Towards Automated Conformance Checking of ebBP-ST Choreographies and Corresponding WS-BPEL Based Orchestrations

Matthias Geiger, Andreas Schönberger and Guido Wirtz

Distributed and Mobile Systems Group, University of Bamberg
Feldkirchenstr. 21, 96052 Bamberg, Germany
E-mail: {matthias.geiger | andreas.schoenberger | guido.wirtz}
@uni-bamberg.de

Abstract

Web Services technologies are a natural candidate for Business-to-Business integration (B2Bi). For cross-organizational processes, the concepts of "choreography" and "orchestration" are important. The term choreography denotes a model of a global view over message exchange scenarios, whereas the term orchestration focuses on models of the local implementation. While WS-BPEL is already kind of a de-facto standard in the field of orchestration languages, there does not exist a standard choreography language. We propose the usage of ebXML BPSS (ebBP) in order to provide choreography modeling at the business level. A frequent problem is to ensure and enforce the consistency and conformance of choreography and orchestration models which is often referred to as "conformance checking". In this paper we examine a way to check the conformance between ebBP-ST (a subset of ebBP) choreographies and corresponding WS-BPEL based implementations. To achieve this check well-known and approved model checking methods and tools are used: First ebBP-ST choreographies are directly transformed into the process algebra CCS. Second, the low level WS-BPEL processes are analyzed for code blocks that implement choreography elements and the sequence of these code blocks is then mapped to CCS, too. Afterwards these formalized representations will be checked for bisimulation equivalence in order to reveal inconsistencies between the choreography and their implementations.

Keywords: SOA, choreography, orchestration, conformance checking

1. Introduction

In the domain of Business-2-Business integration (B2Bi) Web Services technologies provide a suitable solution for integrating the cross-organizational business processes. As in B2Bi scenarios a global coordinator cannot be assumed, the distinction between choreographies and orchestrations is important: Choreographies define a process from a global perspective which may be seen as a communication contract between the integration partners involved. Orchestrations describe the local, executable implementation for each of the partners. In the B2Bi domain the *ebXML Business Process Specification Schema* (ebXML BPSS or ebBP; [8]) is a suitable choice for choreography modeling as it provides several features that are well-suited for the business domain. One core concept of ebBP is the usage of so-called *Business Transactions* (BTs) to model the exchange of a single business document from a sending to a receiving role - optionally followed by a response document and so-called *Business Signals* which notify the senders about the processing status of the exchanged documents. More complex scenarios composed of more document exchanges may be modeled as *Business Collaborations* (BCs). Within BCs, Business Transaction Activities (BTAs) are used to require the execution of BTs and to map BC roles to the roles of the previously defined BTs. The control flow and the ordering between BTAs is modeled in BCs using basic control flow constructs like *Transitions*, *Decisions*, *Forks* and *Joins*.

In [9], an ebBP dialect called ebBP-ST is presented to introduce explicitly modeled *Shared States* (STs) into ebBP choreographies which leads to clearer models in complex scenarios by providing control flow synchronization points.

In order to execute the choreography models defined in ebBP-ST, they have to be transformed into executable orchestrations. In the area of Web Services technologies the *Web Services Business Process Execution Language* (WS-BPEL; [7]) is the de-facto standard for realizing orchestrations which is also used in [9] to implement ebBP-ST choreographies.

For enabling executable orchestrations the integration architecture depicted in figure 1 is assumed between two partners A and B. These partners use backend components to encapsulate business logic and control processes to enforce the correct sequence of message exchanges as defined in ebBP-ST choreographies. In this paper, we investigate automated conformance checking of WS-BPEL based control process implementations and ebBP-ST choreography definitions.

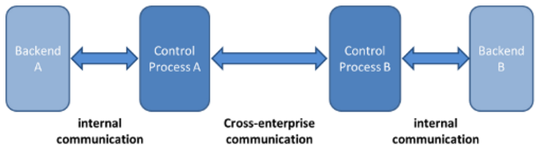

Figure 1. Integration Architecture (adapted from [9])

The paper proceeds as follows: First, in Sec. 2 a use case which serves as a running example is introduced. In section 3, the approach of checking ebBT-ST choreographies and WS-BPEL orchestrations is presented by clarifying which process models are supported, choosing a suitable conformance notion, introducing algorithms to transform ebBP-ST resp. WS-BPEL to the process algebra CCS [5], and describing the actual conformance check. The paper concludes with a discussion of related work and an outlook on ongoing research.

2. Use Case

Throughout the paper an excerpt of a purchasing use case between two partners shall be used to clarify the proposed approach. A visual representation of this ebBP-ST choreography is given in Fig. 2. The use case consists of three different STs and BTAs: The process starts with the ST "*initPurchase*" in which the BTA "*requestQuote*" has to be performed. This BTA defines the exchange of a business document containing a quote request. In the subsequent decision it is checked whether the request could be successfully transfered to the receiver ("*BusinessSuccess*") or not ("*TechnicalFailure*"). In case of a "*TechnicalFailure*", the current ST "*initPurchase*" is not left and the BTA has to be repeated. Otherwise, if the result is "*BusinessSuccess*" the process enters the new ST "*receivedQuote*".

When a quote has been received successfully it may be accepted by a corresponding BTA "*acceptQuote*" which leads to another ST "*concludedContract*" in case this BTA finished with a "*BusinessSuccess*", otherwise the BTA shall be repeated. The other option in ST "*receivedQuote*" is to request another quote by performing the BTA "*requestQuote*" again. The decision afterwards evaluates whether the BTA has been executed successfully or not. Here in both cases the process will be resumed in ST "*receivedQuote*". The difference between both paths is that the ST shall not be left in case the BTA finished with a "*TechnicalFailure*", i.e., the timer defined for the BTA shall not be reset, but shall be reentered with resetting the timer in case of a "*BusinessSuccess*". Note that this fact is not respected in the visualization of the use case in Fig. 2. The last aspect to consider is that a ST may be left by a timeout event. Here, a timeout occuring in ST "*initPurchase*" and "*receivedQuote*" will terminate the process in the end state "*TechnicalFailure*". The further progress of the purchase scenario, e.g., shipment and billing is left out here.

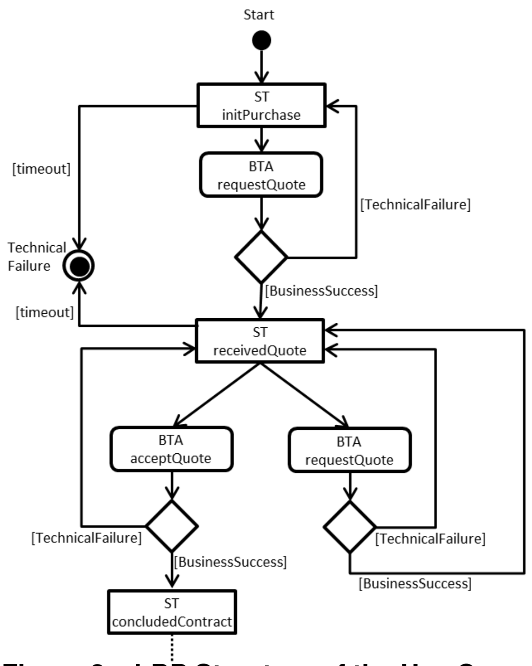

Figure 2. ebBP Structure of the Use Case

3. Conformance Checking ebBP and WS-BPEL

Figure 3 gives a brief overview of the proposed conformance checking approach: First, the given ebBP-ST choreography model and the corresponding WS-BPEL orchestrations which should be checked have to be transformed into a common formal representation. The process algebra CCS is appropriate for the scope of our analysis. After transforming ebBP resp. WS-BPEL into CCS models, the orchestration CCS models can be checked for conformance to the choreography model successively. As conformance is seen as a binary decision in our work, the conformance check returns whether the checked orchestration model conforms to the choreography specification or not.

3.1. Supported Process Definitions

Before describing the approach of checking the conformance between ebBP and WS-BPEL it has to be clarified which kind of processes are supported. As mentioned in the previous section, ebBP-ST models are used to model ebBP

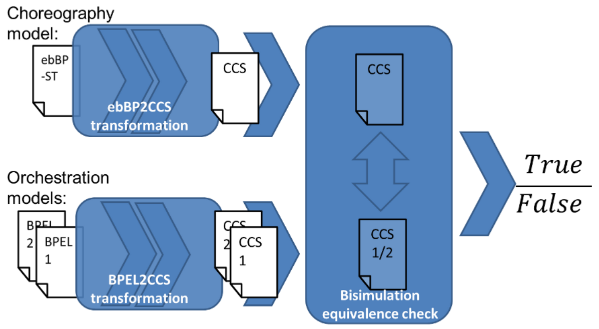

Figure 3. Proposed Approach (Overview)

choreographies. Our approach supports arbitrary ebBP-ST models which are valid regarding the formal definitions and rules for well-formedness in [9]: ebBP-ST models may be formalized as a 5-tuple (R, N, G, ϕ, θ) where R is a set of participants, N is a set of allowed ebBP nodes, G is a set of guards, ϕ is a function that assigns timeouts to each ST and θ is a transition relation which defines all allowed transitions. The set of nodes $N = \{s_0\} \cup ST \cup SBTA \cup DEC \cup T$ contains the ebBP concepts of a starting node, shared states, business transaction activities, decisions and end states which are used to model choreographies. Some important restrictions of ebBP-ST models are that only binary collaborations are supported, ebBP Forks and Joins are not allowed and STs may not overlap. The introduced use case is an example of a rather simple, well-formed model which uses all main aspects of ebBP-ST.

While a formal model of ebBP-ST and its semantics, is given in [9], such a formalization is not provided for the corresponding WS-BPEL implementations. There exist various formalisms for WS-BPEL (see [10]) but all of them are intended to directly formalize low level WS-BPEL language constructs, which would create much too detailed models for our use cases as we are only interested in a formal representation of the control flow between WS-BPEL patterns that realize ebBP choreography concepts. For this purpose the ebBP formalization is adapted to the WS-BPEL realizations: With respect to the set of nodes N it is obvious that all of these aspects have to be represented in WS-BPEL. Concepts such as BTAs may not be directly mapped to WS-BPEL but need to be expressed by more complex combinations of various elements. Possible WS-BPEL patterns to implement a given ebBP-ST choreography are also introduced in [9].

3.2. Conformance Notion - Why Bisimulation?

Conformance checking is not a new problem in the field of Web Services technologies, in fact, conformance checking techniques have been proposed for various combinations of choreography and orchestration languages. But as the notion of conformance is highly dependent on the context there does not exist a commonly accepted definition of the term conformance. A straightforward informal definition is: *An orchestration should be declared as conform to a choreography if all orchestration executions do not violate the predefined protocol.* As conformance checking problems are a specialization of the widely researched process equivalence problem, solutions from this domain can be used in order to specify the conformance notion more precisely. When looking at the message level, it is widely accepted that so-called *trace equivalence* which only analyzes the flow of exchanged messages is too weak to check conformance (e.g., [4]). *Bisimulation equivalence* [6], as another classical process equivalence notion, frequently is regarded as too strict for conformance checks. In most of the related work, the conformance notion distinguishes between incoming and outgoing message flows. An orchestration has to respect all incoming messages in order to be conform to the choreography. In case of outgoing communication, at least one (of possibly more) alternatives has to be implemented in the orchestration. An example is the ST "*receivedQuote*" in our use case, as two different BTAs ("*acceptQuote*" and "*requestQuote*") may be initiated by the potential customer here. The customer does not have to implement the possibility of performing the BTA "*requestQuote*" which means that he always performs "*acceptQuote*" and therefore all quotes have to be accepted (or the process is left by a timeout). Conversely the seller has to implement both execution possibilities as from his viewpoint it is not clear which BTA may be performed next.

However, for our conformance checking scenarios this conformance notion is not suitable as it cannot be applied to our integration architecture: We differentiate between the actual orchestration implementations in the so-called control processes and the existing backend systems that encapsulate business logic. Regarding the incoming information the already proposed conformance condition is not critical: The control processes must be able to react to all possible incoming events as they do not know how the partner will react. But the less strict requirements for outgoing information are only possible if the orchestration is directly deciding which steps should be performed next. In our architecture (cf. Fig. 1), the control processes delegate this decision to the backend systems which trigger the subsequent process flow. So, the control processes must be able to react to all possible backend decisions and must also be able to produce all allowed outgoing communication specified in the choreography. Therefore the conformance requires the rather strict notion of *weak bisimulation equivalence*.

The formal model used to perform conformance checks is the process algebra *Calculus of Communicating Systems* (CCS) developed by Milner [5]. For our purposes, the main aspects of CCS are sufficient to express our integration models: Processes (starting with a capital letter) are defined by an assignment. Linking to processes is done by using the name of a predefined process in a process body. Sequences are built using the "." operator and choices may be created with the operator "+". Therefore, e.g., "`P1 = action1.((action2.P2)+(action3.P3))`" defines the process `P1` in which after performing action "`action1`" either "`action2`" is performed followed by the execution of the process "`P2`" or "`P3`" is performed after "`action3`".

3.3. ebBP-ST to CCS Transformation

The basic principle of transforming an ebBP-ST choreography model into a CCS representation is to define a CCS process for each ST and each end state used in the choreography whereas timeout events, the execution of BTAs and their evaluation in decisions will be represented by CCS actions. Algorithm 1 shows the concrete steps needed in order to transform ebBP-ST choreographies to CCS.

As shown in line 1-3, for each end state $t \in T$ a process is defined using the prefix "`END_`" followed by the name of the state. The process body simply contains a single action named by the corresponding end state in order to distinguish different end states and the CCS *empty* element "`0`". The actual creation of the process is performed by the method `CreateProcess(`*`processName`*`, `*`processBody`*`)` which creates a CCS process with the name *`processName`* containing the *`processBody`*.

In comparison to this, the transformation rules for STs are much more complex: As described before and in [9] a timer should be started when entering a ST. But it also should be possible to reenter an already visited ST without resetting this timer. In order to describe this behavioral difference, two different CCS processes for each shared state are created. An outer process definition "`ST_`*`STname`*" simply performs a `start_timer` action and then links to an inner process definition (named "`INNER_ST_`*`STname`*") which contains the actual control flow logic (stored in variable `processBody`) of a shared state (ll. 28/29, Alg. 1).

BTAs, Decisions and Timeouts are realized in CCS using actions. Therefore each possible BTA in each ST followed by the decision evaluation is added to the ST process body (Alg. 1, line 20) using the method `AddBTA(`*`processBody`*`, `*`btaToAdd`*`)` which combines the allowed BTAs with the "+" operator. BTAs are simply mapped by a single CCS action named like the BTA sequentially followed by the decision. The different decision branches are realized using the CCS choice operator ("+"): First, each guard is translated to a CCS action followed by a link to the CCS process of the subsequent ST and afterwards all of these constructs are combined using the `AddDec` method. A flag f indicates whether a timeout should be reset or not: If $f == \{tt\}$ (l.10) the outer process definition will be used, otherwise it will be linked to the inner process definition of the subsequent ST.

Applying the algorithm to our use case will clarify the output of the algorithm which is presented in listing 1: Line 1 shows the outer process definition "`ST_receivedQuote`", the following lines 2-11 describe the actual control flow logic: The two BTAs allowed and the timeout event are combined as a CCS choice. The execution of a BTA is represented by a CCS action (e.g., "`bta_requestQuote`" in line 7). The decision after this BTA evaluates whether the BTA has been a "*BusinessSuccess*" (BS) or a "*TechnicalFailure*". If the BTA was successful the ST "*receivedQuote*" should be reentered and the timer should be reset. Therefore the referenced CCS process is "`ST_receivedQuote`" (line 9 in List. 1). As the timer should not be reset in case of a "*TechnicalFailure*", the internal process definition "`INNER_ST_receivedQuote`" is referenced in line 8.

The other STs in the use case are transformed in the same way.

Listing 1. CCS Representation of the Use Case (Excerpt)

```
1  ST_receivedQuote = start_timer.INNER_ST_receivedQuote
2  INNER_ST_receivedQuote =
3  (bta_acceptQuote.(
4    (dec_acceptQuote_TF.INNER_ST_receivedQuote)+
5    (dec_acceptQuote_BS.ST_concludedContract)
6    )
7  )+(bta_requestQuote.(
8    (dec_requestQuote_TF.INNER_ST_receivedQuote)+
9    (dec_requestQuote_BS.ST_receivedQuote)
10   )
11 )+(timeout.END_TechnicalFailure)
```

3.4. WS-BPEL to CCS Transformation

The aim of transforming WS-BPEL to CCS is to check whether the WS-BPEL implementations conform to the predefined ebBP-ST choreography models. As mentioned before, we are not interested in low-level formal models but in a representation which allows for this check. For example, there is no need to model the concrete implementation of a BTA including all WS-BPEL *sequences*, *scopes*, *invokes*, etc. in CCS because for our purposes it is only relevant whether a BTA may be performed in a ST or not. So, the most important task for transforming WS-BPEL to CCS is to detect the used patterns which express the different concepts of ebBP and afterwards transforming the sequence of patterns into CCS. When looking at an ebBP *decision*, a possible WS-BPEL implementation pattern may send the previously exchanged *Business Document* to the backend systems (*invoke* statement) which analyze the outcome and the result is stored in a variable (*assign*). Afterwards in a

Algorithm 1: ebBP to CCS Transformation

```
Input: A valid ebBP-ST choreography $(R, (\{s_0\} \cup ST \cup SBTA \cup DEC \cup T), G, \phi, \theta)$ to be transformed
Output: A CCS representation of this choreography
Algorithm:
foreach $t$ in $T$ do
    CreateProcess(“END_” + t.name, “end_”+t.name+“.0”)
end
foreach $st\ in\ ST$ do
    processBody = “ ”;
    foreach $(st, \{tt\}, bta)$ in $\theta$ do
        decisions = “ ”;
        foreach $(bta, \{tt\}, dec)$ in $\theta$ do
            foreach $(dec, g, f, nextST)$ in $\theta$ do
                if $(f == \{tt\})$ then
                    AddDec(decisions, “(dec_”+dec.name+“_”+g.name+“.ST_”+nextST.name+“)”)
                else
                    AddDec(decisions, “(dec_”+dec.name+“_”+g.name+“.INNER_ST_”+nextST.name+“)”)
                end
            end
            foreach $(dec, g, t)$ in $\theta$ do
                AddDec(decisions, “(dec_”+dec.name+“_”+g.name+“.END_”+t.name+“)”)
            end
        end
        AddBTA(processBody, “(bta_”+bta.name+“.(”+decision+“))”)
    end
    if $\exists(st, g^{to}, nextST)$ in $\theta$ then
        processBody += “+(timeout.ST_”+nextST.name+“)”
    end
    if $\exists(st, g^{to}, t)$ in $\theta$ then
        processBody += “+(timeout.END_”+t.name+“)”
    end
    CreateProcess(“ST_”+st.name, “start_timer.INNER_ST”+st.name))
    CreateProcess(“INNER_ST_”+st.name, processBody)
end
CreateProcess(“Start”, “ST_”+firstST.name)
```

series of *if* statements it is checked which result has been evaluated and the switch to the next state is initiated.

After all such patterns and their ordering have been identified, the transformation to CCS is similar as for ebBP-ST. In case of the decision example, each decision branch is transformed to a CCS sequence, containing an action indicating the decision, followed by a reference to the CCS process of the next state.

3.5. Checking the Conformance

After transforming the ebBP-ST choreography model and the two corresponding WS-BPEL orchestrations to CCS, the actual conformance checks can be performed. Therefore each CCS orchestration model has to be checked against the choreography representation for bisimulation equivalence. The result of each bisimulation equivalence check is (see Fig. 3) the binary answer whether an orchestration is conform to the choreography definition or not.

An advantage of this checking approach is that the two different implementations may be analyzed independently, i.e., it is not necessary to analyze the implementation of a partner who possibly does not want to publish his internal orchestration models.

Note that CCS is supported by various model checking tools (e.g. the *Edinburgh Concurrency Workbench* (CWB)[1])

[1]available at: `http://homepages.inf.ed.ac.uk/perdita/cwb/`

and therefore the check can be automatically executed using such a tool. Using the proposed CWB model checker we are able to prove the conformance of the WS-BPEL implementations as well as to detect various intentionally produced faults in orchestrations.

4. Related Work

Conformance checking problems have been investigated for various choreography and orchestration languages:

For example the work of Baldoni et al. (i.a.,[1]) is rather generic and uses automata representations to model choreographies and orchestrations. Martens ([4]) discusses rather extensively suitable conformance notions for conformance checking problems and proposes a formalization of abstract and executable WS-BPEL processes using Petri Nets. A proposal for the *Web Services Choreography Description Language* (WS-CDL) has been developed by Foster et al. ([3]) who use *Finite State Processes* (FSP) to represent WS-CDL and WS-BPEL. The actual check is based on *trace equivalence*. The only work that we are aware of which is dealing with conformance checking ebBP and WS-BPEL is [12] which uses the process algebra *Communicating Sequential Processes* (CSP) to perform so-called traces refinement to check the conformance. The common problem of all these approaches is that they do not use the strict conformance notion we propose in this work. As we have shown above the proposed weaker notions are suitable for the contexts considered in the various papers, but it is not appropriate for the integration scenario we assume.

Another approach which uses a stricter conformance notion when checking WS-CDL choreographies is presented in [11]: The authors derive a new formalism called piX-Model to formally represent WS-CDL choreographies. The piX-Model is based on the well-known π-calculus and open bisimulation is used as conformance notion.

Apart from conformance checking there exist various approaches to formalize WS-BPEL (see [10] for an overview). A WS-BPEL formalization using CCS is presented in [2]. Generally these formalizations are not perfectly suitable for our approach because they provide direct mappings for basic WS-BPEL constructs while we concentrate on those patterns that realize ebBP concepts.

5. Conclusion and Future Work

In this paper, we have introduced an approach for checking the conformance between ebBT-ST choreographies and corresponding WS-BPEL based orchestrations. The key elements of this approach are the proposed transformation algorithms to CCS which allow for a common representation of ebBP-ST and WS-BPEL in order to perform the actual conformance check by a bisimulation equivalence check. Preliminary results show the general applicability and correctness of our approach.

Ongoing research now concentrates on relaxing the requirements concerning the structure of the WS-BPEL processes by using existing WS-BPEL semantics. Furthermore we are working on better usability, such as integrating the proposed tool chain to improve the work flow and direct highlighting of detected conformance issues in the original ebBP-ST and WS-BPEL definitions.

References

[1] M. Baldoni, C. Baroglio, A. K. Chopra, N. Desai, V. Patti, and M. P. Singh. Choice, interoperability, and conformance in interaction protocols and service choreographies. In *8th Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009), Budapest, Hungary, May, 2009*, pages 843–850. IFAAMAS, 2009.

[2] J. Cámara, C. Canal, J. Cubo, and A. Vallecillo. Formalizing wsbpel business processes using process algebra. *Electr. Notes Theor. Comput. Sci.*, 154(1):159–173, 2006.

[3] H. Foster, S. Uchitel, J. Magee, and J. Kramer. WS-Engineer: A Model-Based Approach to Engineering Web Service Compositions and Choreography. In *Test and Analysis of Web Services*, pages 87–119. Springer, 2007.

[4] A. Martens. Consistency between Executable and Abstract Processes. In *2005 IEEE Int. Conf. on e-Technology, e-Commerce, and e-Services (EEE 2005), Hong Kong, China*, pages 60–67. IEEE Computer Society, 2005.

[5] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, 1980.

[6] R. Milner. *Communication and concurrency*. Prentice Hall, Harlow, 1989.

[7] OASIS. *Web Services Business Process Execution Language Version 2.0 (WSBPEL)*, 2.0 edition, April 2007.

[8] OASIS. *ebXML Business Process Specification Schema Technical Specification*, v2.0.4 edition, Oktober 2006.

[9] A. Schönberger, C. Pflügler, and G. Wirtz. Translating shared state based ebXML BPSS models to WS-BPEL. *Int. Journal of Business Intelligence and Data Mining*, 5(4):398–442, 2010.

[10] F. van Breugel and M. Koshkina. Models and Verification of BPEL. Unpublished Draft, Available at: `http://www.cse.yorku.ca/~franck/research/drafts/tutorial.pdf`, September 2006.

[11] G. van Seghbroeck, B. Volckaert, F. D. Turck, B. Dhoedt, and P. Demeester. Web service choreography conformance verification through the pix-model. *Int. J. Cooperative Inf. Syst.*, 19(1-2):1–30, 2010.

[12] W. L. Yeung. A Formal Basis for Cross-Checking ebXML BPSS Choreography and Web Service Orchestration. In *APSCC '08: Proc. of the 2008 IEEE Asia-Pacific Services Computing Conf.*, pages 524–529, Washington, DC, USA, 2008.