlinkType>` from the WSDL by `name`
  2. match `<partnerLink>` by `partnerRole` with `<partnerLinkType><role>`
  3. get `<role>` from found `<partnerLink>`
  4. in the WSDL match `<partnerLink><role>` by `portType` with `<portType>` by `name`
  5. match `<portType><operation>` from the WSDL by `name` with `<invoke>` by `operation` from the BPEL

  (Line 26 to 35)

- *Test case description(s):*
  Invoke-MissingToPart.bpel, Invoke-MissingToPart.wsdl:
  One BPEL-file with a missing `<toPart>` definition for a `<message><part>` in an `<invoke>`.
  Receive-MissingToPart.bpel, Receive-MissingToPart.wsdl:
  One BPEL-file with a missing `<toPart>` definition for a `<message><part>` in an `<receive>`.

3.26 SA00051

- *Short description(s):*

  – <invoke> contains both @inputVariable and <toPart> element

- *SA specification:*
  The inputVariable attribute MUST NOT be used on an invoke activity that contains <toPart> elements.
- *Algorithm:*

```
FOREACH <invoke>
    FAIL IF <toParts> EXISTS AND @inputVariable EXISTS
```

- *Algorithm description:*
  According to SA00052, a BPEL file is not valid if it contains an `<invoke>` element with a `<toPart>` child element as well as a `inputVariable` attribute.

  The algorithm checks for the mere existence of the element and the attribute.
- *Test case description(s):*
  Invoke-ToPartsAndInputVariable.bpel:
  BPEL-file with a <invoke> containing a <toParts> as well as an @inputVariable

3.27 SA00052

- *Short description(s):*
  - <invoke> contains both @outputVariable and <fromPart> element
- *SA specification:*
  The outputVariable attribute MUST NOT be used on an <invoke> activity that contains a <fromPart> element.
- *Algorithm:*

```
FOREACH <invoke>
    FAIL IF <fromParts> EXISTS AND @outputVariable EXISTS
```

- *Algorithm description:*
  According to SA00052, a BPEL file is not valid if it contains an `<invoke>` element with a `<fromPart>` child element as well as a `outputVariable` attribute.

  The algorithm checks for the mere existence of the element and the attribute.
- *Test case description(s):*
  Invoke-FromPartsAndOutputVariable.bpel:
  BPEL-file with a <invoke> containing a <fromParts> as well as an @outputVariable

3.28 SA00053

- *Short description(s):*
  - Corresponding wsdl:<message><part> of <fromPart>@part is missing
- *SA specification:*
  For all <fromPart> elements the part attribute MUST reference a valid message part in the WSDL message for the operation.
- *Algorithm:*

```
 1 $messageAcivity = <invoke> | <receive> | <onMessage> | <onEvent>
 2
 3
 4
 5 FOREACH $messageAcivity<fromParts><fromPart>
 6     SET $operation TO correspondingOperation($messageActivity)
 7     SET $message TO correspondingMessage($operation, 'input')
 8     FAIL UNLESS (ANY $message<part>@name EQUALS <fromPart>@name)
 9
10
11 correspondingMessage: $operation, $messageForm -> wsdl:<message>
12     $operation |=>
13         IF $operation$messageForm@message EQUALS wsdl:<message>@name
14     RETURN wsdl:<message>
15
16 correspondingOperation: $messageActivity -> wsdl:<portType><operation>
17     SET $partnerLink TO correspondingPartnerlink($messageActivity)
18     =>| RETURN correspondingPortTypeOperation($partnerLink, $messageActivity@operation)
19
20 correspondingPartnerlink: $messageActivity -> <partnerLinks><partnerLink>
21     FOREACH <partnerLinks>
22         IF <partnerLinks><partnerLink>@name EQUALS $messageActivity@partnerLink
23              RETURN <partnerLinks><partnerLink>
24
25 correspondingPortTypeOperation(<partnerLink>, $messageActivity@operation) -> wsdl:<
       portType><operation>
26     <partnerLink>   |=>
27         IF wsdl:<partnerLinkType>@name EQUALS <partnerLink>@partnerLinkType
28     => wsdl:<partnerLinkType>
29         IF wsdl:<partnerLinkType><role> EQUALS <partnerLink>@partnerRole
30     => wsdl:<partnerLinkType><role>
31         IF wsdl:<partnerLinkType><role>@portType EQUALS wsdl:<portType>@name
32     => wsdl:<portType>
33         IF wsdl:<portType><operation> EQUALS $messageActivity@operation
34     =>| RETURN wsdl:<portType><operation>
```

- *Algorithm description:*
  SA00053 requires all `part`'s of all message activities' (`<invoke>`, `<reply>`, `<onMessage>`, `<onEvent>`) `<toParts><toPart>` elements in the bpel file to reference valid definitions in the wsdl file.

  To ensure this, the algorithm tries to match the `name`'s of `<message><part>` elements in the WSDL with all `name`'s of the `<toParts><toPart>`'s. (Line 7)

  The `<message>` element in the WSDL is found through matching its `name` with the corresponding `message` of `<portType><operation>` in the same file. (Line 11 to 18)
  This is achieved utilizing the `<partnerLink>`.

The `<partnerLink>` is found via matching the `partnerLink` of `<invoke>` or `<reply>` with each existing `name` of `<partnerLinks><partnerLink>`. (Line 20 to 23)

The `<portType><operation>` itself in the WSDL is found through several steps:

1. match `<partnerLink>` from the BPEL by `partnerLinkType` attribute with `<partnerlinkType>` from the WSDL by `name`
2. match `<partnerLink>` by `partnerRole` with `<partnerLinkType><role>`
3. get `<role>` from found `<partnerLink>`
4. in the WSDL match `<partnerLink><role>` by `portType` with `<portType>` by `name`
5. match `<portType><operation>` from the WSDL by `name` with `<invoke>` by `operation` from the BPEL

(Line 25 to 34)

- *Test case description(s):*
  Invoke-FromPartDifferingFromMessageDefinition.bpel:
  BPEL-file with different <invoke><fromParts><fromPart> as the required wsdl:<message><part> should be.
  OnEvent-FromPartDifferingFromMessageDefinition.bpel:
  BPEL-file with different <eventHandlers><onEvent><fromParts><fromPart> as the required wsdl:<message><part> should be.
  OnMessage-FromPartDifferingFromMessageDefinition.bpel:
  BPEL-file with different <pick><onMessage><fromParts><fromPart> as the required wsdl:<message><part> should be.
  Receive-FromPartDifferingFromMessageDefinition.bpel:
  BPEL-file with different <receive><fromParts><fromPart> as the required wsdl:<message><part> should be.

3.29 SA00054

- *Short description(s):*
  - Corresponding wsdl:<message><part> of <toPart>@part is missing
- *SA specification:*
  For all <toPart> elements the part attribute MUST reference a valid message part in the WSDL message for the operation.
- *Algorithm:*

```
1 $messageAcivity = <invoke> | <reply>
2
3
4
5 FOREACH $messageAcivity<toParts><toPart>
6     SET $operation TO correspondingOperation($messageActivity)
7     SET $message TO correspondingMessage($operation, 'output')
```

```
 8     FAIL UNLESS (ANY $message<part>@name EQUALS <toPart>@name)
 9
10
11
12 correspondingMessage: $operation, $messageForm -> wsdl:<message>
13     $operation |=>
14         IF $operation$messageForm@message EQUALS wsdl:<message>@name
15     RETURN wsdl:<message>
16
17 correspondingOperation: $messageActivity -> wsdl:<portType><operation>
18     SET $partnerLink TO correspondingPartnerlink($messageActivity)
19     RETURN correspondingPortTypeOperation($partnerLink, $messageActivity@operation)
20
21 correspondingPartnerlink: $messageActivity -> <partnerLinks><partnerLink>
22     FOREACH <partnerLinks>
23         IF <partnerLinks><partnerLink>@name EQUALS $messageActivity@partnerLink
24             RETURN <partnerLinks><partnerLink>
25
26 correspondingPortTypeOperation(<partnerLink>, $messageActivity@operation) -> wsdl:<
       portType><operation>
27     <partnerLink>   |=>
28         IF wsdl:<partnerLinkType>@name EQUALS <partnerLink>@partnerLinkType
29     => wsdl:<partnerLinkType>
30         IF PARENT wsdl:<partnerLinkType><role> EQUALS <partnerLink>@partnerRole
31     => wsdl:<partnerLinkType><role>
32         IF wsdl:<partnerLinkType><role>@portType EQUALS wsdl:<portType>@name
33     => wsdl:<portType>
34         IF wsdl:<portType><operation> EQUALS $messageActivity@operation
35     =>| RETURN wsdl:<portType><operation>
```

- *Algorithm description:*
  This algorithm is like an injective function from all the `<toPart>`'s of a outgoing message activity's `<toParts>` element to all `<part>`'s of a WSDL `<message>`. This SA, along with SA00050, states the amount of `<part>` and `<toPart>` are equal.

  To ensure this, the algorithm tries to match the `name`'s of `<message><part>` elements in the WSDL with all `name`'s of the `<invoke><toParts><toPart>` and `<reply><toParts><toPart>`. (Line 7)

  The `<message>` element in the WSDL is found through matching its `name` with the corresponding `message` of `<portType><operation>` in the same file. (Line 11 to 18)
  This is achieved utilizing the `<partnerLink>`.

  The `<partnerLink>` is found via matching the `partnerLink` of `<invoke>` or `<reply>` with each existing `name` of `<partnerLinks><partnerLink>`. (Line 20 to 23)

  The `<portType><operation>` itself in the WSDL is found through several steps:

  1. match `<partnerLink>` from the BPEL by `partnerLinkType` attribute with `<partnerlinkType>` from the WSDL by `name`
  2. match `<partnerLink>` by `partnerRole` with `<partnerLinkType><role>`
  3. get `<role>` from found `<partnerLink>`
  4. in the WSDL match `<partnerLink><role>` by `portType` with `<portType>` by `name`
  5. match `<portType><operation>` from the WSDL by `name` with `<invoke>` by `operation` from the BPEL

(Line 25 to 34)

- *Test case description(s):*
  Invoke-ToPartDifferingFromMessageDefinition.bpel:
  BPEL-file with different `<invoke><toParts><toPart>` as the required `wsdl:<message><part>` should be.
  Reply-ToPartDifferingFromMessageDefinition.bpel:
  BPEL-file with different `<reply><toParts><toPart>` as the required `wsdl:<message><part>` should be.

3.30 SA00055

- *Short description(s):*
  - <receive> contains both @variable attribute and <fromPart> element
- *SA specification:*
  For <receive>, if <fromPart> elements are used on a <receive> activity then the variable attribute MUST NOT be used on the same activity.
- *Algorithm:*

```
FOREACH <receive>
    FAIL IF ((<fromParts> EXISTS) AND (@variable EXISTS))
```

- *Algorithm description:*
  According to SA00055, a BPEL file is not valid if it contains `<receive>` element with a `<fromParts>` child element as well as a `variable` attribute.

  The algorithm checks for the mere existence of the element and the attribute.
- *Test case description(s):*
  Receive-WithFromPartElementAndVariableAttribute.bpel:
  BPEL-file with a <receive> containing a <fromParts> and @variable.

3.31 SA00059

- *Short description(s):*
  - <reply> contains both @variable attribute and <fromPart> element
- *SA specification:*
  For <reply>, if <toPart> elements are used on a <reply> activity then the variable attribute MUST NOT be used on the same activity.
- *Algorithm:*

```
FOREACH <reply>
    FAIL IF <toParts> EXISTS AND @variable EXISTS
```

- *Algorithm description:*
  According to SA00059, a BPEL file is not valid if it contains `<reply>` element with a `<toParts>` child element as well as a `variable` attribute.

  The algorithm checks for the mere existence of the element and the attribute.

- *Test case description(s):*
  Reply-WithToPartElementAndVariableAttribute.bpel
  BPEL-file with a `<reply>` containing a `<toParts>` and an `@variable`.

3.32 SA00063

- *Short description(s):*

  – <onMessage> contains both @variable attribute and <fromPart> element

- *SA specification:*
  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).

- *Algorithm:*

```
FOREACH <onMessage>
    FAIL IF <fromParts> EXISTS AND @variable EXISTS
```

- *Algorithm description:*
  According to SA00063, a BPEL file is not valid if it contains `<onMessage>` element with a `<fromParts>` child element as well as a `variable` attribute.

  The algorithm checks for the mere existence of the element and the attribute.

- *Test case description(s):*
  OnMessage-With-FromPartAndAttributeVariable.bpel:
  BPEL-file with a `<onMessage>` Event containing a `<fromParts>` and `@variable`.

3.33 Problems

3.33.1 Limitations

It is impossible for the tool to validate all processes that can be accepted by a certain BPEL-engine.

1. Containing a `location` attribute is required for `<import>` elements. Otherwise ISABEL could neither load nor inspect the files in question.

2. Each imported file need to state the type of itself in the default namespace. (E.g. a WSDL 1.1 file need to have `xmlns='http://schemas.xmlsoap.org/wsdl/'`) Such files would fail the SA00013Validator otherwise, but should pass the original SA.

3. ISABEL was never meant to validate usages of query and expression languages. Thus BPEL-function/XPATH-expression usages cannot be tested to work properly. If rules are dependent on the capabilities of the engine they are executed on, validating them was also out of scope during ISABEL development.
   The following rules could not be implemented, or only be partially:

   - SA00021 partial, cannot validate `getVariableProperty` BPEL function

   We had to completely ignore the following rules:

   - SA00004, being engine dependent
   - SA00009, being engine dependent
   - SA00026, concerns XPATH/BPEL functions/expressions
   - SA00027, concerns XPATH/BPEL functions/expressions
   - SA00028, concerns XPATH/BPEL functions/expressions
   - SA00029, concerns XPATH/BPEL functions/expressions
   - SA00030, concerns XPATH/BPEL functions/expressions
   - SA00031, concerns XPATH/BPEL functions/expressions
   - SA00033, concerns XPATH/BPEL functions/expressions
   - SA00039, concerns XPATH/BPEL functions/expressions
   - SA00040, concerns XPATH/BPEL functions/expressions
   - SA00041, concerns XPATH/BPEL functions/expressions

3.33.2 Detections

There is no SA00049 Rule defined in the whole BPEL-standard. Also no mention of what happened to the rule, or why it was removed.

3.33.3 Rules yet to be analyzed

Due to time restraints, the following 52 rules are yet to be analyzed, specificated and implemented:

- SA00014 to SA00019

- SA00032
- SA00034 to SA00038
- SA00042, SA00043
- SA00056 to SA00058
- SA00060 to SA00062
- SA00064 to SA00095

Chapter 4

ISABEL

4.1 Architecture

4.1.1 Folder structure

The structure of the project files is shown in figure 4.1.

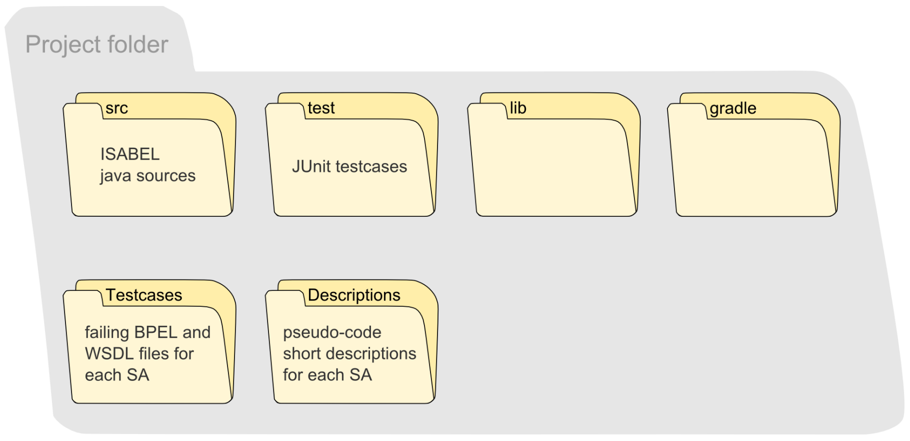

Figure 4.1: ISABEL structure

src The Java source for ISABEL is stored here. Internal package structure is at section 4.1.2 in full detail. This folder contains also one XML file produced with a Perl-script and its appropriate schema. There are two further XML schemas: One is the general schema for XML based files. The other is the schema definition for executable WSDL documents.

Testcases Each SA-valitator needs BPEL and WSDL files, that fail validation, to ensure correctness of the implementation. These are provided within this folder. The test cases are contained in folders, corresponding to the SA-rule, which they are designed to fail.

test This folder holds JUnit tests. Some of them are simple unit tests. But the major part tests the algorithm implementations with the files out of Testcases.

Descriptions The algorithm has a corresponding description in this folder, in a single file per implemented SA-rule. The SA-rules are written down with appropriate error messages within there, as well. Also, there are some PERL-scripts to form different presentation of these files. We used them to generate the program error output messages and the documentation.

lib The modified XOM Library lies here.

gradle Files, used by the gradle-scripts, are located here.

Project folder Within here are two gradle-scripts. One can be used on the Windows command-line. The other can be used in Windows shell emulation (cygwin). There is also a README, containing descriptions on how to use the scripts.

4.1.2 Package structure

There is no complex package structure (cf. figure 4.2), because this is a simple console application. The CommandLineInterpreter, a class to get useful parameters out of the input, as well as the XmlFileLoader belong to the *tool.imports* package. The XmlFileLoader loads all files of a BPEL process into DocumentEntry instances, which are finally stored in a storage class BpelProcessFiles. Later on we need the location of elements and attributes in a process file. The XmlFileLoader has a LocationAwareNodeFactory, for that reason.

The *tool.reports.print* package contains only the ValidationResultPrinter.

In *tool.reports* are several classes to collect the violations, which occur as Violation during the validation process. Violations are summarized within the IsabelViolationCollector, which is our implementation of the ViolationCollector interface. NavigationWarnings are currently unused. See the section future work 5 for planned usage.

The package *tool.validators* contains the 30 implemented validators (e.g. SA00001Validator, SA00020Validator, SA00047Validator, etc.). They are all subclasses of Validator and are handled via the ValidationHandler. Because a lot of methods exist which are required in several validators, we have two classes to containing them to avoid code duplication. The Validator-Navigator is a huge collection of navigation methods. (Here is further room for refactoring improvements. These did not take place due to time restraints.) The OperationHelper is a tiny class to check allowed message exchange patterns.

tool itself contains Isabel, the point of intersection to the application. The exceptions NavigationException and ValidationException are located here, as well, as the enum VerbosityLevel

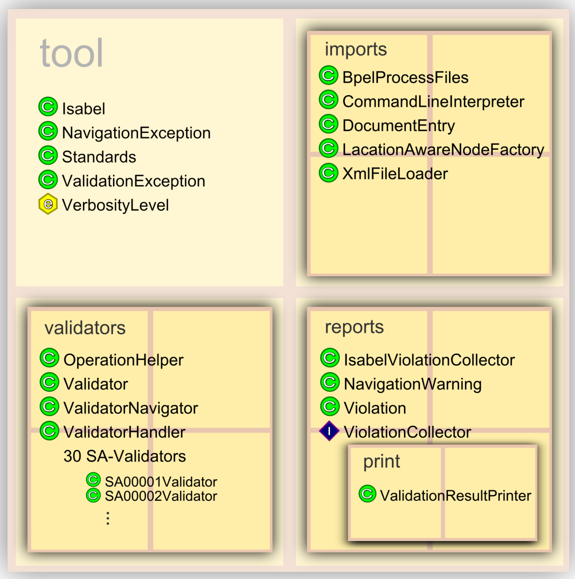

Figure 4.2: Structure of the packages

and the Standards class.

All these packages are locate in the root *de.uniba.wiai.dsg.ss12.isabel*. It contains a simple main method in IsabelTool.

4.1.3 Core class diagram

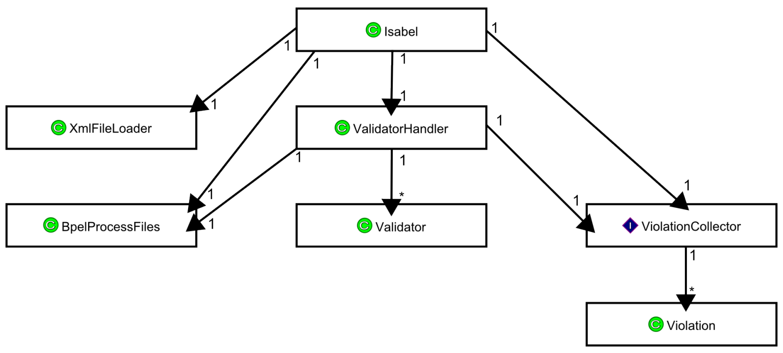

Figure 4.3: Class diagram of the ISABEL core

The class diagram should help to understand following section 4.1.4.

4.1.4 How does it work?

To illustrate the collaboration of the objects, we created a diagram showing the communication between them. The focus of objects within rectangles is on logic. Exceptions are shown in a circle. Ellipses are simple data-types or objects, that have a primary focus on data. The ellipses are given as parameters in functions or given back as the result of functions. This is indicated with the arrows. The arrows, start and end in Validator (blue and orange), may be used many times, depending on the validator and the test file.

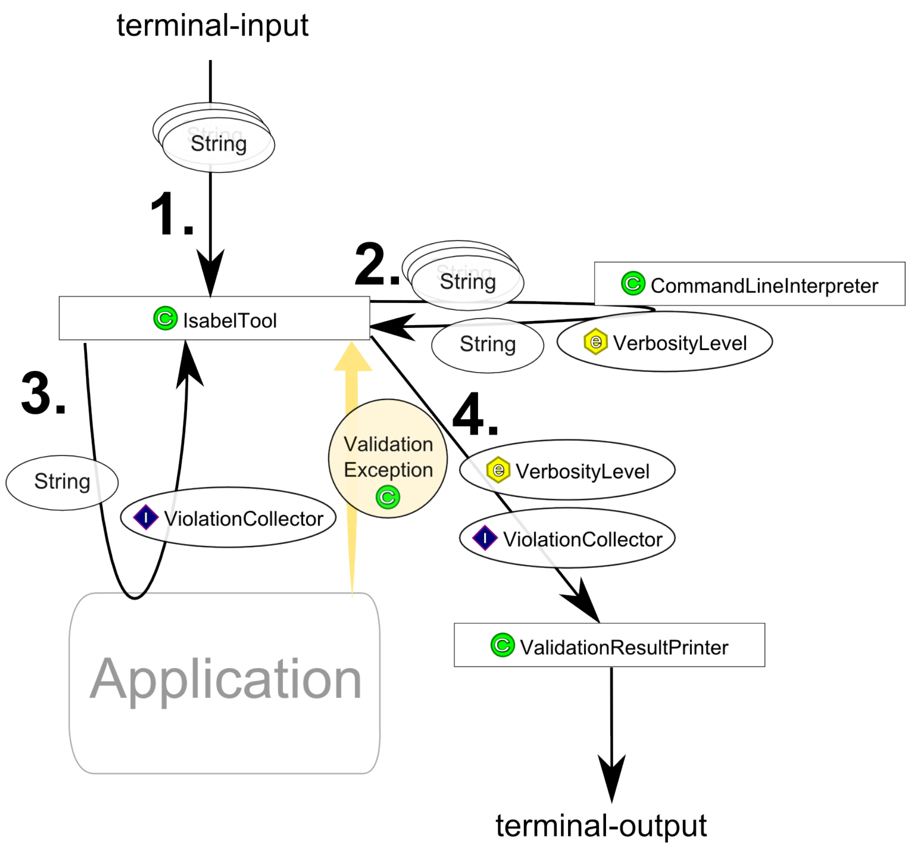

Figure 4.4: Parameter and return types of the user interface

The main function in IsabelTool is called with some String arguments. This String is given to the CommandLineInterpreter which transform the input to the path of the BPEL file as String and the VerbosityLevel. The path is given to the application for validation. After the validation a ViolationCollector is returned. Now the ValidationResultPrinter gets this ViolationCollector and the VerbosityLevel to print the output, which is the end of the tool execution. If the application can not run properly a ValidationException is thrown. An error messages is printed for the user and the tool terminates.

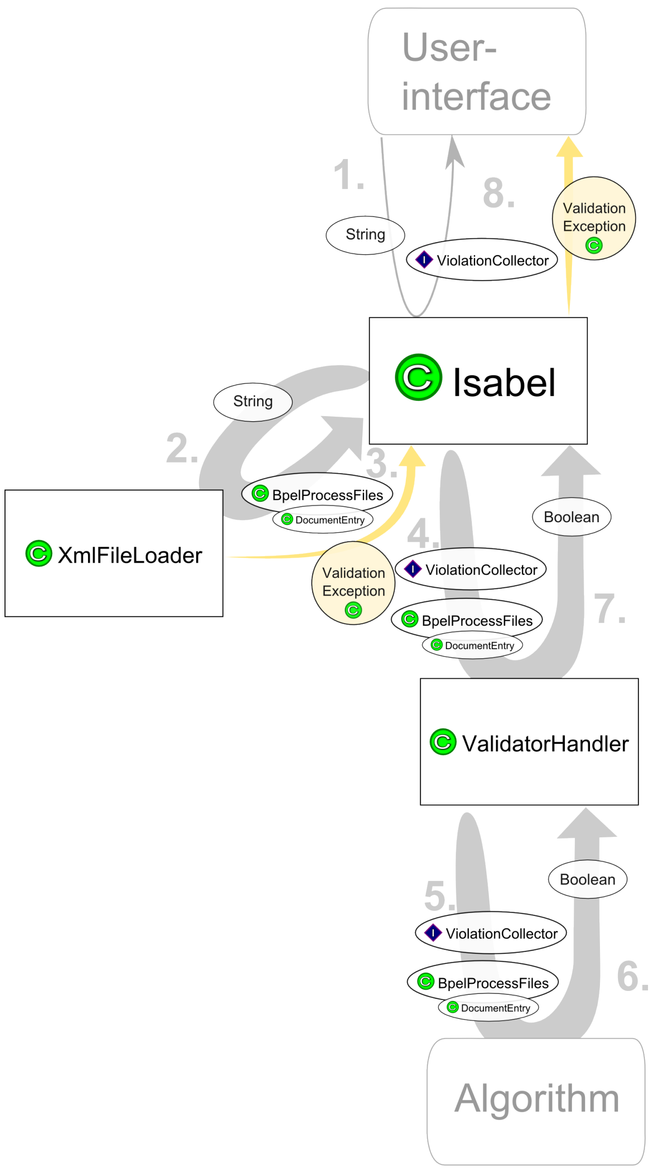

Figure 4.5: Parameter and return types of the core application

As soon as Isabel gets the BPEL file path, it is given to the XmlFileLoader. The loader loads the file from the given path. Then the BPEL file is traversed and every file from `<import>` is loaded as well. These files are traversed and imports are loaded recursively. Every file is packed into a DocumentEntry, which are summarized in BpelPocessFiles. The collection is returned to Isabel. If the XmlFileLoader fails to load an import a ValidationException is thrown.

After that, Isabel gives a created ViolationCollector (in our case an IsabelViolationCollector) and the BpelProcessFiles to the ValidatorHandler.
The handler distributes the parameter to each implemented algorithm.

We have implemented the validation algorithms for:

SA00001	SA00002	SA00003
SA00005	SA00006	SA00007
SA00008	SA00010	SA00011
SA00012	SA00013	SA00020
SA00021	SA00022	SA00023
SA00024	SA00025	SA00044
SA00045	SA00046	SA00047
SA00048	SA00050	SA00051
SA00052	SA00053	SA00054
SA00055	SA00059	SA00063

They return a boolean if the process is valid, according to them. The ValidationHandler returns the logical intersection of the results. Now Isabel returns the ViolationCollector to the user interface. Exceptions are rethrown to the user interface if something went wrong.

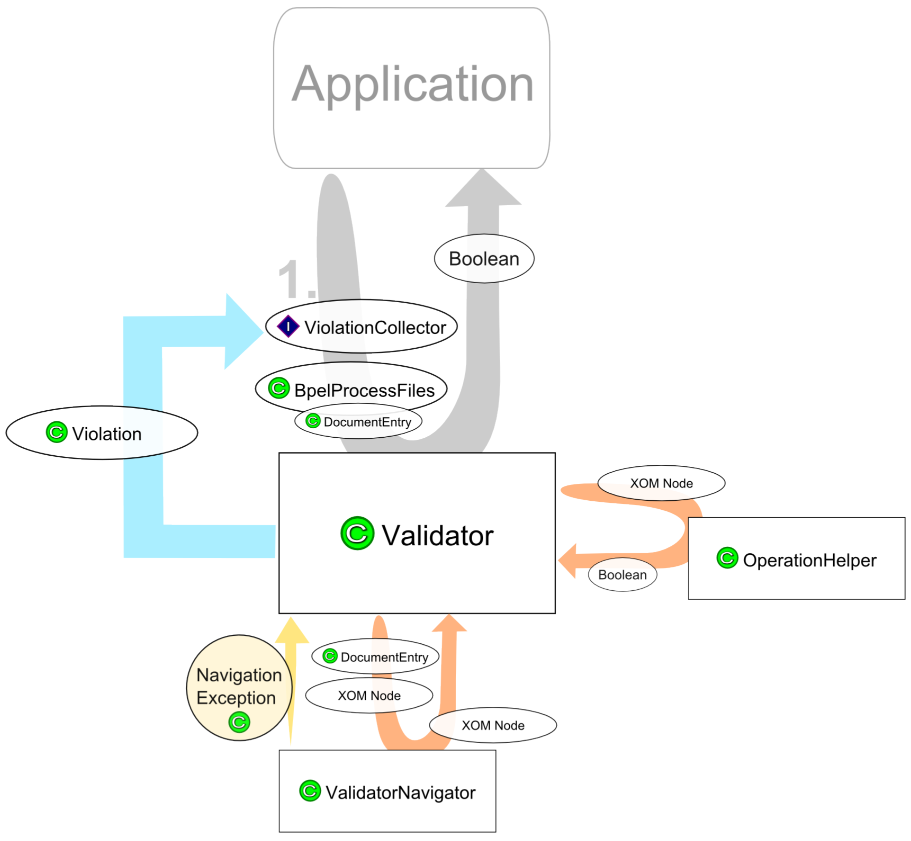

Figure 4.6: Parameter and return types of the algorithm implementation

This part is executed for every validator in its own way. For all validators a ViolationCollector and the BpelProcessFiles are forwarded. Also all violations of a SA-rule are reported to the ViolationCollector.

Some validators use functions of the ValidatorNavigator. These need various parameters out of String, XOM Nodes, DocumentEntrys or parts of it. The return values have similar signatures.

Or they may be a boolean. A common scenario is to navigate from one XOM Node to another. If an error occur during the navigation a NavigationException is thrown. The validators handle those exceptions. Few validators use the OperationHelper. Its functions are called with XOM Nodes and return booleans.

Finally a Validator subclass returns if the BPEL Process was valid, regarding its corresponding rule.

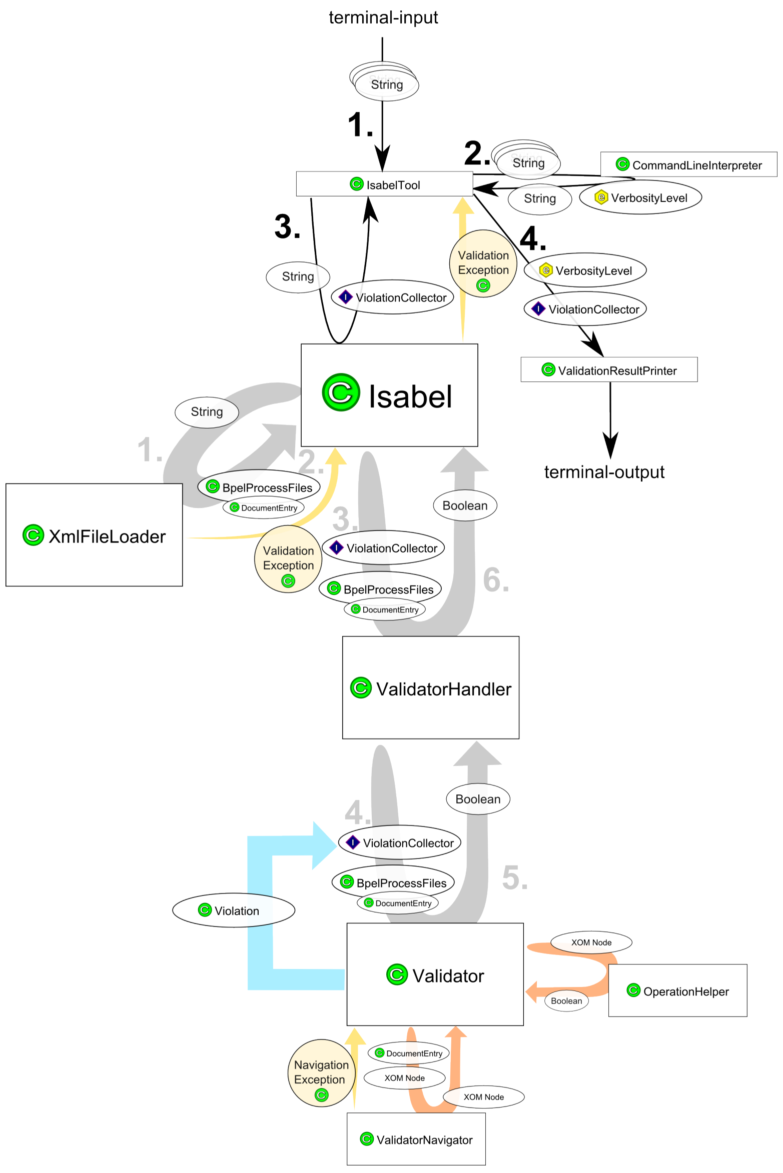

Figure 4.7: Parameter and return types overview

4.2 Isabel API

The Isabel API provides classes to validate an existing BPEL process file. The starting point of this API is the validate method of the Isabel class. This method loads all dependencies of the BPEL process file, initializes all the implemented validators and returns the results.

Listing 4.1: Methods of the `Isabel.java` class

```java
// Validates the file bpel and returns all violation results.
ViolationCollector validate(String bpel);
```

The code to validate a BPEL process is as simple as shown here:

Listing 4.2: How to call a validation

```java
Isabel isabel = new Isabel();
ViolationCollector validationResults = isabel.validate(
    path_to_bpel_file);
```

The `ViolationCollector` collects the results of rule violations detected by a validator. With the getResult method a list of all rule violations can be retrieved and, e.g, used to generate reports.

Listing 4.3: Methods of the ViolationCollector class

```java
// Add rule violation to the list of violations
void add(Violation violation);
void add(String bpelFileName, int ruleNumber, int type, int
    lineNumber, int columnNumber);
List<Violation> getResults();
```

A rule violation is described in the `Violation.java` class by means of the following properties:

`fileName`	The file name where the violation occurred.
`ruleNumber`	The contravened Static Analysis Fault Code (SA) number.
`type`	This property is used to classify the rule violation.
`row`	The line number of the file where the rule violation occurred.
`column`	The column where the rule violation was found.

Listing 4.4: Properties of `Violation.java`:

```java
public final int fileName;
public final int ruleNumber;
public final int type;
public final int row;
public final int column;
```

4.3 XOM

We used a slightly modified version of the original XOM library (`http://www.xom.nu/`), that already had Locator support in NodeFactories retrofitted. This was essential, else it would not have been possible to extract error locations. Line and column number where things went wrong during validation were made spottable.

We further added `foreach` support when working with `Node` Lists. The only change that was needed, was to let `Nodes.java` implement `Iterable<Node>`. Since we had to work with Node lists in almost every validator, this led to much a better code readability, cleared out the probability of off-by-one errors during the implementation phase and reduced development time.

These functionalities have not yet been added to the main development trunk of XOM. A current version of the library in use for ISABEL can be found at `github.com/uniba-dsg/XOM`.

4.4 User manual - How to use ISABEL

4.4.1 CLI, direct

Usage is possible from the commandline in the form of:

```
$ java -jar isabel.jar file.bpel [-f] [-v] [--full] [--verbose]
```

Only the .bpel file path has to be given as parameter, all needed and referenced files (WSDL, XSD) will be loaded automatically. When no further parameters are given, the output consists of error position and a short message.

Parameter order is not important, only the last used parameter is of importance.

Optional parameters:

`-f`
: Same as `-full`.

`--full`
: Output consists of the position of the error, a short specific message and the actual SA rule.

`-v`
: Same as `-verbose`.

`--verbose`
: The position of the error and the description of the actual SA rule are given back.

4.4.2 CLI, through gradle

```
$ gradlew run -Pargs='empty.bpel'  // validates the empty.bpel file directly
$ gradlew test                     // run all unit tests
$ gradlew javadoc                  // generate JavaDoc
$ gradlew eclipse                  // generate Eclipse project files
```
(under some circumstances it is required to fix the jdk import in the eclipse build path)

4.5 Known issues

Currently none known.

Chapter 5

Future work and conclusion

All SA-rules, which can not be validated yet but are possible to (i.e. they are not engine- or expression language specific), should be implemented to make ISABEL more potent. Rules containing each other (partially) should be grouped to get a clearer view on them. This can be done specification- as well as implementation-wise.

The currently unused class NavigationWarning should be put to use in the ViolationCollector when a NavigationException occurs during a validation-navigation to provide better user support. Also the navigation operations themselves could be refactored to be grouped in several classes summarizing similar or related actions. Similar validators have similar navigation needs in common when checking files during validation with ISABEL.

This could actually be quite helpful with finding out how to group SA rules:
Looking up which navigation classes the implemented validators need when validating BPEL processes. This should be easier than trying to group them just by having a look at the BPEL standard, because rules sometimes have very complex requirements. However misunderstandings of SA requirements often show up only *after* a partial or a complete implementation. When implementations are run against the already present test case suite and other validators' tests are failed, this very often indicates a wrongly analyzed rule. One definite fazit of this work is, that analysing rules or specificating algorithms alone is too error prone and absolutely not sufficient.

The class diagram in section 4.3 shows that the architecture needs a little refactoring. The ViolationCollector is not used within the Isabel class, except when returning. As well the BpelProcessFiles is just handed from the XmlFileLoader to the ValidationHandler (which, by the way, is not well named). These are unneeded dependencies indicate architectural design in need of improvement.

Analyzing the collected test data could be used to analyze dependencies among the SA-rules. Results could be visualized utilizing a graph to make rule dependencies clearer.

This could be helpful with future BPEL standards, in case new versions are declared. If a new standard were to be designed more carefully, logical loop holes could be avoided. (Logical corner cases are present in BPEL 2.0.) What are necessary features to reduce the need for query languages? Or it may be required to restrict query language usages to provide a larger freedom to choose any preferred engine.

Overall BPEL 2.0 seemed to be a pretty solid standard compared to others like BPMN from what we have seen so far and works decent. Some of the SA's would be obsolete, if the XML schema definition had been designed with more care. E.g. in one particular case it would have been very easy to restrict the typing of a variable through the XSD already. Instead of three optional attributes, the standardisation committee could have just put one choice element to use instead, making semantic testing like through our implementation unneeded. This was something we experienced more than once.

With ISABEL's development so far a solid foundation is set for further validation rule implementation. Concerning production usage, one third of all structural analysis rules can be checked to be fulfilled by the tested processes, independently from the engine that they are run on.

Since other ventures have never come this far and no known implementations exist to compare, the project, not to forget it being an explorative one, can be declared a success.