Christos Voudouris · Gilbert Owusu
Raphael Dorne · David Lesaint

Service Chain Management

Technology Innovation for the Service Business

Springer

Christos Voudouris
Gilbert Owusu
Raphael Dorne
David Lesaint

British Telecom
Adastral Park, Martlesham Heath
Orion Building pp 1/12
Ipswich, IP53RE
UK

chris.voudouris@bt.com
gilbert.owusu@bt.com
raphael.dorne@bt.com
david.lesaint@bt.com

ISBN-13 978-3-540-75503-6 e-ISBN-13 978-3-540-75504-3

DOI 10.1007/978-3-540-75504-3

ACM Computing Classification (1998): K.6, H.4, J.1, J.7

Library of Congress Control Number: 2007936681

Cover design: KünkelLopka, Heidelberg
Typesetting and Production: LE-TEX Jelonek, Schmidt & Vöckler GbR, Leipzig

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

Chapter 12
Flexible Workflows

B. Jennings[1], A. Finkelstein[1]

[1]University College London, Department of Computer Science

12.1 Introduction

The concept of effective workflows is deeply ingrained in virtually every industry today. As businesses become more distributed, there is a need for massively networked workflow tools which is reflected in the emergence of the Service-Oriented Architecture paradigm. This large collection of standards brings into workflow the concept of linking together disparate Web Services to form composite applications with some integration of human processes. Most implementations of workflow tools, however, require complete adoption at multiple levels of an organisation as well as agreement in standards to enable inter-service communication in heterogeneous environments. This necessitates significant investment in software management solutions, development time and hardware requirements, all of which can lead to inflexible workflow solutions.

This chapter presents an overview of the development of workflow support systems for the last twenty years from the concept of task granularity to Web Service based workflow design and deployment. The review focuses on flexibility and dynamism in workflows in response to change and exceptions. Key elements are identified in the construction of workflows and different approaches to the design of workflow tools are introduced. The need to abstract workflows and separate design from implementation is then looked at both from a modelling and a language perspective. The chapter concludes with recommendations on potential innovation in the workflow/Web Service product space and suggests a lightweight Representational State Transfer (REST) approach. This loosely coupled approach encourages iterative development, less up front design and hence, more flexible workflows.

The chapter is structured as follows. Section 12.2 reviews the key issues in the construction of workflows and the design and deployment of support tools. Section 12.3 surveys previous work done in the field. Section 12.4 looks at contemporary approaches and introduces methods to handle deviations based on a REST approach. Section 12.5 concludes.

12.2 What is Workflow and Workflow Support?

A workflow is a formal, or implementation-specific, representation of a business process. Harrington defines a business process as "... any activity or group of activities that takes an input, adds value to it, and provides an output to an internal or external customer. Processes use an organization's resources to provide definitive results." (Harrington 1991). Organisations typically comprise multiple layers of tasks and services that are co-ordinated to achieve business objectives. Taking an holistic view of organisations is therefore indispensable to understand the notions of workflow and workflow support tools.

12.2.1 Leavitt's Diamond

Leavitt's Diamond provides a good starting point to analyse business processes (Leavitt 1965). As shown in Fig. 12.1, Leavitt's Diamond separates and relates four different aspects of a process, namely, Structure, Technology, People and Task. These aspects are viewed as being integral to a process and of equal importance.

What defines a task, or activity, in a workflow depends upon the granularity at which one chooses to analyse a business process. Once the tasks are identified, an optimal task ordering must be constructed to structure the workflow using algorithms such as the Critical Path Method (DuPont 1950). The tools or technology required to support workflows is another significant component of the analysis. The analysis must also take into account the human factor by considering the people involved in the process and how they will interact with the system. Finally, it must take into account the structure or organisation which represents the set of agents involved in the process.

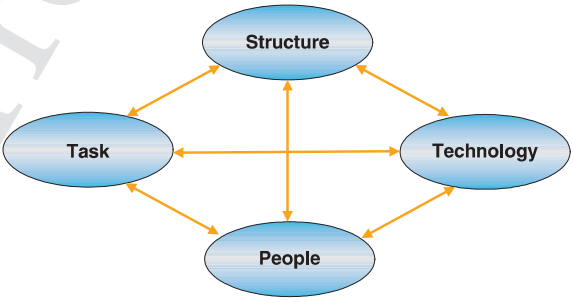

Fig. 12.1 Leavitt's Diamond

12.2.2 A Socio-Cognitive Perspective

Socio-cognitive aspects must be taken into consideration when engineering or deploying workflow technology. A workflow tool will, in some way, replace and in-

teract with employees and customers. The task analysis will be culturally specific (Michelis and Grasso 1994) and, if too restrictive to the individuals interacting with the system, will lead to loss of motivation or abandonment of the system (Hemingway 1998). As quoted from Adams et al. (2003), "Technologies are derived from perceived needs and realised, not in isolation, but through the conventions and norms of their social milieu".

12.2.3 Agents in a Workflow

Today, more so than twenty years ago, a greater number of agents acting in workflows are computer programs or Web Services. Franklin and Glesser define an autonomous agent as a "system situated within and a part of an environment that senses that environment and acts on it" (Franklin and Graesser 1996). This definition does not take into account Web Services as they did not exist at the time of writing. To arrive at a definition of what differentiates a regular computer program from a Web Service is beyond the scope of this chapter.

12.2.4 Control Flow

Another significant aspect in workflow relates to control flow and how computer systems can support collaborative activities and their coordination. Tools that have the ability to monitor and synchronise tasks, enable parallelism and serialism, and handle dependencies and deviation (i. e., exception handling) can generate significant business benefits. For instance, parallelism increases process throughput and reduces latency; handling deviation by providing flexible recovery scenarios minimises business disruptions when tasks end unexpectedly; etc.

12.2.5 Why Are Workflow Support Tools Useful?

As discussed above, workflow support tools have the potential to improve process throughput and reduce overheads in the enterprise, leading towards higher quality of service and lower costs. Workflow tools can also suggest new ways of working to an enterprise by giving the flexibility to model, analyse, and restructure processes. For instance, analysis techniques based on formal models can reveal flaws in workflows such as deadlocks, livelocks, race conditions, unreachable states, and so on. Workflow tools with the ability to handle deviations can make processes more robust and improve the experience of individuals operating within a workflow. Finally, tools providing a suitable level of abstraction can be applied to a variety of business processes, thus allowing enterprises to recoup their investment in workflow technology.

12.3 Previous Approaches

12.3.1 Workflow Management Systems

Workflow tools truly emerged in the eighties and coincided with a move toward a paperless office. The emphasis was on document/data processing and the principle was to capture all the tasks in a process in order to create an optimal workflow. Two ~~notorious~~ examples were the Poise/Polymer system and the Versatile Avionics Shop Test (VAST). Poise and VAST shared a linear view of workflows and imposed a predetermined series of actions on human operators, effectively reducing people to elements in a machine. This was a source of frustration which was compounded by inflexible interfaces and a lack of support for handling deviations.

12.3.2 Running Workflows

The nineties saw the emergence of tools designed to facilitate the running of workflows. A shift from a data-oriented to an agent communication model was deemed necessary from a socio-cognitive and workflow orchestration perspective. The Milan Conversation Model, for instance, advocated the literal capture and transmission of communication between users of a workflow system based on the principle that "any group of persons sharing a social experience, shares a language, a physical space, an agenda, … without which no interpretation is possible" (Michelis and Grasso 1994). The mapping of face-to-face or e-mail communications was used to track a sequence in a workflow. The problem with this model was that it led to predetermined workflows. Other workflow support tools, such as InConcert (McCarthy and Sarin 1993), focused on data flow and adopted a goal orientation methodology providing the ability to link together diverse activities in order to accomplish a task.

12.3.3 Transactional Model

Up to this point, none of the workflow tools had a robust methodology for recovering from failures. The only available strategy was to express all possible points of failure in the workflow itself. While effective, this strategy can produce overly verbose and complex workflow descriptions. ~~Inspired by the success of~~ the transaction model in database management systems, efforts were geared towards the integration of transactional features into workflow tools (Eder and Liebhart 1996). The principle is to use a transaction run-time model to execute compensatory tasks and roll back a process when a task fails. A transactional model can also help to recover from a failure of the workflow tool itself which is essential when dealing with long-running business processes.

In a transactional model, the act of rolling back via compensatory tasks only applies to the tasks themselves. A workflow tool should not be responsible for an agent's internal roll back since this would break the principle of separation of concerns (Dijkstra 1982). In addition, the transaction model must satisfy the ACID properties, i. e., atomicity, consistency, isolation, and durability. Technically, there are two genres of recovery actions, rolling back (undo) and rolling forward (redo). In a rollback recovery, a significant recovery attempt must be made in the event of a failure when a task has dependencies within a structure. All tasks that are part of a structure and have already committed must have a compensatory action to return the process to a consistent state. In a rollforward recovery, if a task within a hierarchy has no dependencies, the remainder of the process can be executed without leaving the process in an inconsistent state. The Workflow Activity Model takes this into account (Eder and Liebhart 1995).

12.3.4 Workflow Modelling

The next evolution in workflow technology was to develop methods for modelling workflow states. Both graphical models based on bipartite graphs (Petri 1962) and algebraic models based on linear textual descriptions (Milner et al. 1992) were developed. The most significant form of graphical modelling came from an adaptation of Petri nets, known as high-level Petri nets (Riemann 1999) to model data flow, time, and task hierarchies within processes. A significant advantage of graphical models over linear models is that they can be created by non-engineers. Another advantage is that the graphical approach separates the modelling of a workflow from its implementation thus making models portable and free from vendor limitations.

12.3.5 Summary

The progression of work up to this point has been freeing elements of the business process into discrete sections. This process is continuing and set to move the abilities of workflow tools further. "The nineties will be marked by the emergence of workflow software, allowing application developers to push the business procedures out of the applications." (van der Aalst 1998). However, introducing a workflow tool should be more than implementing an electronic version of an already flawed system. If the computerisation of a workflow is merely to recreate an existing system and no time is spent on re-engineering the business process itself, one can be left with "automating a fiction" (Sheil 1983).

12.4 Contemporary Approaches

Advances in computing, network and device technologies have led towards a massively distributed style of working and new forms of interactions and collaborations. We present below the new architectural styles and approaches to workflow tools that have emerged in this context.

12.4.1 Why Enterprises Need Service-Oriented Architectures?

In traditional software development, monolithic applications are constructed to suit enterprise business needs through a slow process of requirements elicitation, system design, development, testing and deployment. This methodology leads to tightly coupled systems which require significant resources to alter when business requirements evolve. A newer approach is that of Service Oriented Architectures (SOA). In this architectural style, smaller software components are created and linked together as networked services, or Web Services. In order to facilitate communication, Web Services have well-defined interfaces known as application programming interfaces (APIs) in this context. This approach makes it possible to connect disparate systems created using different tools and languages, thus loosening the coupling between systems and enabling an agile software delivery process.

12.4.2 Workflow Specification Languages

As has been seen from earlier modelling work (e. g., Petri nets), using abstract workflow specification languages is essential to achieve platform- and vendor-independence. We discuss below the design of workflow specification languages in the context of SOA and Web Services and introduce two instances, the Business Process Execution Language (BPEL) and Yet Another Workflow Language (YAWL).

12.4.2.1 Workflow Patterns

Patterns have been originally popularised in the object-oriented community by the Gang of Four book (Gamma et al. 1995). This approach seeks to factorise and express common problems and related solutions in a language-independent way as design patterns that can be reused and applied in different problem domains. Similarly to this research, Russel et al. have formalised and classified workflow patterns into five categories: Control Flow, Structural, State-Based, Advanced Branching and Synchronisation, Cancellation and Multiple Instances (Russell et al. 2003). How best to support these patterns is one of the key challenges in the design of workflow specification languages.

12.4.2.2 BPEL and YAWL

BPEL is the most popular workflow specification language in the enterprise market. It is a notation based on the Extensible Markup Language (XML) for defining business processes via Web Services (IBM et al. 2002). It features a process element that contains other elements, one of which is the concept of a *partnerLink*. This partnerLink allows the definition of other services and roles which allows messages to be passed between services. BPEL also has the concept of exception handling and falls back from exceptions defined in the XML.

One of the key advantages that BPEL offers is interoperability. BPEL is XML-based and therefore agnostic to the platform and execution engine upon which it runs. In relation to BPEL and an Enterprise Service Bus (ESB), there is the concept of *adapter* as a communication mechanism to connect disparate services. Adapters act as an abstraction layer between services. However, many application providers attach significant costs to these types of extensions.

Currently, two versions of BPEL coexist in the market place, BPEL4WS and WSBPEL 2.0. WSBPEL 2.0 is an emerging standard which has not, at time of writing, been finished. Some vendors, however, are already announcing support for WS-BPEL 2.0 in their execution engines. An important point to note about WSBPEL 2.0 is that any processes defined in BPEL4WS will not run on a WSBPEL 2.0 execution engine due to incompatibilities in the schema.

YAWL is an alternative to BPEL which is based on an extension of Petri nets (van der Aalst and ter Hofstede 2005). YAWL has been designed to support a large number of workflow patterns as well as human tasks. Despite these advantages, YAWL lacks industry support. As BPEL and YAWL are incompatible, the cost and complexity of migrating to YAWL would be non-~~negligible~~ for enterprises that have invested in BPEL technologies.

12.4.2.3 Problems with Humans as a Component in the Infrastructure

BPEL was primarily designed for system-to-system communications, not for human-to-system communications. To overcome this limitation, current solutions use a task management service that is external to the BPEL core (Clugage et al. 2006). This service, when instantiated, hands back to the BPEL engine once the assigned task has been completed. An issue with this workaround is that human tasks do not have parity with other tasks. The other issue is that current implementations of external task lists are proprietary which kills service integration and significantly reduces their value in the SOA context.

If a BPEL process is dependant on a human component, it is possible for that process to become stuck, i. e., a process will not resume until a human agent has executed their subset of the process. Under the current BPEL specification, the only way to "unstick" a process would be to undo significant parts of the process which could result in poor performance characteristics, especially if the process is long-running. IBM and SAP BPEL have put forward an extension to WSBPEL 2.0 to address this issue called BPEL4People. BPEL4People has the concepts of activities

and tasks and adds a new type of activity to BPEL called *People Activities*. Again, this presents a significant problem for adoption as other BPEL tools will have to extend their process execution engine to support People Activities.

12.4.2.4 Open Communication Pathway

A Lingua Franca is necessary for any open communication between Web Services in a SOA. Messages must pass between services and be understood by products to enable a composite application workflow framework. It is significant that Web Services communicate via an open mechanism. This levels the playing field for service developers, meaning that an individual developer is on equal terms with a major vendor. This also paves the way for a greater number of services and encourage a communication-oriented development process. As stated by Metcalfe's law, the value of a telecommunications network is proportional to the square of the number of users of the system.

The main communication mechanism for Web Services is XML (W3C 2006). XML is for the most part human readable and different dialects can be created for specific application, or service, needs. Technically, a service must parse an XML schema in order to participate in a collaboration or workflow. One issue is that XML data can sometimes be overly verbose which makes parsing slow. This has led to the increasing popularity of the Javascript Object Notation (JSON) as an alternative communication mechanism (Crockford 2006). JSON is a serialised data format that is capable of expressing the same data constructs as XML but in a far more compact manner.

12.4.3 Modern Architectural Styles

There are currently two main architectural styles for SOA, the WS-* style and the Web-Oriented Architecture/REST-Oriented Architecture style (WOA/ROA). To understand the differences between the two, it is valuable to break down the overall communication mechanisms.

12.4.3.1 WS-* Monolithic Approach

The WS-* approach consists of a multi-layer application stack ~~including~~, from the top down, service orchestration based on BPEL, service definition based on the Web Services Description Language (WSDL), service discovery based on Universal Description Discovery and Integration (UDDI), messaging based on the Simple Object Access Protocol (SOAP), transport based on the Hypertext Transfer Protocol (HTTP/HTTPS), and application servers.

Orchestration of workflows in the WS-* architecture is typically managed by a BPEL execution engine. The engine parses and interprets the BPEL code to

trigger events. WSDL is an XML-based format used to describe the interfaces of the services being orchestrated (W3C 2001). WSDL provides the ability to build composite applications from service descriptions without having to know or care about the way services are implemented. A WSDL file specifically contains the location of a service, the access to the service and what the service provides. It includes an abstract which defines the SOAP messages in a language agnostic manner as well as a descriptor which details the specifics of the service such as type.

UDDI is a platform-independent XML-based registry used to publish and discover Web Services (OASIS 2002). UDDI is interrogated by SOAP messages and provides access to the WSDL files describing the published Web Services. The process of selecting a service can be abstracted using an ESB. However, UDDI is not a plug and play solution for Web Services since it only describes available communication mechanisms, i. e., the composite application's logic still needs to be written around the UDDI description.

The messaging layer is implemented via SOAP (W3C 2007). Data is wrapped in an envelope with a header and body. The envelope is the root element of the data. A problem with this method is that it breaks a fundamental concept of the Web, the Uniform Resource Identifier (URI). Rather than having each service be described by a unique URI, only the base level service is addressable via SOAP and the subsequent services are addressable from that root node. The transport layer is normally based on HTTP or secure HTTP but other transport protocols may be used. The most common application servers are J2EE and .NET with J2EE having a larger share of the enterprise market thanks to popular open source products such as JBoss and GlassFish.

12.4.3.2 WS-* Standards Proliferation

Multiple WS-* standards have been rolled out as products. As many of these standards are being written by the vendors themselves, it is not necessarily in their best interest to ensure that all their products "play nicely" together. If a vendor can acquire lock-in, it will capture more of the market space and consolidate its business offerings and dominance in the product area. This fragmentation of standards and attempts at acquiring lock-in are, at their core, the complete antithesis of the promise of SOA, i. e., loosely coupled, lightweight, componentised services.

Table 12.1 lists some of the WS-* standards being used or under development. The implementation of this stack varies from vendor to vendor. To address this issue, OASIS, as of October 2006, have started to work on a reference architecture. However, difficult challenges lie ahead. As quoted from (Weerawarana et al. 2005), "there is no universally agreed standard middleware, which makes it difficult to construct applications from components that are built using different programming models ... They bring with them different assumptions about infrastructure services that are required, such as transactions and security. As a consequence, interoper-

Table 12.1 Some of the WS-* standards

Security:	WS-Federation
	WS-SecurePolicy
	WS-Security
	WS-Trust
Transaction:	WS-Addressing
	WS-AtomicTransaction
	WS-Coordination
	WS-Eventing
	WS-Notification
	WS-Transaction
Directory access:	WS-Attachments
	WS-Discovery
	WS-MetaDataExchange
	WS-Policy
Messaging:	WS-Addressing
	WS-Eventing
Reliable:	WS-Reliability
	WS-ReliableMessaging
Management:	WS-Management
	WS-ManagementCatalog

ability across distributed heterogeneous platforms such as .NET and J2EE presents a difficult problem."

12.4.3.3 WS-* vs. WOA

The other architectural style that has become the de facto standard in the Web 2.0 world and has been drawing significant attention from the enterprise community is WOA (Andrews 2005).[1] WOA is based on lightweight messaging technologies such as XML/JSON, HTTP and URI. With Silicon Valley startups pushing out products utilising SOA concepts like mashups and open APIs, one has to start to re-evaluate the heavyweight structure of current enterprise products.

Shortened development cycles, loosely coupled services and faster adoption are the promised benefits of SOA. The large number of protocols required in the WS-* approach seems, however, to contradict these precepts. From one perspective, the complexity of the WS-* application stack can be seen as giving the enterprise community the level of sophistication that it demands but from the lightweight, agile camp, it can be seen as a deliberate attempt to lock-in: "So that was a strategy tax that was imposed by the big companies on some of this technology, where they made it more complicated than it needed to be so they could sell tools." (Anderson 2006).

[1] Nick Gall from Gartner Research coined the term Web-Oriented Architecture

Table 12.2 From Web 1.0 to Web 2.0

Web 1.0	Web 2.0
Businesses	Community
Consuming	Contributing
Client/Server	Distributed
Ontologies	Tags
Storing	Collaborating
Roach motels	Open APIs
Eyeballs	Syndication

The WS-* monolithic SOA is designed from a pessimistic view point – the data will be in the wrong format, the service will go down. WOA is more optimistic. The service says how its data will be represented and it is the responsibility of the client or user of the service to insure that the contract is correct for its needs. WOA also has the property of asynchronous contract checking performed on the client end which provides the ability to degrade gracefully if a failure occurs. In the WS-* approach, WSDL mostly checks contract at run-time which leads to a tight coupling between services. While this would be acceptable for a behind-the-firewall deployment, this tight coupling presents issues when opening services up to the Internet.

Sun has recently proposed a new XML-based service description language as a lightweight alternative to WSDL called the Web Application Description Language (WADL) (Hadley 2006). WADL is URI-based rather than node-based. It has specific support for Create, Read, Update, Delete (CRUD) verbs in the schema which is part of the normal transaction process in the Web 2.0/Enterprise 2.0 realm.

From an architectural viewpoint, a significant difference between the WS-* and WOA styles is middleware. WS-* is centred around creating products that act as intermediaries between applications. WOA, on the other hand, is network-oriented and based on the way the Internet has been designed and is being used today in large applications for web sites that serve a significant number of users. One is reminded of the Obasanjo quote: "My website is bigger than your enterprise." (Obasanjo 2006). Although the WOA approach is early in its enterprise usage and is still lacking orchestration capabilities, it represents a significant green field development area for further research.

12.4.3.4 Not Either/Or

When evaluating the WS-* and WOA approaches to SOA and workflow, it should not be considered as an either/or choice as there is a significant difference between green field and legacy development. If an enterprise has legacy systems from a specific application vendor and this vendor has a ESB adaptor to integrate these systems into their SOA platform, this will represent a much easier migration path, whilst preserving the existing investments. However, product interoperability and agility in

delivery remain primary concerns. Although the WS-* camp has more vendor support and products in the market place, WOA arguably proposes a more pragmatic roadmap to achieve the vision of SOA.

12.4.4 Recovery Mechanisms and Techniques

Recovery from deviation in a workflow is still very much at the front of support tools. The more facility a workflow tool has to recover from deviation, the more value it will bring to the enterprise and the greater satisfaction to the users of the tool. "Users are forced to work outside of the system, and/or constantly revise the static process model, in order to successfully support their activities, thereby negating the efficiency gains sought by implementing a workflow solution in the first place." (Adams et al. 1998). While building alternatives in workflow models and utilising transactional recovery models remain popular mechanisms for handling deviation, new approaches based on expert systems, data mining and genetic algorithms, appear promising.

12.4.4.1 Expert Systems

Expert systems, or knowledge-based systems, come from the realm of artificial intelligence (Buchanan and Feigenbaum 1978). In the context of a workflow tool, expert systems extend the concept of building in alternative options at specific points in the execution of a workflow. However, in this instance, the tool itself would have control, rather than embedding it within the specification. If an exception occurs, each node may have a handler which forms a list of possible choices at that point in the current execution of the workflow, i. e., a list of anticipated exceptions. In the event of an exception for which no handler exists, either a new handler may be defined or an existing one may be adapted and stored in the knowledge base for future use. Each handler then acts as a self-contained workflow for dealing with deviation at that point in the superset.

Patterns can also be established by looking at the characteristics of exceptions from different perspectives, e. g., status, activity, event, time. From this basis, a tool could attempt to define the nature of the similarity of the deviation from a conceptual hierarchy to provide weighting. The expert system could then form a decision tree to model the exception path, potentially taking into account algorithms for weighting the decision tree in relation to calculation time, in order to provide more effective exception handling.

The problem with these models is when using such a system at scale, the replication of the knowledge base will be non-trivial. Besides, the construction of such decision trees would optimally occur during run-time. If the data were constructed at compile time, the knowledge contained within such knowledge bases would become stale at run-time, particularly if such a workflow has a large number of instances. The difficulties in implementation of a complex process, with potential for a large

number of knowledge base lookups to form decision trees on multiple nodes in the workflow, could prove inefficient, leading to undesirable performance characteristics.

12.4.4.2 Worklets

One implementation of the exception decision tree concept is worklets (Adams et al. 1998). Worklets are self-contained sub-processes with associated selection and exception handling rules. Worklets also have the concept of exception Ripple Down Rules for any active step in a workflow. This, in effect, is similar to the redo/undo transactional model but provides more flexibility at a finer level of granularity. In this implementation, worklets refer to three types of recoverable deviations: generic, case-dependent with a-priori knowledge, and case-dependent with no a priori knowledge.

12.4.4.3 Ripple Down Rules

In the Ripple Down Rule model (Compton and Jansen 1990), deviations are conceptually arranged in a binary tree format as shown in Fig. 12.2. This forms the basis of the expert system. Each node in the tree has a true or false option which terminates at a successful deviation solution. The refinement or revision of a deviation path can provide new solutions the next time the tree is formed. If no successful terminal node is in the tree, a new ad hoc step could be created as a new leaf in the tree to be added to the catalogue of recovery steps.

One issue with this approach is that it is dependant upon domain experts to provide ad hoc solutions in a timely manner. This might be acceptable if all the

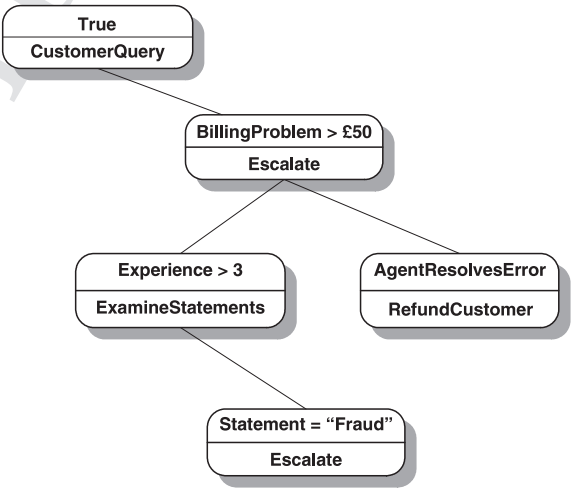

Fig. 12.2 Ripple down rules: a customer billing example

agents of the workflow are human and the workflow is non-critical. However, if some or most of the agents are Web Services, the delay would most likely cause the workflow to abort. Another issue is that, depending on the domain, a large number of deviations may have to be covered in a process. Besides, multiple recovery steps for a similar deviation may be present in the system which could lead to inefficiencies.

12.4.4.4 Data Mining

Data mining (Hartigan 1975), or knowledge discovery, is the process by which patterns are discovered in large volumes of data in an automated manner. By analysing the run-time logs of a workflow, one can derive deviations and performance characteristics. There are two perspectives from which such an analysis may be performed: structure-oriented and performance-oriented. A structure-oriented analysis will reveal modelling problems in the workflow itself, whereas a performance-oriented analysis will show deviations which may lead to bottlenecking behaviour. One can also determine through statistical analysis the frequency of similarly grouped deviations. The noise in the analysis resulting from incorrectly logged events and exceptions may however prove challenging. For example, events may occur multiple times in a process execution so deriving the point at which it occurred may lead to duplicate activities. Hidden activities, events which are not logged but form a fundamental part of the routing process, may also cause issues.

12.4.4.5 Genetic Algorithms

A more experimental approach to deviation recovery is that of genetic algorithms. Genetic algorithms can be used to generate automatically a good, or true, solution to a deviation. Experiments using local and global strategies are reported in (Alves de Medeiros et al. 2005). Local strategies were used to build optimal process models via binary relationships between events, whereas global strategies were based on a one strike search for optimal models. Local strategy cause problems as one stray event can result in damage to the results. Applying genetic algorithms in a global sense, via a Petri net, may present a more effective solution. Whilst an intriguing concept for the automation of recovery from a deviation, this work is still in early phases for adoption into an enterprise-ready solution.

12.4.4.6 Lightweight Mechanisms

Lightweight mechanisms to solve simple deviation problems remain elusive. Rather than attempting to create a highly adaptive global solution, there needs to be a solution, most likely coupled with a lightweight execution engine, that can recover from simple deviations. For instance, in a Web Services context, a tool should be capable

of re-routing to an alternative service when a service goes down rather than aborting the process. It should also have the ability to monitor the response times of different Web Services and load balance accordingly.

12.5 Summary

Looking at workflow support tools, a clear progression is evident. The early tools attempted to run as much of the workflow as they could, in as controlled a manner as possible. This, as was discovered, led to overly complex and brittle processes. Tools based on WS-* or WOA/REST principles now focus on enabling workflows and connecting systems together. Although WS-* technologies have received considerable attention, the proliferation of standards and the lack of interoperability between products remain major obstacles to their adoption. The REST approach which proposes a lightweight messaging infrastructure offers a viable alternative that leading companies are already embracing. For instance, Amazon has large-scale products running on open REST APIs, the Mechanical Turk and Simple Storage Service (S3). The Google Maps API is another example of a Web Service that is used in many third-party applications and product offerings. With the trend towards decentralised, communication-enabled business processes and agile service delivery models, the ability to handle changes and exceptions will become increasingly critical. We believe the REST approach can provide a fertile ground for innovation by drawing upon technologies from the fields of Artificial Intelligence, open interoperable APIs and Data Mining.