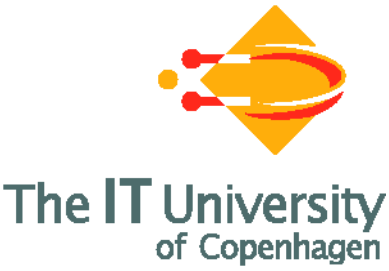

Core BPEL

Semantic Clarification of WS-BPEL 2.0 through Syntactic Simplification using XSL Transformations

Tim Hallwyl
Espen Højsgaard

IT University Technical Report Series **TR-2011-138**

ISSN 1600–6100 **March 2011**

ISSN 1600–6100

ISBN 978-87-794-9187-8

Copies may be obtained by contacting:

IT University of Copenhagen
Rued Langgaards Vej 7
DK-2300 Copenhagen S
Denmark

Telephone: **+45 72 18 50 00**
Telefax: **+45 72 18 50 01**
Web **`www.itu.dk`**

Core BPEL

Semantic Clarification of WS-BPEL 2.0 through Syntactic Simplification using XSL Transformations

Tim Hallwyl* Espen Højsgaard†

Abstract

The Web Services Business Process Execution Language (WS-BPEL) is a language for expressing business process behaviour based on web services. The language is intentionally not minimal but provides a rich set of constructs, allows omission of constructs by relying on defaults, and supports language extensions. Combined with the fact that the language definition does not provide a formal semantics, it is an arduous task to work formally with the language (e.g. to give an implementation).

In this paper we identify a core subset of the language, called Core BPEL, which has fewer and simpler constructs, does not allow omissions, and does not contain ignorable elements. We do so by identifying syntactic sugar, including default values, and ignorable elements in WS-BPEL. The analysis results in a translation from the full language to the core subset. Thus, we reduce the effort needed for working formally with WS-BPEL, as one, without loss of generality, need only consider the much simpler Core BPEL. This report may also be viewed as an addendum to the WS-BPEL standard specification, which clarifies the WS-BPEL syntax and presents the essential elements of the language in a more concise way.

To make the results of this work directly usable for practical purposes, we provide an XML Schema for Core BPEL and a set of XSLT 1.0 transformations that will transform any standard compliant WS-BPEL process into a Core BPEL process. We also provide an online service where one can apply the transformation.

This work is part of the initial considerations on the implementation of a WS-BPEL engine within the Computer Supported Mobile Adaptive Business Processes (CosmoBiz) research project at the IT University of Copenhagen.

*tiha@itu.dk
†espen@itu.dk

Contents

1 Introduction

The Web Services Business Process Execution Language (WS-BPEL) is a language for expressing business process behaviour based on web services. Based on the Extensible Markup Language (XML), the definition of WS-BPEL consists of two parts: A set of XML Schema documents[1], formally defining the syntax of the language, and a document [17], which we shall call the *standard specification*, defining the semantics and some further constraints on the syntax. The standard specification is written in prose and does not give any formal semantics for the language, which makes it harder than necessary to execute and reason about WS-BPEL processes as it is unclear what a WS-BPEL process means.

To remedy this, the academic community has proposed a number of formal semantics for WS-BPEL using a variety of formalisms (see related work below). These works have given a better understanding of WS-BPEL and have enabled formal analysis of WS-BPEL processes such as model checking. But the proposed formalizations are quite complex and comprehensive, which we believe stems from two sources: (a) WS-BPEL has some non-trivial features such as compensation and dead path elimination which are difficult to formalize concisely, and (b) WS-BPEL is (intentionally) not minimal, as for example the `<sequence>`-activity can be encoded as a `<flow>`-activity, and this redundancy carries over to the formalizations.

In this technical report, we investigate how we can remove some of the redundancy of WS-BPEL by showing how some constructs can be seen as *syntactic sugar* on a language core, which we term Core BPEL. By identifying a core language and a way to *desugar* the full language into the core subset, we can ease the tasks of implementing, analyzing, and formalizing the language, as one, without loss of generality, only needs to consider the core language. It should be noted that from a WS-BPEL programmer's perspective, features such as syntactic sugar are convenient and we do not suggest that WS-BPEL should be limited to a language core — we only aim to ease formal treatment of the language, and it is from this perspective that we in this report claim to *simplify* the WS-BPEL language. But we believe that the WS-BPEL user may also gain from this report, as it clarifies the WS-BPEL syntax and provides a more concise presentation of the essential elements of the language. From a more general perspective, this work also demonstrates that it is feasible to present a business process language as a set of primitive constructs from which a larger set of constructs may be built, and that this in fact promotes readability as well as clarifies and simplifies semantics. In particular, we expect a similar approach to be feasible and beneficial for the recent BPMN 2.0 standard [2].

The standard specification mentions many similarities between constructs and even defines some constructs by referring to the definition of other constructs. By analyzing these relations, we come up with desugaring transformations which preserve our understanding of the semantics, substantiated with quotes from the standard specification. Note that, in our analysis, we only consider executable processes, not the so-called abstract processes as these are not intended to be executed and may lack required operational details [17, Abstract]. Also, we disregard memory usage and execution speed, as these are implementation dependent and not treated by the standard.

Due to the lack of formal semantics in the standard specification we cannot prove our desugaring transformation to be semantics preserving in a formal sense; it might very well be the case, that our desugaring transformations do not preserve semantics with respect to formalizations given by the academic community or implementations of WS-BPEL. For instance, the memory usage of a syntactic sugar construct might be different from that of its desugared counterpart, which might lead to differences in exception behaviour. The only formal guarantee we give is that the WSDL types remain the same.

Core BPEL is thus a subset of WS-BPEL which has fewer and simpler constructs, and any valid executable WS-BPEL process can be transformed into an equivalent Core BPEL process by using the transformations presented in this report. We do not claim that the Core BPEL is minimal: there might be further or alternative transformations which yield a smaller language; nevertheless, our experience is that Core BPEL makes implementing and formalizing WS-BPEL significantly simpler.

In addition to syntactic sugar, WS-BPEL supports extensions that are implementation dependent and can be either optional or mandatory. Processes with mandatory extensions must be rejected if the imple-

[1] The XML Schema documents are published in appendix E of the WS-BPEL standard specification [17].

mentation does not support them [17, SA00009], whereas unsupported optional extensions must be ignored by an implementation. Ignoring an extension can be done by removing it from the process before execution, and by providing a generic set of transformations for this, we again ease the implementation task. We therefore examine how optional extensions can be removed from a process by transforming it.

We have implemented the presented transformations as XSLT 1.0 templates. Some of the XSLT templates require access to the Web Service Definition Language (WSDL) files imported by the process.

In the appendix, we list syntax summaries (in the style of the WS-BPEL standard specification) for the WS-BPEL and Core BPEL elements side-by-side for easy comparison (App. A). We also provide an XML Schema which defines the Core BPEL syntax (App. B) and a set of XSLT 1.0 templates which implement the translation from WS-BPEL to Core BPEL (App. D). The schema and transformations are also available at the CosmoBiz project website `http://www.cosmobiz.org` where we also provide a simple web-interface for experimenting with the transformations.

Motivation This investigation is part of the Computer Supported Mobile Adaptive Business Processes (CosmoBiz) project [11], which aims to provide a fully formalized runtime engine for a WS-BPEL-like business process language extended to allow for mobile and adaptive processes. As an initial step towards this goal, we are formalizing and implementing a WS-BPEL engine. By identifying a simpler core language of WS-BPEL, we ease the formalization and implementation. As a by-product, this investigation exposes some of the more subtle aspects of WS-BPEL, thereby making the semantics of the many constructs in WS-BPEL clearer and giving an overview of the many default behaviours specified in the standard specification.

Core BPEL allows WS-BPEL researchers and implementers to work with a simpler and more structured language that is both easier to understand and to reason about — without loss of generality. By providing the necessary transformations and a convenient web-interface, we hope to make our results easily accessible and usable for anyone working with WS-BPEL.

Related work The work presented in this paper is the continuation of Simplified BPEL as proposed by Hallwyl in his master's thesis [10]. The transformation into Simplified BPEL focused on adding implicit activities and default values, to support a so-called "standard-driven" implementation, a conceptual implementation closely mapping the prose descriptions found in the standard specification to code constructs. That aim did not allow us to remove or replace activities from the language. Core BPEL takes this further, by removing, transforming, and replacing constructs, aiming for a core subset of the language.

We are unaware of any other work on identifying a language core for WS-BPEL, but WS-BPEL has received considerable attention from the academic community, resulting in a number of formalizations using different formalisms the most notable being Petri Nets [12, 13, 14], process calculi such as the π-calculus [15] and bigraphs [3], and Abstract State Machines [9, 7, 6, 8].

Structure of the paper We begin with some considerations in Section 2 about how to design and express the transformations from WS-BPEL to Core BPEL. In Section 3, we discuss how to make default attribute values and elements explicit. In Section 4 we discuss the standard attributes and elements of WS-BPEL which are allowed on all activities, and describe how these are only necessary on a few select activities. In Section 5 we identify syntactic sugar in WS-BPEL's activities and the `<process>` construct, and investigate how to desugar them one by one. In Section 6 we discuss the extensibility of WS-BPEL and investigate how to remove optional extensions. Finally, in Section 7, we discuss how to combine the individual transformations into an overall transformation of WS-BPEL processes into Core BPEL and Section 8 concludes the paper with future perspectives.

The appendix contains the syntax summaries for the Core BPEL elements side-by-side with the corresponding WS-BPEL syntax summaries for easy comparison (App. A), the XML Schema for Core BPEL (App. B), an example WS-BPEL process and its Core BPEL equivalent, and the XSLT transformations we have constructed (App. D).

We assume that the reader is familiar with WS-BPEL and XSLT.

2 Transformation Considerations

Before we start discussing how WS-BPEL can be simplified by transformations, we will discuss how we express such transformations and the design goals for our transformations. Also, we will discuss how concurrency must be taken into account when transforming activities, and how to handle the fact that WS-BPEL allows for very general and unconstrained language extensions, since it poses a challenge when we want to perform syntactic manipulations of WS-BPEL processes in general, without knowledge of specific extensions.

2.1 Language

XSLT has been chosen due to its widespread adoption and availability in many programming languages, making it easy to adopt our transformations in implementations. For historical reasons, we use XSLT version 1.0: when Hallwyl commenced his precursory work, XSLT 2.0 had not yet been widely adopted; also, WS-BPEL requires implementations to support XSLT 1.0 transformations [17, Sec. 8.3], and thus by using that version, the effort required by implementers to adopt our transformations is minimal. We are certain, though, that updating the transformations to XSLT 2.0 would make them more readable, but we leave this as an exercise for the reader.

XSLT has the drawback of being somewhat informally specified and untyped leading to inconsistencies between implementations as well as poor help for catching type errors in our transformations. For those reasons, we considered languages like XDuce [18] and CDuce [4], but ended up deciding that the advantages of XSLT outweighs its disadvantages.

2.2 Design Goals

When designing our transformations, we have had three goals in mind:

idempotency: each transformation should only affect elements that are not already valid Core BPEL. In other words, applying the same transformation a second time should not alter the result.

This ensures that the transformations are as specific as possible.

independence: each transformation should be valid for any WS-BPEL process, and should not rely on any of the other transformations being applied before or after itself.

This will allow users to employ a subset of the transformations instead of the whole suite, if they find them useful on their own, though this will of course not yield a Core BPEL process.

simplicity: the transformations should be as readable as possible. Most importantly, this means that we do not require that transformations use Core BPEL elements in their results; for example, several transformations produce `<sequence>`s, which in Core BPEL are encoded as `<flow>`s.

One could worry, that the price for this would be that each transformation might have to be applied several times, and in the worst case that the transformation suite would not terminate. Fortunately, it turns out that there is a simple way to avoid this, which we shall discuss in Section 7.

Also, we have put no effort into optimizing the execution of the transformations in order to keep them as simple as possible.

We believe that we have achieved these goals, except in one case: we have chosen to let the transformations that make default attribute values explicit (cf. Section 3.1) be more general than necessary: they make *all* WS-BPEL default attribute values explicit, though some of these are unnecessary and indeed not a part of Core BPEL, and therefore has to be removed in a separate step. We have made this choice, as we believe those transformations are of general interest and value in their general form, independently of Core BPEL.

2.3 Concurrency Considerations

Since WS-BPEL allows concurrency, e.g. using `<flow>`, we must ensure that our transformations do not alter the concurrency semantics as specified by the standard specification. In particular, since we are concerned with the elimination of syntactic sugar by transforming sugared activities into a composition of more primitive activities, one could fear that we might break atomicity.

The standard specification only puts a few requirements on the concurrent execution of activities: `<assign>` and initialization of correlation sets are required to be atomic:

> "If there is any fault during the execution of an assignment activity the destination variables MUST be left unchanged, as they were at the start of the activity (as if the assign activity were atomic). This applies regardless of the number of assignment elements within the overall assignment activity.
>
> The assign activity MUST be executed as if, for the duration of its execution, it was the only activity in the process being executed." *[17, Sec. 8.4]*

> "However, the initiation of a correlation set is performed in an atomic fashion – in the same sense as that of an `<assign>` operation – ensuring that the correlation set will not be partially initiated." *[17, Sec. 12.8]*

The execution of other activities is not required to be atomic. Thus, as long as our transformations do not split assignments or correlation set initializations into several activities, we do not break any atomicity requirements.

2.4 WS-BPEL Extensibility

WS-BPEL allows extensions in the form of namespace qualified attributes and elements, respectively on and in WS-BPEL elements. When transforming a WS-BPEL element, we cannot in general anticipate which of the resulting WS-BPEL elements each of the extensions relate to, nor whether a given extension still makes sense on the transformed element. For example, when we transform a `<receive>` into a `<pick>` with a single `<onMessage>` (cf. Section 5.3), some extensions might relate to the activity (`<pick>`) whereas others might relate to the messaging element (`<onMessage>`).

Thus, we cannot ensure that the transformations we construct will preserve semantics for extensions nor that the extension instances will be properly placed. And such considerations are out of scope for this report, as our interests are the WS-BPEL language constructs defined by the standard. Note though, that no matter where we place the extension instances, the result will still be valid WS-BPEL. In keeping with our design goal of simplicity, we therefore place the extension instances wherever it is simplest in the XSLT code to put them. If this has unfortunate consequences for specific extensions, the transformations will have to be modified to handle those extensions explicitly.

3 Default Values and Elements

Having optional declarations and defaults means that something is stated when saying nothing. This is a special case of syntactic sugar, where the absence of some syntactic element should be interpreted the same as the presence of a particular instance of that syntactic element. As in the general case of syntactic sugar, making these implicit values explicit makes the language simpler without loosing expressivity.

In this section we briefly discuss the default values in WS-BPEL. For easy reference, they are listed in Table 1 and Table 2.

3.1 Default Attribute Values

We have divided the default attribute values into three groups as follows:

simple defaults: the default value is independent of the context of the construct

global defaults: there is a default value at the `<process>` level, and other constructs default to the value at that level

inherited defaults: there is a default value at the `<process>` level, and other constructs default to the value from the nearest enclosing element with the same attribute

Note that [17, Appendix C] contains a table of default values for attributes. But that table is missing the `ignoreMissingFromData` attribute on the `<copy>` element; it includes the attributes `keepSrcElementName` on `<copy>` and `initializePartnerRole` on `<partnerLink>` which, as argued in Sec. 3.1.1, cannot always be made explicit; and it for some reason also lists some attributes that do not have default values: `location` and `namespace` on `<import>` and `reference-scheme` on `<sfref:service-ref>`.

3.1.1 Simple Defaults

XSLT template: Appendix D.5

There are six optional attributes with default value `"no"` that are specific to the construct that they are part of. These are:

- `createInstance` on `<pick>` and `<receive>` [17, Sec. 11.5, 10.4].
- `ignoreMissingFromData` on `<copy>` [17, Sec. 8.4].
- `initiate` on `<correlation>` [17, Sec. 9.2].
- `isolated` on `<scope>` [17, Sec. 12.8].
- `succesfulBranchesOnly` on `<branches>` [17, Sec. 11.7].
- `validate` on `<assign>` [17, Sec. 8.4].

The keepSrcElementName and initializePartnerRole attributes The `keepSrcElementName` attribute on `<copy>` and `initializePartnerRole` on `<partnerLink>` both have default value `"no"` in [17, Appendix C] and in the case of `keepSrcElementName` the default value is even specified in the XML Schema for WS-BPEL. The default values do not always apply though:

> "An optional `keepSrcElementName` attribute is provided to further refine the behavior. [SA00042] It is only applicable when the results of both from-spec and to-spec are EIIs, and MUST NOT be explicitly set in other cases." *[17, Sec. 8.4.2]*

> "[SA00017] The `initializePartnerRole` attribute MUST NOT be used on a partner link that does not have a partner role; this restriction MUST be statically enforced." *[17, Sec. 6.2]*

Attributes	Default Value	Where
createInstance	no	<pick> <receive>
exitOnStandardFault	no	<process>
	inherited from immediately enclosing scope or process	<scope>
expressionLanguage	*xpath*	<process>
	inherited from <process>	<branches> <condition> <finalCounterValue> <for> <from> <joinCondition> <repeatEvery> <startCounterValue> <to> <transitionCondition> <until>
ignoreMissingFromData	no	<copy>
initiate	no	<correlation>
isolated	no	<scope>
messageExchange	the default <messageExchange> of the closest relevant parallel <forEach>, <onEvent>, or <process>	<onEvent> <onMessage> <receive> <reply>
name	a fresh name	<assign>, <compensate>, <compensateScope>, <empty>, <exit>, <flow>, <forEach>, <if>, <invoke>, <pick>, <receive>, <repeatUntil>, <reply>, <rethrow>, <scope>, <sequence>, <throw>, <validate>, <wait>, <while>
queryLanguage	*xpath*	<process>
	inherited from <process>	<query>
successfulBranchesOnly	no	<branches>
suppressJoinFailure	no	<process>
	inherited from immediately enclosing activity or process	all activities
validate	no	<assign>

Table 1: Default attribute values. Note that *xpath* is an abbreviation for urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0.

Elements	Default Value	Where

`<compensationHandler>`	`<compensationHandler>`
	`  <compensate />`
	`</compensationHandler>`
		`<scope>`

`<completionCondition>`	`<completionCondition />`	`<forEach>`

`<faultHandlers>`	`<faultHandlers>`
	`  <catchAll>`
	`  <sequence>`
	`  <compensate />`
	`  <rethrow />`
	`  </sequence>`
	`  </catchAll>`
	`</faultHandlers>`
		`<scope>`

`<joinCondition>`	`<joinCondition`
	`  expressionLanguage="`*xpath*`">`
	disjunction of links
	`</joinCondition>`
		`<target>`

`<messageExchange>`	`<messageExchange name="`*fresh name*`"/>`	`<process>`, child `<scope>` of `<onEvent>`, parallel `<forEach>`

`<terminationHandler>`	`<terminationHandler>`
	`  <compensate />`
	`</terminationHandler>`
		`<scope>`

`<transitionCondition>`	`<transitionCondition`
	`  expressionLanguage="`*xpath*`">`
	`  true()`
	`</transitionCondition>`
		`<source>`

Table 2: Default elements. Note that *xpath* is an abbreviation for `urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0`.

Thus, the two are attributes are explicitly forbidden from being made explicit in certain cases. We guess that the authors of the WS-BPEL standard specification only intend the default values to apply *if* the attributes may be legally specified; but in that case, specifying a default value for the attribute in the XML Schema is not in keeping with the XML Schema standard specification:

> "*default* specifies that the attribute is to appear unconditionally in the post-schema-validation infoset, with the supplied value used whenever the attribute is not actually present" *[19, Sec. 3.2.1]*

We therefore choose not to have the default values in the Core BPEL schema, as they are not true defaults in the XML Schema sense.

One could make the default values explicit in the cases where they apply, but this would require more complex analyses than we wish to provide with our transformations, and we believe the value of doing so would be limited, as the constructs would not in general become simpler.

3.1.2 Global Defaults

XSLT template: Appendix D.3

The languages for expressions and queries can be specified using the optional `expressionLanguage` and `queryLanguage` attributes on relevant elements. The default for both attributes on `<process>` is `urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0`, while on other activities, the attributes by default inherit their values from `<process>` [17, Sec. 5.2, 8.2].

> "The value of the `queryLanguage` and `expressionLanguage` attributes on the `<process>` element are global defaults and can be overridden on specific constructs, such as `<condition>` of a `<while>` activity, as defined later in this specification." *[17, Sec. 5.2]*

3.1.3 Inherited Defaults

XSLT template: Appendix D.4

The optional attribute `suppressJoinFailure` is allowed on `<process>` and all activities. Its default value on `<process>` is `"no"`, while if it is unspecified on activities, it inherits its value from the nearest enclosing activity/`<process>` [17, Sec. 5.2]. Similarly, the optional attribute `exitOnStandardFault` is allowed on `<process>` and `<scope>`, with a default value of `"no"` on `<process>` while on `<scope>` it by default inherits the value from its nearest enclosing `<scope>`/`<process>` [17, Sec. 5.2].

3.2 Message Exchanges

XSLT template: Appendix D.8

When there is no ambiguity, WS-BPEL allows `messageExchange`'s to be omitted from IMAs[2] and `<reply>`s:

> "If the `messageExchange` attribute is not specified on an IMA or `<reply>` then the activity's `messageExchange` is automatically associated with a default `messageExchange` with no name. Default `messageExchange`'s are implicitly declared by the `<process>` and the immediate child scopes of `<onEvent>` and the parallel form of `<forEach>`. Other occurrences of `<scope>` activities do not provide a default `messageExchange`." *[17, Sec. 10.4.1]*

But the standard specification does not specify in which scope the default `messageExchange` associated with a given IMA or `<reply>` should be found! We see two possibilities for the choice of scope, but there may be more:

1. the nearest enclosing scope, with a default `messageExchange`, of the IMA or `<reply>`.
2. the nearest common enclosing scope, with a default `messageExchange`, of all the IMA's and `<reply>`s that may be part of the same message exchange at runtime.

[2] inbound message activities

The second possibility requires advanced analysis, and is probably undecidable in general. The first possibility seems more plausible to us, as it is in line with the common lexical scoping rules as well as the reasoning for having default `messageExchange`'s declared in the immediate child scopes of `<onEvent>` and the parallel form of `<forEach>`

> "For example each time an `<onEvent>` is executed (i.e. when a new message arrives for processing) it creates a new default `messageExchange` instance associated with each `<onEvent>` instance. This allows a request-response `<onEvent>` event handler to receive messages in parallel without faulting or explicitly specifying a `messageExchange`. Similarly it allows the use of `<receive>`-`<reply>` or `<onMessage>`-`<reply>` pairs in the parallel form of `<forEach>` without the need to explicitly specify a `messageExchange`." *[17, Sec. 10.4.1]*

Thus it seems that it should be the nearest enclosing scope which has a default `messageExchange`. Note that it is not truly the nearest *enclosing* scope in the case of `<onEvent>` (which is an IMA), since it is allowed to refer to its associated scope:

> "When the `messageExchange` attribute is explicitly specified, the resolution order of the message exchange referenced by `messageExchange` attribute MUST be first the associated scope and then the ancestor scopes." *[17, Sec. 10.7.1]*

Assuming this interpretation of default `messageExchange` resolution, we can make the default message exchanges explicit in the following way:

1. Add a new `<messageExchange>` to the `<process>` and to each of the immediate child scopes of `<onEvent>` and the parallel forms of `<forEach>`, each with a unique fresh name (see Sec. 4.1.1 for a discussion of fresh names).

2. For all IMA's or `<reply>`s where the `messageExchange` attribute is unspecified, set the attribute's value to the name of the default `<messageExchange>` of the nearest enclosing scope of the kind listed in 1.

This is the interpretation that we implement, but we acknowledge that other interpretations are possible, as the standard specification is severely lacking on the subject. Note though, that the Core BPEL syntax remains the same regardless of which interpretation one chooses, as long as it can be decided statically.

3.3 Default Handlers

XSLT template: Appendix D.7

When either no fault, compensation, or termination handlers are specified in a `<scope>` activity, default handlers apply [17, Sec. 12.5.1]. These default handlers are shown in Table 2.

3.4 Default Join, Transition, and Completion Conditions

XSLT template: Appendix D.6

The constructs `<targets>`, `<source>`, and `<forEach>` all have optional condition elements:

`<joinCondition>`

> "If no `<joinCondition>` is specified, the `<joinCondition>` is the disjunction (i.e. a logical OR operation) of the link status of all incoming links of this activity." *[17, Sec. 11.6.1]*

As links are statically declared — their source and target activities do not change during execution — we can explicitly declare the default join condition as the disjunction of the link statuses using an XPath expression.

`<transitionCondition>`

> "If the `<transitionCondition>` is omitted, it is assumed to evaluate to true." *[17, Sec. 11.6.1]*

This allows us to declare the default `<transitionCondition>` as the XPath expression "`true()`".

`<completionCondition>`

> "When a `<completionCondition>` does not have any sub-elements or attributes understood by the WS-BPEL processor, it MUST be treated as if the `<completionCondition>` does not exist." *[17, Sec. 11.7]*

Thus the absence of a `<completionCondition>` is equivalent to `<completionCondition />`.

4 Standard Attributes and Elements

XSLT template: Appendix D.23

All WS-BPEL activities have two optional attributes and two optional elements, which in the standard specification are referred to as `standard-attributes` and `standard-elements` respectively, cf. the syntax summaries in Listing 4 and Listing 5 respectively. Making the default values explicit, as discussed in the previous section, simplifies the standard attributes and elements somewhat, but there is no reason to have these attributes and elements on all activities: we might as well move them to a wrapping `<flow>`, thereby restricting the syntax of Core BPEL further. In the following two sections, we discuss how this may be achieved.

4.1 Activity Names

First, we will discuss the standard attribute `name`. It is only in the case of `<scope>` that the value of `name` has any operational semantics specified by the standard (named scopes can be referenced by `<compensateScope>`), but it is also possible that in some cases, the name of an `<extensionActivity>` might be significant; this is not discussed in the standard. For all other activities, we can safely either remove the name or move it to a wrapping `<flow>`; the wrapper cannot be a `<scope>`, since that could break the following requirement in some cases:

> "[SA00092]Within a scope, the name of all named immediately enclosed scopes MUST be unique. This requirement MUST be statically enforced." *[17, Sec. 12.4.3]*

For example, the WS-BPEL fragment in Listing 1 can be transformed into the equivalent fragment in Listing 2 where the names are moved to a wrapping `<flow>`, but using `<scope>`s as wrappers, as in Listing 3, violates [SA00092], whereas a wrapping `<flow>` does not alter the semantics of any activity.

For Core BPEL, we remove the redundant cases of the `name` attribute, i.e. we remove it from all activities but `<scope>` and `<extensionActivity>`.

Now, there is no default value for the `name` attribute, but there is nothing preventing us from assigning an unused (fresh) name to unnamed `<scope>`s (but not `<extensionActivity>`). Doing so makes the `name` attribute on `<scope>` mandatory in Core BPEL.

4.1.1 A Note on Fresh Names

Several of our transformations create named entities, e.g. variables or links, and it is important that we choose names that do not conflict with other named entities (e.g. variable shadowing). Though one could for each transformation analyse exactly how the created named entities might conflict with other entities, we do not believe that such an analysis will bring any significant insight. Instead, we simply generate globally fresh names in the following way:

- before we apply the transformations, we generate a string that is not the prefix of any attribute value in the WS-BPEL process to be transformed; we call this the *fresh prefix*.
- when a fresh name is needed in a transformation, we construct it as the concatenation of the following three parts

  Fresh prefix: the fresh prefix generated above.

  Element id: using the XSLT XPath function *generate-id()* on an element that is being transformed, we obtain an id for that instance of the transformation. If a transformation needs more than one fresh name, it uses a separate element for each fresh name.

  Transformation postfix: a postfix that is unique for that transformation.

Listing 1: Two activities with the same name.

```
<sequence>
  <empty name="foo" />
  <empty name="foo" />
</sequence>
```

Listing 2: Named wrapper.

```
<sequence>
  <flow name="foo">
    <empty />
  </flow>
  <flow name="foo">
    <empty />
  </flow>
</sequence>
```

Listing 3: Scope name clash.

```
<sequence>
  <scope name="foo">
    <empty />
  </scope>
  <scope name="foo">
    <empty />
  </scope>
</sequence>
```

Listing 4: WS-BPEL `standard-attributes`.

```
name="NCName"?
suppressJoinFailure="yes|no"?
```

4.2 Link endpoints

The remaining standard attributes and elements (`suppressJoinFailure`, `<targets>`, and `<sources>`) all relate to link endpoints. As with the `name` attribute, these may simply be moved to a wrapping `<flow>` activity, even in the case of `<scope>` and `<extensionActivity>`.

There is one detail to be aware of though: some `<scope>`s occur in contexts (`<onEvent>` and `<onAlarm>` in `<eventHandlers>`, and `<forEach>`) where other activities are not allowed, and it would therefore seem that we cannot in general move `<targets>` and `<sources>` to a wrapping `<flow>`. But a consequence of the following statement in the standard specification is that those `<scope>`s may not be target nor source of any links:

> A link used within a repeatable construct (`<while>`, `<repeatUntil>`, `<forEach>`, `<eventHandlers>`) or a `<compensationHandler>` MUST be declared in a `<flow>` that is itself nested inside the repeatable construct or `<compensationHandler>`. *[17, Sec. 11.6.1]*

Thus, we *can* move `<targets>` and `<sources>` to a wrapping `<flow>` everywhere they occur, and we do this for Core BPEL.

The `<scope>`s just discussed may still have the `suppressJoinFailure` attribute, though. But since the `<scope>` is not the target of any links, this attribute only serves to set the default value for the child activities of the `<scope>`: cf. Sec. 3.1.3, `suppressJoinFailure` by default inherits its value from the closest enclosing activity. We can therefore remove this attribute from the `<scope>`s in question by propagating the value of the `suppressJoinFailure` attribute to the immediate child activities.

Now, since our transformation will have to propagate the value of `suppressJoinFailure` for the `<scope>`s just discussed, we might as well do the same for activities that have no `<targets>` as they do not need the `suppressJoinFailure` attribute; the transformation does not become more complex due to this and it will lead to fewer wrapping `<flow>`s. This is purely an optimization, which does not affect the syntax of Core BPEL.

Listing 5: WS-BPEL `standard-elements`.

```
<targets>?
  <joinCondition expressionLanguage="anyURI"?>?
    bool-expr
  </joinCondition>
  <target linkName="NCName" />+
</targets>
<sources>?
  <source linkName="NCName">+
    <transitionCondition
      expressionLanguage="anyURI"?>?
      bool-expr
    </transitionCondition>
  </source>
</sources>
```

5 Desugaring Constructs

WS-BPEL provides programmers with a set of shorthands for frequent code patterns, i.e. *syntactic sugar*. For instance, when receiving a message, one often wants to separate the message into its constituent parts to ease further processing; WS-BPEL allows the programmer to specify this data processing concisely as part of the `<receive>` activity, thereby avoiding a sequence of explicit assignments.

The WS-BPEL standard specification does only in a few cases clearly describe shorthands as derived constructs, but it hints at many such relations. We have gone through the standard specification looking for such hints, as well as classical cases of syntactic sugar, and in this section we discuss the cases of syntactic sugar we have identified and we analyse how to desugar them.

The analysis is organized around the elements that we have constructed desugaring transformations for: `<process>`, `<invoke>`, `<pick>`, `<receive>`, `<reply>`, `<scope>`, `<if>`, `<repeatUntil>`, and `<sequence>`. There is a section for each of these constructs, and each section is accompanied by (a) an XSLT template in the appendix which formalizes and implements the described desugaring transformation, and (b) the syntax summary for the element, in the style of the standard specification, as well as the corresponding Core BPEL syntax summary. These are placed in Appendix A and Appendix D, and the header for each section contains references to the appropriate parts of the appendices. Table 3 gives an overview of the desugaring transformations we have constructed.

The elements for which we have not constructed separate desugaring transformations, are listed in Section 5.10; these are also accompanied by their WS-BPEL and Core BPEL syntax summaries, which are not identical, since many optional attributes and elements are made mandatory.

5.1 `<process>`

WS-BPEL syntax summary:	Listing 34
Core BPEL syntax summary:	Listing 35
XSLT template:	Appendix D.14

The `<process>` element serves as the root `<scope>` in the scoped environment hierarchy, and is allowed to contain almost all of the attributes and elements that are also allowed in a `<scope>` element as evident from their syntax summaries (cf. Listing 34 and Listing 64), and according to the standard specification the semantics of those attributes and elements are the same:

> "The `<process>` and `<scope>` elements share syntax constructs, which have the same semantics. However, they do have the following differences:
>
> - The `<process>` construct is not an activity; hence, standard attributes and elements are

Element	Syntactic sugar description	Desugaring transformation
<if>	Allows omitting the <else> element and allows <elseif> elements.	If the <else> element is omitted, insert <else><empty/></else>. Transform the list of <elseif> elements into a nested sequence of <else><if> elements.
<invoke>	Allows local fault and compensation handlers and manipulation of input/output using implicit local variables and implicit assignments.	Make an immediately enclosing <scope> and move handlers to that. Declare the implicit variables explicitly in that <scope> and make the implicit assignments explicit.
<onEvent>	Allows manipulation of input using implicit local variables and implicit assignments.	The standard forbids declaring the variables explicitly, but we make the implicit assignments of each <onEvent> explicit.
<pick>	Allows manipulation of input using implicit local variables and implicit assignments.	Make an immediately enclosing <scope> and declare the implicit variables explicitly in that <scope> and make the implicit assignments of each <onMessage> explicit.
<process>	Is an implicit scope.	Make the <scope> explicit.
<receive>	Corresponds to a <pick> activity which contains only one <onMessage> element.	Replace by a <pick> activity.
<repeatUntil>	Provides conditional iteration, just as <while>, though <repeatUntil> always performs at least one iteration.	Replace by a <while> where the condition is the negation of a temporary variable initialized to *false* and the body is a sequence of the <repeatUntil>'s body followed by an assignment of the value of its condition to the temporary variable.
<reply>	Allows manipulation of output using implicit local variables and implicit assignments.	Make an immediately enclosing <scope> and declare the implicit variables explicitly in that <scope> and make the implicit assignments explicit.
<scope>	Allows in-line variable initialization corresponding to implicit <assign> activities.	Convert the in-line variable initializations to explicit <assign> activities. To preserve the all-or-nothing behaviour of scope-initialization, this requires some additional <scope>s (see Section 5.6 for details).
<sequence>	Provides sequential processing, which can also be achieved with the <flow> activity.	Replace by a <flow> activity with properly defined links between the child activities.

Table 3: Overview of syntactic sugar and transformations.

Listing 6: Process with scope related content.

```
<process>
  <variables>
    <variable name="size" type="xsd:int">
      <from><literal>42</literal></from>
    </variable>
  </variables>
  <catch ...>
    activity
  </catch>
  <catchAll>
    activity
  </catchAll>

  activity

</process>
```

Listing 7: Process with a scope as main activity.

```
<process>
  <scope>
    <variables>
      <variable name="size" type="xsd:int">
        <from><literal>42</literal></from>
      </variable>
    </variables>
    <catch ...>
      activity
    </catch>
    <catchAll>
      activity
    </catchAll>

    activity

  </scope>
</process>
```

not applicable to the <process> construct

- A compensation handler and a termination handler can not be attached to the <process> construct
- The isolated attribute is not applicable to the <process> construct (see section 12.8. Isolated Scopes)"

[17, Sec. 12]

Thus there is nothing preventing us from moving all <scope> related content from a <process> element into an explicitly defined <scope>. This includes the attributes suppressJoinFailure and exitOnStandardFault, which are made redundant for the <process> element by this transformation, since

suppressJoinFailure just serves to set an inherited default value, which is immediately overridden by the new <scope>, and

exitOnStandardFault the situation is the same, as any standard fault that may be trown, will be thrown inside of the new <scope>.

Listing 6 shows an example of <scope> related content in a <process> and Listing 7 shows how the scope is made explicit.

Also, note that two attributes expressionLanguage and queryLanguage serve only to specify default values for the activities of the process, so these can be safely removed after the defaults have been made explicit (i.e. after the transformation discussed in Section 3.1.2). As we shall discuss in Section 7, this is done as a separate transformation, in order to avoid problems when combining this transformation with the transformations of Section 3.1.

5.2 <invoke>

WS-BPEL syntax summary:	Listing 54
Core BPEL syntax summary:	Listing 55
XSLT template:	Appendix D.11

The principal purpose of the <invoke> activity is to invoke an operation offered by a partner. But the

standard specification allows the programmer to do more using this activity, e.g. to specify fault handling which is specific for this invocation.

We have identified three main cases of syntactic sugar for the <invoke> activity:

1. The local declaration of fault or compensation handlers implicitly declares an enclosing <scope> with these handlers.
2. The usage of <toParts> or <fromParts> to map variables to and from message parts implicitly declares an enclosing <scope> with temporary variables and assignments.
3. Referring to an element variable in either <inputVariable> or <outputVariable> also implicitly declares an enclosing <scope> with temporary variables and assignments.

A single <invoke> activity can match all three of these cases, but we examine each of them separately and then, at the end of the section, we examine how they interact.

But before we get to this, we will briefly digress to discuss a redundant attribute, portType, which can be used on the messaging activities (<invoke>, <receive>, <reply>, <onEvent>, and <onMessage>):

> "The portType attribute on the <receive> activity is optional... If the portType attribute is included for readability, 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" *[17, Sec. 5.1]*

As both partnerLink and the role are required for those activities, the portType attribute can only carry redundant information. Thus, there is no reason to keep the portType attribute and consequently it has been removed from the messaging activities.

Case 1: Fault and Compensation Handlers

Fault and compensation handlers are elements of the <scope> activity and when used in an <invoke> they implicitly declare a surrounding <scope> activity as follows:

> "Semantically, the specification of local fault handlers and/or a local compensation handler is equivalent to the presence of an implicit <scope> activity immediately enclosing the <invoke> providing these handlers. The implicit <scope> activity assumes the name of the <invoke> activity it encloses, its suppressJoinFailure attribute, as well as its <sources> and <targets>." *[17, Sec. 10.3]*

This means that whenever either fault or compensation handlers, or both, are declared within an <invoke> activity, it is to be executed as if there was an enclosing <scope> activity. Thus, writing an <invoke> activity along the lines of Listing 8 would be the same as writing out the scope explicitly, as in Listing 9.

Case 2: Mapping Message Parts

The <invoke> activity also allows implicit assignment operations using <toParts> and <fromParts>. <toParts> is used to copy the contents of variables into specified parts of the message to be sent. Symmetrically, <fromParts> is used to copy parts of a received message into specified variables.

The standard specification says the following about the use of <toParts>:

> "By using the <toParts> element, an anonymous temporary WSDL variable is declared based on the type specified by the relevant WSDL operation's input message. The <toPart> elements, as a group, act as the single virtual <assign>, with each <toPart> acting as a <copy>." *[17, Sec. 10.3.1]*

Listing 8: An <invoke> with an implicit <scope>.

```
<invoke ...
  name="bookFlight"
  suppressJoinFailure="yes">
  <targets>
    ...
  </targets>
  <sources>
    ...
  </sources>
    ...
  <catch ...>
    activity
  </catch>
  <catchAll>
    activity
  </catchAll>
  <compensationHandler>
    activity
  </compensationHandler>
  ...
</invoke>
```

Listing 9: Making the <scope> explicit.

```
<scope
  name="bookFlight"
  suppressJoinFailure="yes">
  <targets>
    ...
  </targets>
  <sources>
    ...
  </sources>
  <faultHandlers>
    <catch ...>
      activity
    </catch>
    <catchAll>
      activity
    </catchAll>
  </faultHandlers>
  <compensationHandler>
    activity
  </compensationHandler>
  <invoke name="bookFlight">
   ...
  </invoke>
</scope>
```

Declaring a temporary variable — one that is only visible during the execution of the <invoke> activity — is equivalent to having an immediately enclosing <scope> declaring the variable explicitly. Note that we have to use a fresh name for the variable so that it does not clash with any other variables referenced by the <invoke> activity (cf. Sec. 4.1.1 for a discussion of fresh names).

The virtual <assign> activity can then be declared explicitly in a <sequence> prior to the <invoke> activity. The following quote supports this idea:

> "The virtual <assign> MUST follow the same semantics and use the same faults as a real <assign>." *[17, Sec. 10.3.1]*

Listing 11 illustrates how the implicit <scope> and <assign> activities of Listing 10 are made explicit.

Case 3: Element Variables

The inputVariable and outputVariable attributes, when used, must refer to a WSDL message type variable matching the message to be sent or received respectively [17, SA00048], with one exception:

> "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." *[17, Sec. 10.3]*

The standard specification prescribes that referring to an element variable is equivalent to declaring a temporary message variable and a virtual <assign> activity:

> "The result of using a variable in the previously defined circumstance MUST be the equivalent of declaring an anonymous temporary WSDL message variable based on the associated WSDL

Listing 10: `<invoke>` with implicit assignments.

```
<invoke
  name="orderItems"
  partnerLink="Seller"
  operation="Purchase"
  outputVariable="confirmation">
  <toParts>
    <toPart
      part="address"
      fromVariable="customerAddress"/>
    <toPart
      part="items"
      fromVariable="selectedItems"/>
  </toParts>
</invoke>
```

Listing 11: Making the assignments explicit.

```
<scope
  name="orderItems"
  <variables>
    <variable
      name="in"
      messageType="msg:PurchaseMessage"/>
  </variables>
  <sequence>
    <assign>
      <copy keepSrcElementName="yes">
        <from variable="customerAddress"/>
        <to variable="in" part="address"/>
      </copy>
      <copy keepSrcElementName="yes">
        <from variable="selectedItems"/>
        <to variable="in" part="items"/>
      </copy>
    </assign>
    <invoke
      name="orderItems"
      partnerLink="Seller"
      operation="Purchase"
      inputVariable="in"
      outputVariable="confirmation"/>
  </sequence>
</scope>
```

> message type. The copying of the element data between the anonymous temporary WSDL message variable and the element variable acts as a single virtual <assign> with one <copy> operation whose keepSrcElementName attribute is set to "yes"." *[17, Sec. 10.3]*

This is similar to the usage of <toParts> and <fromParts>. However, in this case we are copying element variables, so the keepSrcElementName attribute on the <copy> operation is set to 'yes':

> "The optional keepSrcElementName attribute of the <copy> construct is used to specify whether the element name of the destination (as selected by the to-spec) will be replaced by the element name of the source (as selected by the from-spec) during the copy operation" *[17, Sec. 8.4]*

5.2.1 Interplay between the cases

We have explored three cases of syntactic sugar for the <invoke> activity. In all three cases, desugaring makes an implicit <scope> explicit and in the last two cases, both implicit variables and virtual <assign> activities are made explicit. As mentioned previously, a single <invoke> could match all three cases, and thus we need to consider how they interact and how to desugar <invoke> as a whole. This is the topic of this section.

Scope Any fault caused by the virtual assignments of message parts (case 2) must be handled by the same local <scope> as the one which is implicitly declared by specifying local fault and compensation handlers (case 1):

> "The virtual <assign> created as a consequence of the <fromPart> or <toPart> elements occurs as part of the scope of the <invoke> activity and therefore any fault that is thrown are caught by an <invoke>'s inline fault handler when defined." *[17, Sec. 10.3.1]*

The standard specification does not describe how to handle errors from virtual assignments caused by a reference to an element variable (case 3). We suspect that this is because errors are not expected, as both the source and target variables are certain to exist and to be of the exact same type. However, implementation specific faults, e.g. 'out of memory', may still occur. As the standard specification does not describe how to handle such faults, it would be consistent to follow the pattern from assignments of parts, i.e. to let the <invoke>'s own <scope> handle any unexpected fault that might occur.

Implicit activities As <toParts> and inputVariable are mutually exclusive, there can be no conflict between their implicit activities; the same goes for <fromParts> and outputVariable. Thus, there will be at most one virtual <assign> activity prior to and following the <invoke> activity respectively, so virtual assignments do not conflict and may be part of the same <sequence> within the same <scope>. The body of the <scope> thus becomes a <sequence> of zero or one prior assignment followed by the core <invoke> activity and zero or one assignments at the end.

5.2.2 Transforming Invoke

The shared <scope> and <sequence> makes it difficult to consider the three cases separately when constructing the XSLT template and as a result, the template becomes somewhat large and complex. To give an overview, we outline the structure of the main template in pseudo code in Listing 12. In the pseudo code, inputElement and outputElement are booleans, indicating whether the input or output variable attributes are referring to element (*true*) or message variables (*false*). Element names, as in XSLT, evaluate to *true* when present, and *false* otherwise.

Listing 12: Transforming <invoke>.

```
if (toParts ∨ fromParts ∨ inputElement ∨ outputElement ∨ catch ∨ catchAll ∨ compensationHandler){

    <scope ...>

        if (toParts ∨ fromParts ∨ inputElement ∨ outputElement)
            Implicit temporary variables in invoke made explicit:
            <variables> ... </variables>

        if (catch ∨ catchAll)
            Move invokes local fault handlers to enclosing scopes
            <catch ...> ... </catch>*
            <catchAll ...> ... </catchAll>

        if (compensationHandler)
            Move invokes local compensation handler to enclosing scope
            <compensationHandler> ... </compensationHandler>

        if (toParts ∨ fromParts ∨ inputElement ∨ outputElement) {

            <sequence>

                if (toParts)
                    <assign>
                        <copy> ... </copy>*
                    </assign>

                if (inputElement)
                    <assign>
                        <copy> ... </copy>*
                    </assign>

                <invoke ...>

                if (fromParts)
                    <assign>
                        <copy> ... </copy>*
                    </assign>

                if (outputElement)
                    <assign>
                        <copy> ... </copy>*
                    </assign>

            </sequence>

        } else {

            Invoke without implict assignment, but with implict scope.
            <invoke ...>

        }

    </scope>

} else {

    A core invoke
    <invoke ...>

}
```

Listing 13: A synchronous `<invoke>`.

```
<invoke
  partnerLink="pl"
  operation="op"
  inputVariable="in"
  outputVariable="out" />
```

Listing 14: Encoding synchronous `<invoke>`.

```
<sequence>
  <invoke
    partnerLink="pl"
    operation="op"
    inputVariable="in" />
  <receive
    partnerLink="pl"
    operation="op"
    variable="out" />
</sequence>
```

5.2.3 A note on synchronous `<invoke>`

It is well known that one can often encode a synchronous communation operation as a pair of asynchronous communication operations. In the case of WS-BPEL, one could for example imagine transforming the synchronous `<invoke>` in Listing 13 to the asynchronous `<invoke>`/`<receive>`-pair in Listing 14.

While these code fragments abstractly accomplish the same request/response operation, they are not interchangeable because WSDL models request/response (and solicit/response) as primitive operation types:

> "WSDL has four transmission primitives that an endpoint can support:
>
> - **One-way.** The endpoint receives a message.
> - **Request-response.** The endpoint receives a message, and sends a correlated message.
> - **Solicit-response.** The endpoint sends a message, and receives a correlated message.
> - **Notification.** The endpoint sends a message.
>
> WSDL refers to these primitives as **operations**. Although request/response or solicit/response can be modeled abstractly using two one-way messages, it is useful to model these as primitive operation types because:
>
> - They are very common.
> - The sequence can be correlated without having to introduce more complex flow information.
> - Some endpoints can only receive messages if they are the result of a synchronous request response.
> - A simple flow can algorithmically be derived from these primitives at the point when flow definition is desired."
>
> *[5, Sec. 2.4]*

Concretely, this means that the two fragments cannot both use the same WSDL definitions for the partner link `pl`, as the definition of `op` in the case of Listing 13 would have to be on the following form:

```
<wsdl:operation name="op" ...>
  <wsdl:input .../>
  <wsdl:output .../>
</wsdl:operation>
```

whereas Listing 14 would require a separate callback operation, and such a transformation would thus change the WSDL interface of the process. For the same reasons, one cannot change a `<reply>` into an asynchronous `<invoke>`, though they both send a message without expecting a reply.

Listing 15: <receive> example.

```
<receive partnerLink="purchasing"
  portType="lns:purchaseOrderPT"
  operation="sendPurchaseOrder"
  variable="PO"
  createInstance="yes">

```

Listing 16: <receive> example desugared.

```
<pick createInstance="yes">
  <onMessage partnerLink="purchasing"
    portType="lns:purchaseOrderPT"
    operation="sendPurchaseOrder"
    variable="PO">
</pick>
```

5.3 <receive>

WS-BPEL syntax summary: Listing 58
XSLT template: Appendix D.15

The <receive> activity is similar to the <onMessage> event of the <pick> activity, as evident from their syntax summaries (cf. Listing 58 and Listing 56) and the following quotes from the standard specification:

> "The <onMessage> is similar to a <receive> activity, in that it waits for the receipt of an inbound message." *[17, Sec. 11.5]*

> "[SA00063] The semantics of the <onMessage> event are identical to a <receive> activity regarding the optional nature of the variable attribute or <fromPart> elements (see also [SA00047]), the handling of race conditions, the handling of correlation sets, the single element- based part message short cut and the constraint regarding simultaneous enablement of conflicting receive actions. For the last case, if two or more receive actions for the same partnerLink, portType, operation and correlationSet(s) are simultaneously enabled during execution, then the standard fault bpel:conflictingReceive MUST be thrown (see section 10.4. Providing Web Service Operations - Receive and Reply). Enablement of an <onMessage> event is equivalent to enablement of the corresponding <receive> activity for the purposes of this constraint." *[17, Sec. 11.5]*

In fact, we have found nothing in the standard specification which indicates any differences between a <receive> activity and a <pick> activity with a single <onMessage> event. Thus, given the above quotes, it seems reasonable to assume that they should be equivalent, and thus we can replace the former with the latter. Listing 16 shows how the <receive> activity example in Listing 15 is desugared into a <pick>.

5.4 <pick>

WS-BPEL syntax summary: Listing 56
Core BPEL syntax summary: Listing 57
XSLT template: Appendix D.13

An <onMessage> element of a <pick> activity makes use of implicit assignments when <fromParts> is used or when its variable attribute refers to an element variable, just as it was the case with <invoke> (though for <invoke> the attribute name is outputVariable). The standard specification defines the syntax and semantics of <fromParts> in <receive> to be the same as in the context of <invoke>:

> "The syntax and semantics of the <fromPart> elements as used on the <receive> activity are the same as specified for the <invoke> activity in section 10.3.1." *[17, Sec. 10.4]*

And as argued in the previous section, <receive> is a special case of <pick>, so the same semantics should apply to <fromParts> in <onMessage>. With respect to the semantics of the variable attribute when it refers to an element variable, the relation to <invoke> is a little less clear. But the requirements on static analysis [17, SA00058] for <receive> are phrased similarly to that of the <invoke> activity [17, SA00048], and thus the discussion from case 3 in Section 5.2 applies to <receive>, and thereby <onMessage>, as well.

Thus, we can reuse the transformation we constructed to handle <fromParts> and element variables in the <invoke> activity, with two minor differences:

- we will have to place the temporary variables for all of the <onMessage>s in a shared <scope> enclosing the <pick>
- the explicit assignments must be placed inside the <onMessage>s

Listing 17 shows an example <pick> with two <onMessage>s using <fromParts> and an element variable, and the example is desugared in Listing 18.

5.5 <reply>

WS-BPEL syntax summary:	Listing 60
Core BPEL syntax summary:	Listing 61
XSLT template:	Appendix D.20

The <reply> activity makes use of implicit assignments when <toParts> is used or when its variable attribute refers to an element variable, just as it was the case with <invoke> (though for <invoke> the attribute name is inputVariable). Indeed, the standard specification defines the syntax and semantics of <toParts> to be the same as in the context of <invoke>:

> "The syntax and semantics of the <toPart> elements as used on the <reply> activity are the same as specified in section 10.3.1. Mapping WSDL Message Parts for the <invoke> activity" *[17, Sec. 10.4]*

With respect to the semantics of the variable attribute when it refers to an element variable, the relation to <invoke> is a little less clear. But the requirements on static analysis [17, SA00058] are phrased similarly to that of the <invoke> activity [17, SA00048], and thus the discussion from case 3 in Section 5.2 applies to <reply> as well.

Thus, we can reuse the transformation we constructed to handle <toParts> and element variables in the <invoke> activity.

5.6 <scope>

WS-BPEL syntax summary:	Listing 64
Core BPEL syntax summary:	Listing 65
XSLT template:	Appendix D.21

The concept of virtual <assign> activities arises again in the descriptions of in-line variable initializations. Variables are declared as part of a <scope> activity. Within the <variable> declaration, a from-spec, as known from assignments [17, Sec. 8.4], may be used to initialize the variable. Listing 19 shows a simple example, initializing an integer variable to the value 42.

The standard specification makes this feature sound entirely trivial:

> "Conceptually the in-line variable initializations are modeled as a virtual <sequence> activity that contains a series of virtual <assign> activities, one for each variable being initialized, in the order they are listed in the variable declarations." *[17, Sec. 8.1]*

I.e. when one or more variables are initialized within the variables declaration, we can make the virtual assignments explicit in a <sequence>, putting the scope main activity at the end of the sequence and letting the sequence become the scope main activity instead. Listing 20 illustrates this approach. While this will initialize the variables before they are used, it does not provide semantic equivalence wrt. fault handling, cf. the requirements for scope initialization:

> "Scope initialization is an all-or-nothing behavior: either it all occurs successfully or a bpel:scopeInitializationFailure fault MUST be thrown to the parent scope of the failed <scope>." *[17, Sec. 12.1]*

This means that we cannot declare the virtual assignments as part of the scope main activity: Any faults from these assignments would be caught by the scope's own fault handler, instead of resulting in bpel:scopeInitializationFailure fault being thrown to the parent scope as required.

Our solution involves three <scope>s:

Listing 17: `<pick>` with implicit assignments.

```
<pick>
  <onMessage
    partnerLink="partnerLink"
    operation="op1">
    <fromParts>
      <fromPart
        part="id"
        toVariable="idVar" />
      <fromPart
        part="description"
        toVariable="descVar" />
    </fromParts>
    <empty />
  </onMessage>
  <onMessage
    partnerLink="partnerLink"
    operation="op2"
    variable="addressVariable">
    <empty />
  </onMessage>
</pick>
```

Listing 18: Making the assignments explicit.

```
<scope>
  <variables>
    <variable
      name="fresh1"
      messageType="..." />
    <variable
      name="fresh2"
      messageType="..." />
  </variables>
  <pick>
    <onMessage
      partnerLink="partnerLink"
      operation="op1"
      variable="fresh1">
      <sequence>
        <assign>
          <copy>
            <from part="id"
                  variable="fresh1"/>
            <to variable="idVar"/>
          </copy>
          <copy>
            <from part="description"
                  variable="fresh1"/>
            <to variable="descVar"/>
          </copy>
        </assign>
        <empty/>
      </sequence>
    </onMessage>
    <onMessage
      operation="op2"
      partnerLink="partnerLink"
      variable="fresh2">
      <sequence>
        <assign>
          <copy keepSrcElementName="yes">
            <from part="addressIn"
                  variable="fresh2"/>
            <to variable="addressVariable"/>
          </copy>
        </assign>
        <empty/>
      </sequence>
    </onMessage>
  </pick>
</scope>
```

Listing 19: Example of a variable initialization.

```
<scope ...>
  <variables>
    <variable name="size" type="xsd:int">
      <from><literal>42</literal></from>
    </variable>
  </variables>
  <flow>
    <invoke .../>
    <invoke .../>
  </flow>
</scope>
```

Listing 20: Naive proposal for transformation of the variable initialization example.

```
<scope ...>
  <variables>
    <variable name="size" type="xsd:int"/>
  </variables>
  <sequence>
    <assign>
      <copy>
        <from><literal>42</literal></from>
        <to variable="size"/>
      </copy>
    </assign>
    <flow>
      <invoke .../>
      <invoke .../>
    </flow>
  </sequence>
</scope>
```

Variable scope: Declares the variables, so they are reachable from both of the other two `<scope>`s, which are put in a `<sequence>` within this scope. It will rethrow any faults, thus making it transparent regarding fault handling, and the default compensation handling will ensure that it is transparent in this regard as well.

Initialization scope: Assigns the initial values to the variables. It has a fault handler that catches all faults and rethrows them as `scopeInitializationFault`s. This ensures that faults from variable initializations will cause a `scopeInitializationFault`, as required.

Main scope: Executes the original main activity. It will also have the fault handlers from the original `<scope>` activity.

Listing 21 shows how the `<scope>` from Listing 19 is desugared.

5.6.1 `<onEvent>`

A `<scope>` may have a set of event handlers associated with it, which are invoked in parallel when their corresponding event occurs. One type of event handler is `<onEvent>` which receives messages much in the same way as `<receive>`:

> "The `<onEvent>` element indicates that the specified event waits for a message to arrive. The interpretation of this element and its attributes is very similar to a `<receive>` activity." *[17, Sec. 12.7.1]*

The differences between `<onEvent>` and `<receive>` stem from the fact that the former is to be executed every time the specified message is received, thus starting several instances of its enclosed `<scope>` and it may therefore use partner links, message exchanges, and correlation sets from within the enclosed `<scope>` activity:

> "The `partnerLink` reference MUST resolve to a partner link declared in the process in the following order: the associated scope first and then the ancestor scopes." *[17, Sec. 12.7.1]*

> "When the `messageExchange` attribute is explicitly specified, the resolution order of the message exchange referenced by `messageExchange` attribute MUST be first the associated scope and then the ancestor scopes." *[17, Sec. 12.7.1]*

Listing 21: Unfolding variable initialization.

```
<scope>
  <variables>
    <variable name="size" type="xsd:int">
  </variables>
  <faultHandlers>
    <catchAll>
      <rethrow/>
    </catchAll>
  </faultHandlers>
  <sequence>
    <scope>
      <faultHandlers>
        <catchAll>
          <throw faultName="scopeInitializationFault"/>
        </catchAll>
      </faultHandlers>
      <assign>
        <copy>
          <from>
            <literal>42</literal>
          </from>
          <to variable="size"/>
        </copy>
      </assign>
    </scope>
    <scope>
      <flow>
        <invoke ... />
        <invoke ... />
      </flow>
    </scope>
  </sequence>
</scope>
```

> "The usage of <correlation> is exactly the same as for <receive> activities, with the following addition: it is possible, from an event handler's inbound message operation, to use correlation sets that are declared within the associated scope." *[17, Sec. 12.7.1]*

For the same reason, the variable(s) used for the incoming message is/are placed in the associated <scope>, but they are implicitly declared and the standard specification specifically forbids making them explicit[3]:

> "Variables referenced by the variable attribute of <fromPart> elements or the variable attribute of an <onEvent> element are implicitly declared in the associated scope of the event handler. [SA00086] Variables of the same names MUST NOT be explicitly declared in the associated scope. This requirement MUST be enforced by static analysis." *[17, Sec. 12.7.1]*

But even so, it is still possible to make the implicit assignments, due to the use of the element attribute or <fromParts>, explicit, in a similar fashion to the transformations of <invoke> and <pick>/<receive> (cf. Sec. 5.2):

> "The syntax and semantics of the <fromPart> elements as used on the <onEvent> element are the same as specified in section 10.4. Providing Web Service Operations – Receive and Reply for the receive activity." *[17, Sec. 12.7.1]*

> "If an element attribute is used then the binding of the incoming message to the variable declared in the <onEvent> event handler occurs as specified for the receive activity (...)." *[17, Sec. 12.7.1]*

The difference is simply that the temporary message variable will be implicitly declared by the <onEvent> and the element variable or message part variables will be placed in its associated <scope>; other than that, the transformation is the same. Listing 22 shows an <onEvent> which uses an element variable, and Listing 23 shows how the implicit variable and assignment is made explicit.

5.7 <if>

WS-BPEL syntax summary:	Listing 52
Core BPEL syntax summary:	Listing 53
XSLT template:	Appendix D.10

The <if> activity allows two classic variants of syntactic sugar: leaving out the <else>-branch, and contracting a nested sequence of <else><if>s to a list of <elseif>s. Listing 24 gives an example and Listing 25 shows the desugared version.

5.8 <repeatUntil>

WS-BPEL syntax summary:	Listing 59
XSLT template:	Appendix D.19

WS-BPEL includes another classic example of redundancy: it includes two repetition constructs, <while> and <repeatUntil>, with only minor differences in their semantics:

> "The <while> activity provides for repeated execution of a contained activity. The contained activity is executed as long as the Boolean <condition> evaluates to true at the beginning of each iteration." *[17, Sec. 11.3]*

> "The <repeatUntil> activity provides for repeated execution of a contained activity. The contained activity is executed until the given Boolean <condition> becomes true. The condition is tested after each execution of the body of the loop. In contrast to the <while> activity, the <repeatUntil> loop executes the contained activity at least once." *[17, Sec. 11.4]*

[3] <fromPart> does not have a variable attribute but a toVariable attribute; we assume that what was intended.

Listing 22: An `<onEvent>` declaration.

```
<onEvent
  partnerLink="consumer"
  operation="getStatus"
  element="xsd:string"
  variable="statusRequest">
  <scope name="event">
    <partnerLinks>
      <partnerLink name="consumer"
       partnerLinkType="..."
       myRole="provider" />
    </partnerLinks>
    <empty name="activity" />
  </scope>
</onEvent>
```

Listing 23: `<onEvent>` desugared.

```
<onEvent
  partnerLink="consumer"
  operation="getStatus"
  messageType="msg:StatusMessage"
  variable="fresh">
  <scope name="event">
    <partnerLinks>
      <partnerLink name="consumer"
       partnerLinkType="..."
       myRole="provider" />
    </partnerLinks>
    <variables>
      <variable name="statusRequest"
                element="xsd:string" />
    </variables>
    <sequence>
      <assign>
        <copy keepSrcElementName="yes">
          <from variable="fresh"
                part="foo" />
          <to variable="statusRequest" />
        </copy>
      </assign>
      <empty name="activity" />
    </sequence>
  </scope>
</onEvent>
```

Listing 24: An `<if>` activity using `<elseif>`.

```
<if>
  <condition>$foo</condition>
  activity1

  <elseif>
    <condition>$bar</condition>
    activity2
  </elseif>
</if>
```

Listing 25: `<if>` desugared.

```
<if>
  <condition>$foo</condition>
  activity1

  <else>
    <if>
      <condition>$bar</condition>
      activity2

      <else>
        <empty/>
      </else>
    </if>
  </else>
</if>
```

As stated in the quote above, the main difference between <while> and <repeatUntil> is that the latter always executes its body at least once before checking the condition, whereas <while> checks the condition first. The second difference is that <repeatUntil> stops looping when its condition becomes true, whereas <while> loops as long as its condition is true.

Either can be seen as a sugared version of the other:

<repeatUntil> → <while>:

The text-book transformation of repeat-until to while is (using pseudo-code for brevity):

```
repeat body until condition   →   sequence
                                    body
                                    while (not condition) do
                                      body
```

This will not immediately work for WS-BPEL, as copying the body might involve making copies of named <scope>s which is problematic: scope names within the same immediately enclosing scope must be unique:

> "[SA00092]Within a scope, the name of all named immediately enclosed scopes MUST be unique. This requirement MUST be statically enforced." *[17, Sec. 12.4.3]*

so one would have to rename all scopes in the body and somehow make sure that compensation works as intended.

Instead, one could imagine doing a less standard transformation which avoids copying the body:

```
repeat body until condition   →   scope
                                    variable first := true
                                    while first or (not condition) do
                                      sequence
                                        body
                                        first := false
```

Note though, that this transformation negates the condition, but WS-BPEL allows any expression language to be used, and there is no requirement that expression languages must have a negation operator! Also, using this approach, one would have to extend our transformations with expression language specific behaviour.

To avoid these issues, we can use the following slightly more complicated transformation:

```
repeat body until condition   →   scope
                                    variable cond_var := false
                                    while (not cond_var) do
                                      sequence
                                        body
                                        cond_var := condition
```

In this transformation, we evaluate the condition expression unchanged, storing the result in the fresh variable cond_var, and then use XPath to negate the stored result. The transformation is thus independent of the expression language. Note that, since it is only recommended that boolean expressions return a value of xsd:boolean, we must, in the general case, evaluate the condition in the following way to ensure that we store a value of type xsd:boolean:

```
if condition then
  cond_var := true
else
  cond_var := false
```

In this case we can optimize the negation of the condition variable and the else-case away, finally arriving at the following transformation:

```
                                          scope
                                            variable cond_var := true
                                            while cond_var do
repeat body until condition   →               sequence
                                                body
                                                if condition then
                                                  cond_var := false
```

<while> → <repeatUntil>:

The reverse transformation can be achieved as follows:

```
while condition do body   →   if condition then
                                repeat body until (not condition)
```

The body is not copied, whereas the condition is but we have found no indications in the WS-BPEL standard that this could raise any problems – but this could depend on the expression language used.

As was the case in the second transformation above, this transformation negates the expression directly, which might not be desirable, and we can use a similar trick to avoid this:

```
                                          if condition then
                                            scope
                                              variable cond_var
while condition do body   →                   repeat
                                                sequence
                                                  body
                                                  cond_var := condition
                                              until (not $cond_var)
```

Thus, we can transform either construct into the other. We choose to use the transformation of <repeatUntil> as this transformation results in slightly shorter code and it also avoids making more than one copy of the condition.

Listing 26 shows an example of a <repeatUntil> activity which is desugared into the code in Listing 27.

5.9 <sequence>

WS-BPEL syntax summary: Listing 66
XSLT template: Appendix D.22

The authors of the standard specification are aware that the language is not 'minimal' and even suggest that the <sequence> activity could be modeled using a <flow> activity:

> "The set of structured activities in WS-BPEL is not intended to be minimal. There are cases where the semantics of one activity can be represented using another activity. For example, sequential processing may be modeled using either the <sequence> activity, or by a <flow> with properly defined links." *[17, Sec. 11]*

Essentially, the <sequence> activity is syntactic sugar for a <flow> construction with sequentially linked child activities. We could manipulate the <targets> and <sources> elements of the child elements directly to include additional sequencing links. But this would involve adding the sequencing links to the <joinCondition>s, i.e. manipulating expressions in arbitrary languages.

A more straightforward alternative is to wrap each of the child activities in a <flow> activity and place the sequencing links there; the default <joinCondition>s will suffice in this case. For example, we can

Listing 26: Example of a `<repeatUntil>`.

```
<repeatUntil>
  <targets>
    <target linkName="l1" />
  </targets>

  <empty name="body" />

  <condition expressionLanguage="expr-lang">
    cond-expression
  </condition>
</repeatUntil>
```

Listing 27: A `<repeatUntil>` rewritten.

```
<scope>
  <targets>
    <target linkName="l1" />
  </targets>
  <variables>
    <variable
      name="cond_var"
      type="xsd:boolean">
      <from expressionLanguage="xpath">
        false()
      </from>
    </variable>
  </variables>

  <while>
    <condition expressionLanguage="xpath">
      not($cond_var)
    </condition>
    <sequence>
      <empty name="body" />
      <if>
        <condition
          expressionLanguage="expr-lang">
          cond-expression
        </condition>
        <assign>
          <copy>
            <from
              expressionLanguage="xpath">
              true()
            </from>
            <to variable="cond_var" />
          </copy>
        </assign>
      </if>
    </sequence>
  </while>
</scope>
```

Listing 28: Example of a <sequence>.

```
<sequence>
  <empty name="A" />
  <empty name="B" />
  <empty name="C" />
</sequence>
```

Listing 29: A <sequence> rewritten.

```
<flow>
  <links>
    <link name="fresh1" />
    <link name="fresh2" />
  </links>
  <flow>
    <sources>
      <source linkName="fresh1" />
    </sources>
    <empty name="A" />
  </flow>
  <flow>
    <targets>
      <target linkName="fresh1" />
    </targets>
    <sources>
      <source linkName="fresh2" />
    </sources>
    <empty name="B" />
  </flow>
  <flow>
    <targets>
      <target linkName="fresh2" />
    </targets>
    <empty name="C" />
  </flow>
</flow>
```

rewrite the <sequence> in Listing 28 into the <flow> of Listing 29. Note that we need fresh names for the links we add, cf. Sec. 4.1.1, that the default <transitionCondition> is sufficient, and that the value of the suppressJoinFailure attribute is irrelevant for sequence flows, since the join condition will always be true, if the preceding activity finishes without propagating a fault; if the preceding activity propagates a fault, the transition condition will not be evaluated and no joinFailure will occur.

5.10 Non-sugared activities

Besides default values, we have not identified any cases of syntactic sugar in the activities listed in Table 4. We will therefore not discuss them further, only summarize the effect of the default value transformations in Section 3, by listing their WS-BPEL and Core BPEL syntax summaries.

5.10.1 A note about <rethrow>

As noted in a draft of the standard specification, <rethrow> is essentially a <throw> with implicit faultName and faultVariable attributes:

> "A fault caught by a <catchAll> handler or by a custom fault handler that does not specify a faultName, may need to be rethrown. However the <throw> activity that requires a faultName can not be used here as the faultName is not available. Hence all fault handlers are allowed to rethrow the original fault with a <rethrow> activity that is defined to be an empty element. In essence <rethrow> can be considered a macro for a <throw> that throws the fault caught by the corresponding fault handler." *[16, Sec. 13.4]*

Activity	WS-BPEL syntax summary	Core BPEL syntax summary
<assign>	Listing 36	Listing 37
<compensate>	Listing 38	Listing 39
<compensateScope>	Listing 40	Listing 41
<empty>	Listing 42	Listing 43
<exit>	Listing 44	Listing 45
<extensionActivity>	Listing 46	Listing 47
<flow>	Listing 48	Listing 49
<forEach>	Listing 50	Listing 51
<rethrow>	Listing 62	Listing 63
<throw>	Listing 67	Listing 68
<validate>	Listing 69	Listing 70
<wait>	Listing 71	Listing 72
<while>	Listing 73	Listing 74

Table 4: Non-sugared activities.

One might therefore wonder, if not <rethrow> could somehow be transformed into <throw>? As pointed out in *loc. cit.*, the faultName is (statically) unavailable inside the <catchAll> handler, since this handler may handle different faults at runtime, and therefore we cannot simply replace a <retrow> with a <throw> in this situation. One might however attempt to transform a <catchAll> into a number of <catch> handlers, by statically determining which faults could be caught by the <catchAll> in question, thereby making the faultName explicit. But alas, we cannot know all the possible faults a <catchAll> might handle, since the standard specification allows arbitrary platform specific faults:

> "There are various sources of faults in WS-BPEL. A fault response to an <invoke> activity is one source of faults, where the fault name and data are based on the definition of the fault in the WSDL operation. A <throw> activity is another source, with explicitly given name and/or data. WS-BPEL defines several standard faults with their names, and there may be other platform-specific faults such as communication failures." *[17, Sec. 12.5]*

Thus we cannot in general eliminate the <catchAll> construct nor the <rethrow> activity.

6 Extensions

XSLT template: Appendix D.17

WS-BPEL supports extensions that are implementation dependent and they can be either optional or mandatory. In this section we look at how one may syntactically remove the use of unsupported optional extensions.

6.1 Extension Declarations

Extensions are declared in an `<extensions>` element as part of the `<process>` element. Each `<extension>` declaration must specify whether the extension is mandatory by using the `mustUnderstand` attribute. If an `<extension>` has `mustUnderstand="yes"` we cannot remove the extension declaration or the uses of the extension. But if the `<extension>` has `mustUnderstand="no"`, we may safely remove all uses of the extension, as discussed in the following sections, and thereafter remove the extension declaration itself.

Note that WS-BPEL identifies extensions by their namespace URI, and that the same extension URI may be declared more than once; if just one such declaration is has `mustUnderstand="yes"`, the extension is mandatory:

> "The same extension URI MAY be declared multiple times in the `<extensions>` element. If an extension URI is identified as mandatory in one `<extension>` element and optional in another, then the mandatory semantics have precedence and MUST be enforced." *[17, Sec. 14]*

6.2 Unsupported Extension Activities

Extension activities, such as vendor-specific activities, can be used by wrapping them in `<extensionActivity>` elements. The standard specification says the following on how to ignore an optional extension activity:

> "If the element contained within the `<extensionActivity>` element is not recognized by the WS-BPEL processor and is not subject to a `mustUnderstand="yes"` requirement from an extension declaration then the unknown activity MUST be treated as if it were an `<empty>` activity that has the standard-attributes and standard-elements of the unrecognized element; all its other attributes and child elements are ignored." *[17, Sec. 10.9]*

In short, the way to ignore an extension activity is to replace it with an `<empty>` activity, with the same standard attributes and elements as the element enclosed in the extension activity. The `<extensionActivity>` element itself cannot have any attributes or elements besides the single enclosed one.

6.3 Unsupported Extension Assign Operations

The `<assign>` activity is used to copy values between variables. Besides the ordinary `<copy>` element, the standard specification allows the usage of implementation specific assignment operations, wrapped in an `<extensionAssignOperation>` element [17, Sec. 8.4].

> "If the element contained within the `<extensionAssignOperation>` element is not recognized by the WS-BPEL processor and is not subject to a `mustUnderstand="yes"` requirement from an extension declaration then the `<extensionAssignOperation>` operation MUST be ignored." *[17, Sec. 8.4]*

This mean we can safely remove optional `<extensionAssignOperation>`s from a process description. If an `<assign>` consists only of optional `<extensionAssignOperation>`s, as in Listing 31, simply removing them will leave an empty, and thus invalid, `<assign>` activity. In that case, we replace the `<assign>` with an `<empty>` activity in the same way as for extension activities.

Besides the standard attributes, an `<assign>` element may have a `validate` attribute, as follows:

Listing 30: An <assign> activity with both an ordinary <copy> and an <extensionAssignOperation>.

```
<assign>
  <copy>
    ...
  </copy>
  <extensionAssignOperation>
    ...
  </extensionAssignOperation>
</assign>
```

Listing 31: An <assign> activity consisting only of an <extensionAssignOperation>.

```
<assign>
  <extensionAssignOperation>
    ...
  </extensionAssignOperation>
</assign>
```

Listing 32: Activity with ignorable extensions.

```
<wait ext:position='2,5'>
  <until>'2006-11-19T19:50-07:00'</until>
  <ext:icon>clock.png</ext:icon>
</wait>
```

Listing 33: After removing extensions.

```
<wait>
  <until>'2006-11-19T19:50-07:00'</until>
</wait>
```

> "The optional validate attribute can be used with the <assign> activity. Its default value is "no". When validate is set to "yes", the <assign> activity validates all the variables being modified by the activity." *[17, Sec. 8.4]*

In the case where we replace the <assign> activity with an <empty> activity, the content of the original <assign> activity solely consists of ignorable assignment operations — thus, no variable would be modified. Therefore it is safe to disregard the validate attribute.

6.4 Attribute and Element Extensions

Besides the two explicitly extendable elements discussed in the previous sections, WS-BPEL supports extension attributes and elements almost everywhere in a WS-BPEL process:

> "WS-BPEL supports extensibility by allowing namespace-qualified attributes to appear on any WS-BPEL element and by allowing elements from other namespaces to appear within WS-BPEL defined elements." *[17, Sec. 5.3]*

As all extensions must be declared in a namespace other than the WS-BPEL namespace it is easy to identify and remove optional extension attributes and elements: simply leave out all attributes and elements declared in the extension namespace.

Listings 32 and 33 illustrate this, showing a <wait> activity with and without its ignorable extensions, respectively.

6.5 Documentation

XSLT template: Appendix D.16

In a sense, documentation is an ignorable extension: <documentation> elements are used to place human readable notes in the process description [17, Sec. 5.3] and they are not assigned any semantics by the standard specification and can thus be left out of the description.

7 Combining the Transformations

We now have a collection of stand-alone transformations, each of which transforms WS-BPEL processes into equivalent WS-BPEL processes with some class of syntactic sugar eliminated. The question is then, how do we combine these transformations, such that we eliminate all syntactic sugar, i.e. we obtain a Core BPEL process? Can we apply the individual transformations sequentially (in some order) or will we have to integrate them into one big transformation?

The challenge comes from the fact that the transformations interfere with each other in the following ways:

use of syntactic sugar: some transformations produce constructs that use syntactic sugar, in order to keep them simple and easily readable; for example, several transformations produce `<sequence>`s or rely on default attributes/elements.

too general: the transformations that make default attribute values explicit are too general: once the defaults have been made explicit, the attributes are only necessary on the specific elements they relate to. For example, `suppressJoinFailure` only affects activities which have a `<targets>` element (which in Core BPEL, can only be `<flow>`), but the transformation in Section 3.1.3 will make the attribute explicit on all activities and `<process>`.

In the case of the first type of interference, we are in the fortunate situation that there are no cyclic dependencies between the transformations: Figure 1 shows that the dependencies between transformations form a DAG. Thus, applying the transformations sequentially according to any topological ordering of that DAG, will result in a WS-BPEL process with no syntactic sugar.

Due to the second type of interference though, we will not obtain a Core BPEL process this way: there will be a number of redundant attributes on various constructs. We see two solutions to this problem:

- we revise the transformations that make defaults explicit, such that they only make defaults explicit on the elements where the attribute value is actually necessary, or
- we simply make a transformation which removes the redundant attributes, which is to be applied after the others. This is sound since the default values for these attributes have been made explicit on the elements where they matter.

We choose the latter option, as it is simple and keeps the other transformations simple and general. The attributes that should be removed are `suppressJoinFailure`, `exitOnStandardFault`, `expressionLanguage`, and `queryLanguage` on `<process>`, and `suppressJoinFailure` on all activities but `<flow>`. The XSLT implementation of this transformation is listed in Appendix D.18.

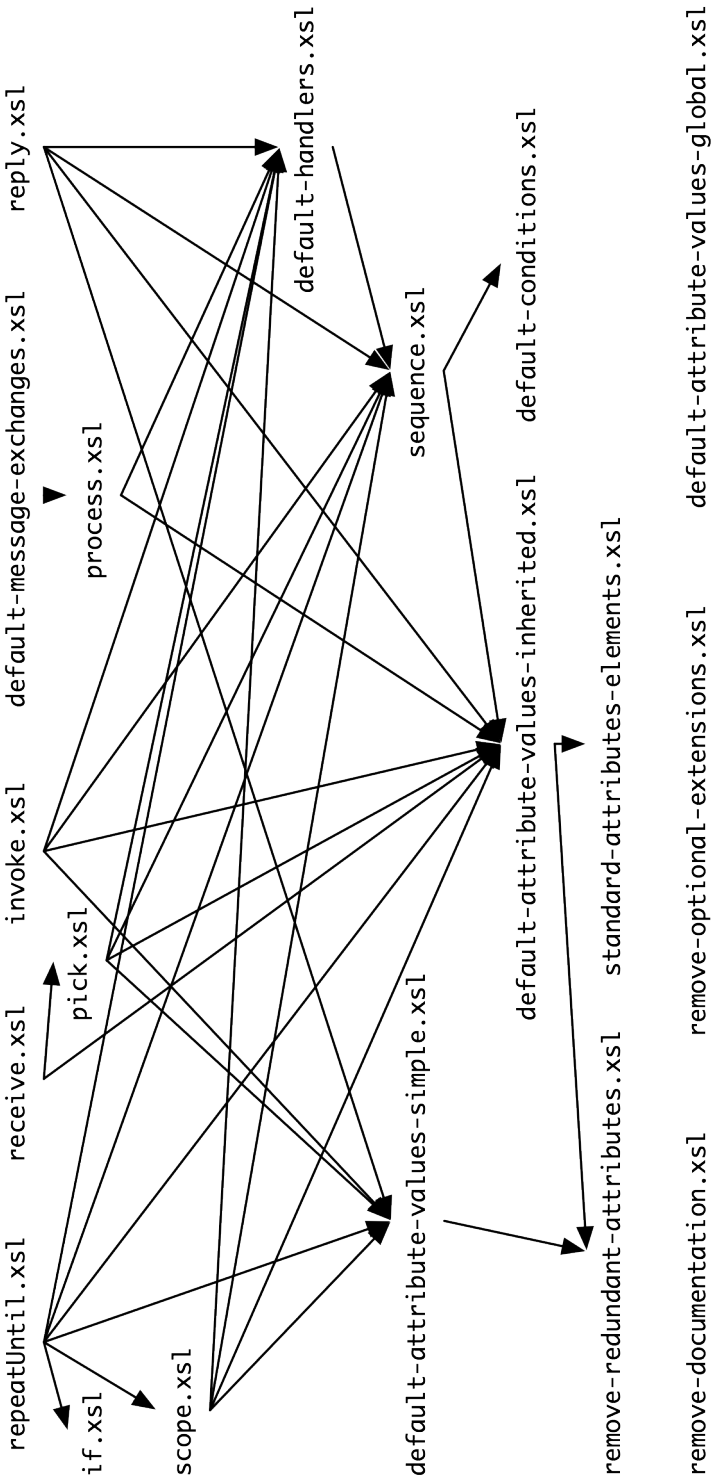

Figure 1: Graph showing which transformations must be applied after other transformations: the graph has an edge $a \rightarrow b$ if transformation a creates syntactically sugared elements of a type that is eliminated by transformation b.

8 Conclusions

We have identified a core subset of the WS-BPEL language, named Core BPEL, and have presented a transformation from WS-BPEL to this core language which preserves semantics according to the informal description in the standard specification, and the WSDL interfaces of processes. Core BPEL has fewer constructs and most constructs have fewer syntactical variations, making it more tractable for formal purposes, such as implementation of execution engines or static analysis, but also easier to understand the features of the language. We have not proved that Core BPEL is minimal, but we are not aware of any further possibilities of reducing or restricting the language syntax.

As part of our analysis, we clarify certain aspects of the syntax of WS-BPEL and provide a more concise presentation of the essential constructs of the language. We therefore hope that this technical report will also prove useful as a cheat sheet for those who are studying the WS-BPEL standard specification.

The transformation from WS-BPEL to Core BPEL consists of a set of independent sub-transformations, each responsible for simplifying a specific part of the language. The subtransformations are given in the form of a set of XSLT 1.0 templates, making them easily adoptable by other researchers and WS-BPEL implementers. The XSLT templates are available at the CosmoBiz website `http://www.cosmobiz.com` where we also provide an online tool for transforming WS-BPEL processes into Core BPEL. As each template performs an independent transformation, users are free to use just a subset of the transformations.

8.1 Future work

There are numerous possible lines of future work. Most importantly, we believe that Core BPEL could serve as a simpler basis for the work the academic community is doing with WS-BPEL; e.g. the numerous WS-BPEL formalizations could probably be simplified, though still remain complete, if they only covered Core BPEL. In the context of the CosmoBiz project, it would be interesting to extend our bigraph formalization of a subset of WS-BPEL [3] to the entire Core BPEL fragment.

Similarly, one might probably simplify implementations of WS-BPEL by only considering Core BPEL; for instance, we expect that Hallwyl's WS-BPEL engine (beepell) [1] can be simplified using this approach. As part of such work, it would also be interesting to investigate if and how this approach would affect performance of implementations.

It would also be interesting to investigate, whether the transformations presented here are semantic preserving with respect to the existing formalizations of WS-BPEL. Similarly, it would be interesting to perform a case study of existing WS-BPEL implementations: do they execute WS-BPEL processes in the same way as their Core BPEL equivalents? In both cases, discrepancies could help pin-point differences in interpretations of the standard as well as errors in our transformations or the formalizations and implementations. This is the natural continuation of Hallwyl's work in his master's thesis [10], where he investigated existing implementations to see if they are consistent in their interpretation of the WS-BPEL standard specification.

We also believe that our approach could be applied to the recent BPMN 2.0 standard [2], with the same benfits as discussed above.

9 References

[1] beepell. Webpage. `http://beepell.com/`.

[2] Business process model and notation (BPMN) version 2.0. Technical report, Object Management Group, January 2011.

[3] Mikkel Bundgaard, Arne John Glenstrup, Thomas Hildebrandt, Espen Højsgaard, and Henning Niss. Formalizing WS-BPEL and Higher Order Mobile Embedded Business Processes in the Bigraphical Programming Languages (BPL) Tool. Technical Report TR-2008-103, IT University of Copenhagen, 2008.

[4] CDuce. Webpage. http://www.cduce.org/.

[5] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web services description language (WSDL) 1.1. W3C Note, W3C, March 2001.

[6] Dirk Fahland. Complete Abstract Operational Semantics for the Web Service Business Process Execution Language. Technical Report 190, Humboldt-Universität zu Berlin, 2005.

[7] Dirk Fahland and Wolfgang Reisig. ASM-based semantics for BPEL: The negative Control Flow. In Danièle Beauquier, Egon Börger, and Anatol Slissenko, editors, *Proceedings of the 12th International Workshop on Abstract State Machines (ASM'05)*, pages 131–151. Paris XII, March 2005.

[8] Roozbeh Farahbod, Uwe Glässer, and Mona Vajihollahi. Specification and validation of the business process execution language for web services. In *Abstract State Machines 2004. Advances in Theory and Practice*, volume 3052 of *Lecture Notes in Computer Science*, pages 78–94. Springer Verlag, 2004.

[9] Roozbeh Farahbod, Uwe Glässer, and Mona Vajihollahi. An abstract machine architecture for web service based business process management. In Christoph Bussler and Armin Haller, editors, *Business Process Management Workshops*, volume 3812 of *Lecture Notes in Computer Science*, pages 144–157. Springer Verlag, 2006.

[10] Tim Hallwyl. Evaluating the BPEL standard specification. Master's thesis, Department of Computer Science, University of Copenhagen, May 2008.

[11] Thomas Hildebrandt (principal investigator). Computer supported mobile adaptive business processes (CosmoBiz) research project. Webpage, 2007. http://www.cosmobiz.org/.

[12] Niels Lohmann. A feature-complete Petri net semantics for WS-BPEL 2.0. In Marlon Dumas and Reiko Heckel, editors, *Proceedings of the 4th International Workshop on Web Services and Formal Methods (WS-FM'07)*, volume 4937 of *Lecture Notes in Computer Science*, pages 77–91. Springer Verlag, 2007.

[13] Niels Lohmann, H.M.W. Verbeek, Chun Ouyang, Christian Stahl, and Wil M. P. van der Aalst. Comparing and evaluating Petri net semantics for BPEL. Computer Science Report 07/23, Eindhoven University of Technology, 2007.

[14] Christian Stahl. A Petri net semantics for BPEL. Informatik-Berichte 188, Humboldt-Universität zu Berlin, 2005.

[15] M. Weidlich, G. Decker, and M. Weske. Efficient analysis of bpel 2.0 processes using π-calculus. In *Asia-Pacific Service Computing Conference, The 2nd IEEE*, pages 266–274, December 2007.

[16] Web services business process execution language version 2.0, working draft. Technical report, OASIS Web Services Business Process Execution Language (WSBPEL) TC, May 2005.

[17] Web services business process execution language version 2.0. Technical report, OASIS Web Services Business Process Execution Language (WSBPEL) TC, April 2007.

[18] XDuce - A Typed XML Processing Language. Webpage. http://xduce.sourceforge.net/.

[19] XML Schema part 1: Structures second edition. W3C Recommendation, W3C, 2004.

A WS-BPEL vs. Core BPEL Syntax Summaries

This section contains syntax summaries for the WS-BPEL process and activity elements side-by-side with the corresponding Core BPEL syntax summaries for easy comparison.

Listing 34: WS-BPEL <process>.

```
<process name="NCName"
  targetNamespace="anyURI"
  queryLanguage="anyURI"?
  expressionLanguage="anyURI"?
  suppressJoinFailure="yes|no"?
  exitOnStandardFault="yes|no"?
  xmlns="http://docs.oasis-open.org/wsbpel
      /2.0/process/executable">
  <extensions>?
    ...
  </extensions>
  <import namespace="anyURI"?
    location="anyURI"?
    importType="anyURI" />*
  <partnerLinks>?
    ...
  </partnerLinks>
  <messageExchanges>?
    ...
  </messageExchanges>
  <variables>?
    ...
  </variables>
  <correlationSets>?
    ...
  </correlationSets>
  <faultHandlers>?
    ...
  </faultHandlers>
  <eventHandlers>?
    ...
  </eventHandlers>
  activity
</process>
```

Listing 35: Core BPEL <process>.

```
<process name="NCName"
  targetNamespace="anyURI"
  xmlns="http://docs.oasis-open.org/wsbpel
      /2.0/process/executable">
  <extensions>?
    ...
  </extensions>
  <import namespace="anyURI"?
    location="anyURI"?
    importType="anyURI" />*
  activity
</process>
```

Scope-related content of a <process> is moved into a new <scope> inside the <process> and the default attribute values defined on the <process> are made explicit on the relevant elements inside the <process> after which the attributes are removed from the <process> element.

Listing 36: WS-BPEL <assign>.

```
<assign validate="yes|no"?
  standard-attributes>
  standard-elements
  (
  <copy keepSrcElementName="yes|no"?
    ignoreMissingFromData="yes|no"?>
    from-spec
    to-spec
  </copy>
  |
  <extensionAssignOperation>
    assign-element-of-other-namespace
  </extensionAssignOperation>
  )+
</assign>
```

Listing 37: Core BPEL <assign>.

```
<assign validate="yes|no">
  (
  <copy keepSrcElementName="yes|no"?
    ignoreMissingFromData="yes|no">
    from-spec
    to-spec
  </copy>
  |
  <extensionAssignOperation>
    assign-element-of-other-namespace
  </extensionAssignOperation>
  )+
</assign>
```

Standard attributes and elements are moved to a wrapping <flow> and default attribute values are made explicit.

Listing 38: WS-BPEL <compensate>.

```
<compensate standard-attributes>
  standard-elements
</compensate>
```

Listing 39: Core BPEL <compensate>.

```
<compensate />
```

Standard attributes and elements are moved to a wrapping <flow>.

Listing 40: WS-BPEL <compensateScope>.

```
<compensateScope target="NCName"
  standard-attributes>
  standard-elements
</compensateScope>
```

Listing 41: Core BPEL <compensateScope>.

```
<compensateScope target="NCName" />
```

Standard attributes and elements are moved to a wrapping <flow>.

Listing 42: WS-BPEL <empty>.

```
<empty standard-attributes>
  standard-elements
</empty>
```

Listing 43: Core BPEL <empty>.

```
<empty />
```

Standard attributes and elements are moved to a wrapping <flow>.

Listing 44: WS-BPEL <exit>.

```
<exit standard-attributes>
  standard-elements
</exit>
```

Listing 45: Core BPEL <exit>.

```
<exit />
```

Standard attributes and elements are moved to a wrapping <flow>.

Listing 46: WS-BPEL `<extensionActivity>`.

```
<extensionActivity>
  <anyElementQName
    standard-attributes>
    standard-elements
  </anyElementQName>
</extensionActivity>
```

Listing 47: Core BPEL `<extensionActivity>`.

```
<extensionActivity>
  <anyElementQName name="NCName" />
</extensionActivity>
```

Standard attributes and elements are moved to a wrapping `<flow>`, except for the `name` attribute.

Listing 48: WS-BPEL `<flow>`.

```
<flow standard-attributes>
  standard-elements
  <links>?
    <link name="NCName" />+
  </links>
  activity+
</flow>
```

Listing 49: Core BPEL `<flow>`.

```
<flow suppressJoinFailure="yes|no">
  <targets>?
    <joinCondition
      expressionLanguage="anyURI">
      bool-expr
    </joinCondition>
    <target linkName="NCName" />+
  </targets>
  <sources>?
    <source linkName="NCName">+
      <transitionCondition
        expressionLanguage="anyURI">
        bool-expr
      </transitionCondition>
    </source>
  </sources>
  <links>?
    <link name="NCName" />+
  </links>
  activity+
</flow>
```

Core BPEL only allows the WS-BPEL standard attributes and elements on `<flow>` and they are therefore inlined in the Core BPEL syntax summary for `<flow>`. Default attribute values and elements are made explicit.

Listing 50: WS-BPEL `<forEach>`.

```
<forEach counterName="BPELVariableName"
  parallel="yes|no"
  standard-attributes>
  standard-elements
  <startCounterValue
    expressionLanguage="anyURI"?>
    unsigned-integer-expression
  </startCounterValue>
  <finalCounterValue
    expressionLanguage="anyURI"?>
    unsigned-integer-expression
  </finalCounterValue>
  <completionCondition>?
    <branches
      expressionLanguage="anyURI"?
      successfulBranchesOnly="yes|no"?>?
      unsigned-integer-expression
    </branches>
  </completionCondition>
  <scope ...>...</scope>
</forEach>
```

Listing 51: Core BPEL `<forEach>`.

```
<forEach counterName="BPELVariableName"
  parallel="yes|no">
  <startCounterValue
    expressionLanguage="anyURI">
    unsigned-integer-expression
  </startCounterValue>
  <finalCounterValue
    expressionLanguage="anyURI">
    unsigned-integer-expression
  </finalCounterValue>
  <completionCondition>
    <branches
      expressionLanguage="anyURI"
      successfulBranchesOnly="yes|no">?
      unsigned-integer-expression
    </branches>
  </completionCondition>
  <scope ...>...</scope>
</forEach>
```

Standard attributes and elements are moved to a wrapping `<flow>` and default attribute values are made explicit.

Listing 52: WS-BPEL `<if>`.

```
<if standard-attributes>
  standard-elements
  <condition expressionLanguage="anyURI"?>
    bool-expr
  </condition>
  activity
  <elseif>*
    <condition
      expressionLanguage="anyURI"?>
      bool-expr
    </condition>
    activity
  </elseif>
  <else>?
    activity
  </else>
</if>
```

Listing 53: Core BPEL `<if>`.

```
<if>
  <condition expressionLanguage="anyURI">
    bool-expr
  </condition>
  activity
  <else>
    activity
  </else>
</if>
```

`<elseif>`s are unfolded to `<else><if>`s, an explicit empty `<else>` branch is added if lacking, standard attributes and elements are moved to a wrapping `<flow>`, and default attribute values are made explicit.

Listing 54: WS-BPEL `<invoke>`.

```
<invoke partnerLink="NCName"
  portType="QName"?
  operation="NCName"
  inputVariable="BPELVariableName"?
  outputVariable="BPELVariableName"?
  standard-attributes>
  standard-elements
  <correlations>?
    <correlation set="NCName"
      initiate="yes|join|no"?
      pattern="request|response|request-
          response"? />+
  </correlations>
  <catch faultName="QName"?
    faultVariable="BPELVariableName"?
    faultMessageType="QName"?
    faultElement="QName"?>*
    activity
  </catch>
  <catchAll>?
    activity
  </catchAll>
  <compensationHandler>?
    activity
  </compensationHandler>
  <toParts>?
    <toPart part="NCName"
      fromVariable="BPELVariableName"/>+
  </toParts>
  <fromParts>?
    <fromPart part="NCName"
      toVariable="BPELVariableName"/>+
  </fromParts>
</invoke>
```

Listing 55: Core BPEL `<invoke>`.

```
<invoke partnerLink="NCName"
  operation="NCName"
  inputVariable="BPELVariableName"?
  outputVariable="BPELVariableName"?>
  <correlations>?
    <correlation set="NCName"
      initiate="yes|join|no"
      pattern="request|response|request-
          response"? />+
  </correlations>
</invoke>
```

The superfluous `portType` attribute is removed, scope-related content as well as implicit temporary variables are moved to a wrapping `<scope>`, implicit assignments are made explicit, standard attributes and elements are moved to a wrapping `<flow>`, and default attribute values are made explicit.

Listing 56: WS-BPEL <pick>.

```
<pick createInstance="yes|no"?
  standard-attributes>
  standard-elements
  <onMessage partnerLink="NCName"
    portType="QName"?
    operation="NCName"
    variable="BPELVariableName"?
    messageExchange="NCName"?>+
    <correlations>?
      <correlation set="NCName"
         initiate="yes|join|no"? />+
    </correlations>
    <fromParts>?
      <fromPart part="NCName"
         toVariable="BPELVariableName" />+
    </fromParts>
    activity
  </onMessage>
  <onAlarm>*
    (
    <for expressionLanguage="anyURI"?>
      duration-expr
    </for>
    |
    <until expressionLanguage="anyURI"?>
      deadline-expr
    </until>
    )
    activity
  </onAlarm>
</pick>
```

Listing 57: Core BPEL <pick>.

```
<pick createInstance="yes|no">
  <onMessage partnerLink="NCName"
    operation="NCName"
    variable="BPELVariableName"?
    messageExchange="NCName"?>+
    <correlations>?
      <correlation set="NCName"
         initiate="yes|join|no" />+
    </correlations>
    activity
  </onMessage>
  <onAlarm>*
    (
    <for expressionLanguage="anyURI">
      duration-expr
    </for>
    |
    <until expressionLanguage="anyURI">
      deadline-expr
    </until>
    )
    activity
  </onAlarm>
</pick>
```

The superfluous `portType` attribute is removed, implicit temporary variables are made explicit in a wrapping <scope>, implicit assignments are made explicit, standard attributes and elements are moved to a wrapping <flow>, and default attribute values are made explicit.

Listing 58: WS-BPEL <receive>.

```
<receive partnerLink="NCName" portType="
    QName"? operation="NCName"
  variable="BPELVariableName"?
      createInstance="yes|no"?
  messageExchange="NCName"? standard-
      attributes>
  standard-elements
  <correlations>?
    <correlation set="NCName" initiate="yes
        |join|no"? />+
  </correlations>
  <fromParts>?
    <fromPart part="NCName" toVariable="
        BPELVariableName" />+
  </fromParts>
</receive>
```

<receive> is transformed into a <pick> with a single <onMessage>.

Listing 59: WS-BPEL <repeatUntil>.

```
<repeatUntil standard-attributes>
  standard-elements
  activity
  <condition expressionLanguage="anyURI"?>
    bool-expr
  </condition>
</repeatUntil>
```

<repeatUntil> is transformed into a <while>.

Listing 60: WS-BPEL `<reply>`.

```
<reply partnerLink="NCName"
  portType="QName"?
  operation="NCName"
  variable="BPELVariableName"?
  faultName="QName"?
  messageExchange="NCName"?
  standard-attributes>
  standard-elements
  <correlations>?
    <correlation
      set="NCName"
      initiate="yes|join|no"? />+
  </correlations>
  <toParts>?
    <toPart
      part="NCName"
      fromVariable="BPELVariableName" />+
  </toParts>
</reply>
```

Listing 61: Core BPEL `<reply>`.

```
<reply partnerLink="NCName"
  operation="NCName"
  variable="BPELVariableName"?
  faultName="QName"?
  messageExchange="NCName">
  <correlations>?
    <correlation
      set="NCName"
      initiate="yes|join|no" />+
  </correlations>
</reply>
```

The superfluous `portType` attribute is removed, implicit temporary variables are made explicit in a wrapping `<scope>`, implicit assignments are made explicit, standard attributes and elements are moved to a wrapping `<flow>`, and default attribute values are made explicit.

Listing 62: WS-BPEL `<rethrow>`.

```
<rethrow standard-attributes>
  standard-elements
</rethrow>
```

Listing 63: Core BPEL `<rethrow>`.

```
<rethrow />
```

Standard attributes and elements are moved to a wrapping `<flow>`.

Listing 64: WS-BPEL `<scope>`.

```
<scope isolated="yes|no"?
  exitOnStandardFault="yes|no"?
  standard-attributes>
  standard-elements
  <partnerLinks>?
    ...
  </partnerLinks>
  <messageExchanges>?
    ...
  </messageExchanges>
  <variables>?
    <variable name="BPELVariableName"
      messageType="QName"?
      type="QName"?
      element="QName"?>+
      from-spec?
    </variable>
  </variables>
  <correlationSets>?
    ...
  </correlationSets>
  <faultHandlers>?
    ...
  </faultHandlers>
  <compensationHandler>?
    ...
  </compensationHandler>
  <terminationHandler>?
    ...
  </terminationHandler>
  <eventHandlers>?
    <onEvent partnerLink="NCName"
      portType="QName"?
      operation="NCName"
      ( messageType="QName" | element="
          QName" )?
      variable="BPELVariableName"?
      messageExchange="NCName"?>*
      <correlations>?
        <correlation set="NCName"
          initiate="yes|join|no"? />+
      </correlations>
      <fromParts>?
        <fromPart part="NCName"
          toVariable="BPELVariableName" />+
      </fromParts>
      <scope ...>...</scope>
    </onEvent>
    ...
  </eventHandlers>
  activity
</scope>
```

Listing 65: Core BPEL `<scope>`.

```
<scope isolated="yes|no"
  exitOnStandardFault="yes|no"
  name="NCName">
  <partnerLinks>?
    ...
  </partnerLinks>
  <messageExchanges>?
    ...
  </messageExchanges>
  <variables>?
    <variable name="BPELVariableName"
      messageType="QName"?
      type="QName"?
      element="QName"? />+
  </variables>
  <correlationSets>?
    ...
  </correlationSets>
  <faultHandlers>
    ...
  </faultHandlers>
  <compensationHandler>
    ...
  </compensationHandler>
  <terminationHandler>
    ...
  </terminationHandler>
  <eventHandlers>?
    <onEvent partnerLink="NCName"
      operation="NCName"
      messageType="QName"?
      variable="BPELVariableName"?
      messageExchange="NCName">*
      <correlations>?
        <correlation set="NCName"
          initiate="yes|join|no"? />+
      </correlations>
      <scope ...>...</scope>
    </onEvent>
    ...
  </eventHandlers>
  activity
</scope>
```

Variable initializations are made explicit, standard attributes and elements are moved to a wrapping `<flow>`

(except `name`), and default attribute values and elements are made explicit.

Additionally, for `<onEvent>`, the superfluous `portType` attribute is removed and implicit temporary variables and assignments are made explicit.

Listing 66: WS-BPEL `<sequence>`.

```
<sequence standard-attributes>
  standard-elements
  activity+
</sequence>
```

`<sequence>` is transformed into a `<flow>`.

Listing 67: WS-BPEL `<throw>`.

```
<throw faultName="QName"
  faultVariable="BPELVariableName"?
  standard-attributes>
  standard-elements
</throw>
```

Listing 68: Core BPEL `<throw>`.

```
<throw faultName="QName"
  faultVariable="BPELVariableName"? />
```

Standard attributes and elements are moved to a wrapping `<flow>`.

Listing 69: WS-BPEL `<validate>`.

```
<validate variables="BPELVariableNames"
  standard-attributes>
  standard-elements
</validate>
```

Listing 70: Core BPEL `<validate>`.

```
<validate variables="BPELVariableNames" />
```

Standard attributes and elements are moved to a wrapping `<flow>`.

Listing 71: WS-BPEL `<wait>`.

```
<wait standard-attributes>
  standard-elements
  (
  <for expressionLanguage="anyURI"?>
    duration-expr
  </for>
  |
  <until expressionLanguage="anyURI"?>
    deadline-expr
  </until>
  )
</wait>
```

Listing 72: Core BPEL `<wait>`.

```
<wait>
  (
  <for expressionLanguage="anyURI">
    duration-expr
  </for>
  |
  <until expressionLanguage="anyURI">
    deadline-expr
  </until>
  )
</wait>
```

Standard attributes and elements are moved to a wrapping `<flow>` and default attribute values are made explicit.

Listing 73: WS-BPEL `<while>`.

```
<while standard-attributes>
  standard-elements
  <condition expressionLanguage="anyURI"?>
    bool-expr
  </condition>
  activity
</while>
```

Listing 74: Core BPEL `<while>`.

```
<while>
  <condition expressionLanguage="anyURI">
    bool-expr
  </condition>
  activity
</while>

```

Standard attributes and elements are moved to a wrapping `<flow>` and default attribute values are made explicit.

B XML Schema for Core BPEL

B.1 core-bpel.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  elementFormDefault="qualified" blockDefault="#all">
  <xsd:annotation>
    <xsd:documentation>
      This is the schema for Core BPEL, a subset of WS-BPEL. This
      Schema is a modified version of the original WS-BPEL XML Schema,
      excluding implicit activities, optional extensions and making
      default values mandatory.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/2001/xml.xsd" />
  <xsd:element name="process" type="tProcess">
    <xsd:annotation>
      <xsd:documentation>
        This is the root element for a Core BPEL process.
      </xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  <xsd:complexType name="tProcess">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element ref="extensions" minOccurs="0" />
          <xsd:element ref="import" minOccurs="0"
            maxOccurs="unbounded" />
          <xsd:group ref="activity" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:NCName" use="required" />
        <xsd:attribute name="targetNamespace" type="xsd:anyURI"
          use="required" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="tExtensibleElements">
    <xsd:annotation>
      <xsd:documentation>
        This type is extended by other component types to allow
        elements and attributes from other namespaces to be added at
        the modeled places.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:any namespace="##other" processContents="lax" minOccurs="0"
        maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:complexType>
  <xsd:group name="activity">
    <xsd:annotation>
      <xsd:documentation>
        All Core BPEL activities in alphabetical order.
        Basic activities and structured activities. Addtional
        constraints: - rethrow activity can be used ONLY within a
        fault handler (i.e. "catch" and "catchAll" element) -
        compensate or compensateScope activity can be used ONLY within
        a fault handler, a compensation handler or a termination
        handler
      </xsd:documentation>
    </xsd:annotation>
    <xsd:choice>
      <xsd:element ref="assign"/>
      <xsd:element ref="compensate"/>
      <xsd:element ref="compensateScope"/>
      <xsd:element ref="empty"/>
      <xsd:element ref="exit"/>
      <xsd:element ref="extensionActivity" />
      <xsd:element ref="flow"/>
      <xsd:element ref="forEach"/>
      <xsd:element ref="if"/>
      <xsd:element ref="invoke"/>
      <xsd:element ref="pick"/>
      <xsd:element ref="reply"/>
      <xsd:element ref="rethrow"/>
      <xsd:element ref="scope"/>
      <xsd:element ref="throw"/>
      <xsd:element ref="validate"/>
      <xsd:element ref="wait"/>
      <xsd:element ref="while" />
    </xsd:choice>
  </xsd:group>
  <xsd:element name="extensions" type="tExtensions" />
  <xsd:complexType name="tExtensions">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element ref="extension" maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="extension" type="tExtension" />
  <xsd:complexType name="tExtension">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:attribute name="namespace" type="xsd:anyURI"
          use="required" />
        <xsd:attribute name="mustUnderstand" type="tBoolean"
          fixed="yes" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="import" type="tImport" />
  <xsd:complexType name="tImport">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:attribute name="namespace" type="xsd:anyURI"
          use="optional" />
        <xsd:attribute name="location" type="xsd:anyURI"
          use="optional" />
        <xsd:attribute name="importType" type="xsd:anyURI"
          use="required" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="partnerLinks" type="tPartnerLinks" />
  <xsd:complexType name="tPartnerLinks">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element ref="partnerLink" maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="partnerLink" type="tPartnerLink" />
  <xsd:complexType name="tPartnerLink">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:attribute name="name" type="xsd:NCName" use="required" />
        <xsd:attribute name="partnerLinkType" type="xsd:QName"
          use="required" />
        <xsd:attribute name="myRole" type="xsd:NCName" />
        <xsd:attribute name="partnerRole" type="xsd:NCName" />
        <xsd:attribute name="initializePartnerRole" type="tBoolean" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="messageExchanges" type="tMessageExchanges" />
  <xsd:complexType name="tMessageExchanges">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
```

```
        <xsd:sequence>
          <xsd:element ref="messageExchange" maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="messageExchange" type="tMessageExchange" />
  <xsd:complexType name="tMessageExchange">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:attribute name="name" type="xsd:NCName" use="required" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="variables" type="tVariables" />
  <xsd:complexType name="tVariables">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element ref="variable" maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="variable" type="tVariable" />
  <xsd:complexType name="tVariable">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:attribute name="name" type="BPELVariableName"
          use="required" />
        <xsd:attribute name="messageType" type="xsd:QName"
          use="optional" />
        <xsd:attribute name="type" type="xsd:QName" use="optional" />
        <xsd:attribute name="element" type="xsd:QName" use="optional" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:simpleType name="BPELVariableName">
    <xsd:restriction base="xsd:NCName">
      <xsd:pattern value="[^\.]+" />
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name="correlationSets" type="tCorrelationSets" />
  <xsd:complexType name="tCorrelationSets">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element ref="correlationSet" maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="correlationSet" type="tCorrelationSet" />
  <xsd:complexType name="tCorrelationSet">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:attribute name="properties" type="QNames" use="required" />
        <xsd:attribute name="name" type="xsd:NCName" use="required" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:simpleType name="QNames">
    <xsd:restriction>
      <xsd:simpleType>
        <xsd:list itemType="xsd:QName" />
      </xsd:simpleType>
      <xsd:minLength value="1" />
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name="faultHandlers" type="tFaultHandlers" />
  <xsd:complexType name="tFaultHandlers">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element ref="catch" minOccurs="0"
            maxOccurs="unbounded" />
          <xsd:element ref="catchAll" minOccurs="0" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="catch" type="tCatch">
    <xsd:annotation>
      <xsd:documentation>
        This element can contain all activities including the
        activities compensate, compensateScope and rethrow.
      </xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  <xsd:complexType name="tCatch">
    <xsd:complexContent>
      <xsd:extension base="tActivityContainer">
        <xsd:attribute name="faultName" type="xsd:QName" />
        <xsd:attribute name="faultVariable" type="BPELVariableName" />
        <xsd:attribute name="faultMessageType" type="xsd:QName" />
        <xsd:attribute name="faultElement" type="xsd:QName" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="catchAll" type="tActivityContainer">
    <xsd:annotation>
      <xsd:documentation>
        This element can contain all activities including the
        activities compensate, compensateScope and rethrow.
      </xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  <xsd:complexType name="tActivityContainer">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:group ref="activity" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="eventHandlers" type="tEventHandlers" />
  <xsd:complexType name="tEventHandlers">
    <xsd:annotation>
      <xsd:documentation>
        XSD Authors: The child element onAlarm needs to be a Local
        Element Declaration, because there is another onAlarm element
        defined for the pick activity.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element ref="onEvent" minOccurs="0"
            maxOccurs="unbounded" />
          <xsd:element name="onAlarm" type="tOnAlarmEvent"
            minOccurs="0" maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="onEvent" type="tOnEvent" />
  <xsd:complexType name="tOnEvent">
    <xsd:complexContent>
      <xsd:extension base="tOnMsgCommon">
        <xsd:sequence>
          <xsd:element ref="scope" />
        </xsd:sequence>
        <xsd:attribute name="messageType" type="xsd:QName"
          use="optional" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="tOnMsgCommon">
    <xsd:annotation>
      <xsd:documentation>
        XSD Authors: The child element correlations needs to be a
        Local Element Declaration, because there is another
        correlations element defined for the invoke activity.
```

```
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="correlations" type="tCorrelations"
            minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="partnerLink" type="xsd:NCName"
          use="required" />
        <xsd:attribute name="operation" type="xsd:NCName"
          use="required" />
        <xsd:attribute name="messageExchange" type="xsd:NCName"
          use="required" />
        <xsd:attribute name="variable" type="BPELVariableName"
          use="optional" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="tCorrelations">
    <xsd:annotation>
      <xsd:documentation>
        XSD Authors: The child element correlation needs to be a Local
        Element Declaration, because there is another correlation
        element defined for the invoke activity.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="correlation" type="tCorrelation"
            maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="tCorrelation">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:attribute name="set" type="xsd:NCName" use="required" />
        <xsd:attribute name="initiate" type="tInitiate" use="required" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:simpleType name="tInitiate">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="yes" />
      <xsd:enumeration value="join" />
      <xsd:enumeration value="no" />
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="tOnAlarmEvent">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:choice>
            <xsd:sequence>
              <xsd:group ref="forOrUntilGroup" />
              <xsd:element ref="repeatEvery" minOccurs="0" />
            </xsd:sequence>
            <xsd:element ref="repeatEvery" />
          </xsd:choice>
          <xsd:element ref="scope" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:group name="forOrUntilGroup">
    <xsd:choice>
      <xsd:element ref="for" />
      <xsd:element ref="until" />
    </xsd:choice>
  </xsd:group>
  <xsd:element name="for" type="tDuration-expr" />
  <xsd:element name="until" type="tDeadline-expr" />
  <xsd:element name="repeatEvery" type="tDuration-expr" />
  <xsd:complexType name="tActivity">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="targets" type="tTargets" />
  <xsd:complexType name="tTargets">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element ref="joinCondition" minOccurs="1" />
          <xsd:element ref="target" maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="joinCondition" type="tCondition" />
  <xsd:element name="target" type="tTarget" />
  <xsd:complexType name="tTarget">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:attribute name="linkName" type="xsd:NCName"
          use="required" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="sources" type="tSources" />
  <xsd:complexType name="tSources">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element ref="source" maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="source" type="tSource" />
  <xsd:complexType name="tSource">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element ref="transitionCondition" minOccurs="1" />
        </xsd:sequence>
        <xsd:attribute name="linkName" type="xsd:NCName"
          use="required" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="transitionCondition" type="tCondition" />
  <xsd:element name="assign" type="tAssign" />
  <xsd:complexType name="tAssign">
    <xsd:complexContent>
      <xsd:extension base="tActivity">
        <xsd:sequence>
          <xsd:choice maxOccurs="unbounded">
            <xsd:element ref="copy" />
            <xsd:element ref="extensionAssignOperation" />
          </xsd:choice>
        </xsd:sequence>
        <xsd:attribute name="validate" type="tBoolean" use="required" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="copy" type="tCopy" />
  <xsd:complexType name="tCopy">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element ref="from" />
          <xsd:element ref="to" />
        </xsd:sequence>
        <xsd:attribute name="keepSrcElementName" type="tBoolean"
          use="optional" />
        <xsd:attribute name="ignoreMissingFromData" type="tBoolean"
          use="required" />
      </xsd:extension>
    </xsd:complexContent>
```

```
</xsd:complexType>
<xsd:element name="from" type="tFrom" />
<xsd:complexType name="tFrom" mixed="true">
  <xsd:sequence>
    <xsd:any namespace="##other" processContents="lax" minOccurs="0"
      maxOccurs="unbounded" />
    <xsd:choice minOccurs="0">
      <xsd:element ref="literal" />
      <xsd:element ref="query" />
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
  <xsd:attribute name="variable" type="BPELVariableName" />
  <xsd:attribute name="part" type="xsd:NCName" />
  <xsd:attribute name="property" type="xsd:QName" />
  <xsd:attribute name="partnerLink" type="xsd:NCName" />
  <xsd:attribute name="endpointReference" type="tRoles" />
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>
<xsd:element name="literal" type="tLiteral" />
<xsd:complexType name="tLiteral" mixed="true">
  <xsd:sequence>
    <xsd:any namespace="##any" processContents="lax" minOccurs="0" />
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="query" type="tQuery" />
<xsd:complexType name="tQuery" mixed="true">
  <xsd:sequence>
    <xsd:any processContents="lax" minOccurs="0"
      maxOccurs="unbounded" />
  </xsd:sequence>
  <xsd:attribute name="queryLanguage" type="xsd:anyURI" />
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>
<xsd:simpleType name="tRoles">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="myRole" />
    <xsd:enumeration value="partnerRole" />
  </xsd:restriction>
</xsd:simpleType>
<xsd:element name="to" type="tTo" />
<xsd:complexType name="tTo" mixed="true">
  <xsd:sequence>
    <xsd:any namespace="##other" processContents="lax" minOccurs="0"
      maxOccurs="unbounded" />
    <xsd:element ref="query" minOccurs="0" />
  </xsd:sequence>
  <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
  <xsd:attribute name="variable" type="BPELVariableName" />
  <xsd:attribute name="part" type="xsd:NCName" />
  <xsd:attribute name="property" type="xsd:QName" />
  <xsd:attribute name="partnerLink" type="xsd:NCName" />
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>
<xsd:element name="extensionAssignOperation"
  type="tExtensionAssignOperation" />
<xsd:complexType name="tExtensionAssignOperation">
  <xsd:complexContent>
    <xsd:extension base="tExtensibleElements" />
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="compensate" type="tCompensate" />
<xsd:complexType name="tCompensate">
  <xsd:complexContent>
    <xsd:extension base="tActivity" />
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="compensateScope" type="tCompensateScope" />
<xsd:complexType name="tCompensateScope">
  <xsd:complexContent>
    <xsd:extension base="tActivity">
      <xsd:attribute name="target" type="xsd:NCName" use="required" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="empty" type="tEmpty" />
<xsd:complexType name="tEmpty">
  <xsd:complexContent>
    <xsd:extension base="tActivity" />
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="exit" type="tExit" />
<xsd:complexType name="tExit">
  <xsd:complexContent>
    <xsd:extension base="tActivity" />
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="extensionActivity" type="tExtensionActivity" />
<xsd:complexType name="tExtensionActivity">
  <xsd:sequence>
    <xsd:any namespace="##other" processContents="lax" />
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="flow" type="tFlow" />
<xsd:complexType name="tFlow">
  <xsd:complexContent>
    <xsd:extension base="tActivity">
      <xsd:sequence>
        <xsd:element ref="targets" minOccurs="0" />
        <xsd:element ref="sources" minOccurs="0" />
        <xsd:element ref="links" minOccurs="0" />
        <xsd:group ref="activity" maxOccurs="unbounded" />
      </xsd:sequence>
      <xsd:attribute name="suppressJoinFailure" type="tBoolean" use="required"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="links" type="tLinks" />
<xsd:complexType name="tLinks">
  <xsd:complexContent>
    <xsd:extension base="tExtensibleElements">
      <xsd:sequence>
        <xsd:element ref="link" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="link" type="tLink" />
<xsd:complexType name="tLink">
  <xsd:complexContent>
    <xsd:extension base="tExtensibleElements">
      <xsd:attribute name="name" type="xsd:NCName" use="required" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="forEach" type="tForEach" />
<xsd:complexType name="tForEach">
  <xsd:complexContent>
    <xsd:extension base="tActivity">
      <xsd:sequence>
        <xsd:element ref="startCounterValue" />
        <xsd:element ref="finalCounterValue" />
        <xsd:element ref="completionCondition" minOccurs="0" />
        <xsd:element ref="scope" />
      </xsd:sequence>
      <xsd:attribute name="counterName" type="BPELVariableName"
        use="required" />
      <xsd:attribute name="parallel" type="tBoolean" use="required" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="startCounterValue" type="tExpression" />
<xsd:element name="finalCounterValue" type="tExpression" />
<xsd:element name="completionCondition" type="tCompletionCondition" />
<xsd:complexType name="tCompletionCondition">
  <xsd:complexContent>
    <xsd:extension base="tExtensibleElements">
      <xsd:sequence>
        <xsd:element ref="branches" minOccurs="0" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="branches" type="tBranches" />
```

```
<xsd:complexType name="tBranches">
  <xsd:complexContent>
    <xsd:extension base="tExpression">
      <xsd:attribute name="successfulBranchesOnly" type="tBoolean"
        use="required" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="if" type="tIf" />
<xsd:complexType name="tIf">
  <xsd:complexContent>
    <xsd:extension base="tActivity">
      <xsd:sequence>
        <xsd:element ref="condition" />
        <xsd:group ref="activity" />
        <xsd:element ref="else" minOccurs="1" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="else" type="tActivityContainer" />
<xsd:element name="invoke" type="tInvoke" />
<xsd:complexType name="tInvoke">
  <xsd:annotation>
    <xsd:documentation>
      XSD Authors: The child element correlations needs to be a
      Local Element Declaration, because there is another
      correlations element defined for the non-invoke activities.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="tActivity">
      <xsd:sequence>
        <xsd:element name="correlations"
          type="tCorrelationsWithPattern" minOccurs="0" />
      </xsd:sequence>
      <xsd:attribute name="partnerLink" type="xsd:NCName"
        use="required" />
      <xsd:attribute name="operation" type="xsd:NCName"
        use="required" />
      <xsd:attribute name="inputVariable" type="BPELVariableName"
        use="optional" />
      <xsd:attribute name="outputVariable" type="BPELVariableName"
        use="optional" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="tCorrelationsWithPattern">
  <xsd:annotation>
    <xsd:documentation>
      XSD Authors: The child element correlation needs to be a Local
      Element Declaration, because there is another correlation
      element defined for the non-invoke activities.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="tExtensibleElements">
      <xsd:sequence>
        <xsd:element name="correlation"
          type="tCorrelationWithPattern" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="tCorrelationWithPattern">
  <xsd:complexContent>
    <xsd:extension base="tCorrelation">
      <xsd:attribute name="pattern" type="tPattern" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="tPattern">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="request" />
    <xsd:enumeration value="response" />
    <xsd:enumeration value="request-response" />
  </xsd:restriction>
</xsd:simpleType>

<xsd:element name="pick" type="tPick" />
<xsd:complexType name="tPick">
  <xsd:annotation>
    <xsd:documentation>
      XSD Authors: The child element onAlarm needs to be a Local
      Element Declaration, because there is another onAlarm element
      defined for event handlers.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="tActivity">
      <xsd:sequence>
        <xsd:element ref="onMessage" maxOccurs="unbounded" />
        <xsd:element name="onAlarm" type="tOnAlarmPick"
          minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
      <xsd:attribute name="createInstance" type="tBoolean"
        use="required" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="onMessage" type="tOnMessage" />
<xsd:complexType name="tOnMessage">
  <xsd:complexContent>
    <xsd:extension base="tOnMsgCommon">
      <xsd:sequence>
        <xsd:group ref="activity" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="tOnAlarmPick">
  <xsd:complexContent>
    <xsd:extension base="tExtensibleElements">
      <xsd:sequence>
        <xsd:group ref="forOrUntilGroup" />
        <xsd:group ref="activity" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="reply" type="tReply" />
<xsd:complexType name="tReply">
  <xsd:annotation>
    <xsd:documentation>
      XSD Authors: The child element correlations needs to be a
      Local Element Declaration, because there is another
      correlations element defined for the invoke activity.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="tActivity">
      <xsd:sequence>
        <xsd:element name="correlations" type="tCorrelations"
          minOccurs="0" />
      </xsd:sequence>
      <xsd:attribute name="partnerLink" type="xsd:NCName"
        use="required" />
      <xsd:attribute name="operation" type="xsd:NCName"
        use="required" />
      <xsd:attribute name="variable" type="BPELVariableName"
        use="optional" />
      <xsd:attribute name="faultName" type="xsd:QName" />
      <xsd:attribute name="messageExchange" type="xsd:NCName"
        use="required" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="rethrow" type="tRethrow" />
<xsd:complexType name="tRethrow">
  <xsd:complexContent>
    <xsd:extension base="tActivity" />
  </xsd:complexContent>
</xsd:complexType>
```

```
<xsd:element name="scope" type="tScope" />
<xsd:complexType name="tScope">
  <xsd:complexContent>
    <xsd:extension base="tActivity">
      <xsd:sequence>
        <xsd:element ref="partnerLinks" minOccurs="0" />
        <xsd:element ref="messageExchanges" minOccurs="0" />
        <xsd:element ref="variables" minOccurs="0" />
        <xsd:element ref="correlationSets" minOccurs="0" />
        <xsd:element ref="faultHandlers" minOccurs="1" />
        <xsd:element ref="compensationHandler" minOccurs="1" />
        <xsd:element ref="terminationHandler" minOccurs="1" />
        <xsd:element ref="eventHandlers" minOccurs="0" />
        <xsd:group ref="activity" />
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:NCName" use="required" />
      <xsd:attribute name="isolated" type="tBoolean" use="required" />
      <xsd:attribute name="exitOnStandardFault" type="tBoolean" use="required" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="compensationHandler" type="tActivityContainer">
  <xsd:annotation>
    <xsd:documentation>
      This element can contain all activities including the
      activities compensate and compensateScope.
    </xsd:documentation>
  </xsd:annotation>
</xsd:element>
<xsd:element name="terminationHandler" type="tActivityContainer">
  <xsd:annotation>
    <xsd:documentation>
      This element can contain all activities including the
      activities compensate and compensateScope.
    </xsd:documentation>
  </xsd:annotation>
</xsd:element>
<xsd:element name="throw" type="tThrow" />
<xsd:complexType name="tThrow">
  <xsd:complexContent>
    <xsd:extension base="tActivity">
      <xsd:attribute name="faultName" type="xsd:QName"
        use="required" />
      <xsd:attribute name="faultVariable" type="BPELVariableName" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="validate" type="tValidate" />
<xsd:complexType name="tValidate">
  <xsd:complexContent>
    <xsd:extension base="tActivity">
      <xsd:attribute name="variables" type="BPELVariableNames"
        use="required" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="BPELVariableNames">
  <xsd:restriction>
    <xsd:simpleType>
      <xsd:list itemType="BPELVariableName" />
    </xsd:simpleType>
    <xsd:minLength value="1" />
  </xsd:restriction>
</xsd:simpleType>
<xsd:element name="wait" type="tWait" />
<xsd:complexType name="tWait">
  <xsd:complexContent>
    <xsd:extension base="tActivity">
      <xsd:choice>
        <xsd:element ref="for" />
        <xsd:element ref="until" />
      </xsd:choice>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="while" type="tWhile" />
<xsd:complexType name="tWhile">
  <xsd:complexContent>
    <xsd:extension base="tActivity">
      <xsd:sequence>
        <xsd:element ref="condition" />
        <xsd:group ref="activity" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="tExpression" mixed="true">
  <xsd:sequence>
    <xsd:any processContents="lax" minOccurs="0"
      maxOccurs="unbounded" />
  </xsd:sequence>
  <xsd:attribute name="expressionLanguage" type="xsd:anyURI"
    use="required" />
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>
<xsd:complexType name="tCondition" mixed="true">
  <xsd:complexContent mixed="true">
    <xsd:extension base="tExpression" />
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="condition" type="tBoolean-expr" />
<xsd:complexType name="tBoolean-expr" mixed="true">
  <xsd:complexContent mixed="true">
    <xsd:extension base="tExpression" />
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="tDuration-expr" mixed="true">
  <xsd:complexContent mixed="true">
    <xsd:extension base="tExpression" />
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="tDeadline-expr" mixed="true">
  <xsd:complexContent mixed="true">
    <xsd:extension base="tExpression" />
  </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="tBoolean">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="yes" />
    <xsd:enumeration value="no" />
  </xsd:restriction>
</xsd:simpleType>
</xsd:schema>
```

C Transformation Example

This appendix shows how an example WS-BPEL process, cf. Section C.1, looks after being transformed by our transformations, cf. Section C.2. The WSDL interface for the processes is listed in Section C.3.

C.1 echo.bpel

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<process
  name="echoService"
  targetNamespace="http://beepell.com/samples/echo"
  xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/
      executable"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/
      process/executable"
  xmlns:srv="http://beepell.com/samples/echo/
      definitions"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <import
    namespace="http://beepell.com/samples/echo/
        definitions"
    location="./echo.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/" />

  <partnerLinks>
    <partnerLink
      name="echoPartnerLink"
      partnerLinkType="srv:echoPartnerLinkType"
      myRole="echoServiceProvider" />
  </partnerLinks>

  <variables>
    <variable
      name="message"
      messageType="srv:echoMessage" />
  </variables>

  <sequence>
    <receive name="receive"
      partnerLink="echoPartnerLink"
      operation="echoOperation"
      createInstance="yes"
      variable="message" />
    <reply
      partnerLink="echoPartnerLink"
      operation="echoOperation"
      variable="message" />
  </sequence>
</process>
```

C.2 echo.cbpel

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<bpel:process
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/
      process/executable"
  name="echoService"
  targetNamespace="http://beepell.com/samples/echo">

  <import
    xmlns="http://docs.oasis-open.org/wsbpel/2.0/
        process/executable"
    xmlns:srv="http://beepell.com/samples/echo/
        definitions"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    importType="http://schemas.xmlsoap.org/wsdl/"
    location="./echo.wsdl"
    namespace="http://beepell.com/samples/echo/
        definitions"/>

  <bpel:scope
    xmlns="http://docs.oasis-open.org/wsbpel/2.0/
        process/executable"
    xmlns:srv="http://beepell.com/samples/echo/
        definitions"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    exitOnStandardFault="no"
```

```
21      isolated="no"
22      name="echoService">
23
24      <partnerLinks>
25        <partnerLink
26          myRole="echoServiceProvider"
27          name="echoPartnerLink"
28          partnerLinkType="srv:echoPartnerLinkType"/>
29      </partnerLinks>
30
31      <bpel:messageExchanges>
32        <bpel:messageExchange
33          name="fresh-prefix-0
                N65537DefaultMessageExchange"/>
34      </bpel:messageExchanges>
35
36      <variables>
37        <variable messageType="srv:echoMessage" name="
              message"/>
38      </variables>
39
40      <bpel:faultHandlers>
41        <bpel:catchAll>
42          <bpel:flow
43            name="fresh-prefix-0N65572FreshActivityName"
44            suppressJoinFailure="no">
45
46            <bpel:links>
47              <bpel:link name="fresh-prefix-0
                    N65570FreshSequenceLink"/>
48            </bpel:links>
49
50            <bpel:flow
51              name="fresh-prefix-0N65577FreshActivityName
                    "
52              suppressJoinFailure="no">
53              <bpel:sources>
54                <bpel:source linkName="fresh-prefix-0
                      N65570FreshSequenceLink">
55                  <bpel:transitionCondition
56                    expressionLanguage="urn:oasis:names:
                          tc:wsbpel:2.0:sublang:xpath1.0">
57                      true()
58                    </bpel:transitionCondition>
59                </bpel:source>
60              </bpel:sources>
61              <bpel:compensate/>
62            </bpel:flow>
63
64            <bpel:flow
65              name="fresh-prefix-0N65587FreshActivityName
                    "
66              suppressJoinFailure="no">
67              <bpel:targets>
68                <bpel:joinCondition
69                  expressionLanguage="urn:oasis:names:tc:
                        wsbpel:2.0:sublang:xpath1.0">
70                  $fresh-prefix-0N65570FreshSequenceLink
71                </bpel:joinCondition>
72                <bpel:target
73                  linkName="fresh-prefix-0
                        N65570FreshSequenceLink"/>
74              </bpel:targets>
75              <bpel:rethrow/>
76            </bpel:flow>
77          </bpel:flow>
78        </bpel:catchAll>
79      </bpel:faultHandlers>
80
81      <bpel:compensationHandler>
82        <bpel:compensate/>
83      </bpel:compensationHandler>
84
85      <bpel:terminationHandler>
86        <bpel:compensate/>
87      </bpel:terminationHandler>
88
89      <bpel:flow
90        name="fresh-prefix-0N65603FreshActivityName"
91        suppressJoinFailure="no">
92
93        <bpel:links>
94          <bpel:link name="fresh-prefix-0
                N65578FreshSequenceLink"/>
```

```
    </bpel:links>

    <bpel:flow
      name="fresh-prefix-0N65608FreshActivityName"
      suppressJoinFailure="no">
      <bpel:sources>
        <bpel:source
          linkName="fresh-prefix-0
              N65578FreshSequenceLink">
          <bpel:transitionCondition
            expressionLanguage="urn:oasis:names:tc:
                wsbpel:2.0:sublang:xpath1.0">
            true()
          </bpel:transitionCondition>
        </bpel:source>
      </bpel:sources>

      <bpel:flow
        name="receive" suppressJoinFailure="no">
        <bpel:pick createInstance="yes">
          <bpel:onMessage
            messageExchange="fresh-prefix-0
                N65537DefaultMessageExchange"
            operation="echoOperation"
            partnerLink="echoPartnerLink"
            variable="message">
            <bpel:empty/>
          </bpel:onMessage>
        </bpel:pick>
      </bpel:flow>
    </bpel:flow>

    <bpel:flow
      name="fresh-prefix-0N65627FreshActivityName"
      suppressJoinFailure="no">
      <bpel:targets>
        <bpel:joinCondition
          expressionLanguage="urn:oasis:names:tc:
              wsbpel:2.0:sublang:xpath1.0">
          $fresh-prefix-0N65578FreshSequenceLink
        </bpel:joinCondition>
          <bpel:target linkName="fresh-prefix-0
              N65578FreshSequenceLink"/>
        </bpel:targets>

        <reply
          messageExchange="fresh-prefix-0
              N65537DefaultMessageExchange"
          operation="echoOperation"
          partnerLink="echoPartnerLink"
          variable="message"/>
      </bpel:flow>
    </bpel:flow>
  </bpel:scope>
</bpel:process>
```

C.3 echo.wsdl

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<wsdl:definitions
  targetNamespace="http://beepell.com/samples/echo/
      definitions"
  xmlns:srv="http://beepell.com/samples/proxy/
      definitions"
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/
      plnktype"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <plnk:partnerLinkType name="echoPartnerLinkType">
    <plnk:role name="echoServiceProvider" portType="srv
        :echoPortType" />
  </plnk:partnerLinkType>

  <wsdl:message name="echoMessage">
    <wsdl:part name="content" type="xsd:string" />
  </wsdl:message>

  <wsdl:portType name="echoPortType">
    <wsdl:operation name="echoOperation">
      <wsdl:input message="srv:echoMessage" />
```

```
21        <wsdl:output message="srv:echoMessage" />
22      </wsdl:operation>
23    </wsdl:portType>
24
25  </wsdl:definitions>
```

D XSLT Transformations

D.1 Overview

	File name	Description
D.2	`constants.xsl`	This is a template library with various constants.
D.3	`default-attribute-values-global.xsl`	Make the global default attribute values explicit. Attributes: `expressionLanguage`, `queryLanguage`.
D.4	`default-attribute-values-inherited.xsl`	Make the inherited default attribute values explicit. Attributes: `suppressJoinFailure`, `exitOnStandardFault`.
D.5	`default-attribute-values-simple.xsl`	Make the simple default attribute values explicit. Attributes: `createInstance`, `ignoreMissingFromData`, `initiate`, `isolated`, `succesfulBranchesOnly`, `validate`.
D.6	`default-conditions.xsl`	Make the default join, transition, and completion conditions explicit.
D.7	`default-handlers.xsl`	Make the default fault, compensation, and termination handlers explicit.
D.8	`default-message-exchanges.xsl`	Add default `<messageExchange>` to `<process>`, the immediate `<scope>` of `<onEvent>`, and parallel `<forEach>`. Also, make the use of these default message exchanges explicit.
D.9	`fresh-names.xsl`	Utility templates for generating fresh names.
D.10	`if.xsl`	Tranform `<elseif>`s into `<else><if>`s and add empty branches to the `<if>`s that lack them.
D.11	`invoke.xsl`	Move the `<scope>`-parts of an `<invoke>` into an explicit enclosing `<scope>`. Also, make temporary variables and assignments, due to the use of `<toParts>`, `<fromParts>`, and/or references to element variables, explicit.
D.12	`onEvent.xsl`	Make temporary variables and assignments, due to the use of `<fromParts>`, and/or references to element variables in `<onEvent>`s, explicit.

D.13 `pick.xsl`

Make temporary variables and assignments, due to the use of `<fromParts>`, and/or references to element variables in `<onMessage>`s, explicit.

D.14 `process.xsl`

Move the scope-parts of a `<process>` into an explicit `<scope>`.

D.15 `receive.xsl`

Transform `<receive>` into `<pick>`.

D.16 `remove-documentation.xsl`

Remove all documentation elements.

D.17 `remove-optional-extensions.xsl`

Remove all optional extensions and their declarations.

D.18 `remove-redundant-attributes.xsl`

Remove the redundant attributes (only sound when default attribute values have been made explicit).

D.19 `repeatUntil.xsl`

Transform `<repeatUntil>` into `<while>`.

D.20 `reply.xsl`

Make temporary variables and assignments, due to the use of `<toParts>`, and/or references to an element variable, explicit.

D.21 `scope.xsl`

Make variable initialization explicit in all scopes (including `<process>`) and make implicit variables and assignments in `<onEvent>`s explicit.

D.22 `sequence.xsl`

Transform `<sequence>`s into `<flow>`s.

D.23 `standard-attributes-elements.xsl`

Move `<targets>`, `<sources>`, and `suppressJoinFailure` from activities to a new wrapping `<flow>`, except for activities that have no `<targets>` or `<sources>`, where we push the value of that attribute to all the child activities. Names are also moved from activities (except `<scope>`s) to a new wrapping `<flow>` and fresh names are added to unnamed `<flow>`s and `<scope>`s.

D.24 `to-from-parts-element-variables.xsl`

Utility templates to make temporary variables and assignments, due to the use of `<toParts>`, `<fromParts>`, and/or references to element variables, explicit.

D.2 constants.xsl

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2
3  <!-- This is a template library with various constants. -->
4
5  <xsl:stylesheet version="1.0"
6    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
7    xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">
8
9    <xsl:variable name="xpathURN">urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0</xsl:variable>
10   <xsl:variable name="WS-BPEL-namespace-URI">http://docs.oasis-open.org/wsbpel/2.0/process/
           executable</xsl:variable>
11
12   <!-- Postfixes for fresh names in different contexts. -->
13   <xsl:variable name="default-message-exchange-postfix">DefaultMessageExchange</xsl:variable>
14   <xsl:variable name="fresh-activity-name-postfix">FreshActivityName</xsl:variable>
15   <xsl:variable name="fresh-sequence-link-postfix">FreshSequenceLink</xsl:variable>
16   <xsl:variable name="tmp-condition-variable-postfix">TmpConditionVar</xsl:variable>
17   <xsl:variable name="tmp-input-message-variable-postfix">TmpInputMessageVar</xsl:variable>
18   <xsl:variable name="tmp-output-message-variable-postfix">TmpOutputMessageVar</xsl:variable>
19
20 </xsl:stylesheet>
```

D.3 default-attribute-values-global.xsl

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2
3  <!-- Make the global default attribute values explicit.
4       Attributes: expressionLanguage, queryLanguage -->
5
6  <xsl:stylesheet version="1.0"
7    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
8    xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">
9
10   <xsl:output indent="yes" method="xml" />
11
12   <xsl:include href="constants.xsl" />
13
14   <xsl:template match="*">
15     <xsl:param name="expressionLanguage" />
16     <xsl:param name="queryLanguage" />
17     <xsl:copy>
18       <xsl:copy-of select="@*" />
19       <xsl:apply-templates>
20         <xsl:with-param name="expressionLanguage" select="$expressionLanguage" />
21         <xsl:with-param name="queryLanguage" select="$queryLanguage" />
22       </xsl:apply-templates>
23     </xsl:copy>
24   </xsl:template>
25
26   <xsl:template match="bpel:process">
27     <xsl:copy>
28       <xsl:copy-of select="@*" />
29       <xsl:apply-templates>
30         <xsl:with-param name="expressionLanguage">
31           <xsl:choose>
32             <xsl:when test="@expressionLanguage">
33               <xsl:value-of select="@expressionLanguage" />
34             </xsl:when>
35             <xsl:otherwise>
36               <xsl:value-of select="$xpathURN" />
37             </xsl:otherwise>
38           </xsl:choose>
39         </xsl:with-param>
40         <xsl:with-param name="queryLanguage">
41           <xsl:choose>
42             <xsl:when test="@queryLanguage">
43               <xsl:value-of select="@queryLanguage" />
44             </xsl:when>
45             <xsl:otherwise>
46               <xsl:value-of select="$xpathURN" />
47             </xsl:otherwise>
48           </xsl:choose>
49         </xsl:with-param>
50       </xsl:apply-templates>
51     </xsl:copy>
52   </xsl:template>
53
54   <!-- Adding missing expressionLanguage attributes -->
55   <xsl:template match="bpel:branches | bpel:condition | bpel:finalCounterValue | bpel:for | bpel:
           from[text()] | bpel:joinCondition | bpel:repeatEvery | bpel:startCounterValue | bpel:to[
           text()] | bpel:transitionCondition | bpel:until">
56     <xsl:param name="expressionLanguage" />
57     <xsl:param name="queryLanguage" />
58     <xsl:copy>
59       <xsl:copy-of select="@*" />
60
61       <!-- if expressionLanguage attribute is missing add it -->
62       <xsl:if test="not(@expressionLanguage)">
63         <xsl:attribute name="expressionLanguage">
64           <xsl:value-of select="$expressionLanguage" />
65         </xsl:attribute>
66       </xsl:if>
67
68       <xsl:apply-templates>
69         <xsl:with-param name="expressionLanguage" select="$expressionLanguage" />
70         <xsl:with-param name="queryLanguage" select="$queryLanguage" />
71       </xsl:apply-templates>
72     </xsl:copy>
73   </xsl:template>
74
75   <!-- Adding missing queryLanguge attributes -->
76   <xsl:template match="bpel:query">
77     <xsl:param name="expressionLanguage" />
78     <xsl:param name="queryLanguage" />
79     <xsl:copy>
80       <xsl:copy-of select="@*" />
81
82       <!-- if queryLanguage attribute is missing add it -->
83       <xsl:if test="not(@queryLanguage)">
84         <xsl:attribute name="queryLanguage">
85           <xsl:value-of select="$queryLanguage" />
86         </xsl:attribute>
87       </xsl:if>
88
89       <xsl:apply-templates>
90         <xsl:with-param name="expressionLanguage" select="$expressionLanguage" />
91         <xsl:with-param name="queryLanguage" select="$queryLanguage" />
92       </xsl:apply-templates>
93     </xsl:copy>
94   </xsl:template>
95
96 </xsl:stylesheet>
```

D.4 default-attribute-values-inherited.xsl

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2
3  <!-- Make the inherited default attribute values explicit.
4       Attributes: suppressJoinFailure, exitOnStandardFault -->
5
6  <xsl:stylesheet version="1.0"
7    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
8    xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">
9
10   <xsl:output indent="yes" method="xml" />
11
12   <xsl:template match="bpel:process">
13     <xsl:copy>
14       <xsl:copy-of select="@*" />
15       <xsl:apply-templates>
```

```
        <xsl:with-param name="exitOnStandardFault">
          <xsl:choose>
            <xsl:when test="@exitOnStandardFault">
              <xsl:value-of select="@exitOnStandardFault" />
            </xsl:when>
            <xsl:otherwise>
              <xsl:value-of select="'no'" />
            </xsl:otherwise>
          </xsl:choose>
        </xsl:with-param>
        <xsl:with-param name="suppressJoinFailure">
          <xsl:choose>
            <xsl:when test="@suppressJoinFailure">
              <xsl:value-of select="@suppressJoinFailure" />
            </xsl:when>
            <xsl:otherwise>
              <xsl:value-of select="'no'" />
            </xsl:otherwise>
          </xsl:choose>
        </xsl:with-param>
      </xsl:apply-templates>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="bpel:scope">
    <xsl:param name="exitOnStandardFault" />
    <xsl:param name="suppressJoinFailure" />

    <xsl:copy>
      <xsl:copy-of select="@*" />

      <xsl:if test="not(@exitOnStandardFault)">
        <xsl:attribute name="exitOnStandardFault">
          <xsl:value-of select="$exitOnStandardFault" />
        </xsl:attribute>
      </xsl:if>

      <xsl:if test="not(@suppressJoinFailure)">
        <xsl:attribute name="suppressJoinFailure">
          <xsl:value-of select="$suppressJoinFailure" />
        </xsl:attribute>
      </xsl:if>

      <xsl:apply-templates>
        <xsl:with-param name="exitOnStandardFault">
          <xsl:choose>
            <xsl:when test="@exitOnStandardFault">
              <xsl:value-of select="@exitOnStandardFault" />
            </xsl:when>
            <xsl:otherwise>
              <xsl:value-of select="$exitOnStandardFault" />
            </xsl:otherwise>
          </xsl:choose>
        </xsl:with-param>
        <xsl:with-param name="suppressJoinFailure">
          <xsl:choose>
            <xsl:when test="@suppressJoinFailure">
              <xsl:value-of select="@suppressJoinFailure" />
            </xsl:when>
            <xsl:otherwise>
              <xsl:value-of select="$suppressJoinFailure" />
            </xsl:otherwise>
          </xsl:choose>
        </xsl:with-param>
      </xsl:apply-templates>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="bpel:assign | bpel:compensate | bpel:compensateScope | bpel:empty | bpel:exit
        | bpel:extensionActivity/* | bpel:flow | bpel:forEach | bpel:if | bpel:invoke | bpel:pick
        | bpel:receive | bpel:repeatUntil | bpel:reply | bpel:rethrow | bpel:sequence | bpel:throw
        | bpel:validate | bpel:wait | bpel:while">
    <xsl:param name="exitOnStandardFault" />
    <xsl:param name="suppressJoinFailure" />

    <xsl:copy>
      <xsl:copy-of select="@*" />

      <xsl:if test="not(@suppressJoinFailure)">
        <xsl:attribute name="suppressJoinFailure">
          <xsl:value-of select="$suppressJoinFailure" />
        </xsl:attribute>
      </xsl:if>

      <xsl:apply-templates>
        <xsl:with-param name="exitOnStandardFault" select="$exitOnStandardFault" />
        <xsl:with-param name="suppressJoinFailure">
          <xsl:choose>
            <xsl:when test="@suppressJoinFailure">
              <xsl:value-of select="@suppressJoinFailure" />
            </xsl:when>
            <xsl:otherwise>
              <xsl:value-of select="$suppressJoinFailure" />
            </xsl:otherwise>
          </xsl:choose>
        </xsl:with-param>
      </xsl:apply-templates>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="*">
    <xsl:param name="exitOnStandardFault" />
    <xsl:param name="suppressJoinFailure" />
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates>
        <xsl:with-param name="exitOnStandardFault" select="$exitOnStandardFault" />
        <xsl:with-param name="suppressJoinFailure" select="$suppressJoinFailure" />
      </xsl:apply-templates>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

D.5 default-attribute-values-simple.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Make the simple default attribute values explicit.
     Attributes: createInstance, ignoreMissingFromData, initiate, isolated, succesfulBranchesOnly,
           validate -->

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">

  <xsl:output indent="yes" method="xml" />

  <xsl:template match="*">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>

  <xsl:template match="bpel:pick[not(@createInstance)] | bpel:receive[not(@createInstance)]">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:attribute name="createInstance">no</xsl:attribute>
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>

  <xsl:template match="bpel:copy[not(@ignoreMissingFromData)]">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:attribute name="ignoreMissingFromData">no</xsl:attribute>
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>
```

```
34
35    <xsl:template match="bpel:correlation[not(@initiate)]">
36      <xsl:copy>
37        <xsl:copy-of select="@*" />
38        <xsl:attribute name="initiate">no</xsl:attribute>
39        <xsl:apply-templates />
40      </xsl:copy>
41    </xsl:template>
42
43    <xsl:template match="bpel:scope[not(@isolated)]">
44      <xsl:copy>
45        <xsl:copy-of select="@*" />
46        <xsl:attribute name="isolated">no</xsl:attribute>
47        <xsl:apply-templates />
48      </xsl:copy>
49    </xsl:template>
50
51    <xsl:template match="bpel:branches[not(@successfulBranchesOnly)]">
52      <xsl:copy>
53        <xsl:copy-of select="@*" />
54        <xsl:attribute name="successfulBranchesOnly">no</xsl:attribute>
55        <xsl:apply-templates />
56      </xsl:copy>
57    </xsl:template>
58
59    <xsl:template match="bpel:assign[not(@validate)]">
60      <xsl:copy>
61        <xsl:copy-of select="@*" />
62        <xsl:attribute name="validate">no</xsl:attribute>
63        <xsl:apply-templates />
64      </xsl:copy>
65    </xsl:template>
66
67  </xsl:stylesheet>
```

D.6 default-conditions.xsl

```
1   <?xml version="1.0" encoding="ISO-8859-1"?>
2
3   <!-- Make the default join, transition, and completion conditions explicit. -->
4
5   <xsl:stylesheet version="1.0"
6     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
7     xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">
8
9     <xsl:output indent="yes" method="xml" />
10
11    <xsl:include href="constants.xsl" />
12
13    <xsl:template match="*">
14      <xsl:copy>
15        <xsl:copy-of select="@*" />
16        <xsl:apply-templates />
17      </xsl:copy>
18    </xsl:template>
19
20    <xsl:template match="bpel:source[not(bpel:transitionCondition)]">
21      <xsl:copy>
22        <xsl:copy-of select="@*" />
23        <bpel:transitionCondition expressionLanguage="{$xpathURN}">true()</bpel:transitionCondition>
24      </xsl:copy>
25    </xsl:template>
26
27    <xsl:template match="bpel:targets[not(bpel:joinCondition)]">
28      <xsl:copy>
29        <xsl:copy-of select="@*" />
30        <bpel:joinCondition expressionLanguage="{$xpathURN}">
31          <xsl:for-each select="bpel:target">
32            <xsl:value-of select="concat('$', @linkName)"/>
33            <xsl:if test="following-sibling::bpel:target"> or </xsl:if>
34          </xsl:for-each>
35        </bpel:joinCondition>
36        <xsl:apply-templates />
37      </xsl:copy>
38    </xsl:template>
39
40    <!-- Add empty completion condition where missing -->
41    <xsl:template match="bpel:forEach[not(bpel:completionCondition)]">
42      <xsl:copy>
43        <xsl:copy-of select="@*" />
44        <xsl:apply-templates />
45        <bpel:completionCondition />
46      </xsl:copy>
47    </xsl:template>
48
49  </xsl:stylesheet>
```

D.7 default-handlers.xsl

```
1   <?xml version="1.0" encoding="ISO-8859-1"?>
2
3   <!-- Make the default fault, compensation, and termination handlers explicit. -->
4
5   <xsl:stylesheet version="1.0"
6     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
7     xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">
8
9     <xsl:template match="*">
10      <xsl:copy>
11        <xsl:copy-of select="@*" />
12        <xsl:apply-templates />
13      </xsl:copy>
14    </xsl:template>
15
16    <xsl:template match="bpel:faultHandler[not(bpel:catchAll)]">
17      <xsl:copy>
18        <xsl:apply-templates select="bpel:catch" />
19        <bpel:catchAll>
20          <bpel:sequence>
21            <bpel:compensate />
22            <bpel:rethrow />
23          </bpel:sequence>
24        </bpel:catchAll>
25      </xsl:copy>
26    </xsl:template>
27
28    <xsl:template match="bpel:scope[not(bpel:faultHandlers) or not(bpel:compensationHandler) or not(
          bpel:terminationHandler)]">
29      <xsl:copy>
30        <xsl:copy-of select="@*" />
31        <xsl:apply-templates select="bpel:targets" />
32        <xsl:apply-templates select="bpel:sources" />
33        <xsl:apply-templates select="bpel:partnerLinks" />
34        <xsl:apply-templates select="bpel:messageExchanges" />
35        <xsl:apply-templates select="bpel:variables" />
36        <xsl:apply-templates select="bpel:correlationSets" />
37
38        <xsl:apply-templates select="bpel:faultHandlers" />
39        <xsl:if test="not(bpel:faultHandlers)">
40          <bpel:faultHandlers>
41            <bpel:catchAll>
42              <bpel:sequence>
43                <bpel:compensate />
44                <bpel:rethrow />
45              </bpel:sequence>
46            </bpel:catchAll>
47          </bpel:faultHandlers>
48        </xsl:if>
49
50        <xsl:apply-templates select="bpel:compensationHandler" />
51        <xsl:if test="not(bpel:compensationHandler)">
52          <bpel:compensationHandler>
53            <bpel:compensate />
54          </bpel:compensationHandler>
```

```
      </xsl:if>

      <xsl:apply-templates select="bpel:terminationHandler" />
      <xsl:if test="not(bpel:terminationHandler)">
        <bpel:terminationHandler>
          <bpel:compensate />
        </bpel:terminationHandler>
      </xsl:if>

      <xsl:apply-templates select="bpel:eventHandlers" />

      <xsl:apply-templates select="*[not(
        self::bpel:targets or
        self::bpel:sources or
        self::bpel:partnerLinks or
        self::bpel:messageExchanges or
        self::bpel:variables or
        self::bpel:candrelationSets or
        self::bpel:faultHandlers or
        self::bpel:compensationHandler or
        self::bpel:terminationHandler or
        self::bpel:eventHandlers
      )]" />
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

D.8 default-message-exchanges.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Add default <messageExchange> to <process>, the immediate <scope> of <onEvent>, and parallel <
      forEach> -->
<!-- Also, make the use of these default message exchanges explicit -->

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">

  <xsl:output indent="yes" method="xml" />

  <xsl:include href="constants.xsl" />
  <xsl:include href="fresh-names.xsl" />

  <xsl:template match="*">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>

  <!-- Find out whether the default message of the given activity is ever used. -->
  <xsl:template name="is-default-message-exchange-used">
    <!-- Activity should be either a <process>, <onEvent>, or <forEach parallel="yes">. -->
    <xsl:param name="activity" select="ancestor-or-self::*[self::bpel:process or self::bpel:onEvent
          or self::bpel:forEach[@parallel='yes']][1]" />

    <!-- Does this activity or one of its descendants, which are _not_ inside
         another scope with a default message exchange, use the default message
         exchange? -->
    <xsl:value-of select="0 &lt; count($activity/descendant-or-self::*
                                       [(self::bpel:reply or self::bpel:receive or self::bpel:
                                             onMessage or self::bpel:onEvent) and
                                        not(@messageExchange) and
                                        ancestor-or-self::*[self::bpel:process or self::bpel:
                                              onEvent or self::bpel:forEach[@parallel='yes']][1]
                                              = $activity])" />
  </xsl:template>

  <!-- Construct the default message exchange name for the given scope
       (which defaults to the closest enclosing scope/process). -->

  <xsl:template name="default-message-exchange-name">
    <xsl:param name="scope" select="ancestor-or-self::*[self::bpel:process or self::bpel:scope[
          parent::bpel:onEvent or parent::bpel:forEach[@parallel='yes']]][1]" />
    <xsl:call-template name="unique-element-name">
      <xsl:with-param name="element" select="$scope" />
      <xsl:with-param name="postfix" select="$default-message-exchange-postfix" />
    </xsl:call-template>
  </xsl:template>

  <!-- Create the default message exchange for the given scope
       (which defaults to the closest enclosing scope/process). -->
  <xsl:template name="default-message-exchange">
    <xsl:param name="scope" select="ancestor-or-self::*[self::bpel:process or self::bpel:scope[
          parent::bpel:onEvent or parent::bpel:forEach[@parallel='yes']]][1]" />
    <bpel:messageExchange>
      <xsl:attribute name="name">
        <xsl:call-template name="default-message-exchange-name">
          <xsl:with-param name="scope" select="$scope" />
        </xsl:call-template>
      </xsl:attribute>
    </bpel:messageExchange>
  </xsl:template>

  <!-- Add default message exchanges to the relevant elements that already have an <messageExchanges
        > element -->
  <xsl:template match="bpel:process/bpel:messageExchanges | bpel:onEvent/bpel:scope/bpel:
        messageExchanges | bpel:forEach[@parallel='yes']/bpel:scope/bpel:messageExchanges">
    <xsl:variable name="is-default-message-exchange-used">
      <xsl:choose>
        <xsl:when test="parent::bpel:process">
          <xsl:call-template name="is-default-message-exchange-used">
            <xsl:with-param name="activity" select=".." />
          </xsl:call-template>
        </xsl:when>
        <xsl:otherwise>
          <xsl:call-template name="is-default-message-exchange-used">
            <xsl:with-param name="activity" select="../.." />
          </xsl:call-template>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>

    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:if test="$is-default-message-exchange-used = 'true'">
        <xsl:call-template name="default-message-exchange" />
      </xsl:if>
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>

  <!-- Add <messageExchanges> elements and default message exchanges to the
       relevant elements that does not already have an <messageExchanges> element -->
  <xsl:template match="bpel:process[not(bpel:messageExchanges)]">
    <xsl:variable name="is-default-message-exchange-used">
      <xsl:call-template name="is-default-message-exchange-used">
        <xsl:with-param name="activity" select="." />
      </xsl:call-template>
    </xsl:variable>

    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:copy-of select="bpel:extensions" />
      <xsl:copy-of select="bpel:import" />
      <xsl:copy-of select="bpel:partnerLinks" />

      <xsl:if test="$is-default-message-exchange-used = 'true'">
        <bpel:messageExchanges>
          <xsl:call-template name="default-message-exchange" />
        </bpel:messageExchanges>
      </xsl:if>

      <xsl:apply-templates select="*[not(
        self::bpel:extensions or
        self::bpel:import or
        self::bpel:partnerLinks or
        self::bpel:messageExchanges
```

```
        )]" />
    </xsl:copy>
  </xsl:template>
  <xsl:template match="bpel:onEvent/bpel:scope[not(bpel:messageExchanges)] | bpel:forEach[@parallel
        ='yes']/bpel:scope[not(bpel:messageExchanges)]">
    <xsl:variable name="is-default-message-exchange-used">
      <xsl:call-template name="is-default-message-exchange-used">
        <xsl:with-param name="activity" select=".." />
      </xsl:call-template>
    </xsl:variable>

    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:copy-of select="bpel:partnerLinks" />

      <xsl:if test="$is-default-message-exchange-used = 'true'">
        <bpel:messageExchanges>
          <xsl:call-template name="default-message-exchange" />
        </bpel:messageExchanges>
      </xsl:if>

      <xsl:apply-templates select="*[not(
        self::bpel:partnerLinks or
        self::bpel:messageExchanges
        )]" />
    </xsl:copy>
  </xsl:template>

  <!-- Make the messageExchange attribute explicit. -->
  <xsl:template match="bpel:onMessage[not(@messageExchange)] | bpel:receive[not(@messageExchange)] |
        bpel:reply[not(@messageExchange)]">
    <xsl:copy>
      <xsl:attribute name="messageExchange">
        <xsl:call-template name="default-message-exchange-name" />
      </xsl:attribute>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>
  <xsl:template match="bpel:onEvent[not(@messageExchange)]">
    <xsl:copy>
      <xsl:attribute name="messageExchange">
        <xsl:call-template name="default-message-exchange-name">
          <xsl:with-param name="scope" select="bpel:scope" />
        </xsl:call-template>
      </xsl:attribute>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

D.9 fresh-names.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Utility templates for generating fresh names. -->

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:param name="freshPrefix" />

  <xsl:template name="unique-element-name">
    <xsl:param name="element" select="." />
    <xsl:param name="postfix" select="''" />
    <xsl:value-of select="$freshPrefix" />
    <xsl:value-of select="generate-id($element)" />
    <xsl:value-of select="$postfix" />
  </xsl:template>

  <xsl:template name="attribute-with-unique-element-name">
    <xsl:param name="attributeName" />
    <xsl:param name="element" select="." />
    <xsl:param name="postfix" select="''" />
    <xsl:attribute name="{$attributeName}">
      <xsl:call-template name="unique-element-name">
        <xsl:with-param name="element" select="$element" />
        <xsl:with-param name="postfix" select="$postfix" />
      </xsl:call-template>
    </xsl:attribute>
  </xsl:template>

</xsl:stylesheet>
```

D.10 if.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Tranform <elseif>s into <else><if>s and add empty branches to the <if>s
     that lack them. -->

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">

  <xsl:output indent="yes" method="xml" />

  <xsl:template match="*">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>

  <xsl:template match="bpel:elseif">
    <xsl:param name="if" />
    <xsl:param name="count" />
    <xsl:param name="index" />

    <xsl:choose>
      <xsl:when test="$index &lt; $count">
        <bpel:else>
          <bpel:if>
            <xsl:copy-of select="*" />
            <xsl:apply-templates select="$if/bpel:elseif[$index + 1]">
              <xsl:with-param name="if" select="$if"/>
              <xsl:with-param name="count" select="$count"/>
              <xsl:with-param name="index" select="$index + 1"/>
            </xsl:apply-templates>
          </bpel:if>
        </bpel:else>
      </xsl:when>

      <xsl:otherwise>
        <bpel:else>
          <bpel:if>
            <xsl:copy-of select="*" />

            <xsl:choose>
              <xsl:when test="$if/bpel:else">
                <xsl:copy-of select="$if/bpel:else" />
              </xsl:when>
              <xsl:otherwise>
                <bpel:else>
                  <bpel:empty/>
                </bpel:else>
              </xsl:otherwise>
            </xsl:choose>

          </bpel:if>
        </bpel:else>
      </xsl:otherwise>
```

```
    </xsl:choose>

  </xsl:template>


  <xsl:template match="bpel:if[bpel:elseif]">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:copy-of select="bpel:condition" />
      <xsl:apply-templates select="*[not(self::bpel:condition | self::bpel:else | self::bpel:elseif)]" />
      <xsl:apply-templates select="bpel:elseif[1]">
        <xsl:with-param name="if" select="self::node()"/>
        <xsl:with-param name="count" select="count(bpel:elseif)"/>
        <xsl:with-param name="index" select="1"/>
      </xsl:apply-templates>
    </xsl:copy>
  </xsl:template>


  <xsl:template match="bpel:if[not(bpel:else | bpel:elseif)]">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates />
      <bpel:else>
        <bpel:empty />
      </bpel:else>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

D.11 invoke.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Move the <scope>-parts of an <invoke> into an explicit enclosing <scope>. -->
<!-- Also, make temporary variables and assignments, due to the use of <toParts>,
     <fromParts>, and/or references to element variables, explicit. -->

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">

  <xsl:output indent="yes" method="xml" />

  <xsl:include href="to-from-parts-element-variables.xsl" />

  <!-- Copy all elements and attributes -->
  <xsl:template match="*">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>

  <!-- Unfold invoke -->
  <xsl:template match="bpel:invoke">
    <xsl:variable name="inputVariable" select="@inputVariable" />
    <xsl:variable name="outputVariable" select="@outputVariable" />

    <xsl:variable name="inputElement" select="count(ancestor::*/bpel:variables/bpel:variable[@name=
          $inputVariable][1]/@element) = 1" />
    <xsl:variable name="outputElement" select="count(ancestor::*/bpel:variables/bpel:variable[@name=
          $outputVariable][1]/@element) = 1" />

    <xsl:choose>
      <xsl:when test="bpel:toParts or $inputElement or bpel:fromParts or $outputElement or bpel:
            catch or bpel:catchAll or bpel:compensationHandler">
        <bpel:scope>
          <xsl:if test="bpel:toParts or $inputElement or bpel:fromParts or $outputElement">
            <bpel:variables>
              <xsl:call-template name="message-activities-temp-variables">
                <xsl:with-param name="messageActivities" select="." />
              </xsl:call-template>
            </bpel:variables>
          </xsl:if>

          <xsl:if test="bpel:catch or bpel:catchAll">
            <bpel:faultHandlers>
              <xsl:copy-of select="bpel:catch" />
              <xsl:copy-of select="bpel:catchAll" />
            </bpel:faultHandlers>
          </xsl:if>

          <xsl:copy-of select="bpel:compensationHandler" />

          <xsl:choose>
            <xsl:when test="bpel:toParts or bpel:fromParts or $inputElement or $outputElement">
              <bpel:sequence>
                <!-- Transform toParts into an assignment, if present -->
                <xsl:if test="bpel:toParts">
                  <xsl:call-template name="copy-to-parts-explicitly" />
                </xsl:if>

                <!-- Create assignment to copy element variable to single part -->
                <xsl:if test="$inputElement">
                  <xsl:call-template name="copy-input-element-explicitly">
                    <xsl:with-param name="inputVariable" select="$inputVariable" />
                  </xsl:call-template>
                </xsl:if>

                <xsl:copy>
                  <xsl:copy-of select="@*[not(namespace-uri() = '' and
                                             (local-name() = 'inputVariable' or
                                              local-name() = 'outputVariable' or
                                              local-name() = 'portType'))]" />

                  <xsl:choose>
                    <xsl:when test="bpel:toParts or $inputElement">
                      <xsl:call-template name="attribute-with-unique-element-name">
                        <xsl:with-param name="attributeName" select="'inputVariable'" />
                        <xsl:with-param name="element" select="." />
                        <xsl:with-param name="postfix" select="$tmp-input-message-variable-postfix" />
                      </xsl:call-template>
                    </xsl:when>
                    <xsl:otherwise>
                      <xsl:copy-of select="@inputVariable" />
                    </xsl:otherwise>
                  </xsl:choose>

                  <xsl:choose>
                    <xsl:when test="bpel:fromParts or $outputElement">
                      <xsl:call-template name="attribute-with-unique-element-name">
                        <xsl:with-param name="attributeName" select="'outputVariable'" />
                        <xsl:with-param name="element" select="." />
                        <xsl:with-param name="postfix" select="$tmp-output-message-variable-postfix" />
                      </xsl:call-template>
                    </xsl:when>
                    <xsl:otherwise>
                      <xsl:copy-of select="@outputVariable" />
                    </xsl:otherwise>
                  </xsl:choose>

                  <xsl:apply-templates select="*[not(self::bpel:catch or
                                                     self::bpel:catchAll or
                                                     self::bpel:compensationHandler or
                                                     self::bpel:toParts or
                                                     self::bpel:fromParts)]"/>
                </xsl:copy>

                <!-- Transform fromParts into an assignment, if present -->
                <xsl:if test="bpel:fromParts">
                  <xsl:call-template name="copy-from-parts-explicitly" />
                </xsl:if>

                <!-- Create assignment to copy single part to element variable -->
```

```
                <xsl:if test="$outputElement">
                  <xsl:call-template name="copy-output-element-explicitly">
                    <xsl:with-param name="outputVariable" select="$outputVariable" />
                  </xsl:call-template>
                </xsl:if>
              </bpel:sequence>
            </xsl:when>

            <xsl:otherwise>
              <xsl:copy>
                <xsl:copy-of select="@*[not(namespace-uri() = '' and
                                            local-name() = 'portType')]" />
                <xsl:apply-templates select="*[not(self::bpel:catch or
                                                   self::bpel:catchAll or
                                                   self::bpel:compensationHandler)]"/>
              </xsl:copy>
            </xsl:otherwise>
          </xsl:choose>

        </bpel:scope>
      </xsl:when>

      <xsl:otherwise>
        <xsl:copy>
          <xsl:copy-of select="@*[not(namespace-uri() = '' and
                                      local-name() = 'portType')]" />
          <xsl:apply-templates />
        </xsl:copy>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

</xsl:stylesheet>
```

D.12 onEvent.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Make temporary variables and assignments, due to the use of
     <fromParts>, and/or references to element variables in <onEvent>s, explicit. -->
<!-- This stylesheet is included by scope.xsl and shouldn't be applied alone. -->

<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">

  <xsl:include href="to-from-parts-element-variables.xsl" />

  <xsl:template match="bpel:onEvent[bpel:fromParts or @element]">
    <xsl:copy>
      <xsl:copy-of select="@*[not(namespace-uri() = '' and
                                 (local-name() = 'portType' or
                                  local-name() = 'element'))]" />

      <xsl:attribute name="messageType">
        <xsl:call-template name="inbound-message-type" />
      </xsl:attribute>
      <xsl:call-template name="attribute-with-unique-element-name">
        <xsl:with-param name="attributeName" select="'variable'" />
        <xsl:with-param name="element" select="." />
        <xsl:with-param name="postfix" select="$tmp-output-message-variable-postfix" />
      </xsl:call-template>

      <xsl:apply-templates select="bpel:correlations" />

      <bpel:scope>
        <xsl:copy-of select="bpel:scope/@*" />

        <xsl:apply-templates select="bpel:scope/bpel:partnerLinks | bpel:scope/bpel:messageExchanges
            " />

      <bpel:variables>
        <xsl:for-each select="bpel:fromParts/bpel:fromPart">
          <bpel:variable name="{@toVariable}">
            <xsl:call-template name="inbound-message-part-typing">
              <xsl:with-param name="messageActivity" select="../.." />
              <xsl:with-param name="part" select="@part" />
            </xsl:call-template>
          </bpel:variable>
        </xsl:for-each>
        <xsl:if test="@element">
          <bpel:variable name="{@variable}" element="{@element}" />
        </xsl:if>
        <xsl:apply-templates select="bpel:scope/bpel:variables/*" />
      </bpel:variables>

      <xsl:apply-templates select="bpel:scope/bpel:correlationSets | bpel:scope/bpel:faultHandlers
              | bpel:scope/bpel:compensationHandler | bpel:scope/bpel:terminationHandler | bpel:
            scope/bpel:eventHandlers" />

      <bpel:sequence>
        <!-- Transform fromParts into an assignment, if present -->
        <xsl:if test="bpel:fromParts">
          <xsl:call-template name="copy-from-parts-explicitly" />
        </xsl:if>

        <!-- Create assignment to copy single part to element variable -->
        <xsl:if test="@element">
          <xsl:call-template name="copy-output-element-explicitly">
            <xsl:with-param name="outputVariable" select="@variable" />
          </xsl:call-template>
        </xsl:if>

        <xsl:apply-templates select="bpel:scope/*[not(self::bpel:partnerLinks or self::bpel:
              messageExchanges or self::bpel:correlationSets or self::bpel:faultHandlers or self
              ::bpel:compensationHandler or self::bpel:terminationHandler or self::bpel:
              eventHandlers)]"/>
      </bpel:sequence>
    </bpel:scope>

    <xsl:apply-templates select="*[not(self::bpel:*)]" />
  </xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

D.13 pick.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Make temporary variables and assignments, due to the use of
     <fromParts>, and/or references to element variables in <onMessage>s, explicit. -->

<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">

  <xsl:output indent="yes" method="xml" />

  <xsl:include href="to-from-parts-element-variables.xsl" />

  <!-- Copy all elements and attributes -->
  <xsl:template match="*">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>

  <xsl:template name="onMessage">
    <xsl:variable name="outputVariable" select="@variable" />
```

```
    <xsl:variable name="outputElement" select="count(ancestor::*/bpel:variables/bpel:variable[@name=
          $outputVariable][1]/@element) = 1" />

    <xsl:copy>
      <xsl:copy-of select="@*[not(namespace-uri() = '' and local-name() = 'portType')]" />
      <xsl:copy-of select="bpel:correlations" />

      <xsl:choose>
        <xsl:when test="bpel:fromParts or $outputElement">
          <xsl:call-template name="attribute-with-unique-element-name">
            <xsl:with-param name="attributeName" select="''variable''" />
            <xsl:with-param name="element" select="." />
            <xsl:with-param name="postfix" select="$tmp-output-message-variable-postfix" />
          </xsl:call-template>

          <bpel:sequence>
            <!-- Transform fromParts into an assignment, if present -->
            <xsl:if test="bpel:fromParts">
              <xsl:call-template name="copy-from-parts-explicitly" />
            </xsl:if>

            <!-- Create assignment to copy single part to element variable -->
            <xsl:if test="$outputElement">
              <xsl:call-template name="copy-output-element-explicitly">
                <xsl:with-param name="outputVariable" select="$outputVariable" />
              </xsl:call-template>
            </xsl:if>

            <xsl:apply-templates select="*[not(self::bpel:fromParts or self::bpel:correlations)]"/>

          </bpel:sequence>
        </xsl:when>
        <xsl:otherwise>
          <xsl:apply-templates select="*[not(self::bpel:correlations)]"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="bpel:pick">
    <xsl:variable name="needs-temp-variables">
      <xsl:call-template name="message-activities-need-temp-variables">
        <xsl:with-param name="messageActivities" select="bpel:onMessage" />
      </xsl:call-template>
    </xsl:variable>

    <xsl:choose>
      <xsl:when test="$needs-temp-variables = 'true'">
        <bpel:scope>
          <bpel:variables>
            <xsl:call-template name="message-activities-temp-variables">
              <xsl:with-param name="messageActivities" select="bpel:onMessage" />
            </xsl:call-template>
          </bpel:variables>

          <xsl:copy>
            <xsl:copy-of select="@*" />

            <xsl:for-each select="bpel:onMessage">
              <xsl:call-template name="onMessage" />
            </xsl:for-each>

            <xsl:apply-templates select="*[not(self::bpel:onMessage)]" />
          </xsl:copy>
        </bpel:scope>
       </xsl:when>

      <xsl:otherwise>
        <xsl:copy>
          <xsl:copy-of select="@*" />
          <xsl:apply-templates />
        </xsl:copy>
      </xsl:otherwise>
    </xsl:choose>

  </xsl:template>

</xsl:stylesheet>
```

D.14 process.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Move the scope-parts of a <process> into an explicit <scope>. -->

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">

  <xsl:output indent="yes" method="xml" />

  <xsl:template match="*">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>

  <xsl:template match="bpel:process[@exitOnStandardFault or
                                    @suppressJoinFailure or
                                    bpel:partnerLinks or
                                    bpel:messageExchanges or
                                    bpel:variables or
                                    bpel:correlationSets or
                                    bpel:faultHandlers or
                                    bpel:eventHandlers]">
    <xsl:copy>
      <xsl:copy-of select="@*[not(namespace-uri() = '' and
                                 (local-name() = 'exitOnStandardFault' or
                                  local-name() = 'suppressJoinFailure'))]" />
      <xsl:copy-of select="bpel:extensions" />
      <xsl:copy-of select="bpel:import" />
      <bpel:scope>
        <xsl:copy-of select="@*[not(namespace-uri() = '' and
                                   (local-name() = 'targetNamespace' or
                                    local-name() = 'expressionLanguage' or
                                    local-name() = 'queryLanguage'))]" />
        <xsl:apply-templates select="*[not(self::bpel:import or self::bpel:extensions)]" />
      </bpel:scope>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

D.15 receive.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Transform <receive> into <pick>. -->

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">

  <xsl:output indent="yes" method="xml" />

  <!-- Copy all elements and attributes -->
  <xsl:template match="*">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>
```

```
18
19   <!-- Unfold receive -->
20   <xsl:template match="bpel:receive">
21     <bpel:pick>
22       <xsl:copy-of select="@createInstance" />
23       <xsl:copy-of select="@name" />
24       <xsl:copy-of select="@suppressJoinFailure" />
25       <xsl:apply-templates select="bpel:targets" />
26       <xsl:apply-templates select="bpel:sources" />
27       <bpel:onMessage>
28         <xsl:copy-of select="@*[not(namespace-uri() = '' and
29                                    (local-name() = 'createInstance' or
30                                     local-name() = 'name' or
31                                     local-name() = 'suppressJoinFailure'))]" />
32         <xsl:apply-templates select="*[not(self::bpel:targets or self::bpel:sources)]" />
33         <bpel:empty/>
34       </bpel:onMessage>
35     </bpel:pick>
36   </xsl:template>
37
38 </xsl:stylesheet>
```

D.16 remove-documentation.xsl

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2
3  <!-- Remove any documentation elements. -->
4
5  <xsl:stylesheet version="1.0"
6    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
7    xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">
8
9    <xsl:output indent="yes" method="xml" />
10
11   <xsl:template match="*">
12     <xsl:copy>
13       <xsl:copy-of select="@*" />
14       <xsl:apply-templates />
15     </xsl:copy>
16   </xsl:template>
17
18   <xsl:template match="bpel:documentation" />
19
20 </xsl:stylesheet>
```

D.17 remove-optional-extensions.xsl

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2
3  <!-- Remove all optional extensions and their declarations. -->
4
5  <xsl:stylesheet version="1.0"
6    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
7    xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">
8
9    <xsl:include href="constants.xsl"/>
10
11   <xsl:output indent="yes" method="xml" />
12
13   <!-- Initiate the removal of optional extensions by collecting the set of
14        namespace URIs for mandatory extensions. -->
15   <xsl:template match="bpel:process">
16     <xsl:call-template name="clean-bpel-node">
17       <xsl:with-param name="mandatory-URIs" select="bpel:extensions/bpel:extension[@mustUnderstand='
             yes']/@namespace" />
18     </xsl:call-template>
19   </xsl:template>
20
21   <!-- Remove the <extensions> element if there are no mandatory extensions. -->
22   <xsl:template match="bpel:extensions[not(bpel:extension[@mustUnderstand='yes'])]" />
23
24   <!-- Remove declarations of optional extensions. -->
25   <xsl:template match="bpel:extension[@mustUnderstand='no']" />
26
27   <!-- Replace optional <extensionActivity>s with <empty>. -->
28   <xsl:template match="bpel:extensionActivity">
29     <xsl:param name="mandatory-URIs" />
30
31     <xsl:choose>
32       <xsl:when test="namespace-uri(*[1]) = $mandatory-URIs">
33         <xsl:copy>
34           <xsl:apply-templates select="@*">
35             <xsl:with-param name="mandatory-URIs" select="$mandatory-URIs" />
36           </xsl:apply-templates>
37           <xsl:apply-templates>
38             <xsl:with-param name="mandatory-URIs" select="$mandatory-URIs" />
39           </xsl:apply-templates>
40         </xsl:copy>
41       </xsl:when>
42       <xsl:otherwise>
43         <bpel:empty>
44           <xsl:apply-templates select="*[1]/@*">
45             <xsl:with-param name="mandatory-URIs" select="$mandatory-URIs" />
46           </xsl:apply-templates>
47           <xsl:apply-templates select="*[1]/child::node()">
48             <xsl:with-param name="mandatory-URIs" select="$mandatory-URIs" />
49           </xsl:apply-templates>
50         </bpel:empty>
51       </xsl:otherwise>
52     </xsl:choose>
53   </xsl:template>
54
55   <!-- Replace <assign>s that will be empty after removing optional
56        <extensionAssignOperation>s with <empty>. -->
57   <xsl:template match="bpel:assign">
58     <xsl:param name="mandatory-URIs" />
59
60     <xsl:choose>
61       <xsl:when test="bpel:copy or bpel:extensionAssignOperation/*[1][namespace-uri() = $mandatory-
             URIs]">
62         <xsl:call-template name="clean-bpel-node">
63           <xsl:with-param name="mandatory-URIs" select="$mandatory-URIs" />
64         </xsl:call-template>
65       </xsl:when>
66       <xsl:otherwise>
67         <bpel:empty>
68           <xsl:apply-templates select="@*[not(namespace-uri() = '' and local-name() = 'validate')]">
69             <xsl:with-param name="mandatory-URIs" select="$mandatory-URIs" />
70           </xsl:apply-templates>
71           <xsl:apply-templates>
72             <xsl:with-param name="mandatory-URIs" select="$mandatory-URIs" />
73           </xsl:apply-templates>
74         </bpel:empty>
75       </xsl:otherwise>
76     </xsl:choose>
77   </xsl:template>
78
79   <!-- Remove optional <extensionAssignOperation>s. -->
80   <xsl:template match="bpel:extensionAssignOperation">
81     <xsl:param name="mandatory-URIs" />
82
83     <xsl:if test="namespace-uri(*[1]) = $mandatory-URIs">
84       <xsl:copy>
85         <xsl:apply-templates select="@*">
86           <xsl:with-param name="mandatory-URIs" select="$mandatory-URIs" />
87         </xsl:apply-templates>
88         <xsl:apply-templates>
89           <xsl:with-param name="mandatory-URIs" select="$mandatory-URIs" />
90         </xsl:apply-templates>
91       </xsl:copy>
92     </xsl:if>
93   </xsl:template>
94
```

```
  <!-- Protect <literal>s. -->
  <xsl:template match="bpel:literal">
    <xsl:param name="mandatory-URIs" />

    <xsl:copy-of select="." />
  </xsl:template>

  <!-- BPEL constructs recursively. -->
  <xsl:template match="bpel:*" name="clean-bpel-node">
    <xsl:param name="mandatory-URIs" />

    <xsl:copy>
      <xsl:apply-templates select="@*">
        <xsl:with-param name="mandatory-URIs" select="$mandatory-URIs" />
      </xsl:apply-templates>
      <xsl:apply-templates>
        <xsl:with-param name="mandatory-URIs" select="$mandatory-URIs" />
      </xsl:apply-templates>
    </xsl:copy>
  </xsl:template>

  <!-- Remove extensions that are in an optional namespace, and leave the rest untouched. -->
  <xsl:template match="*">
    <xsl:param name="mandatory-URIs" />

    <xsl:if test="namespace-uri() = $mandatory-URIs">
      <xsl:copy-of select="." />
    </xsl:if>
  </xsl:template>

  <!-- Remove extension attributes that are in an optional namespace. -->
  <xsl:template match="@*">
    <xsl:param name="mandatory-URIs" />

    <xsl:if test="namespace-uri() = '' or namespace-uri() = $mandatory-URIs">
      <xsl:copy />
    </xsl:if>
  </xsl:template>

</xsl:stylesheet>
```

D.18 remove-redundant-attributes.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Remove the redundant attributes. -->

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">

  <xsl:output indent="yes" method="xml" />

  <xsl:template match="*">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>

  <xsl:template match="*" mode="extensionActivity">
      <xsl:copy-of select="@*[not(namespace-uri() = '' and local-name() = 'suppressJoinFailure')]"
          />
      <xsl:copy-of select="*" />
  </xsl:template>

  <xsl:template match="bpel:extensionActivity">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates mode="extensionActivity" />
    </xsl:copy>
  </xsl:template>

  <xsl:template match="bpel:assign | bpel:compensate | bpel:compensateScope | bpel:empty | bpel:exit
        | bpel:forEach | bpel:if | bpel:invoke | bpel:pick | bpel:receive | bpel:repeatUntil |
       bpel:reply | bpel:rethrow | bpel:scope | bpel:sequence | bpel:throw | bpel:validate | bpel:
       wait | bpel:while">
    <xsl:copy>
      <xsl:copy-of select="@*[not(namespace-uri() = '' and
                                 local-name() = 'suppressJoinFailure')]" />
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>

  <xsl:template match="bpel:process">
    <bpel:process>
      <xsl:copy-of select="@*[not(namespace-uri() = '' and
                                 (local-name() = 'queryLanguage' or
                                  local-name() = 'expressionLanguage' or
                                  local-name() = 'suppressJoinFailure' or
                                  local-name() = 'exitOnStandardFault'))]" />
      <xsl:apply-templates />
    </bpel:process>
  </xsl:template>

</xsl:stylesheet>
```

D.19 repeatUntil.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Transform <repeatUntil> into <while>. -->

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsl:output indent="yes" method="xml" />

  <xsl:include href="constants.xsl"/>
  <xsl:include href="fresh-names.xsl"/>

  <xsl:template match="*">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>

  <xsl:template match="bpel:repeatUntil">
    <xsl:variable name="conditionVariable">
      <xsl:call-template name="unique-element-name">
        <xsl:with-param name="postfix" select="$tmp-condition-variable-postfix" />
      </xsl:call-template>
    </xsl:variable>

    <bpel:scope>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates select="bpel:targets | bpel:sources" />

      <bpel:variables>
        <bpel:variable type="xsd:boolean">
          <xsl:attribute name="name">
            <xsl:value-of select="$conditionVariable" />
          </xsl:attribute>
          <bpel:from expressionLanguage="{$xpathURN}">
            true()
          </bpel:from>
        </bpel:variable>
      </bpel:variables>

      <bpel:while>
        <bpel:condition expressionLanguage="{$xpathURN}">
```

```
          $<xsl:value-of select="$conditionVariable" />
        </bpel:condition>
        <bpel:sequence>
          <xsl:apply-templates select="*[not(self::bpel:targets or self::bpel:sources or self::bpel:
              condition)]" />
          <bpel:if>
            <xsl:apply-templates select="bpel:condition" />
            <bpel:assign>
              <bpel:copy>
                <bpel:from expressionLanguage="{$xpathURN}">
                  false()
                </bpel:from>
                <bpel:to variable="{$conditionVariable}" />
              </bpel:copy>
            </bpel:assign>
          </bpel:if>
        </bpel:sequence>
      </bpel:while>
    </bpel:scope>
  </xsl:template>

</xsl:stylesheet>
```

D.20 reply.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Make temporary variables and assignments, due to the use of <toParts>,
     and/or references to an element variable, explicit. -->

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">

  <xsl:output indent="yes" method="xml" />

  <xsl:include href="to-from-parts-element-variables.xsl" />

  <!-- Copy all elements and attributes -->
  <xsl:template match="*">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>

  <!-- Unfold reply -->
  <xsl:template match="bpel:reply">
    <xsl:variable name="inputVariable" select="@variable" />
    <xsl:variable name="inputElement" select="count(ancestor::*/bpel:variables/bpel:variable[@name=
        $inputVariable][1]/@element) = 1" />

    <xsl:choose>
      <xsl:when test="bpel:toParts or $inputElement">
        <bpel:scope>
          <bpel:variables>
            <xsl:call-template name="message-activities-temp-variables">
              <xsl:with-param name="messageActivities" select="." />
            </xsl:call-template>
          </bpel:variables>

          <bpel:sequence>
            <!-- Transform toParts into an assignment, if present -->
            <xsl:if test="bpel:toParts">
              <xsl:call-template name="copy-to-parts-explicitly" />
            </xsl:if>

            <!-- Create assignment to copy element variable to single part -->
            <xsl:if test="$inputElement">
              <xsl:call-template name="copy-input-element-explicitly">
                <xsl:with-param name="inputVariable" select="$inputVariable" />
              </xsl:call-template>
            </xsl:if>

            <xsl:copy>
              <xsl:copy-of select="@*[not(namespace-uri() = '' and
                                          (local-name() = 'variable' or
                                           local-name() = 'portType'))]" />
              <xsl:call-template name="attribute-with-unique-element-name">
                <xsl:with-param name="attributeName" select="'variable'" />
                <xsl:with-param name="element" select="." />
                <xsl:with-param name="postfix" select="$tmp-input-message-variable-postfix" />
              </xsl:call-template>
              <xsl:apply-templates select="*[not(self::bpel:toParts)]" />
            </xsl:copy>
          </bpel:sequence>
        </bpel:scope>
      </xsl:when>

      <xsl:otherwise>
        <xsl:copy>
          <xsl:copy-of select="@*[not(namespace-uri() = '' and local-name() = 'portType')]" />
          <xsl:apply-templates select="*" />
        </xsl:copy>
      </xsl:otherwise>
    </xsl:choose>

  </xsl:template>

</xsl:stylesheet>
```

D.21 scope.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Make variable initialization explicit in all scopes (including <process>). -->
<!-- Make implicit variables and assignments in <onEvent>s explicit. -->

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">

  <xsl:output indent="yes" method="xml" />

  <xsl:include href="onEvent.xsl" />

  <xsl:template match="*">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>

  <xsl:template match="bpel:process[bpel:variables/bpel:variable/bpel:from]">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates select="bpel:extensions | bpel:import" />
      <xsl:call-template name="scope" />
    </xsl:copy>
  </xsl:template>

  <xsl:template match="bpel:scope[bpel:variables/bpel:variable/bpel:from]"
                name="scope">

    <bpel:scope>
      <bpel:variables>
        <xsl:for-each select="bpel:variables/bpel:variable">
          <xsl:copy>
            <xsl:copy-of select="@*"/>
          </xsl:copy>
        </xsl:for-each>
      </bpel:variables>

      <bpel:faultHandlers>
```

```
        <bpel:catchAll>
          <bpel:rethrow/>
        </bpel:catchAll>
      </bpel:faultHandlers>

      <bpel:sequence>
        <bpel:scope>
          <bpel:faultHandlers>
            <bpel:catchAll>
              <bpel:throw faultName='bpel:scopeInitializationFault'/>
            </bpel:catchAll>
          </bpel:faultHandlers>

          <bpel:assign>
            <xsl:for-each select="bpel:variables/bpel:variable[bpel:from]">
              <bpel:copy>
                <xsl:apply-templates select="bpel:from"/>
                <bpel:to>
                  <xsl:attribute name="variable"><xsl:value-of select="@name"/></xsl:attribute>
                </bpel:to>
              </bpel:copy>
            </xsl:for-each>
          </bpel:assign>
        </bpel:scope>

        <bpel:scope>
          <xsl:copy-of select="@*[not(namespace-uri() = '' and
                                     local-name() = 'targetNamespace')]"/>
          <xsl:apply-templates select="*[not(self::bpel:variables or self::bpel:extensions or self::
                bpel:import)]" />
        </bpel:scope>

      </bpel:sequence>
    </bpel:scope>

  </xsl:template>

</xsl:stylesheet>
```

D.22 sequence.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Transform <sequence>s into <flow>s. -->
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">

  <xsl:output indent="yes" method="xml" />

  <xsl:include href="constants.xsl"/>
  <xsl:include href="fresh-names.xsl"/>

  <xsl:template match="*">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>

  <xsl:template match="bpel:sequence">
    <bpel:flow>
      <xsl:copy-of select="@*" />

      <xsl:apply-templates select="bpel:targets | bpel:sources" />

      <xsl:choose>
        <xsl:when test="1 &lt; count(bpel:assign | bpel:compensate | bpel:compensateScope | bpel:
              empty | bpel:exit | bpel:extensionActivity | bpel:flow | bpel:forEach | bpel:if |
              bpel:invoke | bpel:pick | bpel:receive | bpel:repeatUntil | bpel:reply | bpel:rethrow
              | bpel:scope | bpel:sequence | bpel:throw | bpel:validate | bpel:wait | bpel:while)"
            >
          <xsl:call-template name="sequence-links" />
          <xsl:apply-templates select="bpel:assign | bpel:compensate | bpel:compensateScope | bpel:
                empty | bpel:exit | bpel:extensionActivity | bpel:flow | bpel:forEach | bpel:if |
                bpel:invoke | bpel:pick | bpel:receive | bpel:repeatUntil | bpel:reply | bpel:
                rethrow | bpel:scope | bpel:sequence | bpel:throw | bpel:validate | bpel:wait |
                bpel:while"
                              mode="sequenceChild" />
        </xsl:when>

        <xsl:otherwise>
          <xsl:apply-templates select="bpel:assign | bpel:compensate | bpel:compensateScope | bpel:
                empty | bpel:exit | bpel:extensionActivity | bpel:flow | bpel:forEach | bpel:if |
                bpel:invoke | bpel:pick | bpel:receive | bpel:repeatUntil | bpel:reply | bpel:
                rethrow | bpel:scope | bpel:sequence | bpel:throw | bpel:validate | bpel:wait |
                bpel:while" />
        </xsl:otherwise>
      </xsl:choose>

      <xsl:apply-templates select="*[not(self::bpel:targets or self::bpel:sources or self::bpel:
            assign or self::bpel:compensate or self::bpel:compensateScope or self::bpel:empty or
            self::bpel:exit or self::bpel:extensionActivity or self::bpel:flow or self::bpel:
            forEach or self::bpel:if or self::bpel:invoke or self::bpel:pick or self::bpel:receive
            or self::bpel:repeatUntil or self::bpel:reply or self::bpel:rethrow or self::bpel:scope
             or self::bpel:sequence or self::bpel:throw or self::bpel:validate or self::bpel:wait
            or self::bpel:while)]" />

    </bpel:flow>
  </xsl:template>

  <xsl:template name="sequence-links">
    <bpel:links>
      <xsl:for-each select="bpel:assign | bpel:compensate | bpel:compensateScope | bpel:empty | bpel
            :exit | bpel:extensionActivity | bpel:flow | bpel:forEach | bpel:if | bpel:invoke |
            bpel:pick | bpel:receive | bpel:repeatUntil | bpel:reply | bpel:rethrow | bpel:scope |
            bpel:sequence | bpel:throw | bpel:validate | bpel:wait | bpel:while">
        <xsl:if test="following-sibling::bpel:*">
          <bpel:link>
            <xsl:call-template name="attribute-with-unique-element-name">
              <xsl:with-param name="attributeName" select="'name'" />
              <xsl:with-param name="element" select="." />
              <xsl:with-param name="postfix" select="$fresh-sequence-link-postfix" />
            </xsl:call-template>
          </bpel:link>
        </xsl:if>
      </xsl:for-each>
    </bpel:links>
  </xsl:template>

  <!-- Wrap the children of <sequence>s in <flow>s with appropriate links. -->
  <xsl:template match="*" mode="sequenceChild">
    <bpel:flow>
      <xsl:if test="position() &gt; 1">
        <bpel:targets>
          <bpel:target>
            <xsl:call-template name="attribute-with-unique-element-name">
              <xsl:with-param name="attributeName" select="'linkName'" />
              <xsl:with-param name="element" select="preceding-sibling::*[1]" />
              <xsl:with-param name="postfix" select="$fresh-sequence-link-postfix" />
            </xsl:call-template>
          </bpel:target>
        </bpel:targets>
      </xsl:if>

      <xsl:if test="following-sibling::bpel:*">
        <bpel:sources>
          <bpel:source>
            <xsl:call-template name="attribute-with-unique-element-name">
              <xsl:with-param name="attributeName" select="'linkName'" />
              <xsl:with-param name="element" select="." />
              <xsl:with-param name="postfix" select="$fresh-sequence-link-postfix" />
            </xsl:call-template>
          </bpel:source>
        </bpel:sources>
      </xsl:if>

```

```
      <xsl:apply-templates select="." />
    </bpel:flow>
  </xsl:template>

</xsl:stylesheet>
```

D.23 standard-attributes-elements.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Move <targets>, <sources>, and suppressJoinFailure from activities to a
     new wrapping <flow>, except for activities that have no <targets> or
     <sources>, where we push the value of that attribute to all the child
     activities. -->
<!-- Names are also moved from activities (except <scope>s) to a new wrapping
     <flow> and fresh names are added to unnamed <flow>s and <scope>s. -->

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">

  <xsl:output indent="yes" method="xml" />

  <xsl:include href="constants.xsl"/>
  <xsl:include href="fresh-names.xsl" />

  <xsl:template match="*">
    <xsl:param name="suppressJoinFailure" select="false()" />

    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates>
        <xsl:with-param name="suppressJoinFailure" select="$suppressJoinFailure" />
      </xsl:apply-templates>
    </xsl:copy>
  </xsl:template>

  <xsl:template name="name">
    <xsl:param name="name" />

    <xsl:choose>
      <xsl:when test="normalize-space(string($name)) = ''">
        <xsl:call-template name="attribute-with-unique-element-name">
          <xsl:with-param name="attributeName" select="'name'" />
          <xsl:with-param name="element" select="." />
          <xsl:with-param name="postfix" select="$fresh-activity-name-postfix" />
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
        <xsl:copy-of select="$name" />
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <xsl:template match="*" mode="extensionActivity">
      <xsl:copy-of select="@*[not(namespace-uri() = '' and local-name() = 'suppressJoinFailure')]"
          />
      <xsl:copy-of select="*[not(self::bpel:targets or self::bpel:sources)]" />
  </xsl:template>

  <xsl:template match="bpel:extensionActivity[*[1]/bpel:targets or *[1]/bpel:sources or *[1]/
      @suppressJoinFailure]">
    <xsl:param name="suppressJoinFailure" select="false()" />

    <xsl:choose>
      <xsl:when test="not(*[1]/bpel:targets or *[1]/bpel:sources)">
        <xsl:copy>
          <xsl:copy-of select="*" />
          <xsl:apply-templates />
        </xsl:copy>
      </xsl:when>
      <xsl:otherwise>
        <bpel:flow>
          <xsl:copy-of select="*[1]/@suppressJoinFailure" />
          <xsl:if test="not(*[1]/@suppressJoinFailure) and $suppressJoinFailure">
            <xsl:copy-of select="$suppressJoinFailure" />
          </xsl:if>
          <xsl:copy-of select="*[1]/bpel:targets" />
          <xsl:copy-of select="*[1]/bpel:sources" />
          <xsl:copy>
            <xsl:apply-templates mode="extensionActivity" />
          </xsl:copy>
        </bpel:flow>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <xsl:template match="bpel:flow">
    <xsl:param name="suppressJoinFailure" select="false()" />

    <xsl:copy>
      <xsl:copy-of select="@*[not(namespace-uri() = '' and local-name() = 'name')]" />
      <xsl:if test="not(@suppressJoinFailure) and $suppressJoinFailure">
        <xsl:copy-of select="$suppressJoinFailure" />
      </xsl:if>
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>

  <xsl:template match="bpel:scope">
    <xsl:param name="suppressJoinFailure" select="false()" />

    <xsl:choose>
      <xsl:when test="$suppressJoinFailure">
        <xsl:call-template name="scope-with-links">
          <xsl:with-param name="suppressJoinFailure" select="$suppressJoinFailure" />
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
        <xsl:copy>
          <xsl:copy-of select="@*[not(namespace-uri() = '' and local-name() = 'name')]" />
          <xsl:call-template name="name">
            <xsl:with-param name="name" select="@name"/>
          </xsl:call-template>
          <xsl:apply-templates />
        </xsl:copy>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <xsl:template match="bpel:scope[bpel:targets or bpel:sources or @suppressJoinFailure]"
                name="scope-with-links">
    <xsl:param name="suppressJoinFailure" select="false()" />

    <xsl:choose>
      <xsl:when test="not(bpel:targets or bpel:sources)">
        <xsl:copy>
          <xsl:copy-of select="@*[not(namespace-uri() = '' and
                                      (local-name() = 'suppressJoinFailure' or
                                       local-name() = 'name'))]" />
          <xsl:call-template name="name">
            <xsl:with-param name="name" select="@name"/>
          </xsl:call-template>

          <xsl:choose>
            <xsl:when test="@suppressJoinFailure">
              <xsl:apply-templates>
                <xsl:with-param name="suppressJoinFailure" select="@suppressJoinFailure" />
              </xsl:apply-templates>
            </xsl:when>
            <xsl:otherwise>
              <xsl:apply-templates>
                <xsl:with-param name="suppressJoinFailure" select="$suppressJoinFailure" />
              </xsl:apply-templates>
            </xsl:otherwise>
          </xsl:choose>
        </xsl:copy>
      </xsl:when>
      <xsl:otherwise>
```

```
      <bpel:flow>
        <xsl:copy-of select="@suppressJoinFailure" />
        <xsl:if test="not(@suppressJoinFailure) and $suppressJoinFailure">
          <xsl:copy-of select="$suppressJoinFailure" />
        </xsl:if>
        <xsl:copy-of select="bpel:targets" />
        <xsl:copy-of select="bpel:sources" />
        <xsl:copy>
          <xsl:copy-of select="@*[not(namespace-uri() = '' and
                                     (local-name() = 'suppressJoinFailure' or
                                      local-name() = 'name'))]" />
          <xsl:call-template name="name">
            <xsl:with-param name="name" select="@name"/>
          </xsl:call-template>
          <xsl:apply-templates select="*[not(self::bpel:targets or self::bpel:sources)]" />
        </xsl:copy>
      </bpel:flow>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="bpel:assign
                     | bpel:compensate
                     | bpel:compensateScope
                     | bpel:empty
                     | bpel:exit
                     | bpel:forEach
                     | bpel:if
                     | bpel:invoke
                     | bpel:pick
                     | bpel:receive
                     | bpel:repeatUntil
                     | bpel:reply
                     | bpel:rethrow
                     | bpel:sequence
                     | bpel:throw
                     | bpel:validate
                     | bpel:wait
                     | bpel:while">
  <xsl:param name="suppressJoinFailure" select="false()" />

  <xsl:choose>
    <xsl:when test="$suppressJoinFailure">
      <xsl:call-template name="activity-with-links">
        <xsl:with-param name="suppressJoinFailure" select="$suppressJoinFailure" />
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:copy>
        <xsl:copy-of select="@*[not(namespace-uri() = '' and
                                   local-name() = 'name')]" />
        <xsl:apply-templates />
      </xsl:copy>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="bpel:assign[bpel:targets or bpel:sources or @suppressJoinFailure]
                     | bpel:compensate[bpel:targets or bpel:sources or @suppressJoinFailure]
                     | bpel:compensateScope[bpel:targets or bpel:sources or @suppressJoinFailure]
                     | bpel:empty[bpel:targets or bpel:sources or @suppressJoinFailure]
                     | bpel:exit[bpel:targets or bpel:sources or @suppressJoinFailure]
                     | bpel:forEach[bpel:targets or bpel:sources or @suppressJoinFailure]
                     | bpel:if[bpel:targets or bpel:sources or @suppressJoinFailure]
                     | bpel:invoke[bpel:targets or bpel:sources or @suppressJoinFailure]
                     | bpel:pick[bpel:targets or bpel:sources or @suppressJoinFailure]
                     | bpel:receive[bpel:targets or bpel:sources or @suppressJoinFailure]
                     | bpel:repeatUntil[bpel:targets or bpel:sources or @suppressJoinFailure]
                     | bpel:reply[bpel:targets or bpel:sources or @suppressJoinFailure]
                     | bpel:rethrow[bpel:targets or bpel:sources or @suppressJoinFailure]
                     | bpel:sequence[bpel:targets or bpel:sources or @suppressJoinFailure]
                     | bpel:throw[bpel:targets or bpel:sources or @suppressJoinFailure]
                     | bpel:validate[bpel:targets or bpel:sources or @suppressJoinFailure]
                     | bpel:wait[bpel:targets or bpel:sources or @suppressJoinFailure]
                     | bpel:while[bpel:targets or bpel:sources or @suppressJoinFailure]"
                name="activity-with-links">
  <xsl:param name="suppressJoinFailure" select="false()" />

  <xsl:choose>
    <xsl:when test="not(bpel:targets or bpel:sources)">
      <xsl:copy>
        <xsl:copy-of select="@*[not(namespace-uri() = '' and
                                   (local-name() = 'name' or
                                    local-name() = 'suppressJoinFailure'))]" />

        <xsl:choose>
          <xsl:when test="@suppressJoinFailure">
            <xsl:apply-templates>
              <xsl:with-param name="suppressJoinFailure" select="@suppressJoinFailure" />
            </xsl:apply-templates>
          </xsl:when>
          <xsl:otherwise>
            <xsl:apply-templates>
              <xsl:with-param name="suppressJoinFailure" select="$suppressJoinFailure" />
            </xsl:apply-templates>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:copy>
    </xsl:when>
    <xsl:otherwise>
      <bpel:flow>
        <xsl:copy-of select="@suppressJoinFailure" />
        <xsl:if test="not(@suppressJoinFailure) and $suppressJoinFailure">
          <xsl:copy-of select="$suppressJoinFailure" />
        </xsl:if>
        <xsl:copy-of select="bpel:targets" />
        <xsl:copy-of select="bpel:sources" />
        <xsl:copy>
          <xsl:copy-of select="@*[not(namespace-uri() = '' and
                                     (local-name() = 'name' or
                                      local-name() = 'suppressJoinFailure'))]" />
          <xsl:apply-templates select="*[not(self::bpel:targets or self::bpel:sources)]" />
        </xsl:copy>
      </bpel:flow>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

D.24 to-from-parts-element-variables.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Utility templates to make temporary variables and assignments, due to the
     use of <toParts>, <fromParts>, and/or references to element variables,
     explicit. -->
<!-- NB: this template should _not_ be applied by itself. -->

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype">

  <xsl:include href="constants.xsl" />
  <xsl:include href="fresh-names.xsl" />

  <!-- Get the WSDL message name for either the input or output message of the
       operation in question.
       Assumes that the context node is a message activity. -->
  <xsl:template name="wsdl-message-name">
    <!-- If $input = true then the input message type is returned,
         otherwise the output message type is returned. -->
    <xsl:param name="input" />
    <!-- If $myRole = true then the operation is provided by this process,
         otherwise it is provided by the partner. -->
    <xsl:param name="myRole" />

```

```
    <xsl:variable name="partnerLinkName" select="@partnerLink" />
    <xsl:variable name="operation" select="@operation" />
    <xsl:variable name="partnerLink" select="(self::*[self::bpel:invoke or self::bpel:onMessage or
          self::bpel:receive or self::bpel:reply] |
                                             self::bpel:onEvent/bpel:scope)
                                            /ancestor-or-self::*/bpel:partnerLinks/bpel:partnerLink
                                                [@name=$partnerLinkName][1]" />
    <xsl:variable name="partnerLinkNamespace" select="namespace::*[local-name() = substring-before(
          $partnerLink/@partnerLinkType, ':')]" />
    <xsl:variable name="definitions" select="document(/bpel:process/bpel:import[@importType='http://
          schemas.xmlsoap.org/wsdl/']/@location)/wsdl:definitions[@targetNamespace=
          $partnerLinkNamespace]" />
    <xsl:variable name="portType">
      <xsl:choose>
        <xsl:when test="$myRole">
          <xsl:value-of select="substring-after($definitions/plnk:partnerLinkType[@name=substring-
                after($partnerLink/@partnerLinkType, ':')]/plnk:role[@name=$partnerLink/@myRole]/
                @portType, ':')" />
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="substring-after($definitions/plnk:partnerLinkType[@name=substring-
                after($partnerLink/@partnerLinkType, ':')]/plnk:role[@name=$partnerLink/
                @partnerRole]/@portType, ':')" />
        </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>
    <xsl:variable name="message" select="$definitions/wsdl:portType[$input and @name=$portType]/wsdl
          :operation[@name=$operation]/wsdl:input |
                                         $definitions/wsdl:portType[not($input) and @name=$portType
                                             ]/wsdl:operation[@name=$operation]/wsdl:output" />
    <!-- Expand the namespace -->
    <xsl:variable name="messageNamespace" select="$message/namespace::*[local-name() = substring-
          before($message/@message, ':')]" />
    <xsl:variable name="namespacePrefix" select="name(namespace::*[self::node() = $messageNamespace
          ][1])" />
    <xsl:value-of select="concat($namespacePrefix, ':', substring-after($message/@message, ':'))" />
  </xsl:template>

  <!-- Get the WSDL name of _the_ part in either the input or output message of
       the operation in question.
       Assumes that the context node is a message activity. -->
  <xsl:template name="wsdl-message-part-name">
    <!-- If $input = true then the name of the part in the input message type is returned,
         otherwise the output message part name is returned. -->
    <xsl:param name="input" />
    <!-- If $myRole = true then the operation is provided by this process,
         otherwise it is provided by the partner. -->
    <xsl:param name="myRole" />

    <xsl:variable name="partnerLinkName" select="@partnerLink" />
    <xsl:variable name="operation" select="@operation" />
    <xsl:variable name="partnerLink" select="(self::*[self::bpel:invoke or self::bpel:onMessage or
          self::bpel:receive or self::bpel:reply] |
                                             self::bpel:onEvent/bpel:scope)
                                            /ancestor-or-self::*/bpel:partnerLinks/bpel:partnerLink
                                                [@name=$partnerLinkName][1]" />
    <xsl:variable name="partnerLinkNamespace" select="namespace::*[local-name() = substring-before(
          $partnerLink/@partnerLinkType, ':')]" />
    <xsl:variable name="definitions" select="document(/bpel:process/bpel:import[@importType='http://
          schemas.xmlsoap.org/wsdl/']/@location)/wsdl:definitions[@targetNamespace=
          $partnerLinkNamespace]" />
    <xsl:variable name="portType">
      <xsl:choose>
        <xsl:when test="$myRole">
          <xsl:value-of select="substring-after($definitions/plnk:partnerLinkType[@name=substring-
                after($partnerLink/@partnerLinkType, ':')]/plnk:role[@name=$partnerLink/@myRole]/
                @portType, ':')" />
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="substring-after($definitions/plnk:partnerLinkType[@name=substring-
                after($partnerLink/@partnerLinkType, ':')]/plnk:role[@name=$partnerLink/
                @partnerRole]/@portType, ':')" />
        </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>
    <xsl:variable name="message">
      <xsl:choose>
        <xsl:when test="$input">
          <xsl:value-of select="substring-after($definitions/wsdl:portType[@name=$portType]/wsdl:
                operation[@name=$operation]/wsdl:input/@message, ':')" />
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="substring-after($definitions/wsdl:portType[@name=$portType]/wsdl:
                operation[@name=$operation]/wsdl:output/@message, ':')" />
        </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>
    <xsl:value-of select="$definitions/wsdl:message[@name=$message]/wsdl:part/@name" />
  </xsl:template>

  <!-- Get the WSDL typing attribute of the given part in either the input or
       output message of the operation in question.
       By default assumes that the context node is a message activity. -->
  <xsl:template name="wsdl-message-part-typing">
    <xsl:param name="messageActivity" select="." />
    <xsl:param name="part" />
    <!-- If $input = true then the name of the part in the input message type is returned,
         otherwise the output message part name is returned. -->
    <xsl:param name="input" />
    <!-- If $myRole = true then the operation is provided by this process,
         otherwise it is provided by the partner. -->
    <xsl:param name="myRole" />

    <xsl:variable name="partnerLinkName" select="$messageActivity/@partnerLink" />
    <xsl:variable name="operation" select="$messageActivity/@operation" />
    <xsl:variable name="partnerLink" select="($messageActivity/self::*[self::bpel:invoke or self::
          bpel:onMessage or self::bpel:receive or self::bpel:reply] |
                                                $messageActivity/self::bpel:onEvent/bpel:scope)
                                               /ancestor-or-self::*/bpel:partnerLinks/bpel:partnerLink
                                                   [@name=$partnerLinkName][1]" />
    <xsl:variable name="partnerLinkNamespace" select="$messageActivity/namespace::*[local-name() =
          substring-before($partnerLink/@partnerLinkType, ':')]" />
    <xsl:variable name="definitions" select="document(/bpel:process/bpel:import[@importType='http://
          schemas.xmlsoap.org/wsdl/']/@location)/wsdl:definitions[@targetNamespace=
          $partnerLinkNamespace]" />
    <xsl:variable name="portType">
      <xsl:choose>
        <xsl:when test="$myRole">
          <xsl:value-of select="substring-after($definitions/plnk:partnerLinkType[@name=substring-
                after($partnerLink/@partnerLinkType, ':')]/plnk:role[@name=$partnerLink/@myRole]/
                @portType, ':')" />
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="substring-after($definitions/plnk:partnerLinkType[@name=substring-
                after($partnerLink/@partnerLinkType, ':')]/plnk:role[@name=$partnerLink/
                @partnerRole]/@portType, ':')" />
        </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>
    <xsl:variable name="message">
      <xsl:choose>
        <xsl:when test="$input">
          <xsl:value-of select="substring-after($definitions/wsdl:portType[@name=$portType]/wsdl:
                operation[@name=$operation]/wsdl:input/@message, ':')" />
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="substring-after($definitions/wsdl:portType[@name=$portType]/wsdl:
                operation[@name=$operation]/wsdl:output/@message, ':')" />
        </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>
    <xsl:copy-of select="$definitions/wsdl:message[@name=$message]/wsdl:part[@name=$part]/@*[
          namespace-uri() = '' and
                                                                                    (local-
                                                                                        name
                                                                                        ()
                                                                                        =
                                                                                        'type
                                                                                        'or
```

```
local-name() = 'element')]" />
</xsl:template>

<!-- Determine whether the WSDL part has an element attribute in either the
     input or output message of the operation in question.
     By default assumes that the context node is a message activity. -->
<xsl:template name="wsdl-message-part-is-element">
  <xsl:param name="messageActivity" select="." />
  <xsl:param name="part" />
  <!-- If $input = true then the name of the part in the input message type is returned,
       otherwise the output message part name is returned. -->
  <xsl:param name="input" />
  <!-- If $myRole = true then the operation is provided by this process,
       otherwise it is provided by the partner. -->
  <xsl:param name="myRole" />

  <xsl:variable name="partnerLinkName" select="$messageActivity/@partnerLink" />
  <xsl:variable name="operation" select="$messageActivity/@operation" />
  <xsl:variable name="partnerLink" select="(self::*[self::bpel:invoke or self::bpel:onMessage or
        self::bpel:receive or self::bpel:reply] |
                                          self::bpel:onEvent/bpel:scope)
                                          /ancestor-or-self::*/bpel:partnerLinks/bpel:partnerLink
                                                [@name=$partnerLinkName][1]" />
  <xsl:variable name="partnerLinkNamespace" select="$messageActivity/namespace::*[local-name() =
        substring-before($partnerLink/@partnerLinkType, ':')]" />
  <xsl:variable name="definitions" select="document(/bpel:process/bpel:import[@importType='http://
        schemas.xmlsoap.org/wsdl/']/@location)/wsdl:definitions[@targetNamespace=
        $partnerLinkNamespace]" />
  <xsl:variable name="portType">
    <xsl:choose>
      <xsl:when test="$myRole">
        <xsl:value-of select="substring-after($definitions/plnk:partnerLinkType[@name=substring-
              after($partnerLink/@partnerLinkType, ':')]/plnk:role[@name=$partnerLink/@myRole]/
              @portType, ':')" />
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="substring-after($definitions/plnk:partnerLinkType[@name=substring-
              after($partnerLink/@partnerLinkType, ':')]/plnk:role[@name=$partnerLink/
              @partnerRole]/@portType, ':')" />
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>
  <xsl:variable name="message">
    <xsl:choose>
      <xsl:when test="$input">
        <xsl:value-of select="substring-after($definitions/wsdl:portType[@name=$portType]/wsdl:
              operation[@name=$operation]/wsdl:input/@message, ':')" />
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="substring-after($definitions/wsdl:portType[@name=$portType]/wsdl:
              operation[@name=$operation]/wsdl:output/@message, ':')" />
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>
  <xsl:value-of select="boolean($definitions/wsdl:message[@name=$message]/wsdl:part[@name=$part]/
        @element)" />
</xsl:template>


<!-- Figure out what type an activity's outbound message has.
     Assumes that the context node is a message activity. -->
<xsl:template name="outbound-message-type">
  <xsl:choose>
    <xsl:when test="self::bpel:invoke">
      <xsl:call-template name="wsdl-message-name">
        <xsl:with-param name="input" select="true()" />
        <xsl:with-param name="myRole" select="false()" />
      </xsl:call-template>
    </xsl:when>
    <xsl:when test="self::bpel:reply">
      <xsl:call-template name="wsdl-message-name">
        <xsl:with-param name="input" select="false()" />
        <xsl:with-param name="myRole" select="true()" />
      </xsl:call-template>
    </xsl:when>
  </xsl:choose>
</xsl:template>

<!-- Figure out what type an activity's inbound message has.
     Assumes that the context node is a message activity. -->
<xsl:template name="inbound-message-type">
  <xsl:choose>
    <xsl:when test="self::bpel:invoke">
      <xsl:call-template name="wsdl-message-name">
        <xsl:with-param name="input" select="false()" />
        <xsl:with-param name="myRole" select="false()" />
      </xsl:call-template>
    </xsl:when>
    <xsl:when test="self::bpel:onMessage or self::bpel:receive or self::bpel:onEvent">
      <xsl:call-template name="wsdl-message-name">
        <xsl:with-param name="input" select="true()" />
        <xsl:with-param name="myRole" select="true()" />
      </xsl:call-template>
    </xsl:when>
  </xsl:choose>
</xsl:template>

<!-- Figure out what part an activity's outbound message has.
     Assumes that the context node is a message activity. -->
<xsl:template name="outbound-message-part">
  <xsl:choose>
    <xsl:when test="self::bpel:invoke">
      <xsl:call-template name="wsdl-message-part-name">
        <xsl:with-param name="input" select="true()" />
        <xsl:with-param name="myRole" select="false()" />
      </xsl:call-template>
    </xsl:when>
    <xsl:when test="self::bpel:reply">
      <xsl:call-template name="wsdl-message-part-name">
        <xsl:with-param name="input" select="false()" />
        <xsl:with-param name="myRole" select="true()" />
      </xsl:call-template>
    </xsl:when>
  </xsl:choose>
</xsl:template>

<!-- Figure out what part an activity's inbound message has.
     Assumes that the context node is a message activity. -->
<xsl:template name="inbound-message-part">
  <xsl:choose>
    <xsl:when test="self::bpel:invoke">
      <xsl:call-template name="wsdl-message-part-name">
        <xsl:with-param name="input" select="false()" />
        <xsl:with-param name="myRole" select="false()" />
      </xsl:call-template>
    </xsl:when>
    <xsl:when test="self::bpel:onMessage or self::bpel:receive or self::bpel:onEvent">
      <xsl:call-template name="wsdl-message-part-name">
        <xsl:with-param name="input" select="true()" />
        <xsl:with-param name="myRole" select="true()" />
      </xsl:call-template>
    </xsl:when>
  </xsl:choose>
</xsl:template>

<!-- Figure out what typing a named part of an activity's inbound message has.
     By default assumes that the context node is a message activity. -->
<xsl:template name="inbound-message-part-typing">
  <xsl:param name="messageActivity" select="." />
  <xsl:param name="part" />
```

```
    <xsl:choose>
      <xsl:when test="$messageActivity/self::bpel:invoke">
        <xsl:call-template name="wsdl-message-part-typing">
          <xsl:with-param name="messageActivity" select="$messageActivity" />
          <xsl:with-param name="part" select="$part" />
          <xsl:with-param name="input" select="false()" />
          <xsl:with-param name="myRole" select="false()" />
        </xsl:call-template>
      </xsl:when>
      <xsl:when test="$messageActivity/self::bpel:onMessage or $messageActivity/self::bpel:receive
            or $messageActivity/self::bpel:onEvent">
        <xsl:call-template name="wsdl-message-part-typing">
          <xsl:with-param name="messageActivity" select="$messageActivity" />
          <xsl:with-param name="part" select="$part" />
          <xsl:with-param name="input" select="true()" />
          <xsl:with-param name="myRole" select="true()" />
        </xsl:call-template>
      </xsl:when>
    </xsl:choose>
  </xsl:template>

  <!-- Figure out what whether a named part of an activity's inbound message is
       an element.
       By default assumes that the context node is a message activity. -->
  <xsl:template name="inbound-message-part-is-element">
    <xsl:param name="messageActivity" select="." />
    <xsl:param name="part" />
    <xsl:choose>
      <xsl:when test="$messageActivity/self::bpel:invoke">
        <xsl:call-template name="wsdl-message-part-is-element">
          <xsl:with-param name="messageActivity" select="$messageActivity" />
          <xsl:with-param name="part" select="$part" />
          <xsl:with-param name="input" select="false()" />
          <xsl:with-param name="myRole" select="false()" />
        </xsl:call-template>
      </xsl:when>
      <xsl:when test="$messageActivity/self::bpel:onMessage or $messageActivity/self::bpel:receive
            or $messageActivity/self::bpel:onEvent">
        <xsl:call-template name="wsdl-message-part-is-element">
          <xsl:with-param name="messageActivity" select="$messageActivity" />
          <xsl:with-param name="part" select="$part" />
          <xsl:with-param name="input" select="true()" />
          <xsl:with-param name="myRole" select="true()" />
        </xsl:call-template>
      </xsl:when>
    </xsl:choose>
  </xsl:template>

  <!-- Figure out what whether a named part of an activity's outbound message is
       an element.
       By default assumes that the context node is a message activity. -->
  <xsl:template name="outbound-message-part-is-element">
    <xsl:param name="messageActivity" select="." />
    <xsl:param name="part" />
    <xsl:choose>
      <xsl:when test="$messageActivity/self::bpel:invoke">
        <xsl:call-template name="wsdl-message-part-is-element">
          <xsl:with-param name="messageActivity" select="$messageActivity" />
          <xsl:with-param name="part" select="$part" />
          <xsl:with-param name="input" select="true()" />
          <xsl:with-param name="myRole" select="false()" />
        </xsl:call-template>
      </xsl:when>
      <xsl:when test="$messageActivity/self::bpel:reply">
        <xsl:call-template name="wsdl-message-part-is-element">
          <xsl:with-param name="messageActivity" select="$messageActivity" />
          <xsl:with-param name="part" select="$part" />
          <xsl:with-param name="input" select="false()" />
          <xsl:with-param name="myRole" select="true()" />
        </xsl:call-template>
      </xsl:when>
    </xsl:choose>
  </xsl:template>

  <!-- Make explicit that the message parts are copied to a temporary variable.
       Assumes that the context node is a message activity. -->
  <xsl:template name="copy-to-parts-explicitly">
    <xsl:variable name="uniqueElementName">
      <xsl:call-template name="unique-element-name">
        <xsl:with-param name="element" select="." />
      </xsl:call-template>
    </xsl:variable>

    <xsl:for-each select="bpel:toParts">
      <bpel:assign>
        <xsl:for-each select="bpel:toPart">
          <bpel:copy>
            <xsl:variable name="variableName" select="@fromVariable" />

            <!-- Are both source and target elements? -->
            <xsl:variable name="messagePartIsElement">
              <xsl:call-template name="outbound-message-part-is-element">
                <xsl:with-param name="messageActivity" select="../.." />
                <xsl:with-param name="part" select="@part" />
              </xsl:call-template>
            </xsl:variable>
            <xsl:if test="$messagePartIsElement = 'true' and ancestor::*/bpel:variables/bpel:
                  variable[@name=$variableName][1]/@element">
              <xsl:attribute name="keepSrcElementName">yes</xsl:attribute>
            </xsl:if>

            <bpel:from>
              <xsl:attribute name="variable">
                <xsl:value-of select="@fromVariable" />
              </xsl:attribute>
            </bpel:from>
            <bpel:to>
              <xsl:attribute name="variable">
                <xsl:value-of select="concat($uniqueElementName, $tmp-input-message-variable-postfix
                      )" />
              </xsl:attribute>
              <xsl:attribute name="part">
                <xsl:value-of select="@part" />
              </xsl:attribute>
            </bpel:to>
          </bpel:copy>
        </xsl:for-each>
      </bpel:assign>
    </xsl:for-each>
  </xsl:template>

  <!-- Make explicit that the message parts are copied from a temporary variable.
       Assumes that the context node is a message activity. -->
  <xsl:template name="copy-from-parts-explicitly">
    <xsl:variable name="uniqueElementName">
      <xsl:call-template name="unique-element-name">
        <xsl:with-param name="element" select="." />
      </xsl:call-template>
    </xsl:variable>

    <xsl:for-each select="bpel:fromParts">
      <bpel:assign>
        <xsl:for-each select="bpel:fromPart">
          <bpel:copy>
            <xsl:variable name="variableName" select="@toVariable" />

            <!-- Are both source and target elements? -->
            <xsl:variable name="messagePartIsElement">
              <xsl:call-template name="inbound-message-part-is-element">
                <xsl:with-param name="messageActivity" select="../.." />
                <xsl:with-param name="part" select="@part" />
              </xsl:call-template>
            </xsl:variable>
            <xsl:variable name="variableIsElement">
              <xsl:choose>
                <xsl:when test="../../self::bpel:onEvent"><xsl:value-of select="
                      $messagePartIsElement" /></xsl:when>
                <xsl:otherwise><xsl:value-of select="ancestor::*/bpel:variables/bpel:variable[@name=
                      $variableName][1]/@element" /></xsl:otherwise>
              </xsl:choose>
            </xsl:variable>
            <xsl:if test="$messagePartIsElement = 'true' and $variableIsElement = 'true'">
              <xsl:attribute name="keepSrcElementName">yes</xsl:attribute>
            </xsl:if>
```

```
412
413             <bpel:from>
414               <xsl:attribute name="variable">
415                 <xsl:value-of select="concat($uniqueElementName, $tmp-output-message-variable-
                        postfix)" />
416               </xsl:attribute>
417               <xsl:attribute name="part">
418                 <xsl:value-of select="@part" />
419               </xsl:attribute>
420             </bpel:from>
421             <bpel:to>
422               <xsl:attribute name="variable">
423                 <xsl:value-of select="@toVariable" />
424               </xsl:attribute>
425             </bpel:to>
426           </bpel:copy>
427         </xsl:for-each>
428       </bpel:assign>
429     </xsl:for-each>
430   </xsl:template>
431
432   <!-- Make explicit that the input element is copied to the single part of a
433        temporary variable.
434        Assumes that the context node is a message activity. -->
435   <xsl:template name="copy-input-element-explicitly">
436     <xsl:param name="inputVariable" />
437
438     <xsl:variable name="tmpInputMessageName">
439       <xsl:call-template name="unique-element-name">
440         <xsl:with-param name="element" select="." />
441         <xsl:with-param name="postfix" select="$tmp-input-message-variable-postfix" />
442       </xsl:call-template>
443     </xsl:variable>
444
445     <bpel:assign>
446       <bpel:copy keepSrcElementName="yes">
447         <bpel:from>
448           <xsl:attribute name="variable">
449             <xsl:value-of select="$inputVariable" />
450           </xsl:attribute>
451         </bpel:from>
452         <bpel:to>
453           <xsl:attribute name="variable">
454             <xsl:value-of select="$tmpInputMessageName" />
455           </xsl:attribute>
456           <xsl:attribute name="part">
457             <xsl:call-template name="outbound-message-part" />
458           </xsl:attribute>
459         </bpel:to>
460       </bpel:copy>
461     </bpel:assign>
462   </xsl:template>
463
464   <!-- Make explicit that the output element is copied to the single part of a
465        temporary variable.
466        Assumes that the context node is a message activity. -->
467   <xsl:template name="copy-output-element-explicitly">
468     <xsl:param name="outputVariable" />
469
470     <xsl:variable name="tmpOutputMessageName">
471       <xsl:call-template name="unique-element-name">
472         <xsl:with-param name="element" select="." />
473         <xsl:with-param name="postfix" select="$tmp-output-message-variable-postfix" />
474       </xsl:call-template>
475     </xsl:variable>
476
477     <bpel:assign>
478       <bpel:copy keepSrcElementName="yes">
479         <bpel:from>
480           <xsl:attribute name="variable">
481             <xsl:value-of select="$tmpOutputMessageName" />
482           </xsl:attribute>
483           <xsl:attribute name="part">
484             <xsl:call-template name="inbound-message-part" />
485           </xsl:attribute>
486         </bpel:from>
487         <bpel:to>
488           <xsl:attribute name="variable">
489             <xsl:value-of select="$outputVariable" />
490           </xsl:attribute>
491         </bpel:to>
492       </bpel:copy>
493     </bpel:assign>
494   </xsl:template>
495
496   <!-- Create temporary variables for the current messaging activity if it
497        uses <toParts>, <fromParts> or element variables.
498        Assumes that the context node is a message activity. -->
499   <xsl:template name="message-activity-temp-variables">
500     <xsl:param name="inputVariable" />
501     <xsl:param name="outputVariable" />
502
503     <xsl:variable name="uniqueElementName">
504       <xsl:call-template name="unique-element-name">
505         <xsl:with-param name="element" select="." />
506       </xsl:call-template>
507     </xsl:variable>
508
509     <xsl:variable name="inputElement" select="count(ancestor::*/bpel:variables/bpel:variable[@name=
            $inputVariable][1]/@element) = 1" />
510     <xsl:variable name="outputElement" select="count(self::*[self::bpel:invoke or self::bpel:
            onMessage or self::bpel:receive or self::bpel:reply]/ancestor::*/bpel:variables/bpel:
            variable[@name=$outputVariable][1]/@element |
511                                                     self::bpel:onEvent[@element]) = 1" />
512
513     <xsl:if test="bpel:toParts or $inputElement">
514       <bpel:variable>
515         <xsl:attribute name="name">
516           <xsl:value-of select="concat($uniqueElementName, $tmp-input-message-variable-postfix)" />
517         </xsl:attribute>
518         <xsl:attribute name="messageType">
519           <xsl:call-template name="outbound-message-type" />
520         </xsl:attribute>
521       </bpel:variable>
522     </xsl:if>
523     <xsl:if test="bpel:fromParts or $outputElement">
524       <bpel:variable>
525         <xsl:attribute name="name">
526           <xsl:value-of select="concat($uniqueElementName, $tmp-output-message-variable-postfix)" />
527         </xsl:attribute>
528         <xsl:attribute name="messageType">
529           <xsl:call-template name="inbound-message-type" />
530         </xsl:attribute>
531       </bpel:variable>
532     </xsl:if>
533   </xsl:template>
534
535   <!-- Create temporary variables for the given set of message activities. -->
536   <xsl:template name="message-activities-temp-variables">
537     <xsl:param name="messageActivities" />
538
539     <xsl:for-each select="$messageActivities">
540       <xsl:call-template name="message-activity-temp-variables">
541         <xsl:with-param name="inputVariable">
542           <xsl:choose>
543             <xsl:when test="self::bpel:invoke">
544               <xsl:value-of select="@inputVariable" />
545             </xsl:when>
546             <xsl:when test="self::bpel:reply">
547               <xsl:value-of select="@variable" />
548             </xsl:when>
549           </xsl:choose>
550         </xsl:with-param>
551         <xsl:with-param name="outputVariable">
552           <xsl:choose>
553             <xsl:when test="self::bpel:invoke">
554               <xsl:value-of select="@outputVariable" />
555             </xsl:when>
556             <xsl:when test="self::bpel:onMessage or self::bpel:receive or self::bpel:onEvent">
557               <xsl:value-of select="@variable" />
558             </xsl:when>
559           </xsl:choose>
560         </xsl:with-param>
561       </xsl:call-template>
```

```
    </xsl:for-each>
  </xsl:template>

  <!-- Decide whether temporary variables are needed for the current messaging
       activity if it uses <toParts>, <fromParts> or element variables.
       By default, assumes that the context node is a message activity. -->
  <xsl:template name="message-activity-needs-temp-variables">
    <xsl:param name="activity" select="." />
    <xsl:param name="inputVariable" />
    <xsl:param name="outputVariable" />

    <xsl:variable name="inputElement" select="count($activity/ancestor::*/bpel:variables/bpel:
          variable[@name=$inputVariable][1]/@element) = 1" />
    <xsl:variable name="outputElement" select="count($activity/self::*[self::bpel:invoke or self::
          bpel:onMessage or self::bpel:receive or self::bpel:reply]/ancestor::*/bpel:variables/bpel
          :variable[@name=$outputVariable][1]/@element |
                                                $activity/self::bpel:onEvent[@element]) = 1" />

    <xsl:value-of select="$activity/bpel:toParts or $inputElement or $activity/bpel:fromParts or
          $outputElement" />
  </xsl:template>

  <!-- Decide whether temporary variables are needed for the given set of message activities. -->
  <xsl:template name="message-activities-need-temp-variables">
    <xsl:param name="messageActivities" />

    <xsl:choose>
      <xsl:when test="0 = count($messageActivities/*)">
        <xsl:value-of select="false()" />
      </xsl:when>
      <xsl:otherwise>
        <xsl:variable name="first-activity" select="$messageActivities[1]" />
        <xsl:variable name="first-activity-needs-temp-variable">
          <xsl:call-template name="message-activity-needs-temp-variables">
            <xsl:with-param name="activity" select="$first-activity" />
            <xsl:with-param name="inputVariable">
              <xsl:choose>
                <xsl:when test="$first-activity[self::bpel:invoke]">
                  <xsl:value-of select="$first-activity/@inputVariable" />
                </xsl:when>
                <xsl:when test="$first-activity[self::bpel:reply]">
                  <xsl:value-of select="$first-activity/@variable" />
                </xsl:when>
              </xsl:choose>
            </xsl:with-param>
            <xsl:with-param name="outputVariable">
              <xsl:choose>
                <xsl:when test="$first-activity[self::bpel:invoke]">
                  <xsl:value-of select="$first-activity/@outputVariable" />
                </xsl:when>
                <xsl:when test="$first-activity[self::bpel:onMessage or self::bpel:receive]">
                  <xsl:value-of select="$first-activity/@variable" />
                </xsl:when>
              </xsl:choose>
            </xsl:with-param>
          </xsl:call-template>
        </xsl:variable>

        <xsl:choose>
          <xsl:when test="$first-activity-needs-temp-variable = 'true'">
            <xsl:value-of select="true()" />
          </xsl:when>
          <xsl:otherwise>
            <xsl:call-template name="message-activities-need-temp-variables">
              <xsl:with-param name="messageActivities" select="$messageActivities/*[position() &gt;
                    1]" />
            </xsl:call-template>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:otherwise>
    </xsl:choose>

  </xsl:template>

</xsl:stylesheet>
```