Protocols for Processes: Programming in the Large for Open Systems (Extended Abstract)

Munindar P. Singh, Amit K. Chopra, Nirmit V. Desai, Ashok U. Mallya
{singh,akchopra,nvdesai,aumallya}@ncsu.edu

Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7535, USA

ABSTRACT

The modeling and enactment of business processes is being recognized as key to modern information management. The expansion of Web services has increased the attention given to processes, because processes are how services are composed and put to good use. However, current approaches are inadequate for flexibly modeling and enacting processes. These approaches take a logically centralized view of processes, treating a process as an implementation of a composed service. They provide low-level scripting languages to specify how a service may be implemented, rather than what interactions are expected from it. Consequently, existing approaches fail to adequately accommodate the essential properties of the business partners in a process (the partners would be realized via services)—their *autonomy* (freedom of action), *heterogeneity* (freedom of design), and *dynamism* (freedom of configuration).

Flexibly represented *protocols* can provide a more natural basis for specifying processes. Protocols specify *what* rather than *how*; thus they naturally maximize the autonomy, heterogeneity, and dynamism of the interacting parties. We are developing an approach for modeling and enacting business processes based on protocols. This paper describes some elements of (1) a conceptual model of processes that will incorporate abstractions based on protocols, roles, and commitments; (2) the semantics or mathematical foundations underlying the conceptual model and mapping global views of processes to the local actions of the parties involved; (3) methodologies involving rule-based reasoning to specify processes in terms of compositions of protocols.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures; D.2.10 [**Software Engineering**]: Design; D.2.13 [**Software Engineering**]: Reusable Software; I.2.11 [**Artificial Intelligence**]: Distributed Artificial IntelligenceMultiagent Systems

OOPSLA'04, Oct. 24-28, 2004, Vancouver, British Columbia, Canada.
ACM 1-58113-833-4/04/0010.

General Terms

Standardization, Languages, Design

Keywords

Open Systems, Interaction Protocols, Business Processes

1. INTRODUCTION

We think of business process modeling and enactment as a form of programming in the large [8]. Programming in the large is distinguished from the more common programming in the small in several ways. Programming in the large emphasizes putting together large software components (built by several people over a long period of time) and having a local state, whereas programming in the small is about developing the individual components. Programming in the large is a much more challenging engineering problem.

Business processes, which span multiple autonomous business partners, are an example of programming in the large. Cross enterprise processes, in particular, involve a rich variety of interactions among software components that are independently designed and configured, and which represent independent (and sometimes mutually competitive) business interests. Because Web services simplify interoperation, they have led to a resurgence of interest in technologies for process modeling and enactment.

Current approaches for the modeling and enactment of business processes are woefully inadequate, and reflect the similar inadequacy of programming in the large in open environments. Whereas the initial attempts at programming in the large were based on simple module interconnections, more serious attempts, such as megaprogramming, employed ontologies to address the heterogeneity of the information processes by various components [19].

However, no good abstractions have been developed to model the interactive processing of information by multiple components. Consequently, processes are still specified today as scripts essentially providing the same level of abstraction as developed in the job control languages (JCLs) of the mainframes of the 1950s. The above claim may sound unduly harsh, but while we readily acknowledge progress in technologies for process management, in terms of abstractions for modeling processes the progress is incremental at best. Whether you consider any of the modeling tools in use today, or leading proposed standards such as the Business Process Execution Language for Web Services (BPEL) [1] or the Web Ontology Language for Services (OWL-S) [7], the process abstractions they offer are constructs such as sequence, iterate, fork, and

join. In other words, the abstractions are little more than what you might find in a JCL. Hence our claim above.

But, still, why are such abstractions not adequate for processes? Imperative languages are abstractions that are best suited to programming in the small. They assume the invoked components behave as expected. However, the main problems that arise in processes are problems of programming in the large. Because of environmental effects, exceptions can arise. Because the participants are autonomous, they can act to exploit opportunities, and their behavior may appear to be unexpected to their partners.

The effect of the above is that current approaches for process modeling and enactment have some key, well-known limitations in practice. They are either too rigid (thus frustrating users and causing systemic inefficiencies), or are extremely expensive in time and effort to construct and manage, and usually both. Thus some of the benefits of openness are lost.

2. VISION

Our diagnosis of the above challenges is that they reflect a fundamental problem for programming in the large. The traditional (imperative) scripting constructs specify flows. Clearly any process execution must correspond to a flow. However, this does not mean that we must specify the set of flows explicitly. Instead, we propose that processes be captured in terms of protocols, where each protocol is a flexible encoding of a meaningful set of interactions.

We define *(business) protocols* as publishable specifications of business interactions. We propose to model a business process as a composition of protocols. For example, we can have protocols for negotiation, for payment, for selecting a shipping company, and so on. A protocol is an interface, meaning that it specifies only the key desired aspects of the interactive behavior, not how the interacting parties are implemented.

Each protocol constrains the business partners involved in it. Protocols are modular, i.e., functionally decentralized. For example, a payment protocol between a customer and a merchant would be specified independently of the merchant's inventory fulfillment protocol for ordering goods from its suppliers. Thus, capturing processes via protocols enables us to more easily represent and enact interactions among autonomous business partners.

To model a process, we first identify the protocols using which the different participants interact. For example, a merchant and a customer may interact with each other using a negotiation protocol; the merchant, customer, and payment agency may interact via an escrow protocol; and, the merchant, customer, and shipper may interact through some specialized logistics protocol. When each participant acts according to its local reasoning but respecting the stated protocols, a multiparty business process is enacted but without a global flow necessarily ever having been explicitly encoded.

2.1 Benefits

Our protocol-based approach offers the following natural advantages. One, for process *design*, protocols are naturally reusable whereas complete processes are not. More importantly, protocols lend themselves to modeling abstractions such as specialization and aggregation. Two, for process *enactment*, when protocols are flexible, they enable each party to exercise some discretion in applying its local policies or preferences while obeying a protocol. For example, a merchant may accept only cash for discounted goods and a customer may prefer to pay for goods after using them for a month. This flexibility also enables us to capture and handle business exceptions and opportunities in a natural manner at the level of protocols. Three, for process *monitoring*, protocols provide a clean basis for determining that the interacting parties are complying with the given protocols.

2.2 Trends

Just as network protocols enabled the expansion of the lower layers of the Web architectures, business protocols will enable the development of processes involving autonomous, heterogeneous business partners. For this reason, we expect to see an increasing set of business protocols to be published and custom protocols to be designed. Several business protocols have been defined. Some general-purpose ones are NetBill [18], Secure Electronic Transactions (SET) [17], Internet Open Trading Protocol (IOTP) [11], and Escrow [10].

RosettaNet is a leading industry effort, involving about 400 electronics and telecommunications companies. The RosettaNet [16] Partner Interface Processes (PIPs), of which 107 are currently listed, are business protocols in spirit. These modularly describe several important business interaction scenarios. RosettaNet is in active production use with several billion dollars worth of commerce being conducted over it, e.g., [12]. Another major industry effort is ebXML [9]. ebXML is similar to RosettaNet but more general in style. Some of RosettaNet's components are gradually shifting over to using ebXML, e.g., for messaging formats. ebXML's Business Process Specification Schema (BPSS) describes partner roles and the documents they would exchange. RosettaNet's PIPs map to instances of BPSS. ebXML's Collaboration Protocol Agreement (CPA) describes an agreement (including conversations) between collaborating parties that is derived from their individual profiles.

We find the above trends, along with the well-known expansion of service-oriented computing, extremely encouraging. These trends clearly suggest that industry has understood the problem of developing cross-enterprise information systems, and is showing us researchers the way (in terms of what is important). However, current integration efforts are tedious and expensive, because they require extensive hard-coding. RosettaNet's limitations include that the PIPs specify interactions rather rigidly and do not offer a formal semantics. Further, the PIPs seem to specify some internal operations of each partner. Lastly, the interactions are short (in fact, just involving two parties and typically no more than a request and a response pair) and exceptions, where accounted for, are encoded as separate PIPs. ebXML has the same limitations.

Consequently, although current specifications of the business protocols are lacking in some respects, an increasing set of such protocols is an indication of the significance of our approach. The main reason behind the above limitations is that while business processes apply among autonomous, heterogeneous partners, the programming abstractions are still based on closed systems.

3. KEY TRADITIONAL APPROACHES

We briefly outline the key approaches here. Conventionally, a business process is modeled in a conceptually centralized manner as a global flow. Emerging standards and tools support the specification and enactment of business processes specified as flows, but they cannot escape the fundamental limitations of such representations. One, it is difficult to create and maintain flows: they become complex in the face of exceptions and cannot easily be verified. Two, because a flow represents a central view, it inevitably limits the autonomy and heterogeneity of the business partners involved, leading to a suboptimal treatment of exceptions and opportunities. Three, flow representations are about how a process or composed service is implemented; what we need are interfaces that assure us that independent implementations will interoperate.

It helps to distinguish *orchestration* (how a process is implemented by composing services) from *choreography* (how services interact) [15]. Both BPEL [1] and OWL-S [7] emphasize orches-

tration by encoding flows. Service composition has drawn an increasing amount of attention lately. However, work in this area has concentrated on orchestrating services so as to accomplish a desired composition. For example, McIlraith *et al.* [14] and Cardoso and Sheth [5] show how composed services may be put together. This body of work addresses how a composed service may be implemented, not how an autonomous service would interact with other services.

By contrast, the Web Services Choreography Interface (WSCI) [20] describes "conversations" or constraints on how a service is willing to interact with others, e.g., by requiring login before purchase. Unfortunately, existing choreography approaches rigidly specify a series of message exchanges but without an account of how they could be modularized and combined and how they could be related computationally to the orchestration approaches.

Like choreography, a protocol describes how services interact, but unlike traditional choreography, a protocol would consider the perspective of the interaction rather than of a particular participant. Further, protocols are conceptually composable to yield processes. Lastly, protocols would map into the flows of the participants, where the flows would interact to yield the desired process, thereby fulfilling the purpose of orchestration as well.

4. DISCUSSION

We introduced a technical approach for modeling protocols that provides a natural basis for principled methodologies for designing custom business protocols. The main contributions lie in the formalization of protocol specialization and aggregation. This can further be employed to perform subsumption reasoning and to carry out more interesting operations on protocols, such as splicing. Related work can broadly be classified under the following research areas.

4.1 Why is this an Onward! Paper

Programming in the large as a topic has been around for almost three decades. Business process management has seen a phenomenal resurgence in interest as it has expanded into cross-enterprise settings to serve the needs of e-business. The challenges the open environments pose for process modeling and enactment call for new approaches for software development. We provide a promising such approach. This paper outlines several technical aspects of this approach, indicating that it could be a promising new paradigm for scientific contributions. However, the results might need further refinement before they are ready for conventional forums.

4.2 Existing or Emerging Standards

Among conventional standards, BPEL [1] primarily captures a flow model. BPEL also includes a significant component of data handling. BPEL also includes so-called protocols, which are modeled as processes whose variables are bound late to values. WSCI describes conversations in which a given service may participate. WSCI [20] conversations are not business protocols. Protocols impose interrelated requirements on all participants, not just on the users of a particular service. Each WSCI specification corresponds to a role in our scheme. WSCI Conversations are specified without any semantics, so that transitions and states cannot be reasoned about.

The transactional requirements of the composition could be partially derived from the nature of the composition. A leading approach for transactional support for Web service computations is described in the WS-Coordination [4] and WS-Transaction specifications [3], which has support for Atomic Transactions (ATs) and Business Activities (BAs). Most of our intended applications will fall under business activities. OASIS's Business Transaction Protocol (BTP) introduces an alternative, but similar, framework for coordinating Web transactions [6]. BTP includes atom (all or none) and cohesion (application-specific) transactions. The Web Services Composite Application Framework (WS-CAF) [2] is another industry initiative to support transactional properties of business processes. The above approaches are similar enough for our purposes. They take a flow-oriented stance on processes and attempt to encapsulate certain steps as transactions. Our approach would apply transactional properties at the level of protocols.

Semantic Web efforts have converged into the Web Ontology Language (OWL) [13]. One of the best current works on semantic Web services is OWL-S (derived from DAML-S) [7]. OWL-S is an OWL ontology for services, which includes service grounding, service profile, and process model. The profile is key for specifying and discovering services. The process model describes how a service may be implemented in terms of a set of scripting constructs, such as for sequencing, concurrency, branching, and iteration. In this respect, OWL-S resembles BPEL, which describes processes with which services can be implemented as compositions of others. Although such process specifications may be useful in tools to implement services, they are not directly suited for standardization. In our approach, some of this functionality shifts to protocol specifications.

4.3 Contributions and Directions

To summarize, we have argued that the problems of programming in the large arise with renewed vigor in open environments, as exemplified by the world of cross-enterprise business process management. Current programming abstractions are ineffective for such settings. A simple, but promising, idea is to modularize interactions analogous to the way we traditionally modularize local behaviors: let's have roles and protocols as first-class entities just as classes and methods were in traditional programming. We described key elements of our research program, which in simple terms, seeks to take the above idea to its logical conclusion. We hope to have convinced the reader that a number of benefits can be derived from this exercise.

We don't claim that all important problems are solved or even framed with precision, but we think we are getting there. To this end, we have identified some avenues of research that are especially promising.

Design. Sophisticated process design tools based on our theory of processes and protocols would help designers create process models with greater productivity. Key questions are about design rules and sanity checks on processes and protocols.

Enactment. An execution framework for agents to play roles in different protocols so as to enact a process in a truly distributed manner would maximize the autonomy of business partners. It would eliminate the need for centralized process execution with its concomitant bottlenecks and failure modes.

Monitoring. Compliance verification is a key challenge for open environments. It becomes even more important when the participants are given more autonomy as in our approach. The richer semantics of our approach, in terms of commitments and temporal models, would help us develop stronger results than in other approaches.

Negotiation. We can frame the choice of protocol as a run-time decision to be made by the various participants. In particular,

participants would then use the refined version of a protocol that best matches their local preferences. For example, a customer might choose to deal with a business that requires no authentication steps rather than with a business that offers similar services, but requires a number of authentication steps. In other cases, the parties involved may negotiate with each other to settle upon the particular refinement of a protocol that they would follow in their mutual interactions.

Overall, we believe this could prove to be a fruitful research program for our community.

5. FULL PAPER

The full paper (available from the author) describes our motivations in greater detail, presents our technical framework based on commitments, and introduces our specific approach for modeling and composing protocols. For reasons of space, this extended abstract skips most of these details.

6. ACKNOWLEDGMENTS

This research is supported by DARPA and by the National Science Foundation under grant DST-0139037.

7. REFERENCES

[1] BPEL. Business process execution language for web services, version 1.1, May 2003. www-106.ibm.com/developerworks/webservices/library/ws-bpel.

[2] D. Bunting, M. Chapman, O. Hurley, M. Little, J. Mischkinsky, E. Newcomer, J. Webber, and K. Swenson. Web services composite application framework (WS-CAF), July 2003. http://www.iona.com/ devcenter/ standards/ WS-CAF/ WS-CAF.pdf.

[3] F. Cabrera, G. Copeland, B. Cox, T. Freund, J. Klein, T. Storey, and S. Thatte. Web services transaction (WS-Transaction), Aug. 2002. http://www-106.ibm.com/ developerworks/ webservices/ library/ ws-transpec/.

[4] L. F. Cabrera, G. Copeland, W. Cox, M. Feingold, T. Freund, J. Johnson, C. Kaler, J. Klein, D. Langworthy, A. Nadalin, D. Orchard, I. Robinson, J. Shewchuk, and T. Storey. Web services coordination (WS-Coordination), Sept. 2003. ftp://www6.software.ibm.com/ software/ developer/ library/ ws-coordination.pdf.

[5] J. Cardoso and A. Sheth. Semantic e-workflow composition. *Journal of Intelligent Information Systems (JIIS)*, 12(3):191–225, Nov. 2003.

[6] S. Dalal, S. Temel, M. Little, M. Potts, and J. Webber. Coordinating business transactions on the Web. *IEEE Internet Computing*, 7(1):30–39, Jan. 2003.

[7] DAML-S. DAML-S: Web service description for the semantic Web. In *Proceedings of the 1st International Semantic Web Conference (ISWC)*, July 2002. Authored by the DAML Services Coalition, which consists of (alphabetically) Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne and Katia Sycara.

[8] F. DeRemer and H. H. Kron. Programming-in-the-large versus programming-in-the small. *IEEE Transactions on Software Engineering*, 2(2):80–86, June 1976.

[9] ebXML. Electronic business using eXtensible markup language, 2002. Technical Specifications release, URL: http://www.ebxml.org/specs/index.htm.

[10] Escrow.com. Online escrow process, 2003. http://www.escrow.com/solutions/escrow/process.asp.

[11] IOTP. Internet open trading protocol (IOTP), Oct. 2003. IETF: Internet Engineering Task Force, http://www.ietf.org/html.charters/trade-charter.html.

[12] T. Krazit. Intel conducts $5b in transactions via RosettaNet, Dec. 2002. http://archive.infoworld.com/ articles/ hn/ xml/ 02/12/10/ 021210hnintelrose.xml.

[13] D. L. McGuiness and F. van Harmelen. Web Ontology Language (OWL): Overview. www.w3.org/TR/2003/WD-owl-features-20030210/, Feb. 2003. W3C working draft.

[14] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic Web services. *IEEE Intelligent Systems*, 16(2):46–53, Mar. 2001.

[15] C. Peltz. Web service orchestration and choreography. *IEEE Computer*, 36(10):46–52, Oct. 2003.

[16] RosettaNet. Home page, 1998. www.rosettanet.org.

[17] SET. Secure electronic transactions (SET) specifications, 2003. http://www.setco.org/ set_ specifications.html.

[18] M. A. Sirbu. Credits and debits on the Internet. *IEEE Spectrum*, 34(2):23–29, Feb. 1997.

[19] G. Wiederhold, P. Wegner, and S. Ceri. Toward megaprogramming. *Communications of the ACM*, 35(11):89–99, Nov. 1992.

[20] WSCI. Web service choreography interface 1.0, July 2002. wwws.sun.com/ software/ xml/ developers/ wsci/ wsci-spec-10.pdf.