From WS-CDL Choreography to BPEL Process Orchestration

Jan Mendling[1] and Michael Hafner[2]

[1] Institute of Information Systems and New Media,
Vienna University of Economics and Business Administration - WU Wien, Austria
jan.mendling@wu-wien.ac.at
[2] Quality Engineering Research Group, Institut für Informatik,
Universität Innsbruck, Austria
m.hafner@uibk.ac.at

Abstract The Web Service Choreography Description Language (WS-CDL) is a specification for describing multi party collaboration based on Web Services from a global point of view. WS-CDL is designed to be used in conjunction with the Web Services Business Process Execution Language (WS-BPEL or BPEL). As WS-CDL is a new choreography language, there has been doubt on the feasibility of a transformation to BPEL. In this article, we show how BPEL process definitions of parties involved in a choreography can be derived from the global WS-CDL model and what the limitations of such a derivation are. We have implemented a prototype of the mappings as a proof of concept. The automatic transformation leverages the quality of software components interacting in the choreography as advocated in the Model Driven Architecture concept. The mapping reveals that some information has to be added manually to the generated BPEL, in particular, choice conditions and private activities.
keywords: Choreography, Orchestration, Web Service, Business Process, Workflow, Inter-organizational Processes

1 Introduction

The exchange of structured information between business partners is a crucial means to facilitate coordinated production of goods and services. The increasing use of Web Services for the implementation of inter-organizational scenarios underlines the need for a choreography description language. Choreography languages are utilized to define the rules of a collaboration between parties without revealing internal operations. They allow to specify when which information is sent to which party and which options are available to continue the interaction.

Several specifications have been proposed for defining choreographies. The Web Service Choreography Description Language (WS-CDL) [1] as the latest proposal is based on a meta model and an XML syntax. It is expected to be used in conjunction with the Web Service Business Process Execution Language (WS-BPEL or BPEL) [2]. There are two application scenarios in this context:

first, business entities may agree on a specific choreography defined in WS-CDL in order to achieve a common goal. This WS-CDL choreography is then used to generate BPEL process stubs for each party. In this case, the WS-CDL choreography may be regarded as a global contract to which all parties commit. Second, a business may want to publish its processes' interface to business partners. In this scenario, a choreography description of the internal process has to be generated.

WS-CDL has been criticized for the insufficient separation of meta-model and syntax, the limited support for certain use case categories and its lack of formal grounding [3]. This was taken as a motivation to identify service interaction patterns [4] that might build the foundation of a new choreography language. Besides, it is not clear whether all WS-CDL concepts can be mapped to BPEL [3]. Our paper discusses mappings between WS-CDL and BPEL. The contribution is twofold. First, the mappings can be used to generate BPEL stubs from WS-CDL choreographies and WS-CDL descriptions from BPEL processes, which leverages the re-use of design artifacts as advocated e.g. in the Model Driven Architecture (MDA) approach. Second, the definition of mappings yields insight into potential incompatibilities of both languages. We implemented the mapping in XSLT transformation programs as a proof of concept.

The rest of the article proceeds as follows. Section 2 introduces the concepts of choreography and orchestration, and gives an overview of related work. In Section 3 we give an example of an inter-organizational workflow based on a real use case. Based on this workflow, we then present the main concepts of WS-CDL and BPEL in Sections 4 and 5. Section 6 defines the mappings between WS-CDL and BPEL, and it discusses how BPEL can be generated from WS-CDL. Finally, Section 7 closes the paper with a conclusion and gives an outlook on future research.

2 Choreography and Orchestration of Web Services

Web Services define interfaces for software components that can be accessed via standard Internet protocols. The Web Service Description Language (WSDL) is used to define, among others, operations and message types of a Web Service. The Simple Object Access Protocol (SOAP) provides the mechanism for concrete message interaction with a Web Service. Web Services are stateless in nature, thus supporting only simple message exchange patterns like request-response. This means that Web Services are well suited for messaging in inter-organizational business processes, but they do not directly offer state control.

A key characteristic of inter-organizational business processes is the fact that they involve multiple parties [5]. These parties perform different parts of the overall process, whereas no one has control over the global state. This is similar to multi-application system business processes [6] whose parts are executed by different autonomous systems within one organization. In an inter-organizational setting, the different parties usually have to balance a need for coordination in order to realize the maximum efficiency with a need for privacy of their value-

creating procedures. In order to deal with this requirement, a distinction is made between three interrelated views on the inter-organizational process: global choreography, local choreography, and orchestration [5].

Choreography refers to the message sequences between different parties in an inter-organizational business process [7]. In [8, p.199] the term conversation is used similarly emphasizing Web Service message exchange. A choreography can be described from a global and a local perspective. The global model of a choreography specifies the message exchanges from an overall point of view. Alonso et al. speak of coordination protocols [8] in this context. The local choreography model defines the message interactions from the perspective of one party. A local model is also referred to as interface process, public process, or abstract process [6]. These two perspectives imply that there are multiple local models (for each party one) that correspond to one global choreography. Several dedicated languages have been proposed for choreography including WSCL [9], WSCI [10] or BPSS [11]. Recently, the World Wide Web Consortium has recommended WS-CDL [1] as a new standard for the specification of global choreographies based on Web Services.

Orchestration refers to an executable part of an inter-organizational process that is provided by one party. This executable process (or integration process [6]) interacts both with external Web Services of the other partners and with internal services. Although it contributes to the successful execution of the global choreography, the state of the orchestration process is controlled locally. Several orchestration languages support the definition of executable processes based on Web Services like XLANG [12], WSFL [13], BPML [14], and BPEL. Also XPDL [15] supports Web Service interaction, therefore, it can be utilized for orchestration, too. Recently, BPEL is getting the most attention from the industry. Therefore, it is likely to become the de facto standard for Web Service orchestration.

Currently, there are several research groups working on general issues related to inter-organizational workflow management systems (e.g. [16,17]). A number of contributions discuss standards for specifying service choreographies (e.g., [3]) and propose formal foundations (e.g., [4,16,18]). A common engineering approach in this context is to achieve an agreement between multiple parties on the global choreography (see e.g. [5]). This global model can be used to derive the local choreography models for each party. Each of them can then implement its own orchestration process that complies with the overall choreography. In this article, we do not aim to contribute a novel approach to this field or to develop a new standard. Instead, focusing on Web Services technology, we use existing technology and standards to realize our vision of Model Driven Architecture in the context of inter-organizational workflows. Our work can be regarded as an instantiation (see [19]) of the engineering approach mentioned above by considering real-world choreography and orchestration standards. In this sense, we evaluate the feasibility of deriving BPEL orchestration processes from WS-CDL choreographies as described e.g. in [5]. We reported on the mapping from WS-CDL to BPEL [20]. A recent extension of this work utilizes a knowledge base to

look up private parts of an orchestration that are not explicitly modelled in the choreography [21,22].

Some work has been reported in model-driven development of Web Service processes. [23] describes an implementation, where a local workflow is modeled in a case-tool, exported via XMI-files to a development environment, and automatically translated into executable code for a BPEL engine based on Web Services. In contrast to this approach, we move up one layer of abstraction and start modeling at the choreography level. This is comparable to the approach promoted in [24] where the authors start with a UML extension to generate BPEL. In [25] we propose an approach for integrating security into the development cycle, starting at the choreography level and show how the requirements map through different levels of abstraction. In [26] we link abstract domain-level models to their technical implementation and show how requirements are realized through security components in a target architecture based on Web Services standards. In the following, we present the SECTINO case in order to motivate the transformation from WS-CDL to BPEL.

3 Example of an Interorganizational Process

This section presents a case study from e-government (Section 3.1). The example is first illustrated using a UML 2.0 Activity Diagram (Section 3.2) and integrates a couple of basic and advanced interaction patterns (Section 3.3).

3.1 The Sectino Case Study

We use an example to illustrate various aspects of the relationship between an externally observable choreography and related internal orchestrations of the collaborating partners' nodes. The example captures an inter-organizational process in e-government. It is drawn from a case that was elaborated within the project SECTINO [27]. The project's vision was defined as the development of a framework supporting the systematic realization of e-government related workflows.

The workflow-scenario "Municipal Tax Collection" describes a collaboration interaction between three participants: a tax-payer (the Client), a business agent (the Tax Advisor) and a public service provider (the Municipality). In Austria, wages paid to employees of an enterprise are subject to the municipal tax. According to the traditional process, corporations have to send an annual statement via their tax advisor to the municipality. The latter is responsible for collecting the tax. It checks the declaration of the annual statement, calculates the tax duties and returns a tax assessment notice to the tax advisor. In our case the stakeholders in this public administration process agreed to implement a new online service, which offers citizens and companies to submit their annual tax statements via internet. Due to various legal considerations, the process had to be realized in a peer-to-peer fashion. The workflow specification should ultimately integrate security requirements like integrity, confidentiality and non-repudiation, but this is not shown here. For a deeper insight into Model Driven

Security in the context of inter-organizational workflows, please refer to a series of accompanying publications [25,26,27,28].

3.2 UML 2.0 Activity Diagrams

In the present contribution, we take WS-CDL and BPEL as special cases for global and local workflow languages respectively. The languages are based on XML and by design heavily skewed towards the underlying implementation technology, which in both cases is the Web Services technology stack. We use UML 2.0 Activity Diagrams to illustrate the example process as it provides a means to gain some insights in limitations inherent to UML and WS-CDL.

Figure 1 shows the choreography model as a UML Activity Diagram. It describes the collaboration of the three services in terms of the interactions in which the parties engage. Model information is confined to directly observable behavior, corresponding to the message flow between the participants, the interaction logic and the control flow between the elementary actions. Roughly sketched, the choreography logic as specified in Figure 1 works as follows: a company representing a role `Client` submits a document of type `AnnualStatement` to a company holding the role `TaxAdvisor`. The latter checks the document's formal compliance according to some criteria or guidelines (e.g., by triggering a manual workflow task) and either rejects the document and notifies the `Client` or forwards it as document of type `ProcessedAnnualStatement` to a public service provider holding the role `Municipality`. The latter stipulates the amount of tax duties and returns the document `TaxAssessment` to the `TaxAdivisor`, who in turn forwards a `ProcessedtaxAssessment` to his `Client`. The stakeholders specified that a `Municipality` has to return the response within a 24 hours delay. We envisage an additional scenario, where the response (`TaxAssessment`) is returned directly to the `Client` instead of sending it to the `TaxAdvisor` (not depicted here).

3.3 Interaction Patterns

Patterns provide the means to capture problem-solving insights and generalize the knowledge inherent in proven and reliable solutions in order to facilitate their re-use. Interaction Patterns, as described in [4], have been especially defined in order to formalize the requirements for choreography languages independent of specific languages or technologies. Our example captures the following basic and advanced interaction patterns:

Basic Interaction Patterns correspond to bilateral interactions where a party sends a message to a receiving party (atomic send), and the message optionally triggers a related response (send/receive or receive/send as its dual).

- Send or Receive: in our example in Figure 1 all parties implement a set of Send and Receive patterns.

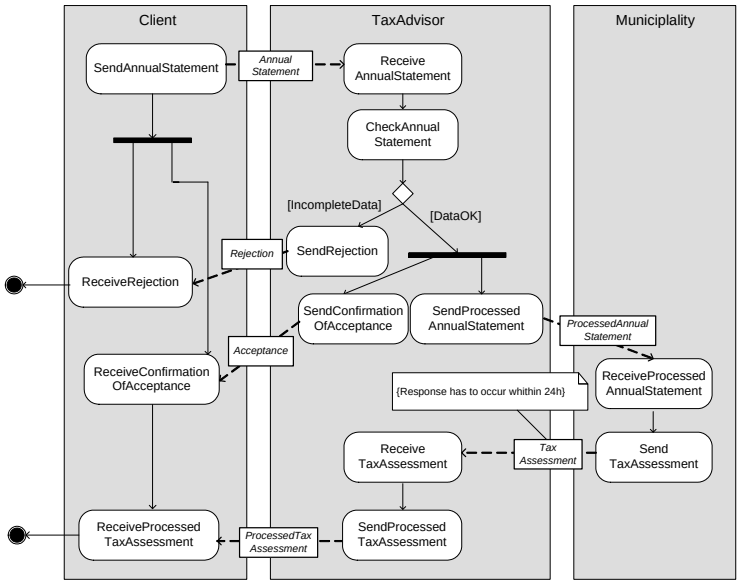

Figure 1. Choreography Model for Municipal Tax Collection

– Send/Receive and Receive/Send patterns relate the two atomic patterns causally. This corresponds to bilateral interaction, showing a document flowing across swim-lanes. As the messages in the choreography all belong to the same instance of the global workflow, a send (receive) pattern that is followed by a receive (send) pattern within the same control flow can implicitly be considered as an instantiation of a send/receive (receive/send) pattern. The case, where a response is sent by a third party corresponds to the pattern *Request with Referral Party*, which is presented in the following.

Advanced Interaction Patterns are composed of basic patterns and specify advanced interaction semantics. Our case study was driven by requirements of a real-life case-study: as a consequence, we had to adapt the scenario to incorporate the following advanced interaction patterns (for a comprehensive survey of patterns, please refer to [4]:

– Contingent Request: the `Municipality` is bound to return the `TaxAssessment` within 24 hours, otherwise an exception is thrown. In UML, this is specified through an informal constraint associated to an object node.
– Request with Referral to a Single Party: in an alternative Scenario not shown in Figure 1, the document `TaxAssessment` should be sent directly to the `Client`, with a `Notification` to the `TaxAdvisor`.

4 An Overview of WS-CDL

In this section, we show how parts of the workflow specifications can be specified with WS-CDL. We first describe the building blocks of a WS-CDL specification corresponding to the choreography in Figure 1 (Section 4.1), we then move on to integrate control-flow activities (Section 4.2) and finally analyze the possibilities to integrate the complex interaction patterns specified in Section 3 (Section 4.3).

4.1 WS-CDL Basic Syntax and Semantics

WS-CDL [1] is a declarative XML-language for the specification of collaboration protocols based on Web Services. It provides a global public view on participants collaborating in a peer-to-peer fashion by offering distributed Web Services in order to achieve a common business goal. The protocol describes their observable behaviour through the order of the messages they exchange as well as the operations they offer. Taking our example "Municipal Tax Collection", Figure 2 shows two parts of a WS-CDL document: the package information and the choreography definition. We only sketch the main concepts, for details, please refer to [1].

Package information: The `package` element is the root of every choreography definition and contains `informationType` definitions for messages and variables (e.g., the document `AnnualStatement` sent from `Client` to `TaxAdvisor` (lines 5-7)) as well as process instance correlation data (lines 2-4). These data types are used within the choreography definition part. A `roleType` represents an actor of the collaboration (e.g., `ServiceProviderRole` (lines 8-11)). This element associates operation names and their WSDL interfaces via the `behavior` element. For example, the `ServiceProviderRole` is expected to implement a `ReceiveAnnualStatement` operation, which is specified in the corresponding WSDL file. The `relationshipType` element pairs two roles and optionally a subset of the behaviour they exhibit: e.g., the `relationshipType ClientTaxAdvisor` associates a `ClientRole` to a `ServiceProviderRole` which is one of two roles a `TaxAdvisor` is expected to implement (lines 16-19). A `participantType` represents a logical grouping of roles. For example, the `participantType TaxAdvisor` implements two roles: on the one hand the `ServiceProviderRole` and on the other hand `ServiceRequesterRole` (lines 20-23). A `channelType` indicates the role the receiver of a message is playing and - optionally - which behaviour he is expected to implement: the `channelType SubmitAnnualStatementChannel` specifies a return channel for responses to a document submission (lines 24-26.). Finally, every package contains one or more choreography definitions (line 37).

Choreography Definition: the core of a collaboration is defined by the element `choreography`, which specifies a set of peer-to-peer interactions. A package can contain one or more choreographies, one being the root choreography (lines 1-2). A choreography first specifies relationships. The example shows two relationships: one between a `TaxAdvisor` and his `Client`, and one between the `TaxAdvisor` and a `Municipality` (lines 3-4).

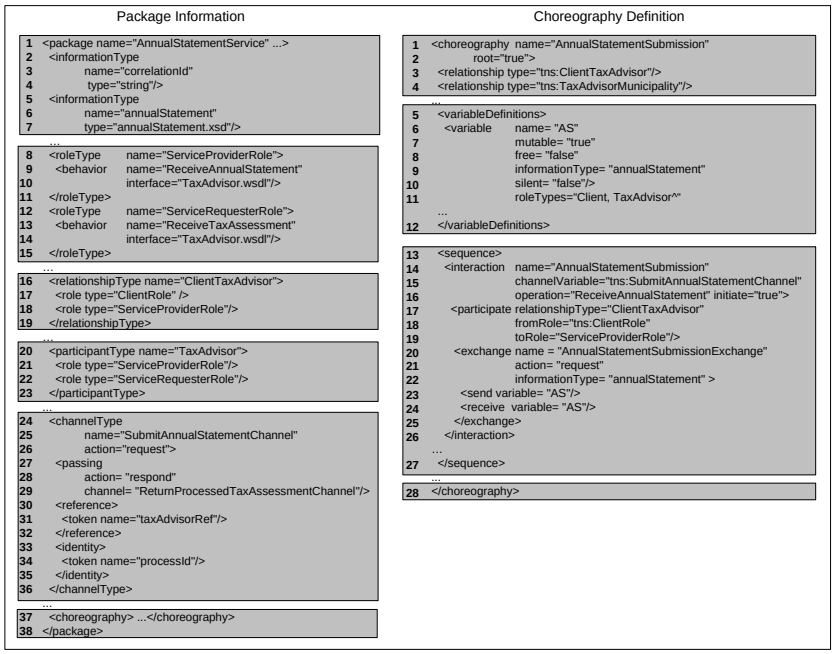

Figure 2. WS-CDL Listings (Basic Constructs)

In a second step, the variables are declared, e.g., the `variable AS` of type `AnnualStatement` will only be used by the `ClientRole` and the `ServiceProviderRole`. The `interaction` element is the building block of communication. It participates in a `relationshipType` and specifies the direction of the message flow: the message flows from the sender (specified as `fromRole`) to the receiver (`ToRole`) if the action attribute of the exchange element is set to `request`. The `exchange` element also captures the name of the operation associated to this interaction. `Interaction` elements can be nested within control-flow activities (e.g., `sequence`, Fig. 2 line 13-27).

4.2 Work-Unit and Control-Flow Activities

A `workunit` activity describes the conditional execution of an activity. An enclosed activity is executed whenever the `guard` condition evaluates to true and can repeatedly be executed by setting a corresponding `repetition` condition. A `workunit` is called blocked, whenever it is bound to wait for the variables to be available before evaluation. In our example, we use `workunit` activites in conjunction with other control-flow activites (e.g., `choice`).

Control-Flow activities can be of three types `sequence`, `parallel` and `choice`. A `sequence` activity describes the execution of two or more activities in sequential order (e.g., Fig. 3, lines 1 and 37). A `parallel` activity describes two or more activities that can be executed in parallel, whereas a `choice` activity

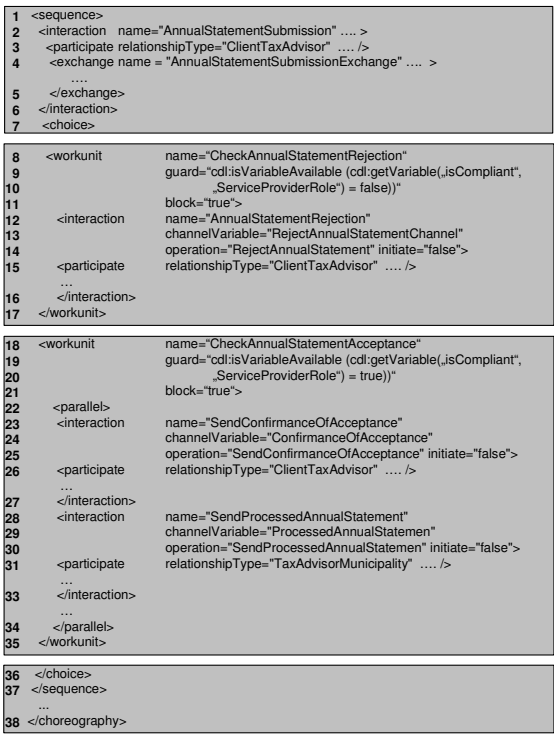

Figure 3. Data Driven Choice (WS-CDL)

defines the execution of a specific activity, depending on the evaluation of data variables or the occurrence of a specific event.

The specification in Figure 1 requires the integration of a *Data-driven Choice Activity*: the `TaxAdvisor` has to decide whether to reject the `AnnualStatement` and return it to the `Client` (by calling the appropriate method on the `Client`'s interface) or to forward it to the `Municipality`. The decision should be based on the evaluation of data variables. The code in Figure 3 shows one of many ways how this decision could be specified in WS-CDL. Each branch was captured in a `workunit` of its own (lines 8-17 and 18-35). We make use of the WS-CDL functions `isVariableAvailable` (e.g., line 9) and `getVariable` (e.g., line 9) to get variable information for the `guard` condition. Based on the assumption that the evaluation task in the `TaxAdvisor`'s application environment would return the boolean variable `isCompliant` as a result of some compliance check, either a notification of rejection (`Rejection`) is sent to the `Client` or the document is forwarded to the `Municipality` and the `Client` simply gets a notification of acceptance (`Acceptance`).

4.3 Advanced Interaction patterns

Contingent Request. For the specification of the requirement that the `Municipality` has to answer within a 24 hours delay, we make use of the same functions as in the previous section and additionally use the WS-CDL function `getCurrentTime()`. The code in Figure 4 shows the the contingent request as a choice between two `workunit`s.

```
<choice>
  <workunit>
                    name="ResponseInTime"
                    guard="cdl:getVariable(„TimeStamp", „/TaxAssessment/TimeStamp",
                      „MunicipalityRole") < cdl:getVariable(„TimeStamp", „/SendAnnualStatement/
                      TimeStamp", „TaxAdvisorRole") + 24:00:00)"
                    block="true">
    <interaction    name="SendProcessedTaxAssessment" ... >
      <participate  relationshipType="ClientTaxAdvisor"  .... />
                    fromRole="tns:TaxAdvisorRole"
                    toRole="ClientRole"/>
                     ....
    </interaction>
  </workunit>
  <workunit>
                    name="ResponseOutOfTime"
                    guard="cdl:getCurrentTime > cdl:getVariable(„TimeStamp", „SendAnnualStatement/
                      TimeStamp", „TaxAdvisorRole") + 24:00:00)"
                    block="true">
    <interaction    name = "DelayException" ...>
      <participate  relationshipType="ClientTaxAdvisor"  .... />
                    fromRole="tns:TaxAdvisorRole"
                      toRole="ClientRole"/>
                     ....
    </interaction>
  </workunit>
</choice>
```

Figure 4. Contingent Request (WS-CDL)

In the first `workunit`, the `guard` condition compares the timestamps of the message sent to the `Municipality` (`ProcessedAnnualStatement`) and the one coming from the `Municipality` (`TaxAssessment`). In case the response occurred within 24h, the workflow proceeds. Otherwise a notification is returned to the `Client` as specified in the second `workunit`. In a third unit, the case when no message at all is returned, could be handled.

Request with Referral to a Single Party. The alternative scenario, where the `Municipality` returns the `TaxAssessment` directly to the `Client` can be simply realized through the passing of a previously defined channel variable (Fig. 5).

5 BPEL Implementation of the Tax Advisor

In this section, we show which parts of the workflow specifications can be specified with BPEL. We first describe the building blocks of a BPEL specification corresponding to the Choreography in Figure 1 (Section 5.1), we then move on to integrate control-flow activities (Section 5.2) and finally analyse the possibilities to integrate the complex interaction patterns specified in Section 2 (Section 5.3).

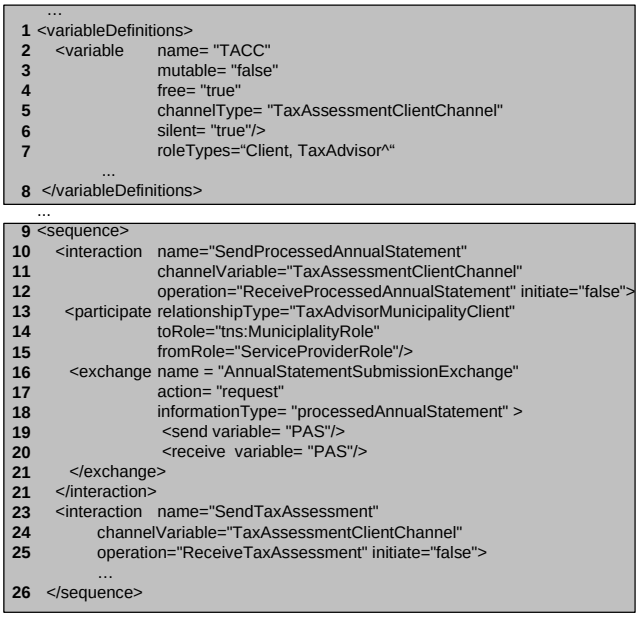

Figure 5. Request with Referral to a Single Party (WS-CDL)

5.1 Standard Workflows with BPEL

BPEL is an XML-based language for the composition of executable business processes based on Web Services. The specification of BPEL can be found in [2]. Listing 6 shows a BPEL process for one of the choreography participants. In order to implement his part of the choreography, the role `TaxAdvisor` orchestrates the sequence of service interactions through a BPEL process called `TaxAdvisorProcess`. The service composition is offered through an interface according to the choreography specification. A `partnerLink` defines internal and external parties (`myRole` and `partnerRole`) that interact with the process instance and the `portTypes` that need to be implemented (see lines 2-6). A `partnerLinkType` is a BPEL extension, which is used in the WSDL definition. It defines two `role`s of a bilateral message exchange and their `portType`s. A `partner` element (lines 7-9) can be used to group `partnerLinks`. Variables (lines 12-14) describe the message types used in a BPEL process. A `variable` is identified by a unique name and is associated to a `messagetype`. Variables store received messages and hold messages to be sent to other parties. A BPEL process describes the execution order of Web Services operations via basic and structured activities. A basic activity is either a message exchange between Web Services or a local operation of a BPEL engine. The example illustrates a `receive` activity (lines 17-25). The activity blocks the process until a matching message arrives. Invocations of remote Web Service operations are modelled as `invoke` activities. Lines 26-32 illustrate an asynchronous one-way invocation. Synchronous

request/response interaction can be expressed by including an additional output variable to store the response. Control flow logic of a BPEL process is defined via structured activities. In our example we use `sequence` for sequential execution. A `correlationSet` describes parts of messages which are unique for a process instance. Aliases to these message parts are called `properties`.

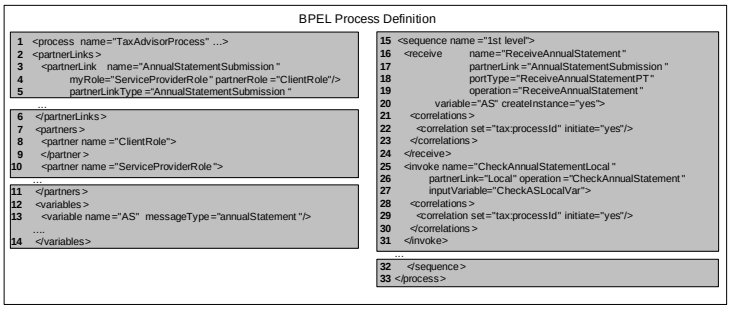

Figure 6. BPEL Listing (Basic Constructs)

5.2 Structured Activities

In BPEL, the order of execution of activities is specified through Structured Activities. Ordinary sequential control is provided by `sequence`, `switch` and `while`, concurrency by `flow` and nondeterministic choice based on external events by `pick`. Figure 7 shows, how the "compliance check issue" is resolved for the `TaxAdvisor` with straightforward use of `switch` and `case` statements, assuming that the variable `isCompliant` is returned as a result of the check and made available to the system in some way.

On the other hand, the `Client` does not know about the decision making of the `TaxAdvisor`. Therefore, the `Client` has to implement a `pick` with alternative `onMessage` constructs for either a rejection or an acceptance message, and we discuss conceptual challenges.

```
1 <switch>
2  <case condition= "bpws:getVariableProperty(isCompliant) = „true"">
3    <flow>
4          <!-- proceed with workflow -->
5    </flow>
6  </case>
7  <case condition="bpws:getVariableProperty(isCompliant) = „false">
8     <flow>
9          <!-- terminate with Rejection -->
10   </flow>
11 </case>
12 </switch>
```

Figure 7. Document Compliance (BPEL)

5.3 Advanced Interaction patterns

Contingent Request. Figure 8 shows the implementation of the 24h response requirement. This occurs through a `pick` activity which awaits an event, in our case either the document `TaxAssessment` - through `onMessage` - or an alarm based on a timer (`onAlarm`).

```
1  <pick>
2    <onMessage partnerLink="Municipality"
3            portType="receiveTaxAssessmentPT"
4            operation="receiveTaxAssessment"
5            variable="lineItem">
6            <!-- activity to forward ProcessTaxAssessment to Client -->
7    </onMessage>
8    <!-- set an alarm to timeout after 24 hours -->

9  <onAlarm for='"P0DT24H"'>
10 <!-- activity to notify Client about timeout-->
11 </onAlarm>
12 </pick>
```

Figure 8. Contingent Request (BPEL)

Request with Referral to a Single Party. The alternative scenario, where the `Municipality` returns the `TaxAssessment` directly is of no importance to the `TaxAdvisor`. He simply waits for some notification message by the `Municipality`.

6 Generating BPEL Stubs from WS-CDL

While the previous sections illustrated how BPEL processes can be modelled in correspondence to a given WS-CDL choreography, this section presents transformation rules from WS-CDL to BPEL. We implemented a transformation program that also includes mapping rules that are not presented here due to space limitations as a proof-of-concept.[3] We use the namespace prefixes cdl: and bpel: to indicate to which specifications the concepts belong.[4]

6.1 Core Concepts

Generally, one WS-CDL document maps to one or more `partnerLinkTypes` (each representing a bilateral communication relationship), multiple `property` and `propertyAlias` definitions related to WSDL interfaces, and at least two BPEL processes for each party involved in the choreography (see Figure 9).

[3] The wscdl2bpel.xslt program is available from http://wi.wu-wien.ac.at/~mendling. It uses the XALAN extensions to generate multiple output files. For details see http://xml.apache.org/xalan-j.

[4] Being aware that their are separate schemas for BPEL processes, partnerLinkTypes, and properties, we use the same bpel: prefix for all the three for better readability.

PartnerLinkTypes: Web Service interactions in a BPEL process rely on the availability of so-called `bpel:partnerLinkTypes`. A `bpel:partnerLinkType` defines the interaction of two parties by giving two related `bpel:role` elements and the `bpel:portType` that implements the role. This concept is very similar to the notion of a `cdl:relationshipType`. Accordingly, one `cdl:relationshipType` maps to one `bpel:partnerLinkType` and the `bpel:role` with its `bpel:portType` is generated from the referenced `cdl:roleType` declaration.

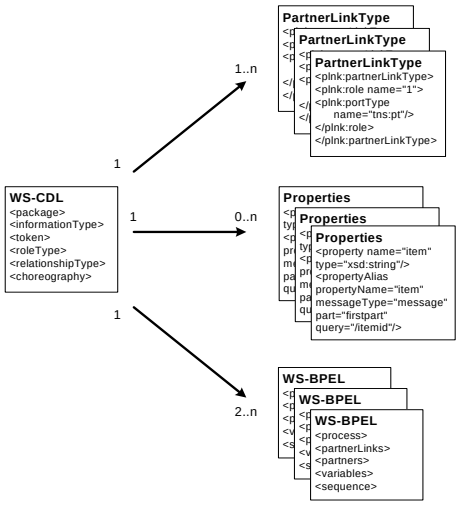

Figure 9. WS-CDL Document Mapping to one or more BPEL Documents

Properties: In BPEL properties play an important role for the correlation of messages and process instances. The `bpel:property` element defines an element that is unique for the process instance and which can be used for correlation. The related `bpel:propertyAlias` element specifies the XPath query to retrieve the `bpel:property` from a message. In WS-CDL `cdl:token` and `cdl:tokenLocator` elements represent the same concept. As only some property declarations are relevant for a specific party involved in the choreography, there needs to be a filter mechanism. We generate separate property files for each `cdl:roleType` including only those `bpel:properties` that are relevant for a party. The WS-CDL does not impose tokens to be defined, because e.g. a simple stateless request-response choreography does not need correlation. Accordingly, it is possible that no property files are generated from the WS-CDL choreography.

BPEL Process: For each party, a separate BPEL stub is generated. A party is either a `cdl:participant` that bundles several `cdl:roleTypes` or a `cdl:roleType` that is not subordinated to a `cdl:participant`. For both, the relevant information to be included in the BPEL files is identified via the `cdl:roleType` or the `cdl:roleType` elements referenced in the `cdl:participant` element.

A party's BPEL process includes declaration blocks with partner links, variables, and correlation sets – information needed by the activities defining the process. The `bpel:partnerLinks` block references the `bpel:partnerLinkType` files. It indicates the party's role in the process via the `bpel:myRole` attribute. The `bpel:variables` are generated from the `cdl:variableDefinitions` and from their references to `cdl:informationType` elements. Variables relevant to a party can be identified via the `cdl:roleTypes` attribute of each variable. `bpel:correlationSets` can be derived from the `cdl:channelType` elements: each channel element yields a `bpel:correlationSet` named after the channel and including the `cdl:token` element of the channel's `cdl:identity` element. The derived correlation sets are only included in those BPEL processes of parties using the respective channel in their interactions. The upper part of Figure 10 summarizes the mapping of the core concepts.

WS-CDL	BPEL	Semantics	Additional Effort
relationshipType	partnerLinkType role (ref. By roleType) portType (ref. By roleType)	Definition of bilateral interaction	
token	property	Message and process instance correlation	
tokenlocator	propertyAlias	Message and process instance correlation	
participant	roleType	For every participants a BPEL stub is generated	
roleType	roleType	Role in interaction	
variableDefinitions	variable	Message variable	
channeltype	correlationSet	Message correlation pattern	
workunit	scope	Execution context	
repeat	condition in a while loop	Repested execution	
guard	condition	Condition to be evaluated	
block = false	switch	Nesting execution units	
block = true	(receive)	Wait until condition becomes true or event happens	some WS-CDL conditions not in BPEL
sequence	sequence	Sequential execution of activity units	
parallel	flow	Simulaneous execution of parallel branches	
choice	case (nested in) switch	Alternative execution units for sender	condition must be added in BPEL
	onMessage (nested in) pick	Alternative execution units for receiver	condition must be added in BPEL
interaction		Bilateral exchange	
action=request	invoke	Invocation by sending party	
action=request	receive	Reception by receiving party	
action=response	invoke	Reply by receiving party	
action=response	receive	Reception by sending party	
timeout	pick, onAlarm (or) onMessage	Concurrent time or message event	
perform	*no mapping*	Bundle interactions to a choreography	
assign	assign (for party in roleTyoe)	Variable assignment	
silentAction	sequence (with nested) empty	Do nothing	activities must be added in BPEL
noAction	empty (for party in roleType)	Do nothing	
finalize	compenstionHandler	Finalizing activities after completion	

Figure 10. Mapping of Core Concepts and Control Flow

6.2 Control Flow

BPEL control flow is defined via scopes, structured and basic activities, the first allowing to nest other activities. WS-CDL uses a similar concept: so-called work units can be related to scopes, ordering structures to structured activities, and WS-CDL basic activities to BPEL basis activities. In the following, we describe the WS-CDL activities and show how they map to BPEL. The bottom part of Figure 10 summarizes the mapping of the control flow elements and additional engineering effort for building the BPEL process.

- `cdl:workunit`: The `cdl:workunit` is related to the `bpel:scope` concept in the sense that it defines a context for consistent execution. Yet, its attributes have a much more direct impact on control flow than the `bpel:scope`. The `cdl:workunit` unifies the concepts of a loop (`cdl:repeat`), of a data event (`cdl:guard`), and a wait (`cdl:block`). The guard and the block are interrelated. If `cdl:block` is true, then the choreography waits for the guard condition to become true before progressing. If set to false, the `cdl:workunit` is skipped. When the `cdl:repeat` condition is true the workunit is considered again for execution depending on the guard. In BPEL the `cdl:block=false` case maps to a `bpel:switch` executing the nested activities if the guard condition is true, otherwise progressing with the next activity subsequent to the `cdl:workunit`. The `cdl:block=true` case is hard to map as BPEL does not know events like "variable becomes available". We propose to use a `bpel:receive` in this case, because it blocks until a message is received and written to a `bpel:variable`. The BPEL engineer needs to add information from where the message is to be received. Finally, a `cdl:repeat` condition is mapped to a `bpel:condition` of a `bpel:while` loop. Note that the `bpel:while` is executed until the `bpel:condition` becomes true, and the `cdl:repeat` indicates repetition as long as the condition is still true.
- `cdl:sequence`: In general the `cdl:sequence` of the global model maps to a `bpel:sequence` of the local model. Yet, if the respective party is involved only in one or in none of the child activities of the `cdl:sequence`, then no local sequences needs to be generated.
- `cdl:parallel`: The `cdl:parallel` maps to a `bpel:flow` element in the local model. Similar to the sequence, if the respective party is involved in zero or one of the parallel branches, then the `bpel:flow` element can be omitted.
- `cdl:choice`: For those parties that can observe the decision condition, the `cdl:choice` is mapped to a `bpel:case` nested in a `bpel:switch` element. In this case the `bpel:switch` can only be left out if the party is not involved in any of the `cdl:choice` nested activities. If the party is involved in one nested activity a `bpel:case` for this activity has to be generated and a `bpel:case` including a `bpel:empty` activity. Note that the `bpel:conditions` of the cases need to be specified manually by the engineer of the BPEL process. For those parties, who receive different messages in response to a decision taken within the scope of another party, the `cdl:choice` has to be mapped to a `bpel:pick` (see Section 5.2).
- `cdl:interaction`: Each `cdl:exchange` of an interaction maps to a web service activity in BPEL (see Figure 11). In this context four cases have to be distinguished depending on the value of the `cdl:action` and whether the current party is mentioned in the `cdl:toRole` or `cdl:fromRole` attribute. In case of a request action a `bpel:invoke` is generated for the party of the `cdl:fromRole` and a `bpel:receive` for the party of the `cdl:toRole`. In case of a response action it is the other way around. If there is a `cdl:timeout` specified, a `bpel:pick` with a concurrent time event to the message receipt has to be specified in place of the `bpel:receive`.

– `cdl:perform`: This activity is not directly mapped to BPEL, but all nested activities of the referenced choreography are transformed and included.
– `cdl:assign`: This activity maps to a `bpel:assign` activity for the party mentioned in the `cdl:roleType` attribute.
– `cdl:silentAction`: This activity indicates that a party must perform some action that is not revealed in the global model. We propose to map it to a `bpel:sequence` with a nested `bpel:empty` activity and a `name` attribute set to "silent action". The engineer of the BPEL process will then have to specify these silent activities before deployment.
– `cdl:noAction`: This activity maps to a `bpel:empty` activity for the party mentioned in the `cdl:roleType` attribute.
– `cdl:finalize`: Finalizing activities can only be started after successful completion of a choreography. In this sense, they are related to the concept of a `bpel:compensationHandler`. Yet, they also involve communication to confirm the completion to other parties. Therefore, they cannot always be mapped to a `bpel:compensationHandler` because the only purpose of the latter is to undo successfully completed actions. Accordingly, we propose to append finalizing activities to the BPEL process of the parties involved.

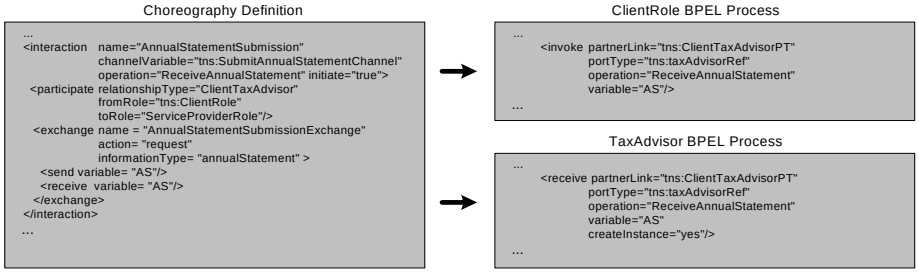

Figure 11. Transformation of the `cdl:interaction` element

With this transformation algorithm, BPEL processes can be generated almost automatically for all parties. Still, a BPEL engineer has to add implementation specific information including conditions for cases of a `bpel:switch` or activities that have been defined as silent activities in the choreography model.

7 Conclusions

The core contribution of this paper was to show how BPEL process definitions for parties involved in a choreography can be derived from a global WS-CDL model. We have implemented a prototype of the mappings as a proof of concept. The automation offers substantial speed-up of the engineering process. Additionally, the automatic generation of BPEL stubs minimizes the risk of inconsistent

process implementations by the parties. The main transformation problems are related to blocking `workunits` and `choices`.

In [21], the transformation of blocking `workunits` depends on the context. If it is nested in a `choice`, the `workunit` is mapped to a `pick`-branch. If it is nested in a `parallel`, the blocking is mapped to `links` from all sources from where the variable could be written. We propose to map such `workunits` to a `receive` in BPEL, because it is not clear how a blocking `workunit` should be treated that is nested e.g. within a `choice` which is itself nested in a `parallel` element. For this reason, we advocate to let a BPEL engineer decide. The problem of WS-CDL's choice is that it may have event-based or data-based decision semantics, or even mixtures of both. In BPEL, both cases have to be mapped either to a `switch` or a `pick`. It is an advantage of languages like Petri nets that allow to represent both cases as a place followed by a transition for each alternative branch.

Furthermore, we integrate advanced interaction patterns into both specification languages. This allowed us to intuitively depict some of the difficulties related to more complex interaction issues. We can observe, that the expressiveness of BPEL outbalances the semantics of WS-CDL when it comes to the integration of advanced interaction patterns. The structured activities of BPEL allow for a more straightforward integration, whereas the specification of the same patterns with WS-CDL's `choice` and `workunit` activities and related `block` conditions are a non-trivial task due to WS-CDL's focus on binary interactions.

We did not analyze all complex service interaction patterns like One-to-Many Send, One-to-Many Receive or Racing Incoming Messages [4] as this has been beyond the scope of article. In [4], the authors partly show that BPEL lacks sufficient support, especially advanced patterns. A comprehensive evaluation of WS-CDL with respect to the interaction patterns is still missing. As a resolution to this issue, we propose the modelling of choreographies with the help of a more abstract language - in the sense of being more independent of underlying technology - like e.g., UML 2.0 Activity Diagrams. Although we only specified the advanced service interaction pattern Contingent Request with the help of an informal constaint associated to the document node in the UML 2.0 Activity Diagram (Figure 1), it would be possible to specify the requirement in a formal way (e.g., with the help of OCL). It is obvious that UML provides enough expressive power to integrate these patterns. Its visual formalism adds significantly to the user convenience. The specification can subsequently be transformed into technology-driven standards like WS-CDL and WS-BPEL. This is part of ongoing research efforts of our research groups.

References

1. Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y.: Web Services Choreography Description Language Version 1.0. W3C Working Draft 17 December 2004, World Wide Web Consortium (2004)
2. Andrews, T., et al.: Business Process Execution Language for Web Services, Version 1.1. , BEA, IBM, Microsoft, SAP, Siebel (2003)

3. Barros, A., Dumas, M., Oaks, P.: A Critical Overview of the Web Service Choreography Description Language (WS-CDL). BPTrends Newsletter **3** (2005)
4. Barros, A.P., Dumas, M., ter Hofstede, A.H.M.: Service interaction patterns. In van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F., eds.: Business Process Management, 3rd International Conference, BPM 2005, Nancy, France, September 5-8, 2005, Proceedings. Volume 3649. (2005) 302–318
5. van der Aalst, W.M.P., Weske, M.: The p2p approach to interorganizational workflows. In Dittrich, K.R., Geppert, A., Norrie, M.C., eds.: CAiSE. Volume 2068 of Lecture Notes in Computer Science., Springer (2001) 140–156
6. Bussler, C.: Enterprise Application Integration and Business-to-Business Integration Processes. In: Process Aware Information Systems: Bridging People and Software Through Process Technology. Wiley Publishing (2005) 61–82
7. Peltz, C.: Web services orchestration and choreography. IEEE Computer **36** (2003) 46–52
8. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services - Concepts, Architectures and Applications. Springer Verlag, Berlin et al. (2003)
9. Banerji, A., Bartolini, C., Beringer, D., Chopella, V., Govindarajan, K., Karp, A., Kuno, H., Lemon, M., Pogossiants, G., Sharma, S., Williams, S.: Web Service Conversation Language (WSCL) 1.0. W3C Note 14 March, World Wide Web Consortium (2002)
10. Arkin, A., Askary, S., Fordin, S., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S., Takacsi-Nagy, P., Trickovic, I., Zimek, S.: Web Service Choreography Interface (WSCI) 1.0. W3C Note 8 August, World Wide Web Consortium (2002)
11. Clark, J., Casanave, C., Kanaskie, K., Harvey, B., Clark, J., Smith, N., Yunker, J., Riemer, K.: ebXML Business Process Specification Schema Version 1.01. Specification, UN/CEFACT and OASIS (2001)
12. Thatte, S.: XLANG: Web Services for Business Process Design. Specification, Microsoft Corp. (2001)
13. Leymann, F.: Web Services Flow Language (WSFL). Specification, IBM Corp. (2001)
14. Arkin, A.: Business Process Modeling Language (BPML). Specification, BPMI.org (2002)
15. Workflow Management Coalition: Workflow Process Definition Interface – XML Process Definition Language. Document Number WFMC-TC-1025, October 25, 2002, Version 1.0, Workflow Management Coalition (2002)
16. van der Aalst, W.: Loosely coupled interorganizational workflows: Modeling and analyzing workflows crossing organizational boundaries. Information and Management **37** (2000) 67–75
17. Grefen, P.W.P.J., Aberer, K., Ludwig, H., Hoffner, Y.: Crossflow: Cross-organizational workflow management for service outsourcing in dynamic virtual enterprises. IEEE Data Eng. Bull. **24** (2001) 52–57
18. Brogi, A., Canal, C., Pimentel, E., Vallecillo, A.: Formalizing web service choreographies. Electr. Notes Theor. Comput. Sci. **105** (2004) 73–94
19. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Quarterly **28** (2004) 75–105
20. Mendling, J., Hafner, M.: From inter-organizational workflows to process execution: Generating bpel from ws-cdl. In: OTM Workshops. Volume 3762 of Lecture Notes in Computer Science., Springer (2005) 506–515

21. Weber, I.: Automation in collaborative business process instantiation. Diploma thesis, Universität Karlsruhe (TH), Institute for Program Structures and Data Organization, http://imweber.de/downloads/Diploma_Thesis--Ingo_Weber.pdf (2005)
22. Weber, I., Haller, J., Mülle, J.A.: Derivation of executable business processes from choreographies in virtual organizations. In: Proc. of XML4BPM 2006, http://wi.wu-wien.ac.at/home/mendling/XML4BPM2006/XML4BPM-Weber.pdf (2006)
23. Gardner, T.: UML Modelling of Automated Business Processes with a Mapping to BPEL4WS. In: Proceedings of the First European Workshop on Object Orientation and Web Services at ECOOP 2003. (2003)
24. Hofreiter, B., Huemer, C.: Transforming umm business collaboration models to bpel. In Meersman, R., Tari, Z., Corsaro, A., eds.: OTM Workshops. Volume 3292 of Lecture Notes in Computer Science., Springer (2004) 507–519
25. Hafner, M., Breu, R., Breu, M., Nowak, A.: Modeling inter-organizational workflow security in a peer-to-peer environment. In: Proceedings of ICWS. (2005)
26. Hafner, M., Breu, R., Breu, M.: A security architecture for inter-organizational workflows: Putting security standards for web services together. In Chen, C.S., et al., J.F., eds.: Proceedings ICEIS. (2005)
27. Breu, R., Hafner, M., Weber, B., Novak, A.: Model driven security for inter-organizational workflows in e-government. In Böhlen, M.H., Gamper, J., Polasek, W., Wimmer, M., eds.: TCGOV. Volume 3416 of LNCS. (2005) 122–133
28. Hafner, M., Breu, R.: Realizing model driven security for inter-organizational workflows with ws-cdl and uml 2.0 - bringing web services, security and uml together. In: to appear in Proceedings of MODELS. (2005)