eFlow: a Platform for Developing and Managing Composite e-Services

Fabio Casati, Ski Ilnicki, Li-Jie Jin,
Vasudev Krishnamoorthy, Ming-Chien Shan
Software Technology Laboratory
HP Laboratories Palo Alto
HPL-2000-36
March, 2000

E-mail: casati,ski,ljjin,vasu,shan@hpl.hp.com

workflow,
service composition,
adaptive,
dynamic change

Today, companies are using the Web to connect with their back-end systems and perform e-commerce transactions. The next chapter of the Internet story is the evolution of today's access/content focused portals into e-services hubs. While many traditional services become available on the Internet as e-services, almost all of them are single point services. In order to offer higher value, end-to-end services, it should be possible to compose, customize, and deploy e-services in a very flexible and efficient way.

To support e-service delivery, we have developed a platform, called eFlow, that provides the service developer with a simple, easy to use, yet powerful mechanism for defining the composite service starting from basic services. Composite services can be preassembled or created on the fly, and can dynamically adapt to changes in the business environment, such as the introduction of new basic services. In addition, eFlow includes components that allow users to monitor, analyze, and modify a service while in execution.

eFlow: a Platform for Developing and Managing Composite e-Services

Fabio Casati, Ski Ilnicki, Li-Jie Jin, Vasudev Krishnamoorthy, Ming-Chien Shan
Software Technology Lab
Hewlett-Packard Laboratories, 1U-4A
1501 Page Mill Road
Palo Alto, CA, 94304
e-mail: casati,ski,ljjin,vasu,shan@hpl.hp.com

Abstract

Today, companies are using the Web to connect with their back-end systems and perform e-commerce transactions. The next chapter of the Internet story is the evolution of today's access/content focused portals into e-services hubs. While many traditional services become available on the Internet as e-services, almost all of them are single point services. In order to offer higher value, end-to-end services, it should be possible to compose, customize, and deploy e-services in a very flexible and efficient way.

To support e-service delivery, we have developed a platform, called eFlow, that provides the service developer with a simple, easy to use, yet powerful mechanism for defining the composite service starting from basic services. Composite services can be pre-assembled or created on the fly, and can dynamically adapt to changes in the business environment, such as the introduction of new basic services. In addition, eFlow includes components that allow users to monitor, analyze, and modify a service while in execution.

1. Introduction

The Web is changing every aspect of our lives, but no area is undergoing as rapid and significant changes as the way businesses operate. Today, large and small companies are using the Web to communicate with their partners, to connect with their back-end systems, and to perform electronic commerce transactions. The next chapter of the Internet story is the evolution of today's e-business and e-commerce into *e-services*. Examples of e-services are bill payment, hotel reservation, customized on-line newspapers, or stock trading.

The e-service environment creates the business opportunity for providing *value-added services*, which are delivered by composing existing e-services, possibly offered by different companies. For instance, an *eMove* composite service could support customers that need to relocate, by composing truck rental, furniture shipments, address change, and airline reservation services, according to the customer's requirements.

In order to enable the delivery of such value-added services, we have developed a platform, called *eFlow*, that supports the specification, deployment, and management of *composite* e-services, i.e., of e-services that are carried out by invoking several other basic or composite services. *eFlow* provides the service developer with a simple, easy to use, yet powerful mechanism for defining services by composing basic ones. Composite services can be pre-defined or created on the fly, according to the customer's needs, and can dynamically adapt to changes in the business environment, such as the introduction of new basic services or the modification or removal of existing ones. In addition, *eFlow* supports the dynamic modification of service definitions while they are operational, which is a fundamental feature in a dynamic environment such as that of Internet-based e-services.

2. The *eMove* scenario

We next present a scenario, called *eMove*, to illustrate the main issues involved in service composition. *eMove* services support several type of relocations: local, interstate, or international. Different basic services are needed for each type of relocation. For instance, local relocations could require a truck rental service, while international relocations may involve air or sea furniture shipment.

In the *eMove* scenario we assume that three service providers A, B, and C, offer several basic services, such as airline ticket reservation, furniture shipment, truck rental, and storage space rental service (see Figure 1). Realizing that relocating customers must typically access many of the above mentioned services to fulfill their requirements, a new service provider (service provider D) decides to support such customers by offering a new service category, called *eMove*, that provides complete relocation services. *eMove* services are operated by invoking other services, possibly offered by service providers A, B, C, or by service provider D itself.

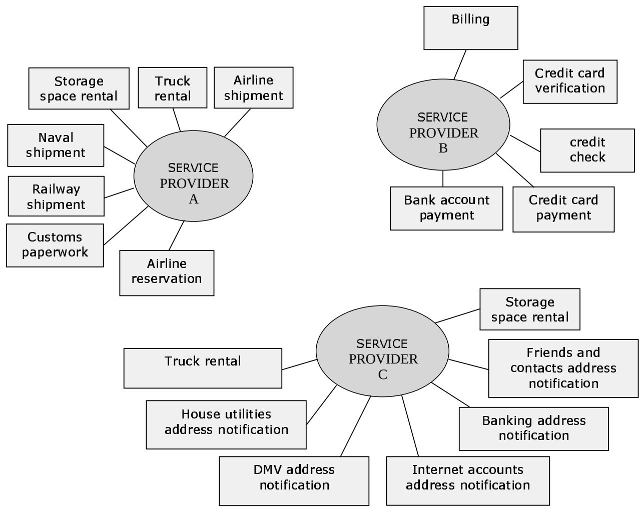


Figure 1 - Basic E-Services offered by Service Providers

Different *eMove* services target different relocation requirements. Customers do not need to be aware of the specific details of each *eMove* service: they simply specify the current location and their destination, and optionally indicate some constraints such as time or budget limit. The service provider will take care of selecting and invoking the proper composite *eMove* service. In the following, we briefly describe the usage of the *eMove* services and which (basic or composite) services are needed to support them. In Section 3 we will show how these services can be modeled and managed with *eFlow*.

eMove Level I (Figure 2) supports customers that have simple relocation requirements. It collects data from the customer (e.g., name, current address, destination address), notifies the change of address (or request termination of service) to all the parties that have relations with the customer (such as the local telephone company, the Department of Motor Vehicles, or the post office), and then bills the customer. e*Move level I* is suited for local relocations, and supports customers that do not require a packing or shipping service. This moving service is composed of two basic services (*Data collection I* and *Billing I*) and of one composite service (*Change of address/termination notification*).

eMove Level II (Figure 3) may be used for domestic relocations. In addition to the features provided by *eMove level I*, it supports customers in renting trucks and in packing and shipping furniture. *eMove level II* also provides an airline reservation service, in order to book the flight for the customer's travel to the destination. As the figure shows, this relocation service uses the *eMove level I* service in order to manage the address change/termination notifications to relevant parties and to perform the related data collection and billing activities. Rentals, shipments, and airline reservations are instead performed through the invocation of suitable basic services.

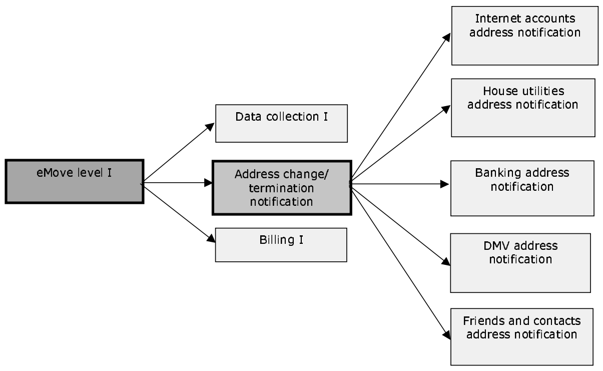

eMove level I
Data collection I
Address change/ termination notification
Billing I
Internet accounts address notification
House utilities address notification
Banking address notification
DMV address notification
Friends and contacts address notification

Figure 2 - eMove Level I service

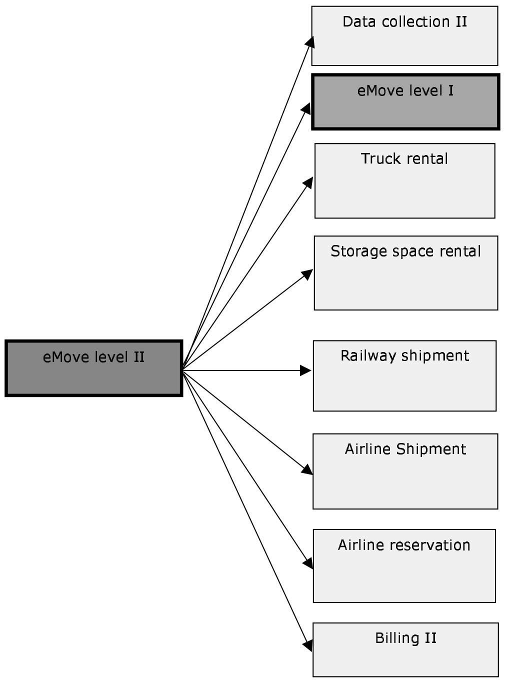


Figure 3 - eMove level II service

Notice that *eMove level II* invokes a *Data Collection II* service, which is different from the *Data Collection I* invoked by *eMove level I*, since it needs to obtain the information required to provide the address change/termination notification service, plus the information related to rentals, shipments, and airline reservations. For instance, Data Collection II requires customers to provide information about the number and size of items to be moved and about the customers' preferred airline for their travel to the new location. Part of this information is needed by the *eMove level I* service, while other is needed by the basic services invoked by *eMove level II*.

Similarly *eMove level II* invokes a *Billing II* service, whose function is to charge the customer for the rentals, shipments, airline reservations, and address change services. Each component service (including *eMove level I*) may be provided by a different company: the *eMove level II* provider is billed by these companies and directly pays them for their services. Then, the *Billing II* activity summarizes the costs and presents a single bill to the customer.

The eMove level II provides a comprehensive service package to support interstate relocations. However, a customer may only need a subset of these services. For instance, he/she may have no need for storage space or may want to separately book the flight ticket from his/her preferred travel agent. *eMove level II* is designed to be very flexible and allows different customers to select the desired subset of features that this composite service can offer. In Section 3 we will show how *eFlow* can support the definition and enactment of composite services with complex requirements such as those of *eMove level II* .

eMove Level III (Figure 4) supports customers that are relocating to a foreign country. In addition to the services provided by e*Move Level II*, *eMove level III* allows for shipment of furniture by sea and handles the paperwork needed to clear the shipped items through customs. Basic services *Data collection III* and *Billing III* provides for the additional data collection and billing requirements not managed by *eMove level II*.

The three mentioned levels of *eMove* services are pre-composed and will post different charges for their usage. In addition, service provider D would also like to offer a service, called *custoMove,* that supports the particular needs of each relocating customer. *custoMove* will allow customers to select all the services they need, and is invoked when none of the pre-packaged services can satisfy the customer's particular moving requirements.

3. Composing and managing e-services with *eFlow*

In *eFlow* a composite service is described as a process that composes other basic or composite services. Visually, a composite service is modeled by a graph that defines the

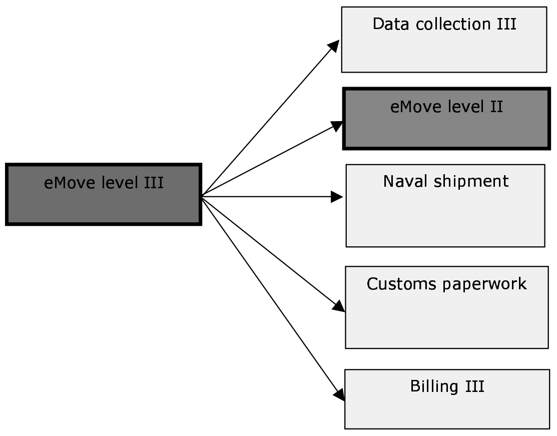


Figure 4 - eMove level III service

overall flow of service execution and data exchange among them. The graph may include two types of nodes: *service* nodes and *decision* nodes. Service nodes represent the invocation of a basic or composite service, while decision nodes specify the alternatives and rules controlling the execution flow. Figures 5 and 6 show the process definitions that model the *eMove level I* and *eMove level II* services introduced in Section 2.

In the figures, basic services are represented by rounded boxes in a light background, while composite services are represented by boxes with a darker background. Filled-in circles represent the starting and ending point of the process, while horizontal bars are one of *eFlow* decision node types, and are used to specify parallel invocation of services and synchronization after parallel service executions.

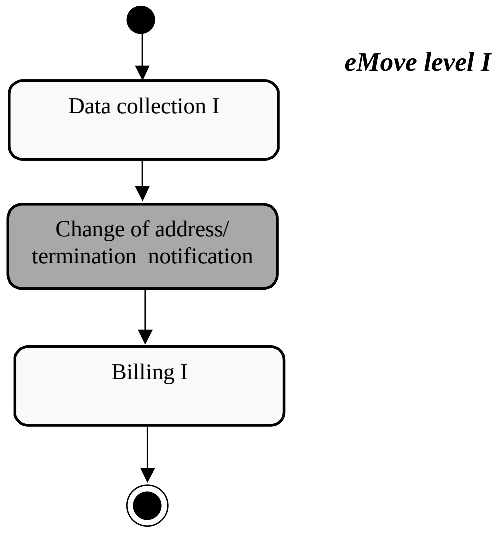
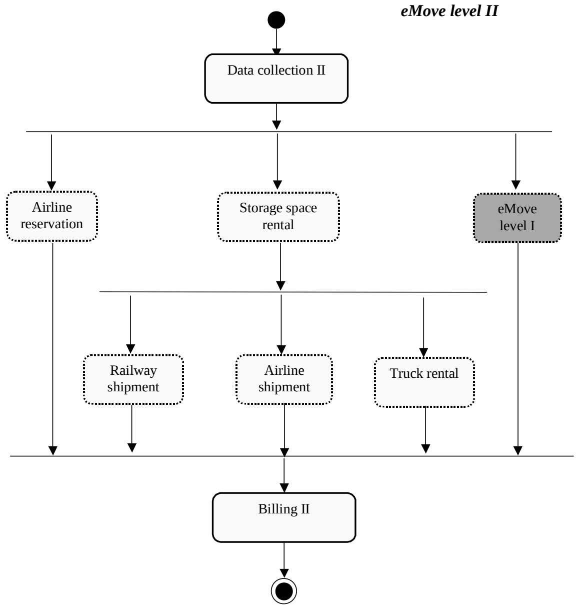


Figure 5 - *eMove* Level I composite service modeled as a process flow

For instance, Figure 6 shows how *eMove level II* can be modeled with *eFlow*. First, the process invokes the *Data collection II* service to ask customers to determine the services they need among those offered by *eMove level II* and collect the data required by those chosen services. The next step in the process involves the parallel invocation of the *Airline reservation*, *Storage space rental*, and *eMove level I* services. Notice that the services will be invoked only if the customer requests them; otherwise they will be skipped. The *skip condition* attribute of service node is used to indicate the customer's choice. In general, if the skip condition holds, a service node is skipped and the flow continues with the activation of the next node. Skippable services are graphically represented by rounded boxes with a dotted border.

After the storage space has been rented, *eMove level II* invokes the requested shipment and truck rental services. Finally, when all invoked services have been completed, *Billing II* is started to bill the customer and collect the payment.



SKIP CONDITIONS
SERVICE Airline reservation: SKIP IF neededServices.AirlineReservation=FALSE
SERVICE Storage space rental: SKIP IF neededServices.Storage=FALSE
SERVICE eMove level I: SKIP IF neededServices.eMoveI=FALSE
SERVICE Railway shipment: SKIP IF neededServices.RailShip=FALSE
SERVICE Airline shipment: SKIP IF neededServices.AirShip= FALSE
SERVICE Truck Rental: SKIP IF neededServices.TruckRental= FALSE

Figure 6 - *eMove* level II process

4. Dynamic service process composition and modification

The e-services environment is highly dynamic: the number and types of services provided through the Internet is growing on a daily basis, as is the number of companies that offer e-services. Therefore, in order to stay competitive, providers of composite services must be able to quickly and effectively modify their processes, in order to adapt to the ever-changing environment.

In addition, the increased, global competition is pushing companies to provide personalized services, to better satisfy the needs of each individual customer. In order to support these requirements, *eFlow* includes several features that enable the dynamic creations of service process definitions as well as the dynamic modifications of service process instances while they are in execution. In the following subsections we illustrate these features.

4.1 Dynamic service node creation

To support the dynamic creation of process definitions for composite services, the *eFlow* model includes the *generic service node*. Generic service nodes support dynamic process definitions for composite services such as *custoMove,* mentioned in Section 2. Unlike ordinary service nodes, generic service nodes are not statically bound or limited to a specific set of services. Instead, they include a configuration parameter that can be set with a list of actual services either at process instantiation time (through the process instance input parameters) or at runtime, by a previously executed service node. The specified services will be executed in parallel or sequentially depending on an *executionMode* attribute of the generic service node.

Figure 7 presents the sample *custoMove* process and shows how the process is instantiated according to the customer's input. In the scenario, customers specify the services they need by accessing the *custoMove* web page and by selecting the checkboxes corresponding to the required services. As the customer submits the form, a new instance of the *custoMove* process is started, and the list of selected services is passed as part of the process instance input parameter, in order to configure the generic service node *Furniture Moving Services.* The new instance will be executed according to the *custoMove* process definition where the generic service node has been replaced by a set of service nodes (whose definition is loaded from the service node repository, defined next) to be executed in parallel, according to the customer's input and to the generic service node specification.

If the flexibility requirements are such that the services to be invoked within a generic service node cannot be determined at service process instantiation time, the resolution of a generic service node can be further delayed until its execution time. In this case, the configuration parameters can be set by previously executed service nodes rather than through the process instance input parameter. Notice that generic nodes are resolved each time they are activated: if the generic service node is within a loop, and a service node in the loop changes the configuration parameters, then the generic node can be resolved into different service nodes for each loop of the execution.

The generic node approach provides considerable flexibility and supports the needed changes of services in a dynamic way to cope with today's changing environments. In particular, it minimizes the effort of changing the process when services are added, modified, or removed, since the generic node dynamically adapts to these changes and retrieves the latest service node definitions from the repository.

4.2 Dynamic service process instance modification

In dynamic operational environments, service process definitions may need to be modified for some of the running instances. For example, we may need to manage errors or exceptional situations, deal with new laws or business policies, or simply to improve the process definition. *eFlow* supports two types of dynamic changes:

Ad-hoc changes: these are modifications that can be applied to a given composite service process while it is in execution. For instance, suppose that the travel arrangements booked by an agent is unavailable due to a strike. Then, the process need to be modified to contact different agents for alternatives. The *eFlow* system supports several types of ad-hoc changes. These include modifications to the process flow graph, modifications to the definitions of service nodes, modification of the routing conditions, and modifications of the value of service node data. Ad-hoc changes only affect the execution of a specific process instance. Other running process instances or the stored process definition will not be affected. In general, this mechanism is used to handle exceptional situations that are not expected to occur again in other executions of this service process.

Dynamic process evolution: *eFlow* allows service designers to modify service process definitions and to apply the changes to a subset of (or to all) the running instances of that service. In addition, the service designer may specify that newly started services should follow the new definition. For instance, consider the situation in which a strike hit a big airline company and is assumed to last for a long period: clearly, it is not practically feasible to separately modify each process instance; instead, with *eFlow*, the service designer can define a new process and specify that all running instances with a given property (in this case, all instances in which the customer has booked a flight with the air carrier hit by the strike) should be migrated to the new version.
The definition of service instances that need to be migrated is performed through a very simple migration language, consisting of a set of rules of the form `IF <condition> THEN MIGRATE TO <version>`. The condition is a predicate over service process data and service process execution state that identifies a subset of the running instances, while `<version>` denotes the process definition version to which instances should be migrated. The set of rules must define a partitioning over the set of active instances, so that each instance is migrated to one version at most. An example of migration rule is:
`IF (selectedAirline="FlyHigh" and travelStatus="booked") THEN MIGRATE TO "A.00.02"`. When executing migration rules, *eFlow* enforces process consistency constraints, so that the migration does not generate any runtime errors. The interested reader is referred to the *eFlow* technical documentation for details [3].

5. Process templates, service nodes, and service data repositories

In order to facilitate service process development, *eFlow* provides a repository of processes, nodes, and data type definitions. The process library includes complete process definitions as well as process templates, i.e., of process skeletons in which some parts of the process are specified while other are *abstract*, i.e., are left undefined. Process templates are useful in order to capture characteristics that may be common to several processes, so that they may be used as a starting point for their development. When specifying a new composite service, a service designer can browse the process library to search for a process or a template of interest and modify or specialize it as needed in order to obtain the desired composite service definition. For instance, many *eMove* processes follow a similar pattern: they begin by collecting data from the customer, they invoke the needed services, and finally they bill the customer. Hence, it would be useful to factorize the common aspects in a process template such as the one shown in Figure 8. As the figure shows, the template may fully define some nodes (such as Data Collection and Billing), but leave other part of the process unspecified. In particular, the dotted box represents a part of the process graph that must be specified in order to obtain a concrete service definition. For simplicity, the figure only shows the flow graph of the template, but in general a template can specify any part of a process, such as process data, service node data, or deadlines. When reusing a template, in addition to the specification of abstract parts, the service designer can also modify or extend the definition of concrete service nodes.

For instance, in order to specify the composite services *eMove* level I, II, and III, the service designer can reuse the template of Figure 8, specify the abstract part of the graph, and possibly add input and output data to the *Data Collection* node so that it can gather the appropriate information from the customer.

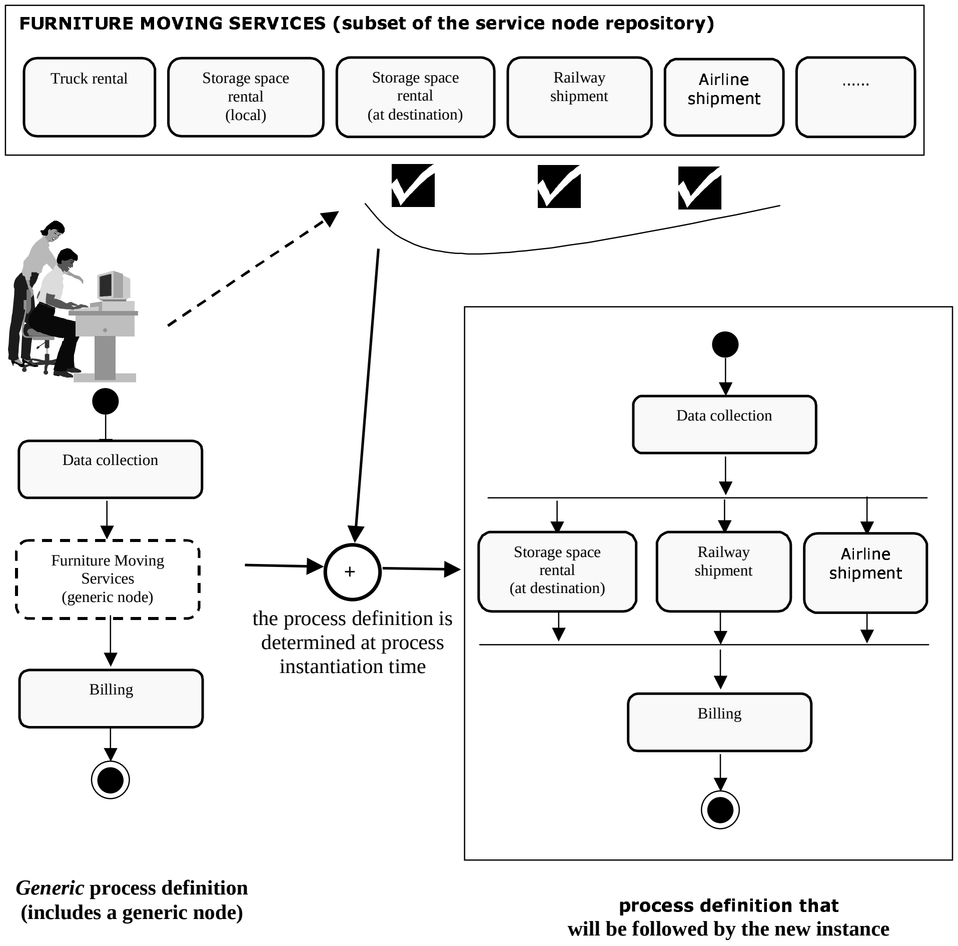

FURNITURE MOVING SERVICES (subset of the service node repository)
Truck rental
Storage space rental (local)
Storage space rental (at destination)
Railway shipment
Airline shipment
......
Data collection
Furniture Moving Services (generic node)
Billing
+
the process definition is determined at process instantiation time
Data collection
Storage space rental (at destination)
Railway shipment
Airline shipment
Billing
Generic process definition (includes a generic node)
process definition that will be followed by the new instance

Figure 7 - The custoMove generic process is fully defined at process instantiation time, based on the customer's choices

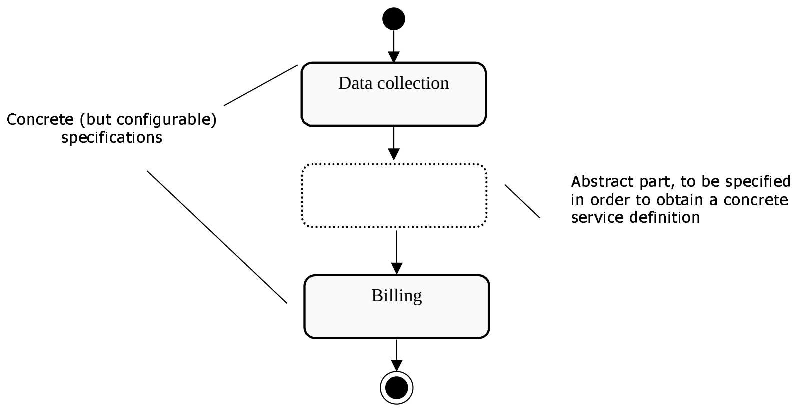

Data collection
Concrete (but configurable) specifications
Abstract part, to be specified in order to obtain a concrete service definition
Billing

Figure 8 - Sample process template definition

Process templates can be divided into groups, and groups can be organized into a hierarchy in order to structure the repository and to simplify its browsing. In addition, they have associated attributes (such as name, description, keywords, specialization/instantiation guidelines, etc) that help the service designer in querying the template repository and in using the selected template.

Another *eFlow* feature that supports reuse is the *service node repository*. Consider again the *eMove* scenario, where different *eMove* composite services may need to invoke the same service during their execution. In this case, many processes may need to include the same (or similar) service nodes. In order to simplify the definition of such composite services, *eFlow* provides a repository where service node definitions can be stored and accessed when defining new processes. As for process templates, service node definitions can be modified or extended when reused in the context of a specific process.

Finally, *eFlow* provides a *data type repository*, to allow the reuse of the same data type across different service nodes and processes. This feature is useful when different service nodes need the same data. For instance, most *eMove* services require general information related to a customer (e.g., name, home address) as well as billing information (e.g., credit card number, expiration date). In order to ease and speed up the definition of service node and process data, *eFlow* includes a data type repository (also structured into groups and groups hierarchies as the other *eFlow* repositories).

Besides providing methods and tools for modeling and managing the flow of service invocations, *eFlow* also supports the designer in handling the interaction with these services. In fact, eFlow provides several adapters enabling access to services that talk different e-commerce protocols, such as OTP, OBI, RosettaNet, or e-speak.

Due to space limitations we are unable to describe the *eFlow* architecture and to provide implementation details. The interested reader is referred to [3].

6. Related Work

Commercial workflow management systems, such as *MQ Wor*kflow [9] or *Staffware2000* [11], do not typically provide support for adaptive or dynamic processes, with the only exception of *InConcert* [5], that does provide some flexibility, although it is limited to allowing ad-hoc changes.

Recently, some approaches to handle dynamic changes have been presented in the literature by the workflow research community. One of the first contributions came from [4], where a correctness criterion for workflow evolution is proposed. The criterion, based on the definition of the set of all valid node sequences, defines when a case can be migrated to a new schema. The paper restricts to a limited set of modifications and does not discuss the handling of instances that cannot meet the correctness criteria. It also does not deal with adaptive process management.

Other contributions to the area of workflow evolution come from [8,10]. In [10], a complete and minimal set of workflow modification operations is presented. Correctness properties are defined in order to determine whether a specific change can be applied to a given instance. If these constraints are violated, the change is either rejected or the correctness must be explicitly restored with exception handling techniques. Liu et al [8] focus instead on a language for workflow evolution, by which the designer can specify which cases should be migrated to which versions, depending on conditions over workflow data. The language is (conceptually) similar to our migration language.

Ad-hoc changes and dynamic evolution are also discussed in [7]. Workflow changes are specified by transformation rules composed of a source graph fragment, a destination graph fragment, and of a condition. The system checks for parts of the process that are isomorphic with the input graph and then replaces the isomorphic graph with the destination graph for all instances for which the condition is verified. The paper also proposes a migration language for managing instance-specific migrations.

A few contributions also come from the software process modeling field. SLANG [1] is a reflective language for software process modeling. A SLANG model includes a set of type definitions and a set of activity definitions. Instances of activities are called *active copies*, and include all the necessary information to execute an activity, namely type and activity definition and the active copy state. EPOS [2] is a software engineering environment intended to support the evolution of large software systems. A process is described by a network of tasks, and composite tasks to be refined into subtask networks can be defined. An Execution Manager checks a task preconditions, executes its associated code, and verifies its postconditions. In [6] the EPOS support to process evolution is presented; EPOS and SPADE offer reflective features since both task definitions and task instances can be accessed like other objects in the system. Changes to a process are therefore performed through modifications to the definition of a composite task; running instances of a task are automatically converted in order to follow the new definition, although they might have to be restarted (losing all the work done) if the modifications cause an instance to be in an inconsistent state.

In designing *eFlow*, we took advantage of these research contributions and extended them in order to cope with a more complex process model such as that of *eFlow*. In addition, we implemented them in a commercial process management system targeted to service composition. Our major contribution lies however in the definition of the adaptive features of *eFlow*. In fact, *eFlow* allows the definition of processes that transparently adapt to changes in the environment with minimal or no user intervention, which is a fundamental requirement for operational environment. The reader interested in the details of the *eFlow* model, the detailed language features that support adaptive processes, and the full specification of the migration language and semantics is referred to [3].

7. Concluding remarks

In this paper we have shown how *eFlow* supports the dynamic composition, enactment, and management of *composite* e-services. As part of our future work, we plan to investigate issues related to dynamic change of process definitions in which transactional/compensation regions are defined. The problem in this context is that transactions and compensation require actions (such as locking of variables and maintenance of additional log data) to be performed at the beginning of the region. Hence, some restrictions need to be imposed on modifications applied while the transactional region is in execution.

In summary, we believe that the *eFlow* platform has the required characteristics and functionality to satisfy the need of Internet-based service providers. *eFlow* is integrated with the Hewlett-Packard e-service strategy, and will be the backbone of many HP e-services platforms. However, *eFlow* is an open technology: it is based on Java and it is compliant with the workflow and Internet standards, such as XML and the Workflow Management Coalition Interface standards. Hence, it can be integrated and used in virtually any IT environment.

References

[1] S. Bandinelli, A. Fuggetta and C. Ghezzi, Software process model evolution in the SPADE environment, IEEE Transactions on Software Engineering, 19(2), 1993.

[2] R. Conradi, E. Osjord, P. Westby and C. Liu, Initial Software Process Management in EPOS, Software Engineering Journal (special issue on Software Development Environments and Factories), Sep 1991.

[3] Hewlett-Packard. *eFlow* Model and Architecture, version 1.0. 1999

[4] S. Ellis, K. Keddara and G. Rozenberg, Dynamic Change within Workflow Systems, Proceedings of the ACM Conference on Organizational Computing Systems (COOCS '95), Milpitas, California, 1995.

[5] Ronni T. Marshak. InConcert Workflow. Workgroup Computing report, Vol 20, No. 3, Patricia Seybold Group, 1997.

[6] M. Jaccheri and Reidar Conradi, Techniques for Process Model Evolution in EPOS, IEEE Transactions on Software Engineering, 19(12), 1993

[7] G. Joeris and O. Herzog. Managing Evolving Workflow Specifications with Schema Versioning and Migration Rules. TZI Technical report 15, University of Bremen, 1999.

[8] C. Liu, M. Orlowska and H. Li. Automating Handover in Dynamic Workflow Environments. Proceedings of CaiSE '98, Pisa, Italy, June 1998.

[9] MQ Series Workflow - Concepts and Architectures, "IBM, 1998

[10] M. Reichert, P. Dadam. Supporting Dynamic Changes of Workflows Without Loosing Control. Technical report 97-07, University of Ulm, 1997.

[11] Staffware Corporation, Staffware2000 White Paper, Available at http://www.staffware.com/home/products/Staffware2000WP.zip, 1999

Fabio Casati is a researcher at HP Labs, Palo Alto. He received the Ph.D. degree from Politecnico di Milano in 1999. His research interests include information systems modeling and design, business process management systems, and e-services hubs. He has published several papers in international journals and conferences, and contributed to several books.

Vasudev Krishnamoorthy is senior architect at Rightworks. He received his MS degree from the University of Texas, El Paso, in 1991. After working at Consilium, in 1996 he joined HP to work on the Changengine workflow system. He moved to HP labs in 1999, before joining Rightworks in the same year.

Ski Ilnicki is a project scientist at HP Labs, Palo Alto. His main research interests are focused on end-to-end security solutions. In the past, he has been engaged in research on network, specifically on secure network printing and secure multicast fabric.

Lijie Jin is a researcher in the Hewlett Packard Laboratories, Palo Alto, California. He joined HP in 1999. He received his PhD degree in computer science from Harbin Institute of Technology, China. His research interests include distributed computing environment, scalable computing architecture and business process management systems.

Ming-Chien Shan is a project manager at HP Labs, Palo Alto. He joined HP in 1985 and managed various projects in object-oriented DBMS, heterogeneous DBMS, workflow, telecom service provision, and E-business operational hub. He received his PhD degree in computer science from UC Berkeley. He has published more than 50 research papers and been granted 11 software patents. Ming-Chien has served as chairperson or program committee member in many conferences.