Web Service Interfaces for Inter-organisational Business Processes
An Infrastructure for Automated Reconciliation

Giacomo Piccinelli
Hewlett-Packard Laboratories
Stoke Gifford Park
Bristol BS34 8QZ, UK
gicomo_piccinelli@hp.com

Wolfgang Emmerich
Dept. of Computer Science
University College London
Gower Street, London WC1E 6BT, UK
w.emmerich@cs.ucl.ac.uk

Christian Zirpins, Kevin Schütt
Dept. of Computer Science (VSIS)
University of Hamburg
Hamburg, Germany
zirpins@informatik.uni-hamburg.de

Abstract

For the majority of front-end e-business systems, the assumption of a coherent and homogeneous set of interfaces is highly unrealistic. Problems start in the back-end, with systems characterised by heterogeneous mix of applications and business processes. Integration can be complex and expensive, as systems evolve more in accordance with business needs then with technical architectures. E-business systems are faced with the challenge to give a coherent image of a diversified reality. Web Services make business interfaces more efficient, but effectiveness is a business requirement of at least comparable importance.

In this paper, we propose a technique for automatic reconciliation of the Web Service interfaces involved in inter-organisational business processes. The working assumption is that the Web Service front-end of each company is represented by a set of WSDL and WSCL interfaces. The result of our reconciliation method is a common interface that all the parties can effectively enforce. Indications are also given on ways to adapt individual interfaces to the common one. The technique was embodied in a prototype that we also present.

1. Introduction

The high level of internal automation that companies have achieved is a major driver for the development of e-business systems. Information systems are a pervasive reality inside business operations, and the automation of business-to-business (B2B) interaction is recognised as the next objective for corporate IT. Countless products as well as open-access technologies become available on a daily basis. Standardisation initiatives flourish at any level, from enabling technology (e.g. XML, SOAP and WSDL) to business-level ontology (e.g. ebXML, RosettaNet). New interaction and intermediation models gain increasing acceptance (e.g. electronic services, electronic marketplaces).

From its early appearance three years ago, the electronic service model has been gaining increasing consensus as a way to tackle the complexity of e-business systems and solutions. The initial history of the Internet saw technology come first, and business applications follow. In the case of electronic services, the business objectives came first. Electronic services are about business processes and resources made available over the Internet, in order to enable seamless interaction and dynamic creation of business solutions. As for today, technology is only at the early stages in terms of enabling the full vision for electronic services. The development of the Web Service stack represents an important step towards making electronic services a reality.

The work we present focuses on Web Services, and their use in B2B integration solutions. Web Services currently support the externalisation of atomic business capabilities. Description models for Web Service interfaces like WSDL (Web Service Definition Language) and WSCL (Web Service Conversation Language) support the externalisation of access points (WSDL) to a business service as well as basic interaction patterns (WSCL). In the case of a flight reservation service, a WSDL interface can model a flight-availability request. The invocation of the Web Service related to such interface could return the list of available flights to a certain destination. A WSCL interface can instead model the interaction involved in the payment for the flight purchase. A number of messages might be exchanged in relation to the actual payment, triggering a conversational interaction between Web Services. The complete interface for the flight reservation service can be modelled as sets of WSDL and WSCL interfaces. The focus of our work is the automatic reconciliation of the Web Service interfaces externalised by e-business systems.

Section 2 contains a brief overview of Web Services and related initiatives in the space of business-to-business integration (B2Bi). Section 3 depicts a typical scenario for e-business systems, highlighting possibilities and challenges for Web Services and introducing the problem of process reconciliation. In Section 4, we propose a

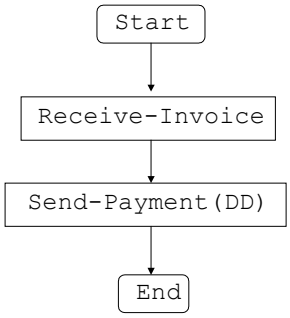

Figure 1.a: Interaction process for TravelSmart

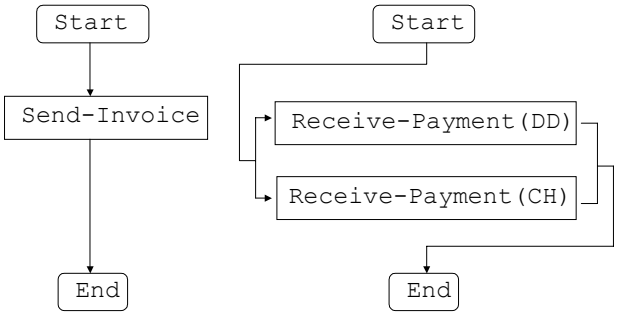

Figure 1.b: Interaction processes for AirWeb

technique for automatic reconciliation of business-service interfaces. The assumption is a service description based on WSDL and WSCL. The technique is based on the concept of workflow inheritance. Section 5 describes the prototype embodying the proposed technique. In Section 6, the prototype is applied to an example of interface reconciliation. Section 7 describes related works. The closing section includes considerations on current results, and indications on future developments.

2. B2B integration and Web Services

Business-to-business integration revolves around interaction and coordination. In terms of interaction, information must be exchanged efficiently and interpreted univocally. In terms of coordination, business processes need to align on mutually acceptable cooperation logic. For a long time, solutions based on the EDI (electronic data interchange) model have addressed the integration problem for closed and static clusters of companies. The challenge is now to extend the basic principles of EDI to open and dynamic clusters of business partners.

In the EDI world, business interaction is based on private networks and proprietary protocols. The main problem with traditional EDI is flexibility. Sharing a common ontology and business practices is a prerequisite for cooperation. Still, the acceptance should be on a wide scale. As a reference case, we can consider the travel industry and expand the context for the flight reservation example introduced in the previous section. A travel agent TravelSmart may be working with a specific set of airlines. A new airline AirWeb can enter the market, and offer special retail margins for the first six months of operation. TravelSmart should be able to start trading with AirWeb almost immediately. After the initial six months, the retail margins offered by AirWeb become less attractive. TravelSmart should be able to stop trading with AirWeb, without concerns for upfront investments. Unrealistic in a traditional EDI context, dynamic business-to-business integration is a key objective for electronic services.

RosettaNet [18] and ebXML [22] demonstrate the potential of open standards for a common business ontology. Initiatives such as OMG's MDA (Model Driven Architecture) [14] promote common architectural patterns for business solutions. UDDI (Universal Description Discovery and Integration) [21] enables dynamic discovery of business partners. Still, Web Services represent the most noticeable effort in terms of open and dynamic integration. From a business perspective, the service-oriented model [19] redefines the modularisation criteria for business capabilities. Business offer is represented as service modules, which can be combined into customer-specific solutions. Business solutions can involve multiple parties, each contributing only specific capabilities. From a technology perspective, the Web Service stack leverages the flexibility of XML to enable the automatic processing of service-related information. From SOAP (Simple Object Access Protocol) [6] and WSDL (Web Service Description Language) [8] to XLANG [20] and WSCL (Web Service Conversation Language) [4], Web Services are rapidly moving from access logic to service delivery logic. The convergence of Web Services and the Semantic Web initiatives [23] promises further improvements in the space of rich definitions of business services.

3. B2B integration and process reconciliation

The canonical subdivision for a business information system is based on the notions of back-end and front-end. The back-end includes applications and processes directly linked with the production of goods and services, as well as the interaction with suppliers. The front-end includes

applications and processes involved in sales and other forms of customer interaction. Taking TravelSmart as example, the interaction with AirWeb for the purchase of a number of seats on a flight is part of the back-end. The interaction with a group of holidaymakers for the sale of a package holiday is part of the front-end. For AirWeb, the interaction with TravelSmart is part of the front-end. B2Bi refers to the integration between back-end of business customers and the front-end of their suppliers. In the example, B2Bi is about TravelSmart and AirWeb. As a step towards dynamic B2Bi, the focus of our work is on automatic reconciliation for the business interaction processes enforced by different companies.

A concrete example of process reconciliation can be derived from the interaction processes used by TravelSmart and AirWeb. Focusing on invoicing and payment, Figure 1 captures a high-level representation of the interaction processes required by the e-business systems of TravelSmart (Figure 1.a) and AirWeb (Figure 1.b). The interaction logic for TravelSmart involves waiting for the invoice, and then paying by direct debit. The interaction logic for AirWeb is to send an invoice, and to allow payment by either direct debit or electronic cheque. From AirWeb's perspective, invoicing and payment are two distinct processes that can occur in any order. Despite the structural differences in terms of interaction processes, the interaction logic of the two companies is clearly compatible. The essential piece of information required by AirWeb is that it should initiate the interaction with TravelSmart by using the invoicing process. AirWeb may also be interested to know that TravelSmart will never use one of the payment options. The essential piece of information for TravelSmart is that the interaction process it requires is perfectly matched by AirWeb.

Interaction processes are an essential component of a business offer, and the capability to adapt them consistently with the delivery capability of the company results in tangible competitive advantage. The end goal of process reconciliation is to identify common ground as well as possible incompatibilities in the interaction logic of the business partners. In particular, the objective is to provide indications to each party on the adjustments required to the respective interaction processes in order to adhere to mutually acceptable patterns of interaction. The outcome of the reconciliation translates into contractual obligations.

The challenge posed by electronic services is to automate the reconciliation procedure for interaction processes. The human factor remains crucial, but human contribution should be more in terms of modelling the reconciliation logic then on the application of such logic to specific reconciliation instances.

4. Process reconciliation technique

Summarising the content of the previous sections, the new generations of e-business systems are increasingly adopting service-oriented models and technology. Interaction processes become integral part of the interface exposed to business partners, and the capability to mutually adapt interaction patterns constitutes a key competitive advantage.

In this section, we propose a technique for the reconciliation of the interaction requirements of complementary business interfaces. The working assumption is that the interaction layer is implemented with Web Service technology.

4.1. Background

For illustration purposes, we assume a scenario in which two companies (A and B) have already agreed on the business content of the interaction. Company A could be TravelSmart, which already knows that AirWeb (company B) sells flights. The e-business systems of both companies support Web Services, and each company has described the preferred interaction processes as a set of WSDL and WSCL descriptions (I_A and I_B respectively). The reconciliation technique produces a pair of sets (I_{A1}, I_{B1}) of WSDL and WSCL descriptions that captures commonly acceptable interaction processes for A and B. The technique also produces indications on the way the WSDL and WSCL interfaces in the initial offer of each company can be used in order to enforce the common interaction processes.

The proposed approach focuses on the technical capabilities of interaction between the Web Service interfaces of the two companies. Specific business requirements may locally invalidate this assumption, hence the need for some form of business validation of the result produced. For example, AirWeb may require payment before invoicing from new partners like TravelSmart. The business validation issue is outside the scope of the work presented in this paper.

The idea for the reconciliation of I_A and I_B is to build new processes out of mutually acceptable modifications and combinations of existing processes. The approach does not completely solve issues raised by fundamental results of computability theory, such as the fact that the equivalence between processes is in general only semi-decidable. Still, the scope of this work is the support of business interaction. We rely on the assumption that businesses expose meaningful descriptions of their interaction requirement. Interaction processes are aligned with core production processes, and they are likely to inherit the structural simplicity of internal workflows.

4.2. Process unification

The general approach for the reconciliation of I_A and I_B is based on unification techniques derived from logic programming [13]. Individual processes are considered as clauses (facts) in a theory. Sets of processes such as I_A and I_B as well as their union are considered as theories. Fundamental difference in terms of unification rules is that matching is done between complementary elements instead of equals. For example, a `Send-Invoice` is matched with a `Receive-Invoice`. Every element in I_A is unified in a theory deriving from the combination of I_A and I_B, and the result added to I_{A1}. The combination of I_A and I_B is not a standard union of sets. The way in which I_A and I_B are used is described in more detail in the following section. Similarly, every element in I_B is unified in the combined theory, and the result added to I_{B1}. In practice, the construction of the sets I_{A1} and I_{B1} progresses in parallel. If the unification is not possible, nothing is added to I_{A1} or I_{B1} and a new element is considered.

The core of the unification technique we devised is based on an extension of the concepts and techniques for workflow inheritance proposed by van der Aalst in [2]. Van der Aalst focuses particularly on internal business processes, and their need to adapt to changing business conditions. Apart from the shift in focus towards interaction processes, the new dimension we add is the evolution of a process specification explicitly driven by other process specifications. Traditionally, research on workflow evolution has focussed more on execution history [9].

The transformation model proposed by van der Aalst provides two transformation methods called blocking and hiding. Blocking a task in a process implies that the task is no longer executed. In practical terms, the task is removed from the process description. Hiding a task in a process implies that the execution of the task can be ignored. The task can still be executed, but the execution does not affect the process. In both cases, the task virtually disappears from the process; but the impact can be profoundly different. In case of blocking, the transitions connected to that task are also erased. Entire branches of the process can be severed from the main tree and become unreachable. In case of hiding, the transitions are still available. The flow of the process simply traverses the task, independently from its execution. For example, we can consider the simple processes in (Figure 1). Hiding task `Receive-Invoice` in TravelSmart makes the process compatible with the payment process for AirWeb. Blocking `Receive-Invoice` in TravelSmart would instead make the `Send-Payment` task unreachable preventing the compatibility with AirWeb.

Our unification model extends the basic principles of hiding and blocking along different lines. First, we consider interaction steps instead of normal tasks. The concept of equal is replaced by the concept of complementary; extensively explored in frameworks such as Milner's CCS [15]. Second, we take a different approach to hiding. Instead of hiding the task where it appears explicitly, we introduce it explicitly also where it is implicitly present. Third, we consider multiple processes at the same time. The chain reaction triggered by the unification procedure can involve multiple elements from both I_A and I_B.

4.3. Unification procedure

Given a process p_A in I_A, p_A is used as a seed for the construction of two complementary (sets of) processes S_A and S_B to be added respectively to the reconciled interfaces I_{A1} and I_{B1}. The case of I_B is symmetric. The algorithm for constructing S_A and S_B revolves around the following procedure:

- Given *m* the first move in p_A, find an element p_B in I_B that begins with the complementary move *m** (the match is between send and receive)
- If p_B is found, create two temporary processes p_{A1} and p_{B1} starting respectively with *m* and *m**
- Proceed evaluating the structural compatibility of p_A and p_B, evolving p_{A1} and p_{B1} in accordance to the rules for blocking and hiding (description follows) for the cases in which there is no direct compatibility between the current processes
- If p_{A1} and p_{B1} reach a complete stage (all the moves are matched), go to the following step. If a complete stage is not reachable, start from the beginning with a different element in I_A
- Add the resulting sets S_A and S_B to I_{A1} and I_{B1}

As anticipated, blocking and hiding need to be adjusted to the interaction requirements for Web Services and the properties of WSDL and WSCL. The blocking of a move *m* in p_A is applied only if two tests fail at the same time. First, there must be no matching move among the reachable next steps of the current complementary process p_B. The definition of reachable depends on the structural connection with the current steps (e.g. sequential or conditional) [17]. Second, there must be no other p_{B*} elements in I_B having *m** as first move. In the first case, the compatibility is found directly within p_A and p_B. In the second case, p_{B*} can be virtually connected with p_B and temporarily recreating the conditions for direct compatibility. The second case is also relevant to the hiding technique.

When compatibility is achieved by merging more than one process, the classic hiding rule would require hiding

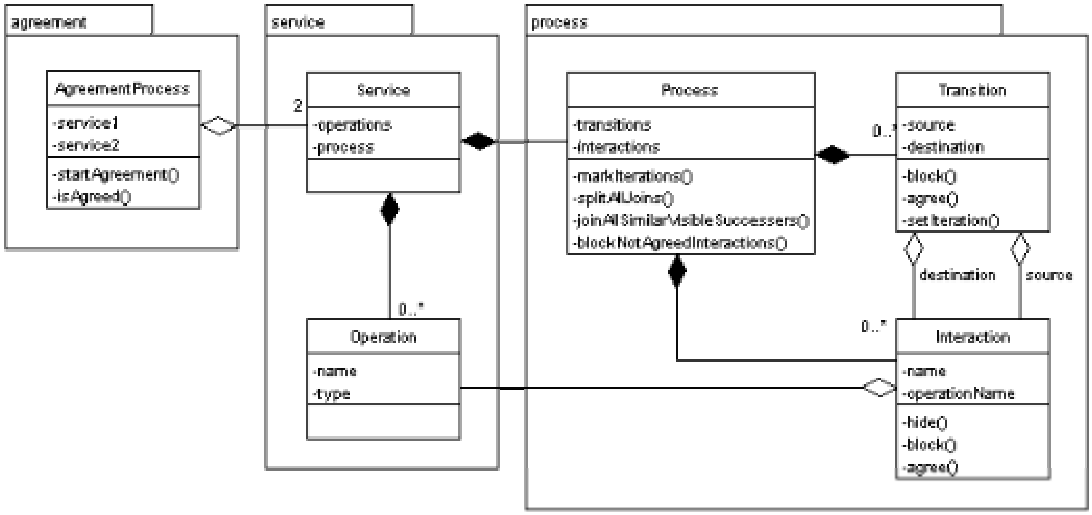

Figure 2: UML Class Diagram of the Process Reconciler

the moves in the current process that are matched by the new branch of the virtual process. We instead adopt the opposite approach, and explicitly represent the virtual branch of the complementary process. The objective is to capture the fact that different Web Service interfaces are used accordingly to a wider interaction process. We leave the possibility to discard the extra information acquired to an optional step in the reconciliation process.

During the execution of the algorithm, new processes from I_B can be used to extend p_B in the attempt to match p_A. At the same time, processes from I_A can be used to extend p_A in the attempt to match p_B. The use of the term cross-unification tries to capture the idea of an intertwined evolution for p_A and p_B.

4.4. Computational complexity

The computational cost of the unification procedure can be expressed in terms of the number of matches performed between process steps. The number of matches gives an indication of the time as well as the space required by the algorithm. We consider N the number of processes in I_A and M the number of processes in I_B.

In the optimal case, there is a direct one-to-one matching between the processes of the two companies. In this case the complexity is O(max(N,M)). The worst case can be constructed by introducing indirect loops. For example, I_A could include a process defined as the sequence of the steps `Send-Invoice` and `Receive-Payment`. I_B could include a process composed by the only step `Receive-Invoice`, and a second process defined as the sequence of the steps `Send-Payment` and `Receive-Invoice`. Such configuration would create an infinite loop of invoicing and payments. Standard loop-control mechanisms are in place in the actual implementation of the algorithm. In the worst case, all processes on both sides are modelled in a way that causes loops. In practice, such case is highly unrealistic. The complexity of the worst case is $O((N*M)^{(N*M)})$.

In the average case, the match between the processes of the two companies requires some form of composition and adaptation of the various processes. The presence of loops is limited in proportion to the overall number of processes involved in the reconciliation. We estimate the complexity of the average case as O(N∗M).

5. Prototype

The process reconciliation concept is embedded in a prototype we refer to as Process Reconciler that was implemented using Java technology and the DOM model for XML processing. It is engineered as a generic standalone service that can be used by various parts of an e-business platform for tasks like semi-automatic reengineering of the business-level interaction processes as well as subsequent control and management of technical Web Services interoperation. Thus, it exposes both manual and programmatic interfaces, consisting of an interactive tool, a Java API as well as a Web Service interface. Standard XML editing tools can be used to manage WSDL and WSCL documents.

(Figure 2) shows the architecture of the Process Reconciler using a UML class-diagram. It mainly consists of three major packages: `service`, `process`, and `agreement`. The `service` package covers the internal representation of static Web Service interfaces derived from WSDL specifications. It primarily includes a `Service` class, which holds a collection of `Operation`

type objects. The latter represent all external operations a specific service is capable of dealing

```
<Conversation name="AirWeb_Payment"
initialInteraction="Start"
finalInteraction="End">
<ConversationInteractions>
    <Interaction interactionType="Empty"
id="Start" />
    <Interaction interactionType="ReceiveSend"
id="receive_pay_by_debit">
        <operation name="pay_by_debit"/>
</Interaction>
    <Interaction interactionType="ReceiveSend"
id="receive_pay_by_cheque">
        <operation name="pay_by_cheque"/>
</Interaction>
    <Interaction interactionType="Empty"
id="End" />
</ConversationInteractions>
<ConversationTransitions>
    <SourceInteraction href="Start"/>
<DestinationInteraction
href="receive_pay_by_debit"/>
                <SourceInteractio
n href="Start"/> <DestinationInteraction
href="receive_pay_by_cheque"/>
    <SourceInteraction
href="receive_pay_by_debit"/>
<DestinationInteraction href="End"/>
    <SourceInteraction
href="receive_pay_by_cheque"/>
<DestinationInteraction href="End"/>
</ConversationTransitions>
</Conversation>
-----------------------------------------------
<portType name="InterfaceExample1">
       <operation name="invoice">
       <output message="Invoice"/>
</operation>
```

Figure 3.a: Part of AirWeb's payment process

with, discriminating interaction types that can be inbound, outbound or both.

A `Service` instance is always associated with a set of interaction processes. The information on service choreography is derived from WSCL process specifications and represented in the `process` package. This package includes a `Process` class, which handles collections of `Transition` type objects and `Interaction` type objects together expressing the interaction flow. Interactions are linked to associated operations. The explicit modelling of operations and interactions as distinct entities derives form the need to handle multiple occurrences of the same operation in a single related interaction flow. The same operation can be used in different moves of a process.

The actual unification procedure is implemented within the `agreement` package. The objective of the prototype was to show the feasibility of the reconciliation technique. Hence, basic techniques are used for activities such as loop handling and exploration of possibility spaces. Specific procedures enforce a simple form of normalisation for the process specifications. The main objective of this normalisation is to remove redundancies (e.g. identical first moves after a conditional branching point). The algorithm is then applied to the resulting sets of processes. In the end, Web Service interface specifications of the reconciled interaction processes for both service partners are being generated.

6. Example

In this section, we apply the Process Reconciler to the example of TravelSmart and AirWeb. AirWeb is proposing two interaction processes as shown in (Figure 1.b). The first process p_B contains only one outbound interaction `send_invoice`. P_B expresses the option of AirWeb to send an invoice at any time. The second process p_C represents the possibility for a business partner to choose between two types of payment. The option is expressed by the alternative inbound interactions `receive_pay_by_debit` and `receive_pay_by_cheque`. These processes are represented as Web Service interfaces using WSDL and WSCL. An extract of them is shown in (Figure 3.a).

TravelSmart is proposing the interaction process p_A (Figure 1.a). The process captures the fact that the company will first wait for an invoice (inbound interaction `receive_invoice`), and then pay by debit (outbound interaction `send_pay_by_debit`). The sequencing of the interaction steps can be quite important for TravelSmart. An extract from the WSCL definition of the process is presented in (Figure 3.b).

Using the Process Reconciler, the interaction processes of the two companies are used to define a mutually acceptable interaction ground. The reconciliation procedure starts from the processes proposed by TravelSmart. The first step is the match between `receive_invoice` in p_A and `send_invoice in` p_B. Next, the reconciliation mechanism tries to find the counterpart for `send_pay_by_debit of` p_A. The match is made possible by p_C, which is virtually merged with p_B. The procedure terminates with a successful result. The WSCL process represented in (Figure 3.b) represents also the result produced by the Process Reconciler.

The reconciliation procedure can then proceed with the processes proposed by AirWeb. The reconciliation of AirWeb process p_C fails, because it is not possible to match one of the starting interactions with TravelSmart process p_A. However, reconciliation succeeds for p_B where `send_invoice` matches `receive_invoice`. Because the end of p_A is not reached, p_C is considered for the next round of matching. As in the previous case, the match of one of the conditional branches of p_C with the remaining part of p_A succeeds. Looking at this last step in more detail, `receive_pay_by_debit` is matched to

receive_pay_by_debit and receive_pay_by_cheque is blocked. The part of the WSCL result produced by the Process Reconciler for AirWeb is similar to the one for TravelSmart, but send and receive operations are swapped.

```
<Conversation name="TravelSmart_Payment"
initialInteraction="Start"
finalInteraction="End">
<ConversationInteractions>
    <Interaction interactionType="Empty"
id="Start" />
    <Interaction interactionType="Receive"
id="receive_invoice">
        <operation name="invoice"/>
</Interaction>
    <Interaction interactionType="SendReceive"
id="send_pay_by_debit">
        <operation name="pay_by_debit"/>
</Interaction>
    <Interaction interactionType="Empty"
id="End" />
</ConversationInteractions>
<ConversationTransitions>
    <SourceInteraction href="Start"/>
<DestinationInteraction href="receive_invoice"/>
    <SourceInteraction href="receive_invoice"/>
        <DestinationInteraction
href="send_pay_by_debit"/>
    <SourceInteraction
href="send_pay_by_debit"/>
<DestinationInteraction href="End"/>
</ConversationTransitions>
</Conversation>
```

Figure 3.b: TravelSmart's payment process

The example highlights the issue of business validation mentioned in previous sections. Process p_A is a technically acceptable common ground for the e-business systems of the two companies. Still, the option to use the interaction pay_by_cheque is something that might be useful to TravelSmart, and a key feature in the offer of AirWeb. The value of such an option could justify an extension for the web interface of TravelSmart.

At the current stage of development, Web Service standards and technology focus entirely on the structural aspects of communication. Initiatives like the Semantic Web [5] show the interest from both the industry and the research community in addressing this type of problems. We are currently exploring the use of specific metadata to indicate a weight value for both individual moves and process regions. For example, the need to block a move with high weight could then result in a warning to be raised to the business validation level.

7. Related Work

The modelling of component behaviour is central to Web Services [10]. The focus is on the specification of transaction processing, business rules, and related forms of business logic. Examples representation models include RosettaNet [18], ebXML [22], the Business Process Modeling Language (BPML) [7], and the Business Transactions Protocol (BTP) [16]. Workflow [24] represents the main reference point in terms of both model and technology. Behavioural evolution [2, 9] and inter-organizational use [1] of workflows are particularly relevant to Web Services.

Based on the workflow model, different efforts are ongoing for the specification of Web-Service interaction and orchestration. The focus is on message flow. Main examples of such efforts are the Web Service Conversational Language (WSCL) [4], the Web Services Flow Language (WSFL) [12], and the Extensible Language (XLANG) [20]. An example of modelling and management of conversational Web Services can be found in [11]. General contributions to the behavioural specification of services come also from the semantic web initiative [5]. As part of DAML (DARPA Agent Markup Language), DAML-S is directly related to semantic description of Web Services, containing a process model for service operation [3].

The main issue with current approaches to behavioural descriptions for Web Services is dynamic change. Change requires a layer of meta-information that is almost completely absent at this stage. For example, information is required about the fact that parts of a behavioural specification are optional or mandatory. We propose that an explicit notion of adaptation capability is required in order to support the dynamic nature of Web Service.

8. Conclusions

Web Services play a fundamental role in the definition and implementation of B2B interaction processes. Using formalisms such as WSDL and WSCL, companies can capture and expose their interaction logic as sets of processes. The problem we address in this paper is the reconciliation of the processes exposed by distinct business partners.

The aspects of the process reconciliation we focus on are essentially two. The first aspect is the definition of a common set of processes that the parties can use for the interaction. The second aspect is the specific adaptation required to the processes of each party in order to support the common processes. Each aspect introduces specific requirements for the reconciliation algorithms, and conflicts arise. The tradeoffs are between the breadth of the common processes and the amount of internal change required to support them. The algorithm we propose takes a balanced approach towards both types of requirements. Common processes are built out of compatible portions of existing processes. Prerequisite for the effectiveness of the algorithm is the use of a common ontology (e.g. RosettaNet) by the parties.

The algorithm focuses on process reconciliation from a technical perspective. Validation of the resulting processes from a business perspective is a crucial requirement for a complete solution. We perceive explicit management of process-level policies as a fundamental element for business validation. Process-level policies are among the main lines of development for our work.

Bibliography

[1] W. M. P. v. d. Aalst and K. Anyanwu, "Inheritance of Interorganizational Workflows to Enable Business-to-Business E-commerce", In *Proc. 2nd Int. Conf. on Telecommunications and Electronic Commerce (ICTEC)*. Nashville, Tennessee, USA, 1999, pp. 141-157.

[2] W. M. P. v. d. Aalst and T. Basten, "Inheritance of workflows: an approach to tackling problems related to change", *Theoretical Computer Science*, vol. 270, 2002, pp. 125-203.

[3] A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila, D. L. Martin, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng, "DAML-S: Semantic Markup For Web Services", In *Proc. Internationl Semantic Web Working Symposium (SWWS)*, 2001.

[4] A. Banerji, C. Bartolini, D. B. A. Chopella, K. Govindarajan, A. Karp, H. Kuno, G. P. M. Lemon, S. Sharma, and S. Williams, "Web Services Conversation Language (WSCL) 1.0", W3C Technical Note, 2002.

[5] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web", *Scientific American*, vol. 284, 2001, pp. 34-43.

[6] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocol (SOAP) 1.1", W3C Note, 2000.

[7] BPMI, "Business Process Management Initiative", http://www.bpmi.org, 2002.

[8] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1", W3C Note, 2002.

[9] P. Heinl, S. Horn, S. Jablonski, J. Neeb, K. Stein, and M. Teschke, "A comprehensive approach to flexibility in workflow management systems", In *Proc. Work Activities Coordination and Collaboration (WACC)*, San Francisco, California, 1999, pp. 79-88.

[10] H. Kuno, "Surveying the E-Services Technical Landscape," in *Proc. Second International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems (WECWIS 2000)*. Milpitas, California, USA: IEEE Computer Society, 2000.

[11] H. Kuno and M. Lemon, "A Lightweight Dynamic Conversation Controller for E-Services", In *Proc. 2nd Int. Workshop on Advance Issues of E-Commerce and Web-Based Information Systems*, Milpitas, California, USA, 2001.

[12] F. Leymann, "Web Services Flow Language (WSFL 1.0)", IBM, 2002.

[13] J. W. Lloyd, *Foundations of Logic Programming*, 2nd edition, Springer, 1987.

[14] J. Miller and J. Mukerji, "Model Driven Architecture", Object Management Group (OMG), 2001.

[15] R. Milner, "A Calculus of Communicating Systems", In *LNCS 92*, Springer-Verlag, 1980.

[16] OASIS, "Business Transactions", http://www.oasis-open.org/committees/business-transactions, 2002.

[17] G. Piccinelli, "Exposing Models of Behaviour of E-Service Components," in *Proc. 6th London Communication Symposium*. London, UK, 2001.

[18] RosettaNet, http://www.rosettanet.org, 2002.

[19] A. Sillitti, T. Vernazza, and G. Succi, "Service Oriented Programming: a New Paradigm of Software Reuse", In *Proc. of the 7th Int. Conference on Software Reuse*, *LNCS*, 2002.

[20] S. Thatte, "XLANG - Web Services for Business Process Design", Microsoft, 2002.

[21] UDDI, "Universal Description, Discovery and Integration", http://www.uddi.org, 2002.

[22] UN/CEFACT and OASIS, "Electronic Business XML", http://www.ebxml.org, 2002.

[23] W3C, "Semantic Web Initiative", http://www.w3.org/2001/sw, 2002.

[24] WFMC, "Workflow Management Coalition" http://www.wfmc.org, 2002.