Information Technology and Management 6, 61–87, 2005

Towards Adaptive Workflow Enactment Using Multiagent Systems

PAUL A. BUHLER * pbuhler@cs.cofc.edu
College of Charleston, Department of Computer Science, 66 George Street, Charleston, SC 29424, USA

JOSÉ M. VIDAL vidal@sc.edu
University of South Carolina, Computer Science and Engineering, Columbia, SC 29208, USA

Abstract. Advances in Information Technology have created opportunities for business enterprises to redesign their information and process management systems. The redesigned systems will likely employ some form of workflow management system. Workflow management systems exactly enact business processes described in a process description language. Unfortunately, such strict adherence to the prescribed workflow makes it impossible for the system to adapt to unforeseen circumstances. We firmly believe that the historic trajectory of software development paradigms and IT advancements will establish multiagent systems as the workflow enactment mechanism of the future. In this paper we provide a critical survey of workflow, workflow description languages, web services and agent technologies. We propose that workflow description languages and their associated design tools can be used to specify a multiagent system. Specifically, we advance the idea that the Business Process Execution Language for Web Services (BPEL4WS) can be used as a specification language for expressing the initial social order of the multiagent system, which can then intelligently adapt to changing environmental conditions.

Keywords: Web services, software composition, workflow management systems, multiagent systems

1. Introduction

Advances in Information Technology (IT) are creating opportunities for business enterprises to redesign their information and process management systems. Foundational technologies for a universal enterprise integration platform are emerging. The refinement of service-oriented architectures and the emergence of web-enabled, semantically described services allow us to envision a future where these Web services become the next generation of enterprise components. Recently, the term servicization [29] has been coined to discuss the act of converting existing enterprise applications into Web services. This new enterprise software vision will require new integration strategies. "The traditional programmed interactions between people and software are [being] replaced by task-focused interactions that are dynamic and flexible" [29, p. 216]. This places new demands on software architectures because they will need to support computing with "dynamically-formed, task-specific, coalitions of distributed autonomous resources" [6, p. 99]. These changes are a logical consequence of the seminal work

* Corresponding author.

in coordination technology done by Gelertner. As a result of Gelertner's work, the old computer science adage *Applications* = *Algorithms* + *Data* is being replaced by *Applications* = *Computation* + *Coordination* [30].

It is now generally accepted that Gelertner was correct when he theorized that computation was orthogonal to coordination [7]. This orthogonality was implied by DeRemer, who wrote in 1976, "Structuring a large collection of modules to form a 'system' is an essentially distinct and different intellectual activity from the construction of the individual modules [themselves]" [4]. From these perspectives, a software system is viewed as an ensemble of coordinables and their orchestrated interactions. Coordinables are entities that function as independent units of computation. The coordinated interaction of the computational units produces the desired behavior of the system. Obvious parallels to workflow systems exist; the workflow activities are the coordinables and business processes coordinate their interaction.

Leymann asserts that workflow construction can be viewed as a two-level programming problem [20, p. 217]. His view is that the implementation of workflow activities is akin to traditional programming, or programming in the small. Activities encapsulate well-defined functionality that typically involves low-level data access routines and algorithmic processing. In contrast, the building of the workflow's process model is akin to programming in the large. The process model prescribes coordination rules by providing a means to express the sequencing of the activities and the flow of data amongst them.

We advocate the synthesis of Gelertner's and Leymann's points of view. We believe that the statements *workflow* = *activities* + *processes* and *applications* = *computation* + *coordination* are equivalent. The paper presents our contribution to enterprise integration, the use of multiagent systems for flexible enactment of enterprise workflows. Our view can be summarized by the aphorism *Adaptive Workflow Engines* = *Webservices* + *Agents*. In this context, the Web services provide the computational resources and the Agents provide the coordination framework. We propose the use of the Business Process Execution Language for Web Services (BPEL4WS) as a specification language for expressing the initial social order of the multiagent system.

In this paper a brief background of enterprise software is presented. This is followed by a section that examines the relationship between predominate software abstractions and enterprise software architecture. An examination of workflow systems motivates the discussion of BPEL4WS, DAML-S and agents. Finally, the paper concludes with a discussion of related and future work.

2. Enterprise software

Business enterprises are organizations that perform collective work. In order to achieve necessary operational efficiencies, the work needs to be governed by processes that define the flow of work throughout the organization. Regulated business processes are important because they reduce transactional costs when compared to ad-hoc approaches. In fact, according to Coase [47], the existence of the enterprise itself is dependent upon its

ability to achieve lower internal transactional costs than the cost of performing the same work in open markets. Of course enterprises cannot service every need internally, because at some point the overhead burden of these operations exceeds the acquisition cost from the marketplace. When this occurs, businesses form partnerships and cooperative agreements with one another.

Economic arguments and competitive pressures have stimulated business to spend heavily on IT. IT holds the promise of reducing transactional costs via management of business processes and information. Initial IT spending was used to procure enterprise applications that manage information relating customers, suppliers, partners and employees; these key entities are those with whom the enterprise interacts and transacts. Enterprise applications are categorized by the types of information and interactions they manage. For example, Customer Relationship Management (CRM), Supply Chain Management (SCM) and Enterprise Resource Planning (ERP) are the traditional categories of enterprise applications.

After enterprise applications were in place, IT spending targeted Enterprise Application Integration (EAI) initiatives. EAI allows businesses to further leverage their investment in enterprise software by providing the infrastructure to enable the sharing of data across organizational, system, and application boundaries. EAI improves the visibility and flow of information within the organization, thus increasing the enterprises responsiveness to marketplace demands.

2.1. Enterprise software architecture trends

Since enterprise integration solutions are software driven, it is important to examine them within the context of software development abstractions. This discussion will illuminate the interdependence of enterprise software architecture, IT, and software development abstractions.

The evolution of programming paradigms has been occurring for half a century. Programming has transitioned from the earliest days of hard-wired machines to today's component and agent-based approaches. Table 1, a modified version of [25, table 1], clearly demonstrates that software abstractions have been evolving toward implementation methodologies that feature increasing levels of localization. Localization is a side effect of encapsulation and it is important because it reduces the interdependence between units of code. When units of code are tightly coupled, they become tangled and difficult to independently deploy. Ideally, software developers work with abstractions that enable the design and implementation of systems with units of code that exhibit loose coupling and high functional cohesion [27]. These features are desirable because they enable greater software reuse.

Over time, enterprise software architecture has evolved in step with IT trends and the influence of software abstractions; figure 1, based in part upon [5, figure 1], charts this evolution. In the mid-1970's structured programming allowed developers to modularize their software in more controllable ways. In the late 1970's and early 80's, relational databases entered the marketplace. These two developments enabled soft-

Table 1
Evolution of programming paradigms.

	Monolithic	Structured	Object oriented	Component based	Agent oriented
How does a unit behave? (Code)	**External**	Local	Local	Local	Local
What does a unit do when it runs? (State)	**External**	**External**	Local	Mixed[a]	Local
When does a unit run? (Control)	**External**	**External**	**External**	**External**	Local
	→ Increasing software localization				

[a] Components are predominately stateless; however, stateful components do exist, e.g. stateful session beans. For this reason, State is designated as 'Mixed' in the Component Based column.

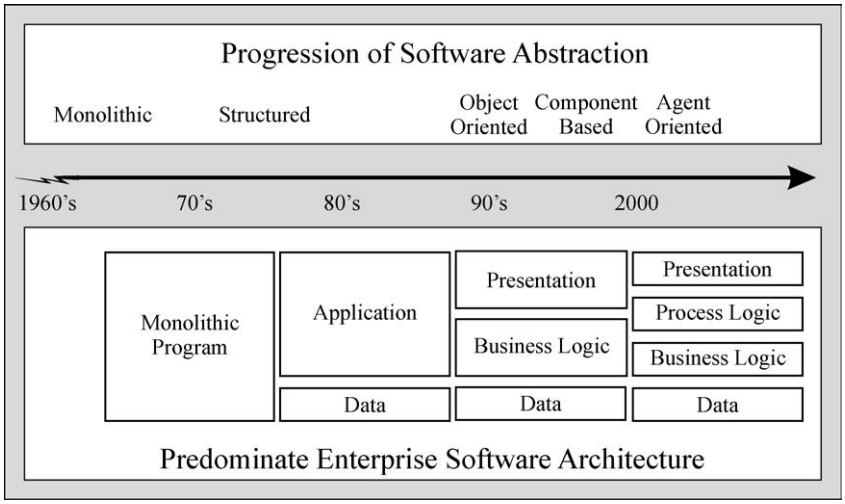


Figure 1. Evolution of enterprise software architectures.

ware architectures that separated the application logic from the data that the application processed. This separation was leveraged by the client/server computing model which became predominate with the introduction of Local Area Network (LAN) technology and the shift toward desktop computing.

In the late 80's and early 90's, the widespread adoption of networks and Graphical User Interfaces (GUIs) brought about the next architectural shift. GUI programming employs an event-driven programming model, which is best managed by the Model-View-Controller (MVC) software architecture. MVC enforces separation of concerns: the Model encapsulates the application state which is maintained as data; the Controller defines the application's behavior in response to GUI events; and the View is responsible for the presentation of the model to the user. MVC architectures are deeply rooted in object-oriented technology.

In the late 90's and early 2000's, the Internet became an integration platform for enterprise applications. The MVC architecture is the basis for the familiar N-tier server-side architecture found in today's Internet based enterprise applications. Many of these server-side applications utilize the Java 2 Platform, Enterprise Edition (J2EE™) [33], which provides a component-based modular architecture. As an integration platform, the Internet has proven to be very flexible. Open standards, which reduce vendor lock-in and increase interoperability, are enabling corporate e-business initiatives. As businesses integrate across organizational boundaries, it becomes important to separate the 'public' process logic from the 'private' business logic. The process logic specifies the order and conditions under which things get done; whereas, the business logic specifies what gets done. Business Process Management (BPM) software is an emerging classification of integration software that treats business processes as first-class entities.

In figure 1, it can be seen that the agent-oriented software abstraction is destined to have an impact on enterprise software architectures. Currently, much attention is being focused on Web services and their suitability to BPM applications. A closer examination of Web services in the context of enterprise software follows.

2.2. Web services as enterprise software components

Software components are created in order that they may be composed. As stated by Szyperski [34], "Composition enables prefabricated 'things' to be reused by rearranging them in ever new composites". Composing an application via reuse of existing software assets can dramatically reduce the development time. Likewise, the quality of the application will also increase, if the reused software has previously been proven through testing and successful deployment. Obviously any software engineering technique that has the ability to reduce development time while simultaneously increasing the quality of the product is noteworthy.

The range of Component-Based Software Engineering (CBSE) practice can be constrained by the definition of a software component. The definition of a software component is hotly debated; a sampling of common definitions can be found in [10,11,28,34]. For the purposes of this paper, the definition presented by Heineman will be used. This definition was selected for several reasons: it has undergone extensive review and revision; the definition is architecturally neutral in that it does not favor any specific implementation language or component model; and it is abstract enough to be inclusive of the other commonly referenced definitions of a software component. The software component definition found in [10, p. 7] is:

> A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard.

A component model consists of a collection of standards that govern the interaction and composition of software components that conform to the model. Standards are essential to the concept of open systems, which are simply a collection of interacting

Table 2
Component model standards [45].

Standards for	Description
Interfaces	Specification of component behavior and properties; definition of Interface Description Languages (IDL)
Naming	Global unique names for interfaces and components
Meta Data	Information about components, interfaces, and their relationships; APIs to services providing such information
Interoperability	Communication and data exchange among components from different vendors, implemented in different languages
Customization	Interfaces for customizing components. User-friendly customization tools will use these interfaces
Composition	Interfaces and rules for combining components to create larger structures and for substituting and adding components to existing structures
Evolution Support	Rules and services for replacing components or interfaces by newer versions
Packaging and Deployment	Packaging implementation and resources needed for installing and configuring a component

software and hardware components. The interaction within the open system is defined by interface specifications that are complete, publicly available and non-proprietary [24]. Table 2 summarizes the basic elements of a component model.

From a workflow perspective, a composite software system can be viewed as a sequence of services operating upon data. Ideally these services should be language, platform and location independent [8]. Such services would then be interoperable, where interoperability is characterized by the "ability of two or more software components to cooperate despite differences in language, interface, and execution platform" [44]. Web services represent a new class of interoperable, web-enabled software service. Several specifications have been developed that form the basis of a component model for Web services; specifically, SOAP (Simple Object Access Protocol), WSDL (Web Service Description Language) and UDDI (Universal Description, Discovery, and Integration). These specifications are used to invoke, describe, publish, and discover Web services. They also embrace an open systems point of view: XML (eXtensible Markup Language) is utilized to exchange data in a neutral format and component communication occurs via open transport protocol like HTTP.

Two important features of Web services are that they have a network-addressable interface and they can be used to represent business concepts or services. These two characteristics are essential to the concept of enterprise components as introduced in [11]. Web services have become the enterprise components for the next generation of enterprise-level software and are now viewed as the new building blocks for enterprise software systems. As with any building block metaphor, the challenge is how to arrange

or compose the building blocks into larger structures. As indicated in table 2, composition standards are a critical element of a robust component model. From the enterprise application perspective, compositions of Web services can be viewed as a workflow; section 3 will introduce appropriate workflow concepts.

3. Workflow management systems

The Workflow Management Coalition (WfMC) is an international standards-setting organization of workflow vendors, users, analysts and university/research groups. The WfMC has been responsible for the creation of a workflow reference model and a glossary of standardized workflow terminology. These resources will be used to define workflow concepts with more precision. Several key terms are important to understanding the nature of workflow and to provide an underpinning for a discussion of contemporary trends in workflow management systems. The WfMC Terminology and Glossary [39] document provides the following definitions:

> *Workflow* – the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.
>
> *Business Process* – a set of one or more linked procedures or activities which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships.
>
> *Process Definition* – the representation of a business process in a form which supports automated manipulation, such as modeling, or enactment by a workflow management system. The process definition consists of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated IT applications and data, etc.
>
> *Workflow Management System* – a system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications.

In summary, a process definition is an abstract representation of a business process that can be consumed by a workflow management system in order to enact the workflow.

3.1. Adaptive workflow in context

Traditionally, workflow management systems have not been designed for dynamic environments requiring adaptive response. Currently, the need for adaptive workflow is being driven by the demands of e-commerce in both B2B and B2C space. Initial B2B automation activities were centered on Electronic Data Interchange (EDI) initiatives.

More recent work in the B2B space has focused on the development and deployment of ebXML (electronic business XML). With both EDI and ebXML the collaborating business partners predefine the terms of their electronic interaction. As discussed by Jenz, these technologies enforce regulated B2B interaction and as such, they create closed communities of business partners [43]. In comparison, views toward virtual organizations require flexible, on-the-fly alignment of business partners; in other words, adaptive workflow capabilities. These loose collaborations of business partners operate in open, non-regulated B2B/B2C scenarios [43]; intuitively, pre-negotiated collaboration agreements are a hindrance in these environments.

Available workflow management systems span a range of capability. This is not surprising since businesses in any segment can benefit from workflow management. For example the insurance industry benefited greatly from document management systems, which reduce physical paperwork, increase the availability of documents, and control the flow of information during processing. As shown in figure 2(a), adapted from [20, p. 10], workflows can be broadly categorized by their business value and repetition rates. Focusing on the two highest value workflow categories, figure 2(b), a modification of [42, p. 193], differentiates collaborative and production-oriented workflows.

Collaborative and production-oriented workflows are distinguished by measures of structure and centricity. Collaborative workflows are information centric. Typically, human interpretation of information drives the workflow in a loosely structured manner. Collaborative workflows are sometimes described by the term Computer Supported Cooperative Work (CSCW); groupware and other shared workspace tools are often the vehicles for CSCW. In comparison, production workflows are process driven due to their highly repetitive nature. To achieve the efficiency required of production workflow, the processes are highly structured. Today's agile manufacturing environments are controlled by Manufacturing Execution Systems (MES) that schedule production based upon highly detailed process plans. As indicated in figure 2(b), the requirements of adaptive workflow fall between these two broad categories.

Adaptive workflows need to react to changing environmental conditions. Currently, businesses change their workflows through two primary mechanisms: Business Process Reengineering (BPR) and Continuous Process Improvement (CPI). Figure 2(c), adapted from [42, p. 239], illustrates the difference between BPR and CPI. BPR is the periodic analysis and subsequent redesign of the intra- and inter-business processes used by an organization. BPR is used to overhaul processes in order to create operational efficiencies that improve quality and save time and cost. Conversely, CPI focuses on continuous improvement through the application of small and orderly changes. Workflows are continuously examined in order to find ways to increase quality and reduce waste. Adaptive workflows respond to changing conditions through adaptive change. As shown in figure 2(c), adaptive change is not constrained by measures of frequency or impact.

Current workflow initiatives have embraced the Web service model. Given the current state of technology, Web service based workflows typically are deployed behind corporate firewalls and are used for intra-organizational workflow. The reason for this is that Web service specifications are weak in regards to issues of security, transaction

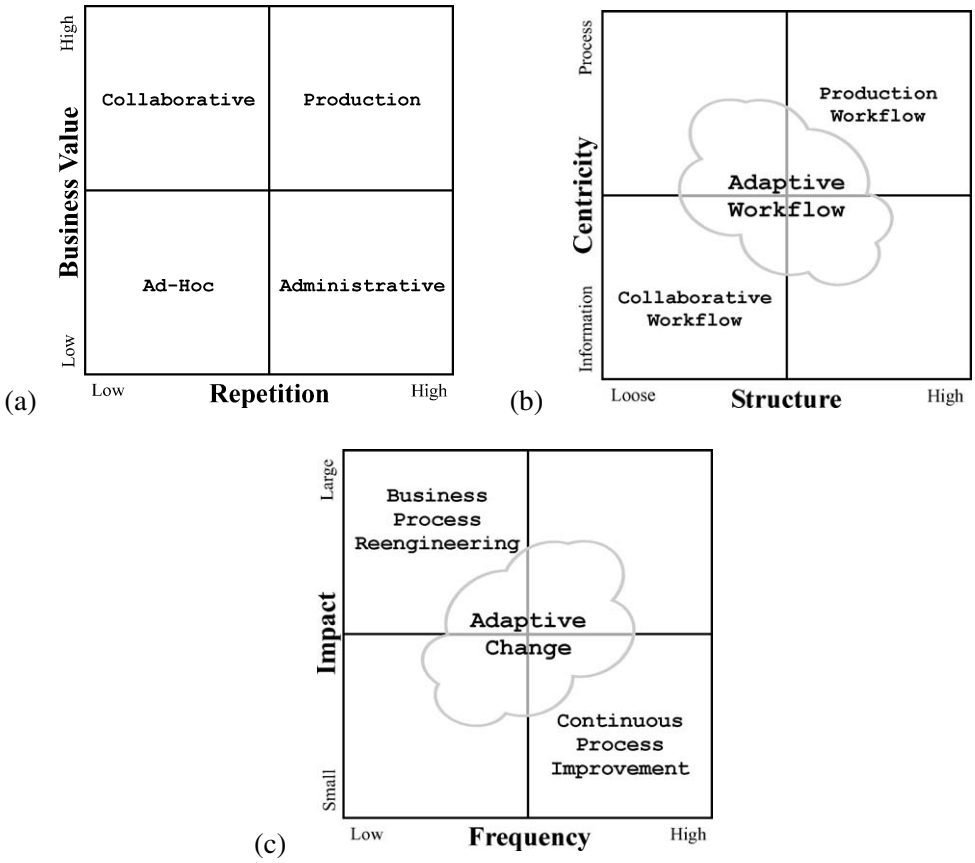


Figure 2. Workflow perspectives.

management, internationalization etc. Inevitably, as standards evolve to address these deficiencies, workflows will transition from the domain of intranets to that of the Internet. This transition will be accompanied by a new set of problems.

When an intranet-based workflow system executes, it does so with a collection of services that are owned and managed by the same organization. In this environment, service interruptions are infrequent and typically scheduled due to consolidated system management. In contrast, Internet-based workflows must be designed for resilient operation as service partners periodically become unavailable due to decentralized system management and the lack of network service guarantees. The evolution from intra- to inter-net based workflows will increase the design and run-time complexity, since the coordination mechanism must become more fault tolerant.

3.2. Workflow reference model

The Workflow Reference Model describes a generic architecture for workflow management systems [40]. The model, depicted in figure 3, shows the functional components

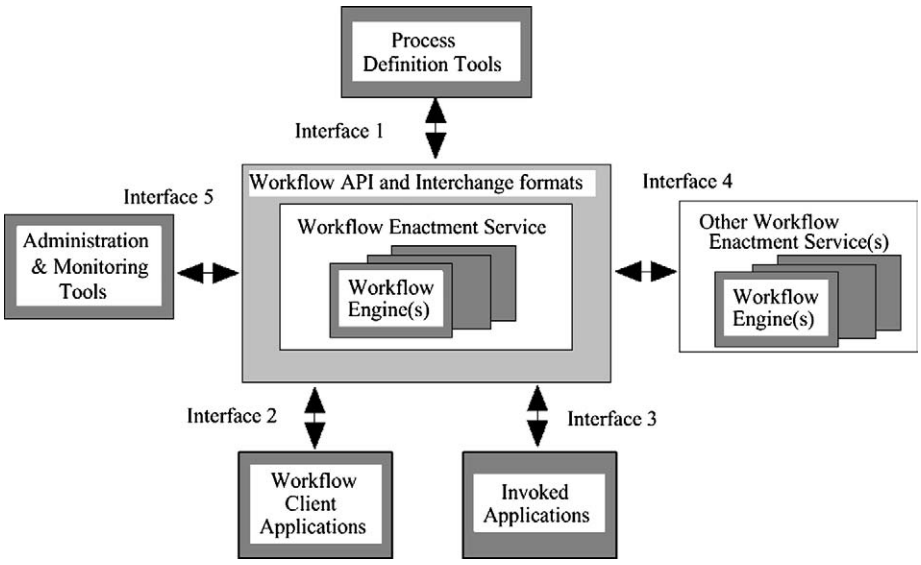


Figure 3. WfMC Reference Architecture © WfMC.

of a workflow system and identifies the major interfaces between them. Although the workflow reference model was created in 1995, it still provides a relevant architecture for discussing workflow management systems today.

As illustrated, a process definition of a business process is generated by Process Definition Tools. These tools typically allow a process designer to create a diagrammatic representation of business processes. The diagrams are then saved in a Process Description Language (PDL) that is received by the Workflow Enactment Service via interface 1. The enactment services utilize one or more local workflow engines that interpret and execute the PDL. While the workflow is being enacted, it can be administered and monitored via interface 5. The executing business process may interact with other automated business processes via interface 4. Interface 2 provides a mechanism for engaging a human participant in the workflow process, whereas interface 3 is designed for invoking applications without human intervention.

3.3. Workflow tools

As discussed in section 2.1, enterprise software architecture has evolved over time. Likewise, according to Singh and Huhns, the concept of workflow technology has developed in a generational manner; each generation leveraging the computational capabilities and management theories of their time. They identify four generations of workflow advancement and project a fifth generation based upon agent technology. The four generations they identify are: manual, closed, database-centric, and workflow tools oriented [32]. Contemporary approaches to business process automation are primarily focused on 4th generation approaches, that is, they are workflow tools oriented.

The majority of workflow tools target process definition. Business processes can be complex and difficult to comprehend. If a business process is incomprehensible, it cannot be effectively communicated, analyzed nor checked for correctness or completeness. UML activity diagrams can be used to represent workflows. These diagrams provide a high-level graphical description of the various task dependencies. The diagrams were designed to be human-readable. The activity diagram notation supports the expression of concurrent activity flows, the use of conditional branching, and the mapping of activities to specific actors. However, activity diagrams are insufficient for the purposes of enactment by an automatic system.

In the past few years, PDLs have been heavily influenced by XML and Web services. There are several ongoing initiatives that are defining XML-based PDLs to describe workflows composed of Web services. Although XML is human readable, its strength is in providing an unambiguous mechanism for the exchange of information. Fortunately, workflow design tools insulate the modeler from the complexity of the underlying PDL. The current generation of workflow design tools allows the process modeler to generate executable workflows directly from their visual representation. BPEL4WS, an XML-based language for expressing the composition of Web services, is poised to become the target PDL for the next generation of workflow design tools.

4. BPEL4WS

During the summer of 2002, IBM, Microsoft and BEA released a new PDL named BPEL4WS [50]. BPEL4WS represents the merger of two other PDLs, IBM's Web Services Flow Language (WSFL) and Microsoft's XLANG. BPEL4WS provides both graph-based and block-based control structures, making it capable of representing a wide range of control flows. Van der Aalst has compared the expressiveness of several PDLs, and has confirmed that BPEL4WS represents the union of WSFL and XLANG [41]. This merger has created the market consolidation necessary to make BPEL4WS the de facto standard for expressing workflows consisting of Web services.

BPEL4WS can be used to describe executable business processes and abstract processes. Abstract processes are used to create behavioral specifications consisting of the mutually visible messages exchanged between transacting parties executing a business protocol. BPEL4WS relies upon the following XML-based specifications: WSDL 1.1, XML Schema 1.0, XPath 1.0 and WS-Addressing.

Structurally, a BPEL4WS file describes a workflow by stating whom the participants are, what services they must implement in order to belong to the workflow, and the control flow of the workflow process. The BPEL4WS process model is built on top of the WSDL 1.1 service model and assumes all primitive actions are described as WSDL portTypes. That is, a BPEL4WS description describes the choreography of a set of messages all of which are described by their WSDL definitions. Importantly, WSDL is also used to describe the external interface to the workflow. This allows BPEL4WS to be compositionally complete, which means that the composition of Web services are exposed as a single Web service eligible to participate in other compositions [30].

	`<?xml version="1.0"?>`
⟨**definitions**⟩ specifies one or more services.	`<definitions name="CurrencyExchangeService"`
	`targetNamespace="http://www.xmethods.net/sd/CurrencyExchangeService.wsdl"`
⟨**types**⟩ section provides information about any complex data types used in the document. Note: the ⟨types⟩ section is not present in this example because only simple types are used.	`xmlns:tns="http://www.xmethods.net/sd/CurrencyExchangeService.wsdl"`
	`xmlns:xsd="http://www.w3.org/2001/XMLSchema"`
	`xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"`
	`xmlns="http://schemas.xmlsoap.org/wsdl/">`
⟨**message**⟩ is an abstract definition of the data being communicated. This Web service defines two messages: getRateRequest and getRateResponse.	`<message name="getRateRequest">`
	`<part name="country1" type="xsd:string"/>`
	`<part name="country2" type="xsd:string"/>`
	`</message>`
	`<message name="getRateResponse">`
	`<part name="Result" type="xsd:float"/>`
	`</message>`
⟨**portType**⟩ provides an abstract set of operations supported by the endpoints. ⟨**operation**⟩ describes the action provided by the service. This service has an operation named getQuote that takes a getRateRequest message and returns a getRateResponse message.	`<portType name="CurrencyExchangePortType">`
	`<operation name="getRate">`
	`<input message="tns:getRateRequest"/>`
	`<output message="tns:getRateResponse"/>`
	`</operation>`
	`</portType>`
⟨**binding**⟩ describes how the operation is invoked. It specifies the protocol and data format for the operation and messages.	`<binding name="CurrencyExchangeBinding" type="tns:CurrencyExchangePortType">`
	`<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>`
	`operation name="getRate">`
	`<soap:operation soapAction=""/>`
	`<input>`
	`<soap:body use="encoded" namespace="urn:xmethods-CurrencyExchange"`
	`encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>`
	`</input>`
	`<output>`
	`<soap:body use="encoded" namespace="urn:xmethods-CurrencyExchange"`
	`encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>`
	`</output>`
	`</operation>`
	`</binding>`
⟨**service**⟩ specifies the port address(es) of the binding.	`<service name="CurrencyExchangeService">`
	`<documentation>`
	`Returns the exchange rate between two currencies`
	`</documentation>`
⟨**port**⟩ defines a communication endpoint.	`<port name="CurrencyExchangePort" binding="tns:CurrencyExchangeBinding">`
	`<soap:address location="http://services.xmethods.net:80/soap"/>`
	`</port>`
	`</service>`
	`</definitions>`

Figure 4. A WSDL file for a currency exchange rate Web service.

Since BPEL4WS is highly dependent upon WSDL, a closer examination of the structure of a WSDL file is in order. Figure 4 presents the contents and an explanation of a WSDL file for a publicly available currency exchange rate Web service.

Figure 5 presents a skeletal view of the primary sections of a BPEL4WS file. The ⟨partners⟩ section declares the different parties that participate in the workflow process. Each partner is given a service link type and the role it will perform as part of the service link. The service link types are named entities that provide a mapping between role names and WSDL portTypes that the role must support. The ⟨variables⟩ section defines the variables used by the process. Variables store past messages and are needed to maintain the state of the workflow process as it executes. The ⟨faultHandlers⟩ section describes the fault handlers used by the workflow. All faults generated by the workflow must be given a name. The fault handlers define the activities that will be performed when a fault is raised. The process definition of the workflow occurs after the fault handlers section and before the close process tag.

A workflow process is defined in BPEL4WS using activity constructs. The primary constructs are:

– ⟨sequence⟩ specifies that its contents must be executed in the order presented;
– ⟨flow⟩ specifies that the contents are executed in parallel;

⟨**process**⟩ specifies a process.	`<process>`
⟨**partners**⟩ section declares the different parties that participate in the workflow. ⟨**partner**⟩ defines a workflow participant	`<partners>` `<partner/>` `</partners>`
⟨**variables**⟩ section defines the data variables used by the process. ⟨**variable**⟩ defines a named variable, associated with a WSDL message type.	`<variables>` `<variable/>` `</variables>`
⟨**faultHandler**⟩ section which defines the activities that should execute in response to a fault which occurs during the enactment of the process. ⟨**catch**⟩ handles a specific fault. ⟨**catchAll**⟩ default handler for faults that are not specifically caught.	`<faultHandlers>` `<catch/>` `<catchAll/>` `</faultHandlers>`
	`<!-- Workflow Definition Occurs Here -->`
	`</process>`

Figure 5. Primary BPEL4WS sections.

– ⟨while⟩ indicates that an activity must be repeated until the given criteria has been met;
– ⟨switch⟩ allows the conditional execution of one of many activities;
– ⟨pick⟩ blocks until a specified message arrives or a time-out occurs; when either one occurs its associated activity is executed;
– ⟨receive⟩ designates that a message is to be received;
– ⟨reply⟩ sends a message;
– ⟨invoke⟩ construct invokes an operation on a specified partner, portType pair;
– ⟨throw⟩ is used to raise a fault;
– ⟨wait⟩ pauses the execution for a specified time; and
– ⟨links⟩ are used to specify complex ordering constraints. Specifically, each link has a ⟨source⟩ and ⟨target⟩ construct. The target only execute after the source has finished.

Figure 6 contains a sample BPEL4WS workflow definition with the structuring activity tags in boldface. This service provides the ability to obtain a stock quote from the NYSE in any currency. For example, the service could be called to obtain a quote for shares of IBM expressed in Swiss Francs. An analysis of this workflow reveals that the service is composed from three other Web services. The three external Web services are the stockQuoteProvider, the currencyExchangeProvider, and the simpleFloatMultProvider. A narrative description of the steps in the workflow follows:

1. A service request message is received from the requestor.
2. The stock symbol is copied from the request message and the stockQuoteProvider is invoked to obtain the quote.

```
<sequence>
  <receive name="request partner="requestor"
           portType="tns:stockLookupProcessPT" operation="requestLookup"
           variable="reuest" createInstance="yes">
  </receive>
  <flow>
    <assign>
      <copy>
        <from variable="request" part="symbol"/>
        <to variable="stockQuoteProviderRequest" part="symbol"/>
      </copy>
    </assign>
    <invoke name="getStockQuote" partner="stockQuoteProvider"
       portType="nsl:StockQuotePortType" operation="getQuote"
       inputVariable="stockQuoteProviderRequest"
       outputVariable="stockQuoteProviderResponse">
    </invoke>
    <assign>
      <copy>
        <from expression="'usa'"/>
        <to variable="currencyExchangeProviderRequest" part="country1"/>
      </copy>
      <copy>
        <from variable="request" part="country"/>
        <to variable="currencyExchangeProviderRequest" part="country2"/>
      </copy>
    </assign>
    <invoke name="getCurrencyExchange" partner="currencyExchangeProvider"
       portType="ns2:CurrencyExchangePortType" operation="getRate"
       inputVariable="currencyExchangeProviderRequest"
       outputVariable="currency ExchangeProviderResponse">
    </invoke>
  </flow>
  <assign>
    <copy>
      <from variable="stockQuoteProviderResponse" part="Result"/>
      <to variable="simpleFloatMultProviderRequest" part="f1"/>
    </copy>
    <copy>
      <from variable="currencyExchangeProviderResponse" part="Result"/>
      <to variable="simpleFloatMultProviderRequest" part="f2"/>
    </copy>
  </assign>
  <invoke name="doSimpleFloatMult" partner="simpleFloatMultProvider"
       portType="ns3:SimpleFloatMult" operation="multiply"
       inputVariable="simpleFloatMultProviderRequest"
       outputVariable="simpleFloatMultProviderResponse">
  </invoke>
  <assign>
    <copy>
      <from variable="simpleFloatMultProviderresponse" part="multiplyReturn"/>
      <to variable="response" part="Result"/>
    </copy>
  </assign>
  <reply name="response" partner="requestor"
       portType="tns:stockLookupProcessPT" operation="requestLookup"
       variable="response">
  </reply>
</sequence>
```

Figure 6. An example BPEL4WS Workflow definition.

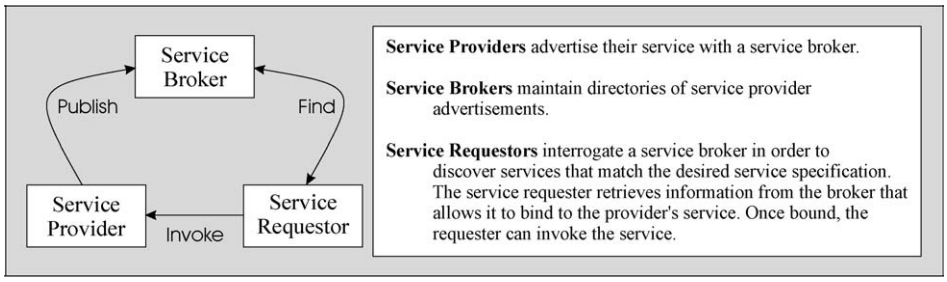


Figure 7. Service-oriented architecture model.

3. The country name is copied from the request message and the currencyExchange-Provider is invoked to obtain the rate of exchange between US dollars and the foreign currency
4. The returned quote and the exchange rate are sent to the simpleFloatMultProvider, which multiplies these values to convert the quote to the foreign currency.
5. The converted quote is copied into the reply message which is sent to the service requester.

Note that steps 2 and 3 have no data dependencies; therefore they can execute concurrently. This is specified in the BPEL4WS by placing steps 2 and 3 within a ⟨flow⟩ block.

5. DAML-S

As compelling a technology as BPEL4WS is, it does not fully address the challenge of composing workflow systems from Web services. BPEL4WS is focused on expressing the mechanics of the workflow, but not its semantics. Web services are compatible with Service-Oriented Architectures (SOA), which support run-time discovery and invocation. Ultimately, SOAs require semantic service descriptions in order to be broadly functional. Figure 7 illustrates the elements of a SOA.

The semantic web initiative is developing technologies for locating web resources based upon their semantic content. Included in this vision is DAML-S, a specification for providing semantic markup for Web Services. DAML-S is being designed to support the following Web Service related tasks: discovery, invocation, composition and interoperation, and execution monitoring [37]. DAML-S provides a machine-interpretable, ontology-backed semantic description of both atomic and composite Web services. As stated in [23], the markup provides:

– declarative advertisements for service properties and capabilities which can be used for automatic service discovery;
– declarative APIs for individual Web Services that are necessary for automatic Web Service execution; and
– declarative specifications of the prerequisites and consequences of individual service use that are necessary for automatic service composition and interoperation.

As discussed, DAML-S is being designed to ease composition issues by providing information that will allow Web Services to smartly interoperate. For a discussion of the relationship of DAML-S to other standards like UDDI, WSDL, and ebXML see [36].

A short example will provide justification of the need for a semantic markup language like DAML-S. Consider the following use-case scenario: "A London-based firm that wants to automate purchasing. Upon execution of an initial request for the purchase of 1,000 units of some item, the resulting query passes over the network to a Web service run by the supplier in Germany. That supplier in turn calls a Web service that calculates shipping cost, a second that computes tax, and a third that converts between pounds and euros. The Germany supplier then returns a consolidated quote to the original caller in London" [31]. Intuitively, an automated workflow engine could make use of the CurrencyExchangeService Web service, corresponding to the WSDL file presented in figure 4, to convert the price from euros to pounds. Or could it?

Without DAML-S or some other semantic annotation, the answer is no. Even if the service were located via a UDDI search, an automated workflow engine would not know what data to provide upon invocation. To the human reader, the names (country1 and country2) and the types (string) of the arguments in the getRateRequest message provides some contextual clues as to what these variables should contain; however, to an automated process this syntactic information is meaningless. However, if the service was described with a semantically rich notation, it might declare something like: CurrencyExchangeService is a Web service that accepts two ISO 3166-1-Alpha-2 country codes and returns an exchange rate whose value is a floating point number. Given the proper ontological structures, it would be possible to perform automated reasoning. From the structure of the requester's address, it would be likely to conclude that London is a city. Inference rules that establish that cities are found in countries could derive that London is found in a country named United Kingdom. Finally, countries have attributes and in our ontology, one of them is the ISO 3166-1-Alpha-2 country codes, which for the United Kingdom is GB.

Semantic markup of the service also provides information about what function the service performs. If the workflow enactment mechanism had knowledge about the role the CurrencyExchangeService played in the overall process, it would be possible to adapt the workflow in intelligent ways. If the German parts supplier in our example were supplying parts to another company in Germany, the CurrencyExchangeService obviously would not be required. A more sophisticated level of reasoning would need to occur if the German company was selling parts to a customer in France. In the ontology, countries also have an attribute of ISO 4217 Currency Type Symbol. This information could provide the adaptive workflow mechanism knowledge that both Germany and France use the same EUR currency type; therefore invoking the CurrencyExchangeService would be an unnecessary operation.

Given the demonstrated importance of semantic markup, it is worth considering the relationship between DAML-S and BPEL4WS. The DAML-S service model overlaps with BPEL4WS functionality; specifically, the Process Ontology contains the concept

of a Composite Process. In DAML-S, composite processes can be recursively decomposed into a set of Web services that are related to one another via control constructs. The DAML-S control constructs are block-structured and therefore lacks the ordering flexibility provided by BPEL4WS links. We suggest that BPEL4WS and DAML-S are compatible due to the fact that BPEL4WS is compositionally complete. BPEL4WS exposes a single WSDL interface for the composite process it contains and could therefore be marked-up in DAML-S as an atomic process. This results in the composite process itself, rather than its internal processing being described in DAML-S. This is the same mechanism used by DAML-S for handling DAML-S composite processes. A DAML-S composite process is transformed into a simple process via the *collapse* property. The simple process is mapped to an atomic process by the *realizedBy* property. In DAML-S, atomic processes are grounded to WSDL operations [35].

6. Agent-based workflow approaches

In our earlier work [2], we established a relationship between semantically described Web services and agents. Our vision is to use Semantic Web services as external behaviors for proactive agents. Huhns further distinguishes between Web services and agents. Some of the distinctions he provides are: Web services know only about themselves, they do not possess any meta-level awareness; Web services are not designed to utilize or understand ontologies; Web services are not capable of autonomous action, intentional communication, or deliberatively cooperative behavior [12]. In contrast, agents possess all of these capabilities.

As previously introduced, Singh and Huhns, anticipate that 5th generation workflow systems will employ agent-based technologies [32]. Others share this view, specifically [9,21,22]. To place this in perspective, an agent is a system that exhibits properties like: situatedness, autonomy, reactivity, pro-activeness, and social ability [48]. The social metaphor gives power to the agent-oriented paradigm; it is one of the characteristics that makes the agent abstraction particularly suitability for developing complex, distributed systems [16,17]. If a collection of sociable agents, representing individual services, cooperate and coordinate they would have the capability to enact any workflow that is composed of the represented services. In other words, agents have the capability to dynamically form social structures through which they share commitments to the common goal of workflow enactment. The individual agents, through their coordinated interactions achieve globally coherent behavior; they act as a collective entity known as a multiagent system.

Workflow enactment by a multiagent system can be viewed as an act of cooperative problem solving. “Cooperative problem solving occurs when a group of autonomous agents choose to work together to achieve a common goal” [49]. For cooperative problem solving to occur, an agent in the multiagent society must recognize that the best path to achieving a goal is to enlist the help of other agents. Social commitments arise when one agent makes a commitment to another. Typically a social commitment comes

about due to a social dependency. As defined in [13, p. 113] a social dependence can be defined as:

$$(\text{SocialDependence } x\ y\ a\ p) \equiv (\text{Goal } x\ p) \wedge \neg(\text{CanDo } x\ a) \wedge (\text{CanDo } y\ a) \wedge ((\text{DoneBy } y\ a) \Rightarrow \text{Eventually } p),$$

where meaning agent x depends on agent y with regard to act a for realizing state p, when p is a goal of x and x is unable to realize p while y is able to do so.

As indicated, for such a social dependency to be established, agent x and agent y must be able to reason about their ability to perform act a, and have knowledge that the performance of a will establish state p. The concept of first-order ability states that for agent x to have first-order ability regarding the establishment of state p, it must know explicitly whether $\exists a((\text{CanDo } x, a) \wedge ((\text{DoneBy } x\ a) \Rightarrow \text{Eventually } p))$ [49, p. 150]. If agent x desires to achieve state p, but knows $\neg(\text{FirstOrderAbility } x,\ p)$, then it must solicit assistance in order to attain the goal.

We believe that the advent of the semantic web and the emergence of a Web Services component model can facilitate agent-based workflow management in dynamic real-time environments. If agents use semantically described Web Services, then the semantic service descriptions become the basis for determining the agent's first-order abilities. Likewise, a common semantic markup for Web Services will facilitate effective communication between agents. Social agents that have access to an ontology-backed semantic description of their behaviors should be better able to proactively coordinate themselves at the macro-level.

6.1. BPEL4WS for multiagent systems

Unfortunately, despite the promise of the semantic web, its application in deployable, commercial applications is still distant. Given this fact, is the concept of using multiagent systems for workflow enactment still viable? The answer to this question is a resounding yes, for even without semantic information, agents can employ their autonomy to perform local workflow optimizations. Of course, a troubling issue remains. As discussed in the previous section, agents need semantic descriptions of their behaviors in order to reason about their own capabilities and the capabilities of the other agents in the system. This reasoning ability is crucial for cooperative problem solving to occur. If agents are unaware of their first-order ability, how is it possible for a multiagent system to organize itself?

We propose a novel approach, which is to use a BPEL4WS process description to impose an initial social order upon a collection of agents. Since BPEL4WS describes the relationships between the Web services in the workflow, agents representing the Web services would know their relationships a priori. Notably, the relationships between the Web services in the workflow are embedded in the process logic of the BPEL4WS file. A mechanism to extract this relational information is required if it is to be used to coordinate the interactions of the agents. Our strategy is to construct a Petri Net (PN) for the workflow, which is then partitioned based upon partner information. Agents

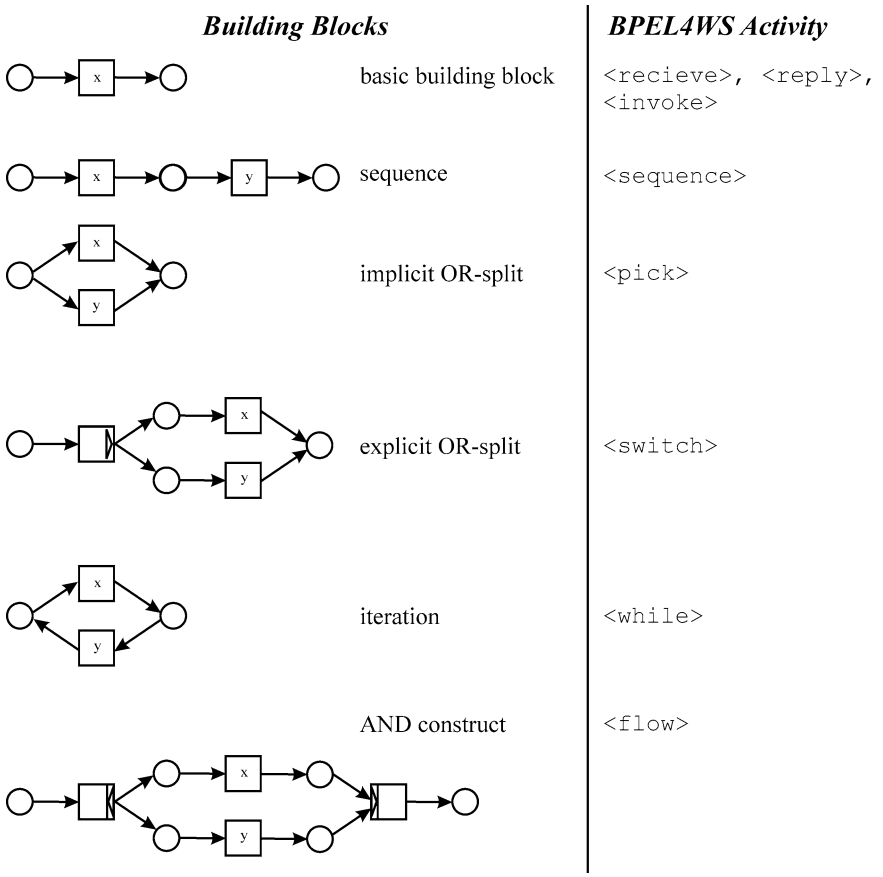


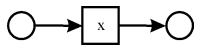
Figure 8. Building blocks mapped to BPEL4WS activities.

within a multiagent system represent each partner and enact the workflow in a distributed manner.

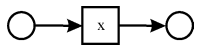
PNs are recognized as a useful workflow modeling tool. The reader is referred to [26] for a collection of PN related material. Their unambiguous and precise semantics allow a workflow model to be analyzed. Such analysis can prove many properties about a workflow process, including the absence of livelock and deadlock conditions. For our purposes, an assumption is made that the BPEL4WS process description represents a well-formed workflow process. Van der Aalst presents a methodology to generate PNs via the repetitive replacement of elemental PNs with other PNs [42]. In order to leverage this replacement property, a collection of elemental building blocks is required. Figure 8 based in part on [42, figure 4.11], illustrates a collection of PN building blocks along with a mapping to the appropriate BPEL4WS activity.

Although the mapping from building blocks to BPEL4WS activities is incomplete, it is sufficient to represent many workflows. It is our opinion that after the basic workflow process is converted to PN form, it can be augmented with the ⟨link⟩, ⟨terminate⟩, ⟨wait⟩

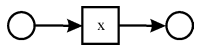
etc. activities. To demonstrate the application of the replacement property, the workflow description found in figure 6, will be used to illustrate the conversion of BPEL4WS to PN form. To begin, represent each BPEL4WS basic activity as a building block.

A. 

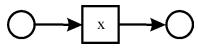
where x is the ⟨`receive`⟩ activity named request.

B. 

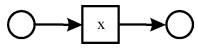
where x is the ⟨`invoke`⟩ activity named getStock Quote.

C. 

where x is the ⟨`invoke`⟩ activity named getCurrencyExchange.

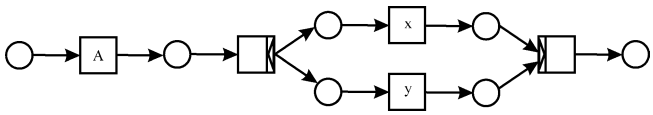
D. 

where x is the ⟨`invoke`⟩ activity named doSimpleFloatMult.

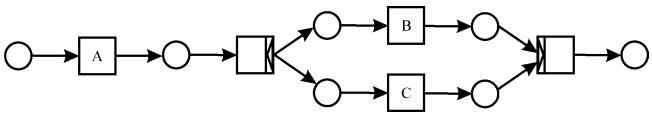
E. 

where x is the ⟨`reply`⟩ activity named response.

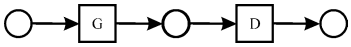
Utilizing the mapping presented in figure 8, repeatedly apply the replacement property, consuming the BPEL4WS process description. Starting with a sequence building block, replace x with A and replace y with a flow construct, yielding F.

F. 

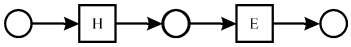
Starting with F, replace x with B and replace y with net C, yielding G.

G. 

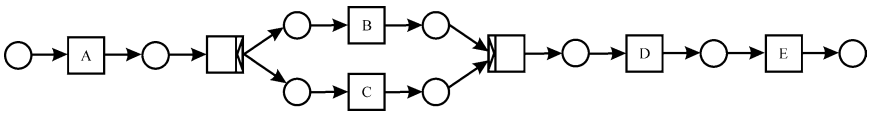
In a sequence building block, replace x with G and y with D, yielding H.

H. 

In a sequence building block, replace x with H and y with E, resulting in I.

I. 

The final PN in expanded form:



After the BPEL4WS file has been converted to PN form, it can be decomposed to establish the relationships between the Web services. This is accomplished by traversing the PN in the forward direction, noting the input and output tokens consumed and

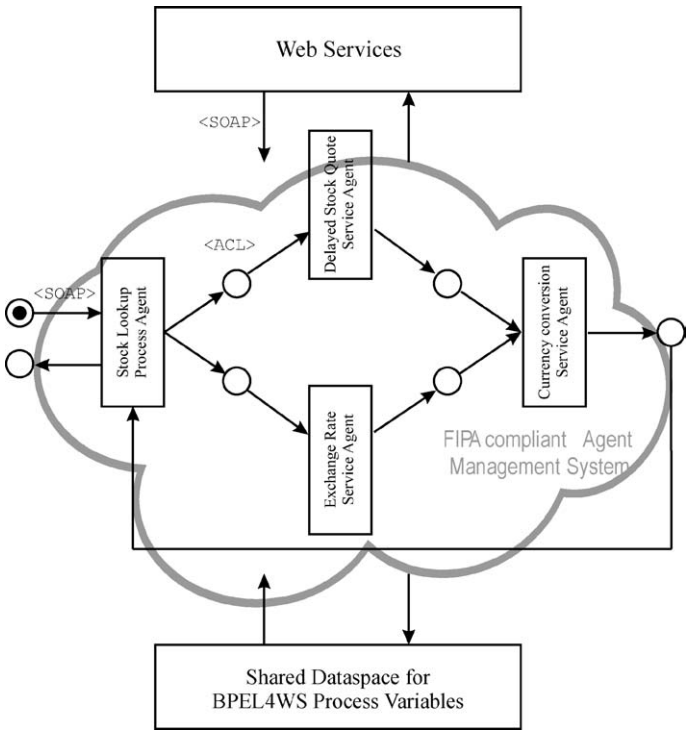


Figure 9. Architecture of the multiagent enactment mechanism for the example workflow.

sent by each partner. This information, along with variable manipulation instructions is sufficient to allow the agents to coordinate the enactment of the workflow.

The architecture for the multiagent enactment of the example workflow is illustrated in figure 9. This architecture relies upon the work of The Foundation for Intelligent Physical Agents (FIPA), which can be thought of as a component model that enables agents from heterogeneous origins to collaborate in open environments [38]. In this architecture, the following communications pathways exist:

- agent to agent communication occurs via FIPA's Agent Communication Language (ACL) and is facilitate by a FIPA compliant Agent Management System (AMS);
- agent to Web service communication is accomplished via SOAP messages;
- agent to shared dataspace communication utilizes appropriate protocols/interfaces provided by the dataspace. The dataspace is used to store BPEL4WS process variables, which maintain the state of the workflow process.

A few example scenarios will demonstrate how the agents can adapt the workflow based upon knowledge acquired from: cached interaction histories with their respective Web services, semantic markup, or coded by the application developer.

1. At time zero, a request for a quote of IBM stock in Euro currency is received. No interaction history is present and the entire workflow executes.

2. A request for IBM in US dollars occurs two minutes later. Since the conversion is between like currencies, the Exchange Rate Service Agent doesn't invoke the exchange rate Web service. Likewise, since the delayed stock quote Web service has a QOS guarantee of 15 minutes, the Delayed Stock Quote Service Agent determines that the cached quote for IBM is valid; therefore the underlying Web service is not invoked.
3. Five minutes later a request containing IBM and Swiss Francs arrives. This invocation of the workflow utilizes the cached quote for IBM, but requires that the currency exchange rate be computed.

6.2. Multiagent workflow enactment as an autonomic system

We believe that IBM's Autonomic Computing initiative provides an interesting vantage point from which to consider adaptive workflow. As noted in IBM's Autonomic Computing Manifesto [14], complexity itself is a byproduct of automation; workflow management systems by their very definition are the automation of a business process. One of the tenants of the autonomic computing initiative is to remove the complexity from the end-user and embed it in the infrastructure of the system. Sophisticated self-governing processes then manage the infrastructure. These processes possess several key characteristics; among them are: self-configuration, self-optimization, self-healing, and self-preservation. Each of these characteristics speaks to the need for adaptation that is designed to achieve specific goals.

Using multiagent systems for workflow enactment is only the first step that enables the exploration of many other fundamental questions. As noted in [18] autonomic systems will consist of autonomic elements that will have policy driven relationships with one another. If the BPEL4WS workflow description were interpreted as a strict policy statement, then a static enactment mechanism is appropriate; however, if interpreted as a policy guideline, multiagent enactment mechanisms provide a degree of process agility.

A sampling of some of the questions to be explored:

– How might the concept of adjustable autonomy be used to enable multiagent enactment across the spectrum of workflow types, from collaboration to production? In production workflows, multiagent implementation may provide execution-monitoring advantages; even without the agents possessing a high-degree's of autonomy. On the other end of the spectrum, agents that monitor the interaction of the participants in a CSCW scenario could potentially discover interaction patterns, formalize process rules and utilize their autonomy to enact elements of the ad-hoc workflow without manual intervention (self-configuring).
– How might agents leverage workflow design tools that can capture the business logic and rationale for service selection and flow? This meta-process information could latter be utilized by the autonomous agents for process redesign (self-optimization). Having a design specification for the MAS provides self-knowledge, which could be leveraged for self-optimization. For example, agents can use the workflow description to determine the impact of hypothetical changes, or use it, along with knowledge of available resources, to find other resources that can be exploited.

– How might BPEL4WS be extended to allow the specification of multiple, functionally equivalent partners at each end of the service link? In a supply chain management scenario, the agents could use this information to tailor the workflow to deliver different QOS levels based upon cost, time or quality constraints (self-configuring, self-optimizing). Likewise, the list of partners might represent primary and secondary service providers; in the event of primary partner failure, the workflow could automatically engage the secondary partner (self-healing).
– How might an agent's active monitoring of service invocation patterns be useful for the purposes of detecting/correcting inappropriate service access? (self-protection) Perhaps, agents could use a BPEL4WS process description to identify normal behavior and signal everything else as abnormal. Abnormal behaviors would have to be further analyzed to determine if they are a real threat or a legitimate deviation enacted by the agents in an effort to optimize the system's behavior.
– How might the abstract process notion be useful as a specification that can be instantiated by agents? (self-configuring) An abstract process definition is non-deterministic and does not specify under what conditions each branch is chosen. As such, it can be used by agents to determine the set of "legal" actions and leaves the choice to the agent's reasoning. One can envision the use of abstract specifications (if made very flexible) as very high-level system behavioral limits. The agents would then be free to implement any specific system behavior that falls within this space.

7. Related developments

Adaptive workflow capabilities, achieved through multiagent enactment mechanisms, will be influenced by developments related to: BPM software and PDL developments, Web services, the semantic web, and Agent-Oriented Software Engineering (AOSE). The pace of change in each of these areas is quickening as commercial entities strive to capture early market share and consortia like WfMC, BPMI, and W3C struggle to maintain their relevance. In the BPM solution space, this scramble is being driven by market analysis that predicts the BPM market will be worth $6.32 billion in 2005, up from $2.26 billion in 2001 [3]. Interestingly, evidence that establishes the need for self-configuring, self-optimizing BPM systems is found in this same research report, which shows that for every dollar spent on BPM software in 2001, three dollars were spent on related professional integration services.

In the domain of PDL development, we feel that BPEL4WS will become the de facto standard as soon as Microsoft and IBM retool their product offerings for release in 2003. IBM has released BPWS4J on their Alphaworks site [15]. BPWS4J provides a preview of the capabilities that will be available in future versions of WebSphere Studio Application Developer Integration Edition and the WebSphere Application Server. BPWS4J consists of an Eclipse based graphical editor for designing workflows expressed in BPEL4WS PDL. The BPEL4WS workflow descriptions can then be executed on the BPEL4WS workflow engine. Both the WfMC and BPMI have release statements

indicating that their own process description languages, XPDL and BPML respectively, are more capable than the BPEL4WS specification; however, they embrace BPEL4WS as a positive development for the BPM industry [1,46].

Although not at the same frenetic pace, developments are also occurring in the space of Web services, the semantic web and AOSE. Regarding Web services, the WSDL and SOAP specifications are completing an update cycle. The semantic web is transitioning ontology languages from DAML + OIL to the new Web Ontology Language (OWL). The field of AOSE is beginning to pay close attention to the Web service developments. FIPA has formed a technical committee to propose an integration strategy for FIPA compliant agents to interoperate with Web services.

On the academic front, several researchers are working at the intersection of agents and workflow. Specifically, [9,22,32] have written about the potential benefits of introducing agent technology into workflow enactment mechanisms. In [22, p. 575], Marinescu discusses the use of the Bond agent architecture to enact a workflow description captured in XPDL. Most closely related to our vision of using contemporary BPM tools and Web services for multiagent systems design is the work described in [19]. In this paper, Korhonen et al. describe the creation of a workflow ontology that is used to describe both agents and Web services. They hope to build a workflow enactment mechanism that can utilize the ontology to bridge the communications gap between agents and Web services.

8. Conclusion and future work

In this paper, our goal has been to contextualize thoughts of multiagent systems as a workflow enactment mechanism. We predict that the landscape of enterprise integration will undergo dramatic changes in the next 3–7 years as Web services usher in a new era and BPM applications replace traditional EAI efforts. In these turbulent times historic perspective is necessary for setting appropriate expectation levels. For the reasons presented in sections 2 and 6, we firmly believe that the historic trajectory of software development paradigms and IT advancements will establish multiagent systems as the workflow enactment mechanism of the future. The business community, while being inundated with Web service hype and PDL confusion need to remain vigilant for the emergence of disruptive technologies in the BPM application area. Agent-oriented software is maturing at the same time as the semantic web activity. The combination of these two technologies may truly establish the Internet connected virtual enterprises of the future. In the meantime, agent researchers need to seriously investigate the use of workflow design tools and PDLs for producing multiagent system specifications.

We have much work to do to demonstrate an adaptive, multiagent-based workflow engine. Our current activities are focused on two fronts: the design and implementation of a generic agent/Web service interface, and a complete mapping of BPEL4WS constructs onto a multiagent system decomposition. We anticipate having both of these tasks complete in the first quarter of 2004. Following these activities, the inclusion of se-

mantically described Web services into the workflows will enable experimentation with autonomic system concepts since robust agent reasoning about service capabilities will be possible.

Acknowledgements

This work is supported by the U.S. National Science Foundation under grant no. IIS-0092593 (CAREER award).

References

[1] BPMI.org, BPML|BPEL4WS: A convergence path toward a standard BPM stack (August 15, 2002).
[2] P. Buhler and J.M. Vidal, Semantic Web services as agent behaviors, in: *Agentcities: Challenges in Open Agent Environments*, eds. B. Burg, J. Dale, T. Finin, H. Nakashima, L. Padgham, C. Sierra and S. Willmott (Springer, Berlin, 2003) pp. 25–31.
[3] S. Cowley, BPM market primed for growth *InfoWorld* (September 23, 2002).
[4] F. DeRemer and H. Kron, Programming in the large versus programming in the small, IEEE Transactions on Software Engineering 2(2) (1976) 80–87.
[5] eAI Journal, Business process logic: Half-empty or half-full? http://www.eaijournal.com/Article.asp?ArticleID=629&DepartmentID=7.
[6] D. Garland, Software architecture: A roadmap, in: *The Future of Software Engineering* (ACM Press, New York, NY, 2000) pp. 91–101.
[7] D. Gelertner and N. Carriero, Coordination languages and their significance, Communications of the ACM 35(2) (1992) 97–107.
[8] G. Glass, *Web Services, Building Blocks for Distributed Systems* (Prentice Hall PTR, Upper Saddle River, NJ, 2002).
[9] M. Griss, Software agents as next generation software components, in: *Component-Based Software Engineering: Putting the Pieces together*, eds. G.T. Heineman and W.T. Councill (Addison-Wesley, Boston, 2001) pp. 641–657.
[10] G.T. Heineman and W.T. Councill, Definition of a software component and its elements, in: *Component-Based Software Engineering: Putting the Pieces together*, eds. G.T. Heineman and W.T. Councill (Addison-Wesley, Boston, 2001) pp. 5–19.
[11] P. Herzum and O. Sims, *Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise* (Wiley, New York, 2000).
[12] M.N. Huhns, Agents as Web services, IEEE Internet Computing 6(4) (2002) 93–95.
[13] M.N. Huhns and L.M. Stephens, Multiagent systems and societies of agents, in: *Multiagent Systems: A Modern Approach to Distributed Artifical Intelligence*, ed. G. Weiss (MIT Press, Cambridge, MA, 1999) pp. 79–120.
[14] IBM, Autonomic computing: IBM's perspective on the state of information technology, http://www.research.ibm.com/autonomic/manifesto/.
[15] IBM, BPWS4J, http://www.alphaworks.ibm.com/tech/bpws4j.
[16] N.R. Jennings, An agent-based approach for building complex software systems, Communications of the ACM 44(4) (2001) 35–41.
[17] N.R. Jennings, On agent-based software engineering, Artifical Intelligence 177 (2000) 277–296.
[18] J.O. Kephart and D.M. Chess, The vision of autonomic computing, IEEE Computer 36(1) (2003) 41–50.
[19] J. Korhonen, L. Pajunen and J. Puustijarvi, Using Web services and workflow ontology in multi-agent systems, in: *Workshop on Ontologies for Multi-Agent Systems* (2002).

[20] F. Leymann and D. Roller, *Production Workflow: Concepts and Techniques* (Prentice Hall PTR, Upper Saddle River, NJ, 2000).
[21] Z. Maamar and J. Sutherland, Toward intelligent business objects, Communications of the ACM 43(10) (2000) 99–101.
[22] D.C. Marinescu, *Internet-Based Workflow Management: Toward a Semantic Web* (Wiley-Interscience, New York, 2002).
[23] S.A. McIlraith, T.C. Son and H. Zeng, Mobilizing the semantic Web with DAML-enabled Web services, in: *Semantic Web Workshop* (2001).
[24] B.C. Meyers and P. Oberndorf, *Managing Software Acquisition: Open Systems and COTS Products* (Addison-Wesley, Boston, 2001).
[25] H.V.D. Paranak, Go to the ant: Engineering principles from natural multi-agent systems, Annals of Operations Research (1997).
[26] Petri Nets World, http://www.daimi.au.dk/PetriNets/.
[27] S.L. Pfleeger, *Software Engineering: Theory and Practice* (Prentice Hall, Upper Saddle River, NJ, 2001).
[28] J. Sametinger, *Software Engineering with Reusable Components* (Springer, New York, 1997).
[29] M. Sawhney and J. Zabin, *The Seven Steps to Nirvana: Strategic Insights into eBusiness Transformation* (McGraw-Hill, New York, 2001).
[30] J.-G. Schneider, M. Lumpe and O. Nierstrasz, Agent coordination via scripting languages, in: *Coordination of Internet Agents: Models, Technologies, and Applications*, eds. A. Omicini, F. Zambonelli, M. Klusch and R. Tolksdorf (Springer, New York, NY, 2001) pp. 153–175.
[31] C. Shirky, Web services and context horizons, IEEE Computer 35(9) (2002) 98–100.
[32] M.P. Singh and M.N. Huhns, Multiagent systems for workflow, International Journal of Intelligent Systems in Accounting, Finance and Management 8 (1999) 105–117.
[33] Sun Microsystems, Inc. Java 2 Platform, Enterprise Edition, http://java.sun.com/j2ee/.
[34] C. Szyperski, *Component Software: Beyond Object-Oriented Programming* (Addison-Wesley, New York, 2002).
[35] The DAML Services Coalition, DAML-S: Web service description for the semantic Web, in: *The 1st International Semantic Web Conference (ISWC)* (2002).
[36] The DAML Services Coalition, DAML-S and related technologies, http://www.daml.org/services/daml-s/0.7/survey.pdf.
[37] The DAML Services Coalition, DAML-S: Semantic markup for Web services, http://www.daml.org/services/daml-s/0.7/daml-s.pdf.
[38] The Foundation for Intelligent Physical Agents, www.fipa.org.
[39] The Workflow Management Coalition, Terminology & Glossary, Document No. WFMC-TC-1011, http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf.
[40] The Workflow Management Coalition, The Workflow Reference Model, Document No. TC00-1003, http://www.wfmc.org/standards/docs/tc003v11.pdf.
[41] W. van der Aalst, Don't go with the flow: Web services composition standards exposed, IEEE Intelligent Systems 18(1) (2003).
[42] W. van der Aalst and K.M. van Hee, *Workflow Management: Models, Methods, and Systems* (MIT Press, Cambridge, MA, 2002).
[43] WebServices.Org, The 'big boys' unite forces – What does it mean for you? http://www.webservices.org/index.php/article/articleview/633/1/24/.
[44] P. Wegner, Interoperability, ACM Computing Surveys 28(1) (1996) 285–287.
[45] R. Weinreich and J. Sametinger, Component models and component services: Concepts and principles, in: *Component-Based Software Engineering: Putting the Pieces together*, eds. G.T. Heineman and W.T. Councill (Addison-Wesley, New York, 2001) pp. 33–48.
[46] WfMC, Press release (September 12, 2002).

[47] O.E. Williamson, S.G. Winter and R.H. Coase, *The Nature of the Firm: Origins, Evolution, and Development* (Oxford University Press, New York, 1991).
[48] M. Wooldridge, Agents and software engineering, AI*IA Notizie XI(3) (1998) 31–37.
[49] M.J. Wooldridge, *Reasoning about Rational Agents* (MIT Press, Cambridge, MA, 2000).
[50] XML Cover Pages, Business Process Execution Language for Web Services (BPEL4WS), http://xml.coverpages.org/bpel4ws.html.