Developing E-Services for Composing E-Services

Fabio Casati, Mehmet Sayal, and Ming-Chien Shan

Hewlett-Packard Laboratories
1501 Page Mill Road, 1U-4
Palo Alto, CA, 94304 USA
{casati,sayal,shan}@hpl.hp.com

Abstract. The Internet is rapidly becoming the preferred mean through which companies provide services to businesses and customers. A large number of e-services, including for instance stock trading, customized newspapers, real-time traffic report, or itinerary planning, is already available on the Web, and the type and number of e-services grows on a daily basis. In order to support the development and deployment of e-services, software vendors are developing e-services frameworks and platforms, that provide a language for *describing* an e-service, and then allow service providers to *register*, *advertise* and securely *deliver* e-services to (authorized) users. A *composite* e-service is an e-service defined by composing other basic or composite e-services. As the e-service paradigm becomes popular and more and more applications are developed or deployed as e-services, the need and opportunity for defining composite service become manifest. This paper presents a specific type of e-service (or, rather, a meta e-service) called *Composition* E-Service (CES), that allows the definition, execution, management, and monitoring of composite e-services. We first describe the advantages and the functionality of such a service. Next, we present the language used for specifying the composition, also discussing why existing workflow languages are not suitable for this purpose. Finally, we present the architecture and implementation of the CES we developed to deliver the service on top of the e-services platform *e-speak*. An analogous architecture and implementation strategy can be followed with any other e-services platform.

1 Introduction

Today, the Internet is not only being used to provide information and perform e-commerce transactions, but also as the platform through which services are delivered to businesses and customers. The explosion of the number and type of services as well as service providers requires mechanisms and frameworks that support providers in developing and delivering e-services and support consumers in finding and accessing them. Several software vendors and consortia are providing models, languages, and interfaces for describing e-services and making them available to users. Such frameworks usually allow the specification of business functions or applications in terms of their properties, which can be generic (such as the service name and location) or service-specific (such as the *car size* for a car rental service). Depending on the framework, the properties are represented by Java vectors or XML documents.

K.R. Dittrich, A. Geppert, M.C. Norrie (Eds.): CAiSE 2001, LNCS 2068, pp. 171–186, 2001.

In addition, vendors also provide software platforms (called E-Services Platform, or simply ESP in the following) that allow service providers to register and advertise their services and allow authorized users to lookup and access registered services (see Fig. 1). Examples of such platforms are BEA eCollaborate, WebMethods Enterprise, Sun Jini, Microsoft .net, and HP e-speak.

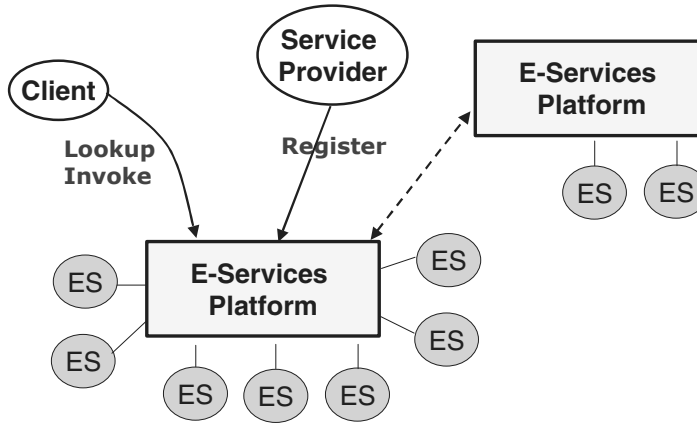

Fig. 1. E-Services platforms allow providers to register e-services and users to lookup and invoke them. Ovals labeled *ES* represent registered e-services.

These approaches enable the uniform representation, search, and access of business applications, both those used for internal operations (such as ERP operations, DBMSs, CRM, SCM, etc) and the ones that are made available to customers, typically via the Web.

The uniform representation and implementation of applications according to a homogeneous e-service framework creates the opportunity for *composing* individual, web-accessible e-services (possibly offered by different companies) into pre-packaged, value added, *composite* e-services. For instance, a provider could offer a travel reservation service by composing hotel and flight reservation services, or it could offer an itinerary planning service by composing road map services, weather services, traffic prediction services, and "utility" services to collect data from the user via the Web or send e-mail notifications.

Although composite services could be developed by hard-coding the business logic using some programming language, service providers would greatly benefit from a service composition tool that could ease the task of composing e-services, managing and monitoring them, and making them available to authorized users. This issue is similar to that of workflow applications, where the alternative is hard-coding the flow logic or using a Workflow Management System (WfMS). The advantages of service composition and workflow management tools versus hard-coding (for many practical applications) have been discussed elsewhere in the literature and will not be presented here (the interested reader is referred to [2, 4, 8]).

The traditional approach to providing a composition facility, advocated by workflow and Enterprise Application Integration (EAI) vendors, consists in offering a development environment targeted to the enterprise IT personnel. We decided to follow a different approach, that consists in providing composition functionality as an

e-service itself (or, rather, a meta-service, since it is a service for developing services). By making it an e-service, the service composition facility can be advertised, discovered, delivered, managed, and protected by end-to-end security analogously to any other e-service, thereby exploiting all the advantages and features provided by the ESP. In addition, the ability of defining and deploying composite services is not limited to the ESP's owner, but can be offered to other businesses and customers, thereby relieving them from the need of maintaining a composition system that may be onerous to buy, install, and operate. In the following we will refer to this meta-service e-service as *composition* e-service, or simply CES.

In this paper we present the design, architecture, and implementation of the *CES*. We first introduce the notion of composition as an e-service and provide an overview of the functionality and behavior of such a service. Then, we discuss the characteristics of composite services, and analyze similarities and differences with respect to workflow processes. This discussion will introduce and motivate our choices in the definition of the service composition model. Next, we present the architecture and implementation of the composition e-service we have developed on top of the e-services platform *e-speak*. An analogous architecture and implementation strategy can be followed with any other e-services platform, and therefore provides a viable solution for software vendors and solution providers that need to develop a composition facility.

2 Service Composition as an E-Service

This section first briefly describes ESPs (in order to make this paper self-contained), and then introduces the basic functionality of a CES.

2.1 Basic ESP Functionality

ESPs typically allow service providers to *register* services, and allow authorized users to *lookup* and *invoke* registered services. In order to make services searchable and accessible to customers, service providers must register the service definition with the ESP, and possibly with advertising services. As part of the registration process, the service provider gives information about the service, such as the service name, the methods (operations) that can be performed on the service along with their input/output parameters, or the list of authorized users. Note that, in most service models, a service may provide several methods (operations) to be invoked as part of its *interface*. For instance, an e-music service may allow users to *browse* or *search* the catalog, to *listen* to songs, or to *buy* discs or mp3 files.

In addition, the service provider specifies who is the *handler* of the service, i.e., the application that must be contacted in order to request service executions. Depending on the service model and the ESP, the service handler can be identified by providing a URI (such as in e-speak) or by giving a proxy java object that will take care of contacting the handler (such as in Jini). Customers may look for available services by issuing *service selection queries*, that may simply search services by name, or can include complex constraints on the service properties as well as ranking criteria in

case multiple services satisfy the search criteria. Service selection queries return a reference to one or more services that can be used to invoke them.

2.2 CES Functionality

This section describes the behavior of the composition e-service. A CES sits on top of an e-services platform and allows users to:

- Register and advertise definitions of composite services with the ESP and make them available to authorized users just like any other e-service. Composite services are defined in a Composite Service Description Language (CSDL), whose features will be presented later in the paper.
- Invoke (start executions of) composite services. The CES will execute the service on behalf of the user by appropriately invoking the component services as defined by the CSDL specifications.
- Monitor and manage composite services. The CES allows the modification or deletion of composite service definitions as well as running instances. Customers and service providers can monitor/track the execution of on-going instances as well as completed composite service executions.

In order to register a composite service, the service provider must give the same information needed to register a basic service (except for the handler - see below) to the CES, so that the composite service can be registered and made available to authorized users. In addition, the service provider gives the CSDL specifications to define how services should be composed[1].

Fig. 2 shows the composite service registration process for a composite service called *FoodOnWheels* (described in the following section): a provider that wants to define a new composite service invokes the *register* method of the CES by sending the service description (service information plus CSDL) as parameter. The CES then registers the composite service with the ESP in order to make it available as an e-service to the other (authorized) customers. The registration with the e-service platform is analogous to any other service registrations, and therefore the CES must provide all the required information describing the e-service and restricting access to it. In particular, it should also specify who is the handler for the service.

When a client needs a food delivery service, it queries the service repository to find out which services are available, asking the ESP to rank the services according to the specified criteria and return the best one. If the best service happens to be FoodOnWheels, then a reference to this service is returned. As for any other service, the client can then query the service description stored in the repository and perform method invocations on this service (see Fig. 3). The client has no knowledge that the service is in fact composite.

Figures 2 and 3 represent what happens "conceptually" from the CES users' viewpoint. When discussing the implementation, we will show that what happens behind the scenes is actually slightly different, but users are unaware of these differences, and the behavior of the system is as described above.

---

[1] A CES may also provide built-in, *utility* services, that provide frequently needed functionality, such as e-mail notifications or generation of web forms for collecting input data.

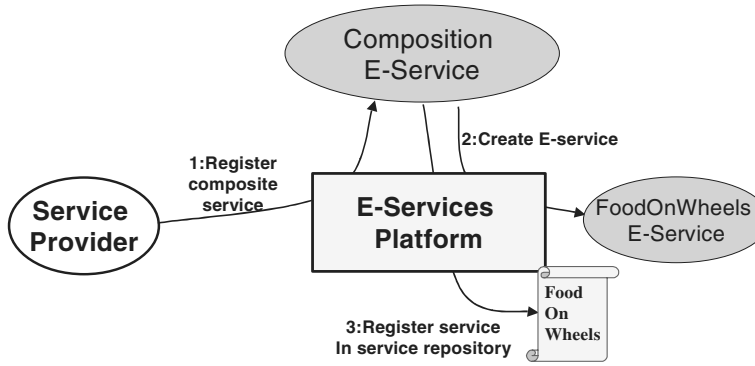

Fig. 2. Registration of a composite service, made available as an e-service

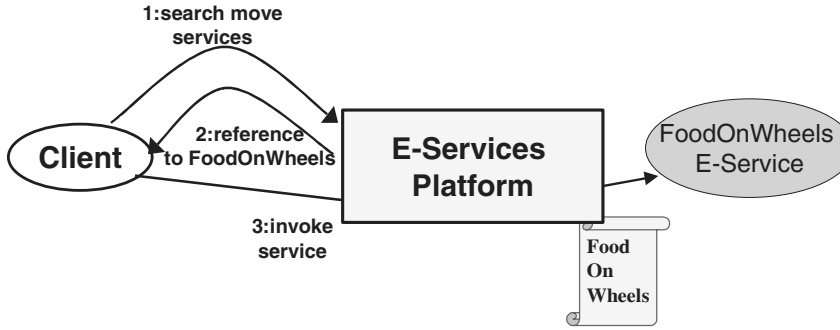

Fig. 3. Service selection and invocation

Service providers can update or delete a service definition, resulting in a corresponding update or deletion of the service registration on the ESP. Note that, technically, the definitions on the ESP are "owned" by the CES. This prevents service providers from directly updating or deleting composite service descriptions on the ESP, bypassing the CES and causing inconsistencies between information stored at the CES level and at the ESP level. The CES also allows service provider to monitor the status of service executions (note that since any composite service is itself an e-service, monitoring features provided by CES are in addition to whatever mechanism is provided by the e-services platform for service monitoring). The CES allows service providers to check how many services are in execution, at what stage they are in the execution (i.e., which path in the execution flow they have followed, which service is currently being invoked, what is the value of composite service data, etc.). CES monitoring capabilities are similar to those provided by WfMSs.

Services created by the CES also include method calls that allow users to control service executions. More specifically, users can *pause*, *resume*, and *cancel* a service execution (see Fig. 4). Note that while service providers interact with the CES, clients of composite services only interact with the services through the service reference they got as a result of the lookup, as with basic e-services.

Finally, we observe that the CES should be able to compose any service that is reachable through the ESP on top of which it is developed. Advanced ESPs such as e-

speak are capable of searching and accessing e-services delivered through ESPs of different kinds, either natively or through gateways provided with the platform. Hence, we conveniently rely on the capability of the ESP to access e-services running on top of heterogeneous ESP platforms rather than re-developing the same interoperability features.

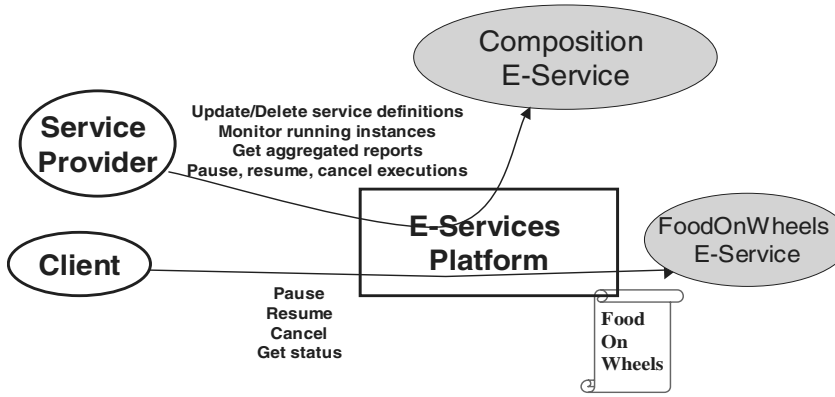

Fig. 4. Service providers can manage definitions and monitor and control executions, while service users can control executions (to the extent allowed by the provider)

3 Composite Service Definition Language

This section presents the service composition model and language. We first discuss the characteristics of a composite service and we underline the differences between workflow and e-service composition. Then, we present the composite service description language.

3.1 Workflows and E-Service Composition

This section introduces the main characteristics of composite services. In particular, we introduce them in terms of differences with respect to workflow applications. In fact, in many ways, a composite service is similar to a workflow: in order to define a composite service, the provider mainly needs to specify the *flow* of service invocations (i.e., the services to be invoked, their input and output data, and their execution dependencies). Similarly, in a workflow, the designer must specify the flow of work (i.e., the work items to be executed, their input and output data, and their execution dependencies).

Hence, an option that we had considered for CSDL was to simply use an existing workflow modeling language. However, a language and system for service composition has many different requirements with respect to workflows. We list the main differences below:

- *Service selection:* Nodes in traditional workflow graphs represent administrative or production work items, assigned to human or automated resources. Often, workflow models also impose a resource model, based on roles and/or organizational levels. Selecting a resource typically involves selecting an employee or an enterprise application by means of a (possibly rich and expressive) resource language that identifies authorized resources depending on the roles they play and on the level they belong to.
- Similarly, nodes in an e-service environment represent service invocations. As part of the service node definition, the provider specifies the service to be invoked. Although conceptually this is similar to selecting a resource for a work item, the e-service environment has very different concepts and requirements: there is typically no fixed "organizational model" or resource taxonomy. The service is selected depending on its properties, and the selection criteria are specified in the query language supported by the e-services platform, which is usually quite powerful and flexible. A service composition language should support and facilitate the definition of service selection criteria for each node in the flow, allowing also criteria that depend on the specific instance in execution (i.e., are sensible to the instance-specific data, such as the customer name or geographical location). Note that, while in principle it is possible to follow the "workflow approach" (i.e., identify and classify services in advance and then specify work assignments through some role expression), this is not required due to the presence of a (homogeneous) service repository in the ESP and of a service query and selection language. Besides not being required, the workflow approach is also not advised. In fact, the e-service environment is very dynamic and services are introduced, modified, or deleted very often. Hence, the content and structure of the repository would have to be updated all the time.
- *Input and output data*: in workflows, input and output data are typically specified by a set of variable names. The semantics is that the value of the input variables at the time the node is started is passed to the selected resource, and node execution results are inserted into the output variables. Communication between the WfMS and the resources is done through adapters, that understand the syntax and semantics of the data and perform the required data mappings.
  E-services, depending on the platform on top of which they run, typically communicate in java or XML, and these two languages dictate the rules and the syntax for data exchanges. Therefore, facility for processing Java and XML objects and transferring them to and from the invoked e-services must be provided. Also in this case, while in principle it would be possible to follow the "workflow approach" and develop adapters that bridge the composition environment and the e-services to get rid of data mapping issues (at the cost of transferring the problem onto the adapters), this is luckily not needed. In fact, e-services running on top of ESPs share the same service model and parameter passing semantics, so that it is possible to take this into account in the service composition model and provide facility for communicating with e-services as prescribed by the ESP, thereby avoiding the need for adapters. Indeed, this is a considerable advantage, given that developing adapters is difficult and tedious job, as demonstrated by the cost of commercial system integration platforms. In addition, it simplifies the use of the CES, since developers may define and deploy a composite service by simply sending a single file that includes all the business

logic. There is no need of changing the configuration of several different systems, as it happens with current WfMSs[2].

- *Dynamic environment*: Unlike "traditional" business processes, composite e-services have to cope with a highly dynamic environment, where new services become available on a daily basis. In order to stay competitive, service providers should offer the best available service in every given moment to every specific customer. Clearly, it is unfeasible to continuously change the flow to reflect changes in the business environment, since these occur too frequently and modifying a composite service definition can be a delicate and time-consuming activity. Ideally, composite services should be able to transparently adapt to changes in the environment and to the needs of different customers with minimal or no user intervention. Workflow systems do not typically offer these capabilities.
- *Black boxes vs multi-methods interfaces*: typically a work item in a workflow represents the invocation of a business function. The work item is a black box from the workflow viewpoint. Instead, an e-service may have several states and state transitions, caused by method invocations. Interacting with an e-service requires operations to be performed at the service level (e.g., search and authentication) and operations to be performed at the method level (e.g., method invocations).
- *Security:* current workflow technology has very little support for security. Often, there is no encryption, and access is controlled by means of usernames and passwords. This is due to the genesis of WfMSs as systems for managing the work in a restricted and controlled environment, within a corporation. In the Internet and e-service environment the security requirements are different, and in particular e-services may require the use of certificates, which therefore should be also supported by the service composition model and language.
- *Business-to-business interactions*: a number of standards (e.g., RosettaNet, cXML, CBL) are being defined in order to support business-to-business interactions, possibly limited to specific, vertical markets (such as RosettaNet for the IT industry). Many applications that support such standards are being or have been developed, and it is likely that many service composition applications will interact with services that follow one of these standards. A CSDL should facilitate the composition of such services as well as their invocation, checking that the appropriate protocol is followed and that exceptions are thrown when deviations from the protocol are recognized. Many workflow models and systems do not provide such kind of support, although many vendors are moving in this direction.

3.2 CSDL Definition

This section presents the Composite Service Description Language. Although CSDL reuses some of the concepts developed by the workflow community, it has several innovative features that make it suitable for service composition:

---

[2] The adapter approach can still be followed, if the users so desire, by embedding the mapping semantics into suitable e-services.

1. It has a two-level service composition model, that distinguishes between invocation of services and of operations within a service. This is important since some aspects of the business logic are specific to a service and need to be specified at the service level, while others are instead specific to each method invocation, as detailed in the following.
2. The language allows the definition of how to send XML documents as input to service invocations, and of how to map XML results into composite service data items. This is important since we expect most of the interactions among e-services to occur in the form of XML documents.
3. A flexible mechanism to handle certificates is provided, to enable the definition of which certificates should be sent to a service.
4. A number of adaptive and dynamic features are provided, to cope with the rapidly evolving business and IT environment in which e-services are executed.
5. Facilities for B2B interactions are provided, in the form of service templates that can be reused by composite service designers, so that they do not need to be concerned with technical details about the standard.
6. The *entire* business logic can be defined within a single XML document, thereby making easy and practical to provide and use composition as an e-service.

CSDL originates from concepts developed in a previous HP project, called *eFlow* [3], that we have extended to take into account the characteristics of ESPs and of the e-services they support. Here we will only present the innovative aspects of CSDL.

Overview. A composite service is described as a process schema that composes other basic or composite services. A schema is modeled by a graph, which defines the order of execution among the nodes in the process. At the top level, the graph may include *service, decision,* and *event* nodes. Service nodes represent invocations of basic or composite services; decision nodes specify the alternatives and rules controlling the execution flow, while event nodes enable composite services to send and receive several types of notifications. A *composite service instance* is an enactment of a composite service schema. The same composite service may be instantiated several times, and several instances may be concurrently running.

As an example, consider a *FoodOnWheels* service, that delivers any kind of food to customers' doors. The graphical description of the composite service is shown in Fig. 5. In the figure, boxes represent service nodes while diamonds represent decision nodes. The entry points are represented by right-pointing triangles, while end points are denoted by left-pointing triangles. *FoodOnWheels* receives order from customers and, if the customer has a valid credit card, it selects one or more restaurants that provide the requested food (unless the customer specifies a preference) by accessing a restaurant selection service. Then, it picks up the food at the restaurants and delivers it to the customers at the requested time, through a food delivery service. Next, the customer's credit card is charged, by invoking a credit card payment service. A composite service is textually specified by an XML document.

A composite service may include the definition of *input*, *output*, and *local* data items (sometimes also called flow variables in the following). Input data items are parameters passed to the composite service at activation time. Output data items represent data returned to the caller at service completion. Input and output data items can also be used for routing purposes within composite service execution and for transferring data among service nodes. Local data items are neither input nor output,

but are only used within the composite service to perform routing decision or to transfer data among nodes. The types of variables can be any basic Java type (e.g. String or Integer), a Java Vector, a generic Object, or an XML document. Each composite service instance has a local copy of the flow variables.

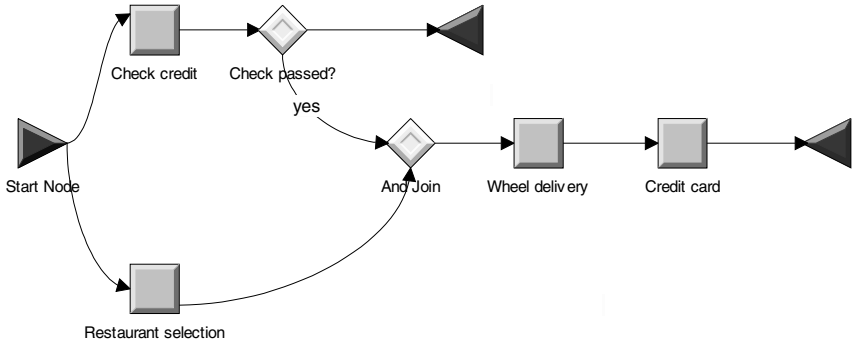

Fig. 5. Graphical representation of the FoodOnWheels composite service

Besides the graph that defines the flow of service invocations, the definition of the composite service also includes security-related specifications. In particular, the definition of a composite service includes information about the certificates to be used throughout the flow within service invocations, in case the ESP and the invoked e-service support or even require the use of digital certificates. By default, the composite service invokes component services with the privileges (i.e., the certificate) of the composite service definer. However, the designer may specify that services should be invoked with the privileges of the composite service users, or with the privileges specified by the content of a flow variable (for instance, the certificate to be used may be passed to the composite service as one of its input parameters).

Service Nodes. Service nodes represent invocations of a given service. The service to be invoked is specified by a *search recipe,* defined in the query language supported by the ESP. As the service node is started, the search recipe is executed, returning a reference to a specific service. Recipes can be configured according to the specific service instance in execution*:* every word in the search recipe that is preceded by a percentage sign "%" is expected to be a reference to a flow variable, and will be replaced by the value of that variable at the time the service node is started. This allows the customization of the search recipe according to the value of flow variables. Note that different activations of a service node may result in the selection of different services. However, sometimes the designer needs to specify a service node that should reuse the same service invoked by another service node. The composition service model allows this by enabling the definition of a *Service Reuse* attribute that includes the name of the service node whose service reference is to be reused.

The definition of the service node may include the certificate to be used when invoking the service's methods. The definition at the service level overrides the one done at the top (i.e., composite service) level. Since it is assumed that all invocations

on the same service will use the same certificate, there is no provision for the definition of a certificate at the method invocation level.

Flow of Method Invocations. E-services, in most ESP models, will have an interface that allows several operations to be invoked on them. In order to achieve their goals, clients of these services will typically have to invoke several operations (i.e., call several methods) on the same service. Correspondingly, CSDL allows the designer to specify, within a service node, the *flow of method invocations* to be performed on a service. For instance, if we are accessing the e-music service, we may want to specify that we search for a given song (invoking the *search* method) and, if the price for the disc that includes the song is lower than a limit, then we buy the whole disc (*buyDisc* method), otherwise we simply download the mp3 file of that song only, paying the requested fee (*BuySong* method). To simplify both the language and the implementation, the method flow is specified with the same syntax (and semantics) of the top-level flow of services, with the only difference that here we are concerned with the flow of *method nodes* instead of service nodes. If only one method needs to be invoked, then the designer needs not specify the flow structure, but only a single method node. In addition, we also allow the definition of service nodes that have no method nodes inside. In fact, in a few cases, the designer might only want to execute a search recipe and get the results, possibly without invoking any method on the selected service. For instance, a node may simply need to get a service name or handle in order to pass it to another service.

Method Nodes. A method node defines the method to be invoked on a service and its input data, how to handle the reply (and specifically how to suitably map the reply message into flow variables), and how to handle exceptions that may occur during the method invocation. The name of the operation to be invoked can be statically specified, or it can be taken from the value of a flow variable, as usual specified by a string preceded by the percentage sign. The input data to be sent to the method are specified by a list of variable names or values. In case of variable names, the value of the variable at the time the node is started is sent as input to the method.

If a method invocation on a service returns a result (e.g., an integer or an XML document), then the designer needs to specify how information in the document can be extracted and inserted into flow variables. In case the method output is a (basic or complex) Java object, then the mapping is simply specified by describing the name of the flow variable to which this value should be copied. For example, method *CheckCredit* returns a Boolean value defining whether the credit check on the customer is positive or negative. In CSDL, this is defined as follows:

```
<Method-Output> <Var-Mapping Flow-Var="Confirmation" />
</Method-Output>
```

Since it is likely that most of the output data will be a string containing an XML document, CSDL provides additional support for XML, and in particular it allows the designer to specify how fragments of the XML output document can be mapped into flow variables. A flow variable name assumes the value identified by an XSL transformation or an XQL query on the output document. In the case of XQL queries, if the flow variable is of type XML, then the XQL query may actually return a set of elements, or a document. Otherwise, CSDL requires the query to identify a single

element or attribute, or an exception is raised. For instance, the following mapping specifies that the XQL query `customerList/customer[0]` should be applied to the method output, and the result of the query should be put into variable "customer":

```
<Method Output>
<Var-Mapping Flow-Var="customer"  Conversion-Rule=
"customerList/customer[0]" Rule-Type="XQL" />
</Method Output>
```

The definition of the query may be static or may include references to flow variables, as usual preceded by the percentage sign.

4 The *Composition* E-Service Prototype

This section presents the CES prototype, developed at HP for composing *e-speak* e-services. The same design can however be adopted for any other ESP. The prototype is built on top of a commercial workflow engine (and specifically of HP Process Manager) that handles the execution of the flow. The need of using a commercial workflow engine came from the requirement we had of building a *robust* prototype in a very short time, that ruled out the possibility of developing one by ourselves. Note that only the engine was needed for our purposes, so we removed all other HP Process Manager components to get a lighter and faster system[3].

Another key component of the architecture is the *gateway*, that enables the interaction between the workflow engine and the ESP, performing the appropriate mappings and implementing CSDL semantics that could not be supported by the workflow engine, as discussed below.

Fig. 6 shows the components of the prototype and in particular how they handle composite service registrations. The CES front-end responds to calls from service providers and clients (even if the latter are unaware of the fact that they are communicating with the CES). When a service provider registers a service, the CES front-end first translates CSDL into the language of the selected workflow engine. The translation generates a process where nodes correspond to method invocations on the ESP or on the selected e-services. However, since CSDL is in fact much richer than traditional workflow languages, the translation is a fairly complex procedure and requires the insertion of several "helper" nodes and data items that, in conjunction with the operations performed by the gateway (that has knowledge of the semantics of such helper nodes), enable the correct implementation of the CSDL semantics. Examples of issues we have to deal with in the translation include mapping the two-level (service and method) CSDL model into a single-level one and rewriting the input and output data items of nodes so that they can have all the information required to build XML documents and to map back XML replies into process data.

For instance, consider the single problem of mapping the CSDL two-level service model into a traditional workflow model. In order to map a service node, we need to insert a node that implements the search recipe (i.e., sends the service selection query to the ESP), and to define the data items needed for storing and sending certificate information. In addition, different method invocations occur in the context of the

[3] Note that using a WfMS for *implementing* the CES is not in contrast with our previous discussion on the unsuitability of a workflow language for *modeling* composite services.

same "session" with the service. Hence, we need to define and properly initialize process data items that can carry session IDs from node to node. Note that this problem could not have been solved by simply defining a subprocess, both because the need for defining service selection nodes and certificate nodes still remain, and because nodes in a subprocess do not have access to the variables of the main process (unless they are passed as input parameters, but even in that case the parameters are passed by value and not by reference). Where it was not possible to map appropriately, we encoded part of the semantics in the gateway. For instance, XQL queries are performed by the gateway. The gateway is also in charge of replacing references to flow variables in XML documents (i.e., those items preceded by the "%" symbol) with the actual value.

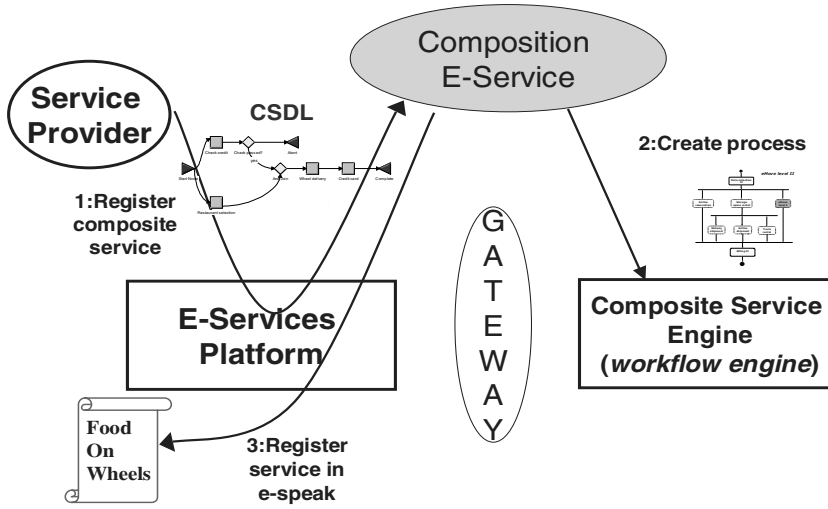

Fig. 6. Components of the CES prototype and how they handle registrations

We also mention that in this prototype we did not map adaptive features of CSDL, event nodes, and exceptions (we have only mapped deadline-related exceptions). These features will be introduced in the next version of the CES.

After the mapping has been completed and the process is installed on the workflow engine, the CES registers the new service with the e-speak ESP. As the Figure shows, the CES itself is the handler for the newly registered service. However, this does not change the validity of the scenario depicted in Figures 2 and 3. Indeed, clients simply communicate with the (composite) e-service through the reference they get, and they are not concerned with how the service is implemented on the server side.

When a client invokes a composite service (see Fig. 7), the CES starts the corresponding process in the workflow system (the mapping between the composite e-service name and the process name is defined at registration time and stored within the CES). Activities in the workflow correspond to method invocations on a given service involved in the composition. From a workflow perspective, all activities are assigned to the gateway. The gateway receives indication of what to do by the workflow engine as part of the activity definition, along with data items that provide (a) context information about the service on which method calls are being or have to

be placed (e.g., service references, search recipes, certificates, and mapping information to process the XML document returned by the method and update the value of flow variables) and (b) the value of the parameters to be passed as part of the method invocation.

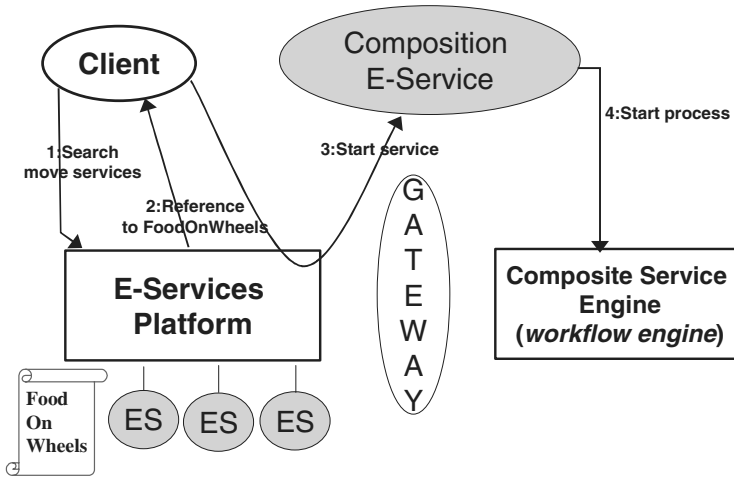

Fig. 7. Invocation of composite services

When the gateway receives work by the engine, it activates a new thread in order to process the work. The thread waits for the reply from the service, executes the mapping rules, and sends the results back to the engine. All the state information is maintained by engine, and the gateway does not persist anything. This choice is motivated by the fact that the engine logs all state changes, so there is no need for a persistent gateway.

Observe that, as experienced by WfMS vendors, building a commercial-strength workflow engine is not an easy job, especially if it includes tracking, monitoring, and business transaction functionalities and has demanding requirements in terms of availability and performance. Hence, we believe that the architecture characterized by the reuse (or possibly the adaptation) of a commercial workflow engine is the alternative that most ESP vendors will follow. Indeed, this is the path followed by HP, whose middleware offering includes e-speak and Process Manager.

We expect that in the future, as ESPs add more functionality in terms of high availability, load balancing, monitoring, and support for business transactions, the need for integrating commercial workflow engines will progressively reduce, and the development from scratch of an interpreter designed and optimized for CSDL will become realistic.

5 Related Work

To the best of our knowledge, there is no commercial composition/process management system that can perform e-service composition and satisfy the

requirements stated in Section 3, neither among traditional workflow management systems (such as *MQ Wor*kflow [10], *InConcert* [7], or *Staffware2000* [11]), nor among newly developed, open, XML- and web-based systems such as *Forte' Fusion* [9] and *KeyFlow* [6].

E-services platform themselves do not provide service composition capabilities, although all vendors declared interest in moving into this space. The only exception is WebMethods, who provide a very simple composition language for composing WebMethods' services [12]. The language allows the definition of flows that are a subset of what is allowed by traditional workflow management system (basically it can only model sequential or conditional flows where services are statically bound to service nodes), and therefore does not have many of the features presented in this paper. On the other hand, it is well suited for compositions that have simple requirements and it is quite easy to use.

Within the research community, approaches that are more closely related to the work presented here have been proposed by Georgakopoulos at al. [5] and by Benatallah at el. [1]. The first paper proposes a service-oriented process model targeted at enabling cross-organizational processes. The paper also presents a service model, where services are described by a state machine that specifies the valid "logical" states of a service and the valid state transition, caused by either method invocations or by transitions performed internally by the service. The paper differs from ours in that it focuses on the service model and only briefly sketches the service composition model. Instead, we assume that the service model is provided by the ESP, and we focus on the composition. In addition, the paper does not deal with certificates and data mappings/extraction while communicating with the e-services.

Benatallah et al. propose a framework for creating and maintaining virtual enterprises, where component enterprises share e-services. The main focus of the paper is on a model for managing service communities. However the paper also deals with service composition, and proposes an ECA-rule based approach for defining the composition. Our work differs in that CSDL has a graph-based approach to specify the composition. In addition, the paper also does not deal with search recipes, certificates, and data mappings and extraction, which are critical in our approach.

6 Concluding Remarks

This paper has presented the functionality and implementation of an e-service for composing e-services. The main contributions of this paper are:

- The idea and notion of providing composition functionality as an e-service, to be used not only by the owner of the ESP, but also by any (authorized, and possibly paying) user.
- A discussion of the characteristics of composite e-services and of their differences with respect to "traditional", workflow-like composition.
- The definition of a composition model suitable for e-services.
- The description of our prototype implementation, that shows an approach that can be reused for implementing composition on top of any ESP.

This effort is the initial part of a long-term work that has the purpose of developing a lightweight engine that can execute CSDL services, based on the assumption that

future ESPs will take care of providing load-balancing, monitoring, tracking, and of other functionality. In addition, we plan to integrate more concepts taken from *eFlow*, including generic nodes, multiservice nodes, and dynamic conversation selection.

References

1. B. Benatallah, B. Medjahed, A. Bouguettaya, A. Elmagarmid, and J. Beard. Composing and Maintaining Web-based Virtual Enterprises. Procs. of the VLDB-TES Workshop, Cairo, Egypt (2000)
2. F. Casati and M.C. Shan. Process Automation as the Foundation for E-Business. Procs. of VLDB2000, Cairo, Egypt (2000)
3. F. Casati, S. Ilnicki, L.J. Jin, and M.C. Shan. *eFlow*: an Open, Flexible, and Configurable System for Service Composition. Procs. of WECWIS, Milpitas, CA, USA (2000)
4. D. Georgakopoulos, M. F. Hornick, and A. P. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. Distributed and Parallel Databases 3(2) (1995)
5. D. Georgakopoulos, H. Schuster, D. Baker, and A. Cichocki. Process-based e-service Integration. Procs. of the VLDB-TES Workshop, Cairo, Egypt (2000)
6. Keyflow Corp. Workflow Server and Workflow Designer (1999)
7. Ronni T. Marshak. InConcert Workflow. Workgroup Computing report, Vol 20, No. 3, Patricia Seybold Group (1997)
8. F. Leymann, D Roller. Production Workflows. Addison Wesley (2000)
9. J. Mann. Forte' Fusion. Patricia Seybold Group report (1999)
10. IBM. MQ Series Workflow - Concepts and Architectures (1998)
11. Staffware Corporation, Staffware2000 White Paper (1999)
12. WebMethods Inc. WebMethods Enterprise (2000)