Efficient Integration of Web Services with Distributed Data Flow and Active Mediation

David Liu
Electrical Engineering
Stanford University
davidliu@stanford.edu

Jun Peng
Civil & Env. Engineering
Stanford University
junpeng@stanford.edu

Kincho H. Law
Civil & Env. Engineering
Stanford University
law@stanford.edu

Gio Wiederhold
Computer Science
Stanford University
gio@cs.stanford.edu

ABSTRACT

This paper presents a loosely coupled service-composition paradigm. This paradigm employs a distributed data flow that differs markedly from centralized information flow adopted by current service integration frameworks, such as CORBA, J2EE and SOAP. Distributed data flows support direct data transmission to avoid many performance bottlenecks of centralized processing. In addition, active mediation is used in applications employing multiple web services that are not fully compatible in terms of data formats and contents. Active mediation increases the applicability of the services, reduces data communication among the services, and enables the application to control complex computations. The benefits of distributed data flow and active mediation are illustrated with various applications, such as dynamic type conversion, result extraction, and engineering application. It is shown that active mediation, combining with distributed data flows, can greatly improve the performance of an application utilizing multiple web services.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures – *information hiding, patterns*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software – *distributed systems, information networks, performance evaluation*; H.2.5 [**Database Management**]: Heterogeneous Databases – *Data translation, program translation*.

General Terms

Algorithms, Performance, Design, Languages, Experimentation, Standardization, Theory.

Keywords

Web services; service integration; direct data transmission; active mediation; mobile class; handheld services

1. INTRODUCTION

We experience a continuing increase in both the size and the performance of computer networks. As networks become pervasive and ubiquitous, all computing facilities can be accessed from any geographic location. This development enables the use of remote software services over the web. Web services is now one of the foundations of a vision called utility computing – the idea that computing power should be as easy to tap into as the electrical grid. However, unlike the electrical grid where the power is generated by limited number of sources, the computing grid has numerous machines and services. Integrating related services to form composed application requires complex planning, control, monitoring, and execution.

There are many issues associated with the current service composition frameworks, and two noticeable ones are interface incompatibility and performance. Web services are normally provided as software services managed by independent service providers [6, 31]. They are globally heterogeneous and adhere to a variety of conventions for control and data. Even when standards are promulgated, such as SQL, the precise meaning and scope of the output will not necessarily match the expectations of another service. A prime example of available web services today are information providers, which expose their functionalities through XML, SQL, and report generators, but are not geared to interoperate with other services, as analytical services or predictive simulations [28]. In a typical composed application, all results from one web service have to be shipped to the application site, handled there, and then shipped to the next web service. In most cases, the centralized data-flow approach is inefficient for integrating large-scale software services. This inefficiency is implicit in all common composition protocols, such as CORBA, DCOM, J2EE, SOAP, and Microsoft .NET.

In order to deal with the issues associated with the current service composition frameworks, we demonstrate a Flow-based Infrastructure for Composing Web Services (FICAS) [15]. FICAS is implemented as a collection of software modules that support the construction of web services, facilitate the functional composition of web services into composed application, and conduct the execution of performance-enhanced applications. FICAS addresses three design concerns: (1) scalability – integration and management of large number of web services in the service composition infrastructure; (2) performance – high efficiency in the execution of composed applications; and (3) ease of composition – effective and convenient specification of service

compositions by the application programmers. FICAS uses distributed data-flows to achieve greatly improved scalability and performance without sacrificing ease of composition.

FICAS applies the concept of active mediation to enhance efficient execution of applications employing composed services. Active mediation allows code to be provided to remote services to resolve format and content incompatibilities [17]. Without being able to delegate such a capability to the remote service such incompatibilities have to be resolved at the application site. Active mediation exploits the notion of mobile code [12] to provide for unforeseen remote information processing. Specifically, matching, reformatting, rearranging, and mapping of data being sent or received among services can be embodied in mobile code, and shipped by the composed application to the remote service as needed. Remote services that can accept active mediation now have the ability to adapt their behavior to the client requests. Active mediation distributes a class of computations within the service framework, and reduces the amount of data traffic significantly by moving computations closer to the data.

Active mediation, enabled by the mobile class and distributed dataflow, is an effective approach to resolve service interface incompatibilities and improve the performance of the composed application. The concept of active mediation will have a major impact on the semantic web [4]. In cases where the composed application is operated on a handheld device, the cost of shipping intermediate data to and from the handheld can become the bottleneck of overall system. Planned properly, mobile classes can be placed onto an appropriate service to minimize the amount of data communications. This is especially beneficial when the controlling node of a composed application is on a low bandwidth device. Such low bandwidth mobile devices are very attractive to manage complex scenarios in access to government services [14], engineering [16], regulations [16], and healthcare [4].

2. SERVICE INTEGRATION FRAMEWORK

Web services execute processes that involve one or more software applications along with their domain data. Web services are composed in a loosely-coupled fashion to allow flexible integration of heterogeneous systems in a variety of domains. There has been much significant research in service composition, particularly in creating uniform ways of describing, deploying, and accessing applications [10]. Many standards have been proposed to represent processes using web services, such as BPEL4WS [1], WSCL [3], and DAML-S [2]. FICAS contributes a software composition paradigm that supports distributed data flow with active mediation and addresses the performance implications of service composition.

2.1 SOFTWARE SERVICE MODEL

The composition of multiple web services into a composed application consists of three phases. First, existing services that provide composable functionalities are catalogued. Existing services that decide to participate will expose their interfaces. Missing services are constructed, or, rather, their construction by others is encouraged or contracted. Second, the composed application is specified so that it will employ the most suitable combination of web services. Issues to be considered when composing web services include scalability of the services, robustness of the services, security of the service interaction, effective and convenient specification of the compositions, and performance of the composed applications. Third, the composed application is executed as often as needed.

In FICAS, web services are specified as a homogeneous model that promotes communication and cooperation with each other. Figure 1 illustrates the web service metamodel, which consists of a service core, an input event queue, an output event queue, an input data container, and an output data container:

- The service core represents the core functionality of the web service. It is responsible for performing computation on the input data elements and generating resultant data elements. We can often wrap existing software applications into a service core.
- Events (messages) are exchanged between services to control the flow of web service executions. Asynchronous execution of web services is achieved by using queues for event processing. The default queuing protocol in FICAS is FIFO (first in and first out), so event messages are processed in the order they arrive, but other methods are being developed [20].
- The data containers are groupings of input and output data elements for the web service. Input data elements are fetched from the input data container and processed by the service core. The generated data elements are put into the output data container. The data containers enable web services to look up generated data elements.

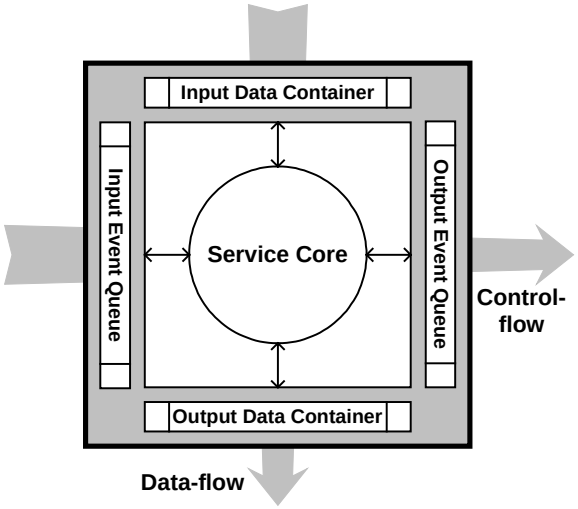

Figure 1: FICAS web service metamodel

Web services export the service functionalities contained in the encapsulated software applications. Although the service functionalities differ, the way by which the functionalities are exported is similar for all our web services. The web services share many common components, such as the event queues and the data containers. In addition, the interactions among the components are largely identical. Hence, the construction of web services can be significantly simplified by building the common components into a standard module. We call that module, which facilitates the encapsulation of software applications into web services, the web service wrapper.

In FICAS, the web service wrapper has been implemented in Java. The Java classes and interfaces are incorporated into a Java library. With the FICAS web service wrapper, the adaptation of a software application into a web service is simplified to defining the `ServiceCore` interface, which has functions for setting up, executing and terminating a service. The *setup()* method defines the actions of the application when the service is initialized; the *execute()* method is called when the service is invoked, triggering the application to process the data in the containers; and the *terminate()* method is called when the service is terminated. Each method takes three parameters: the *inputcontainer* provides the reference to the input data container of the autonomous service; the *outputcontainer* provides the reference to the output data container of the autonomous service; and the *flowid* identifies the flow to which the service request belongs. With the references to the data containers and the flow identifier of the request, the software application can look up the input parameters from the input data container and generate the results into the output data container.

2.2 MEDIATION

Services are usually built by leveraging existing software capabilities and information resources. These resources have had incompatibilities in many e-commerce and e-business applications. Mediators are intelligent middleware that sit between the information sources and the clients [27, 30]. Mediators reduce the complexity of information integration and minimize the cost of system maintenance. They provide integrated information, without the need to integrate the actual information sources. Specifically, mediators perform functions such as accessing and integrating domain-specific data from heterogeneous sources, restructuring the results into objects, and extracting appropriate information.

Figure 2(a) illustrates the mediation architecture, which conceptually consists of three layers. The information source provides raw data through its source access interface. The mediation layer resides between the information source and the information client, performing value-added processing by applying domain-specific knowledge processing. The information client accesses the integrated information via the client access interface. The architecture of the information-oriented service can be mapped to a processing-oriented architecture, as shown in Figure 2(b). The application software corresponds to the information source layer, the service wrapper corresponds to the mediation layer, and the composed application matches the information consumer layer. The software is accessed through the application-specific interface. The service wrapper obtains specifications from the services and exposes its capabilities through the access protocol.

In traditional mediators, code is written to handle information processing tasks at the time the mediators are constructed. Such mediators are static, and only modified when the sources change interfaces or behavior. Static mediators are appropriate when resource behavior is known at construction time. In contrast, the active mediators introduced in this paper allow clients to adapt the services, in particular their interfaces.

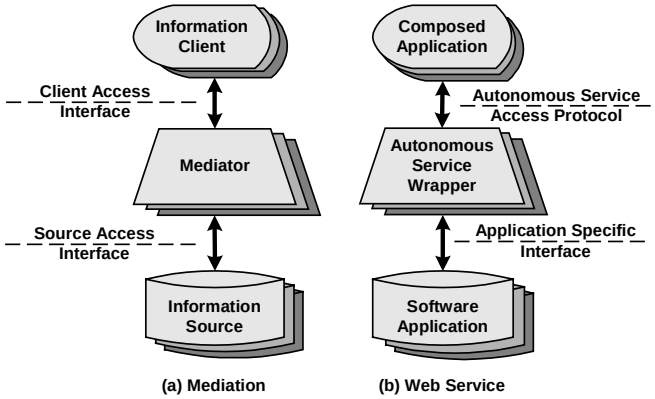

Figure 2: Conceptual layers in mediation and in service integration

3. DISTRIBUTED DATA-FLOW

In FICAS, the composed application controller has the responsibility for managing the control-flows. Based on an execution plan, the controller executes and schedules web services by managing and coordinating the choice, timing, sequence, and dependencies of control events. The purpose of scheduling is to improve the Qualify of Service (QoS) of the composed application. Many techniques used to improve QoS for distributed workflows have been proposed for web service processes [5, 32]. In our current implementation, we focus on minimizing the aggregate data-communication cost among services.

3.1 SEPARATING CONTROL AND DATA FLOWS

A distinguishing characteristic of FICAS is its distributed data-flow model, which allows direct data-flow to occur among remote services. In the common web services management, the site of the composed application is the central hub for all the control and all the data traffic, so that there is both centralized control and centralized data-flow. The distributed data-flow model provides better performance and scalability than the centralized data-flow model when data is significant. The distribution of data communications exploits the network capacity among the services, and avoids bottlenecks at the composed application. Especially when the composed application resides on a mobile device, relying on centralized data-flow would severely stress its limited bandwidth. Control, i.e., the invocation of a remote service, remains centralized in FICAS. We find it difficult to apply proposed distributed control-flow models effectively to conduct service composition. There remain many technical challenges to construct distributed operational code segments.

The separation of control-flow and data-flow is also presented in several emerging service composition standards, for example, BPEL4WS [3], WSCL [6] and DAML-S [4], demonstrating that the importance of separating control-flow and data-flow is being recognized. The idea of separating data-flow from control-flow can also be seen in some distributed workflow environments. For instance, Exotica/FMQM adopts distributed workflow execution and data management for distributed workflow applications [2, 26]. However, data flow in those environments is typically supported by a set of loosely synchronized replicated databases instead of managed direct transfers.

Figure 3 shows a schematic composed application in FICAS where the data are directly exchanged among web services. By distributing data-flows, FICAS eliminates the focused, redundant, and heavy-duty data traffic caused by the forwarding of everything through the composed application. The distributed data-flow model utilizes the communication network among web services, and thus alleviates communication loads on the composed application. Furthermore, FICAS allows computations to be distributed efficiently to where data resides, so that the data can be processed without incurring communication traffic.

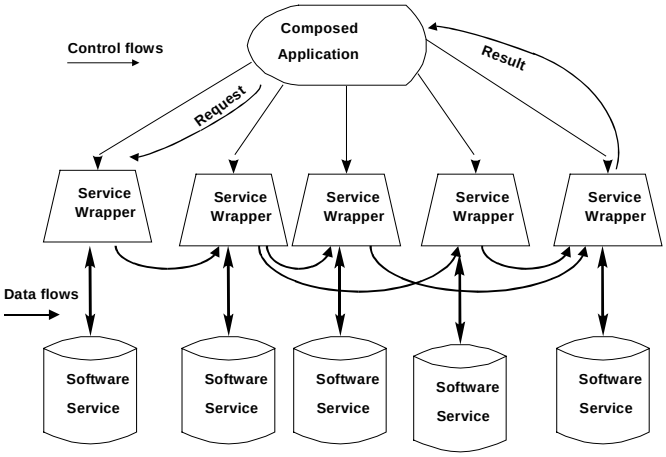

Figure 3: Sample control flow and data flow in FICAS

3.2 PLANNING DISTRIBUTED DATA FLOWS

Optimizing the placement of data processing to minimize data transfer has been of interest for distributed database systems [9, 23]. Query optimisation of distributed databases requires deciding where to ship the data and where to perform query operations. A similar concept is adopted in FICAS to plan the distributed data-flows. There are three steps in generating an execution plan. First, the composed application is analysed to discover data dependencies among web services. Then, a data dependency graph is constructed to identify independent data-flows. Finally, based on the data dependency graph, the controller then builds an execution plan.

The program segment in Figure 4 shows implicit data dependencies between web services. For instance, invocation of *Service3* takes *A* and *B* as inputs, which are the outputs of the invocations of *Service1* and *Service2*, respectively. Hence, *Service3* is data dependent on *Service1* and *Service2*. The data dependencies among the web services are analysed when the program is interpreted. The dependencies are mapped into a data dependency graph (DDG) as shown in Figure 5. The nodes represent service invocations, and the directed arcs represent data dependencies between service invocations. Each directed arc points to the dependent service and is tagged with the data elements exchanged between the pair of services. For example, the arc between *Invocation1* and *Invocation3* represents that *Invocation3* is dependent on *Invocation1*, with *A* being the data element passed from *Invocation1* to *Invocation3*.

The composed application execution plan is represented by the event dependency graph (EDG), as shown in Figure 6. The node in the EDG contains an outgoing control event from the controller. The arc establishes a predecessor-successor relationship between a pair of events. The successor event cannot be sent until the action taken by the predecessor event is completed, i.e., the controller receives the response of the predecessor event. The controller uses the EDG to coordinate the execution of the composed application. Invocation nodes in the DDG can be directly mapped into the *INVOKE* event nodes in the EDG. The mapping from the arcs in the DDG to the event nodes in the EDG is more complex.

Figure 6 shows the mapping scheme where data communications are directed among dependent web services. The controller functions merely as a coordinator for the events that control the data communication activities. Each directed arc in the DDG is mapped onto a *MAPDATA* event node with arcs connecting the predecessor and successor event nodes. For instance, the arc tagged with A in the DDG (shown in Figure 5) is mapped onto the *MAPDATA(A, Service1, Service3)* event node in the EDG (shown in Figure 6). This example shows a simple example of service execution, i.e. predecessor-successor relationships. More complicated control constructs, such as switch and loop, are also specified in FICAS to describe a variety of services [15].

```
Invocation1 = Service1.invoke()
Invocation2 = Service2.invoke()

A = Invocation1.extract();
B = Invocation2.extract();

Invocation3 = Service3.invoke(A, B)

C = Invocation3.extract();

Invocation4 = Service4.invoke(C)
D = Invocation4.extract();
```

Figure 4: Sample program segment for service integration

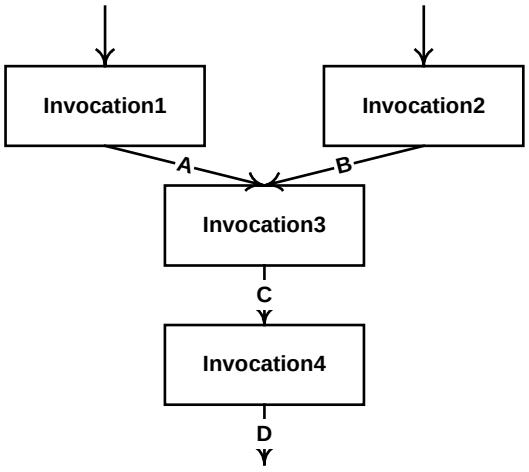

Figure 5: Sample DDG

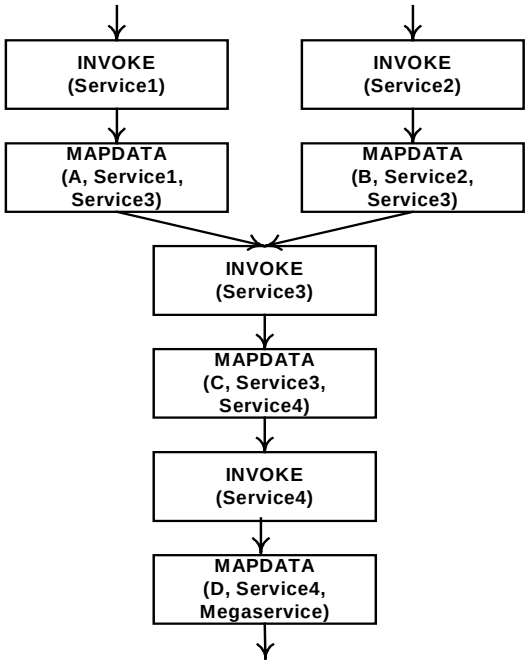

Figure 6: EDG with Distributed Data-flows

The presented scheduling algorithm is applicable to relatively static data flows, in which the volume of data flows can be estimated before the execution of web services. Major perturbations may require re-analysis. Similar control constructs have also been included in other service composition protocols. For example, DAML-S defines a set of control constructs that consists of Sequence, Split, Split + Join, Choice, Unordered, Condition, If-Then-Else, Iterate, Repeat-While, and Repeat-Until [2].

4. MOBILE CLASSES AND ACTIVE MEDIATION

While FICAS gains substantial performance from direct data communication among the services, it still requires the capability provided in mediator nodes to map incompatible sources or to integrate information from diverse sources. In the semantic web setting, where we expect many and diverse providers, we cannot expect that each service will deliver results that will be fully compatible and useful to further services that the composed application will need to invoke.

Active mediation applies the notion of mobile code [12] to support unforeseen remote information processing. Specifically, matching, reformatting, rearranging, and mapping of data being sent or received among services can be embodied in mobile code, and shipped by the composed application to the remote service as needed. Remote services that can accept active mediation now have the ability to adapt their behavior to the client requests. For instance, an information client can forward a compression routine to a service so that queried information is compressed before returned. A service that computes monthly taxes due state-by-state can have chronological transaction data rearranged and summarized as it needs. In general, with active mediation to provide client-specific functionalities, services can be viewed as if they were intended for the specific needs of the client.

4.1 MOBILE CLASSES

A mobile class is an information-processing module that can be dynamically loaded and executed. Conceptually, a mobile class is a function that takes some input data elements, performs certain operations, and then outputs a new data element. The underlying programming support of mobile class in FICAS is similar to that of the mobile agent technology [7, 26]. They both utilize executable programs that can migrate during execution from machine to machine in a heterogeneous network. Through a set of domain independent active and intelligent registries, called mobile agents, web services can be discovered for interoperation [24]. However, the mobile agents are self-governing in that they decide when and where to migrate on their own. On the other hand, the mobile class in FICAS is an integral part of the service composition framework. Since mobile classes are controlled by composed application, their management and deployment becomes easier. There are also efforts of applying mobile agent approach to lightweight process workflow [13, 25]. Such mobile code, which is used to carry out process-based dynamic adaptation, has been called worklet [13].

Mobile classes can be implemented in many general-purpose programming languages [8, 11, 18]. In our work, Java is chosen as the specification language for mobile classes. First, Java is suitable for specifying computational intensive tasks. There are many available standard libraries that provide a wide range of computational functionalities. Second, Java has extensive support for portability. Java programs can be executed on any platform that incorporates a Java virtual machine. Third, Java supports dynamic linking and loading. Java class files are object files rather than executables in the traditional sense. Linking is performed when the Java class files are loaded onto the Java virtual machine. Compiled into a Java class, the mobile class can be dynamically loaded at runtime.

Figure 7 presents the MobileClass interface, which contains a single function that represents the functionality of a mobile class. The *execute()* function takes a vector of data elements as the input and generates a data element as the output. The *execute()* function is overloaded by a mobile class to provide specific processing functionality. Figure 8 shows the definition of the DataElement class representing a data element, which is used for data exchange among services in FICAS. Since services and mobile classes use the same representation for data, data can be exchanged among the services and the mobile classes without any conversion. Internally, a data element is represented in XML. There are two constructors for DataElement, one for creating an empty data element, the other for creating a data element based on its XML representation. The class provides functions to query the type and the size of the data element. In the case that the data element is of a primitive type (i.e., Boolean, integer, real, or string), functions are provided to set, fetch and compare values for the data element. Otherwise, the content of the data element can be fetched as a byte array.

```
public interface MobileClass {
  public DataElement execute(Vector params);
}
```

Figure 7: Definition of the MobileClass Interface

```
public class DataElement {
  public DataElement();
  public DataElement(Document doc);

  // Return XML document representation
  Document doc();
  // Return byte array representation
  byte[] getByteArray();
  // Return a string in XML printout form
  String toString();

  // Return the size of the element
  int getSize();
  // Return the type of the element
  int getType();

  DataElement setValue(boolean value);
  DataElement setValue(double value);
  DataElement setValue(int value);
  DataElement setValue(java.lang.String value);
  DataElement setValue(byte[] arr);

  boolean getBooleanValue();
  int getIntValue();
  double getRealValue();
  String getStringValue();

  int compare(DataElement e);
  boolean eq(DataElement e);
  boolean ge(DataElement e);
  boolean gt(DataElement e);
  boolean le(DataElement e);
  boolean lt(DataElement e);
  boolean ne(DataElement e);
}
```

Figure 8: Definition of the DataElement Class

4.2 ACTIVE MEDIATION

To enable active mediation in FICAS, a composed application needs to be able to invoke mobile classes on a service, and the service needs to support the execution of the mobile classes. To allow a composed application to coordinate the invocation of mobile classes on services, we have introduced a Mobile Class event, which is sent from a composed application to a service to invoke a mobile class. A service supports the execution of mobile classes through the incorporation of an active mediator. Figure 9 illustrates the architecture of the active mediator:

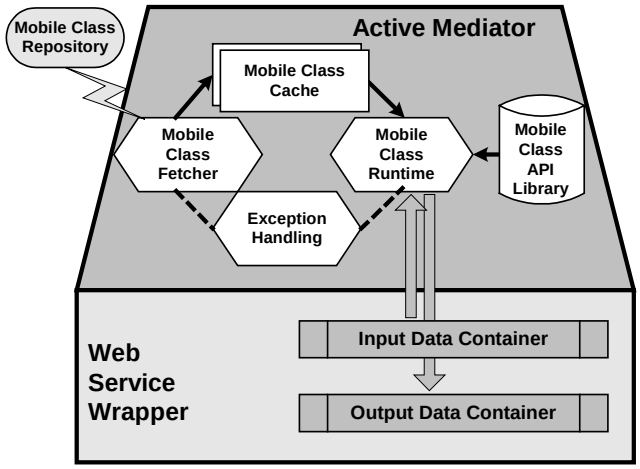

Figure 9: Architecture of active mediation

- The Mobile Class Fetcher is responsible for loading the mobile class. The source location of the Java class is specified by the mclass-name in the MCLASS event. The loaded Java class is stored into the Mobile Class Cache.
- The Mobile Class Cache is a temporary storage for mobile classes. The Mobile Class Cache is used to avoid the duplicate loading of a mobile class. It is looked up every time before any mobile class is loaded. The Mobile Class Fetcher is used to load the Java byte code only when the cache miss occurs.
- The Mobile Class Runtime is the execution engine for the mobile classes. To execute a mobile class, the Mobile Class Runtime loads the Java class from the Mobile Class Cache and invokes the *execute()* function. The input parameters of the *execute()* function are looked up from the Input Data Container. The result of the *execute()* function is put into the Output Data Container.
- The Mobile Class API Library stores the utility classes that make the construction of mobile classes more convenient. For instance, the Java Development Kit (JDK) library is provided as part of the Mobile Class API Library.
- The Exception Handling module provides error handling for the loading and the execution of the mobile class.

Upon receiving a Mobile Class event, a service directs the Mobile Class Fetcher to load the mobile class into the Mobile Class Cache. The *execute()* function of the mobile class is then invoked to process data local to the service. Since the service wrapper handles the interchange of the data among services, the active mediator is only concerned with the data processing that is local to the service.

4.3 PLACEMENT OF MOBILE CLASSES

The choice of which web service executes the mobile class affects how the data-flows are formed. The placement of the mobile class therefore has significant impact on the performance of the composed application. An example, shown in Figure 10, is used to demonstrate such impact. The application has two web services, S1 and S2, and one mobile class. The mobile class *FILTER* takes a large string as input, filters through the content, and returns a string that consists of every 10th character of the input string. We have three potential placement strategies, as shown in Figure 10:

- Strategy 1: By placing the mobile class FILTER at the web service that hosts the controller, we can construct the execution plan as shown in Figure 10(a). S1 generates the data element A and passes it to the mobile class for processing. The processed result B is then sent to S2 for further processing.
- Strategy 2: Placing the mobile class FILTER at S1 constructs the execution plan as shown in Figure 10(b). S1 generates the data element A and processes it locally using the mobile class. The result B is sent from S1 to S2 for further processing.
- Strategy 3: By placing the mobile class FILTER at S2, we can construct the execution plan as shown in Figure 10(c). S1 generates the data element A and passes it to S2. S2 processes A locally using the mobile class to generate the result B.

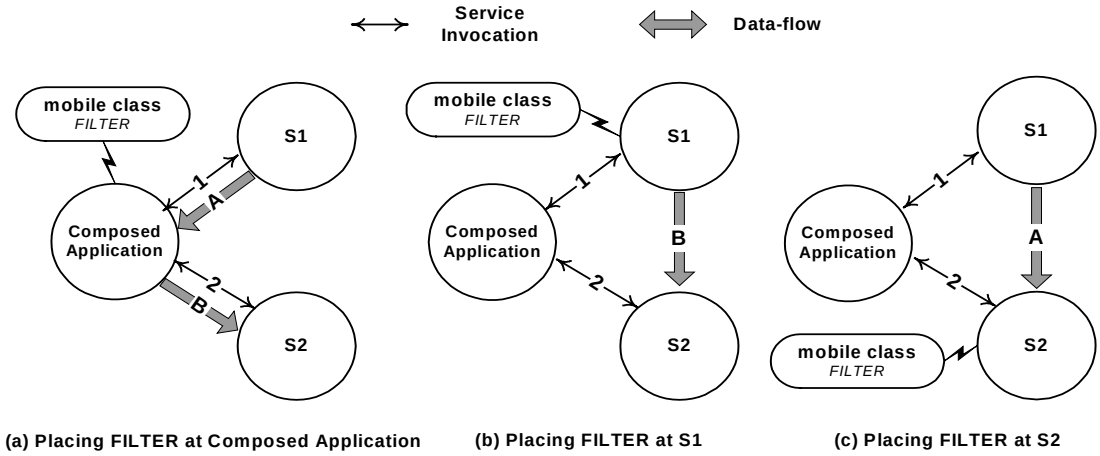

Figure 10: Execution plans with different placements for the mobile class

To compare the strategies, we assume that the performance of loading and executing the mobile class is the same on all web services. Strategy 1 requires both the input data element A and the output data element B to be transmitted among the composed application and the web services. Thus Strategy 1 incurs the most communication traffic compared to the other two strategies. Strategy 2 and Strategy 3 differ in the data sent between the web services. Since the data element B is one tenth the volume of data element A, Strategy 2 incurs the least amount of communication traffic and has the best performance.

The above observation can be generalized as the following algorithm to determine the optimal placement of a mobile class. The criterion used is minimizing the amount of data transmission among web services. For a mobile class, each input data element to the mobile class is represented as a pair, (S_i, V_i), where S_i is the web service that generates the ith input data element, and V_i is the volume of the data element. The output is a (S_0, V_0) pair, where S_0 is the destination web service to which the result of the mobile class will be sent, and V_0 is the size of the data element. Two observations can be made. First, the sum of V_i remains the same regardless where the mobile class is executed. Second, by placing the mobile class on the web service S_i, we can eliminate the corresponding data-flow volume V_i as the data element is local to the web service. Therefore, the optimal placement of the mobile class is the web service S_i that has the largest aggregated V_i.

Figure 11 shows the LDS (Largest Data Size) algorithm that selects the web service that generates and consumes the largest volume of data for a given mobile class. The algorithm first computes the total amount of data attributed to each unique web service. Then, the web service with the largest data volume is selected as S_{max}, which represents the optimal placement for the mobile class. The LDS algorithm is applicable when the input and output data sizes are known. For a situation where the output data size of a mobile class is only determined after the execution of the mobile class, we need to estimate the output data size. We view the output data size of a mobile class as a function of the input data sizes: $S_O = f(S_A, S_B, \ldots)$. The function f is called the sizing function of the mobile class, where S_O is the output data size and S_A, S_B are the input data sizes. The sizing function may be stored along with the Java byte codes in the mobile class repository. The controller can then use the sizing function to estimate the output data size for running the LDS algorithm.

The LDS algorithm assumes that the network links among all web services are of comparable performance. When this assumption does not hold, a more complicated model can be adopted to minimize the aggregated time. Various network parameters, such as topology of network and bandwidth of network channels, can have impact on the performance of the composed application, and other algorithms (for example, see [22]) can also be implemented in FICAS.

```
INPUT: input pairs(S1, V1), …, (Sn,Vn)
       output pair (S0, V0)
OUTPUT: Smax
METHOD:
   Vmax=0
   for every unique S in input and
   output pairs
      V=0
      for i=0,…,n
         if Si==S
            V=V+Vi
      if V>Vmax
         Smax=S
         Vmax=V
```

Figure 11: LDS Algorithm for Optimal Mobile Class Placement

5. EXAMPLE APPLICATIONS

Several examples are presented in this section to demonstrate that FICAS is well suited for integrating web services that exchange large amount of data and for solving interface incompatibility. The distribution of data-flows and active mediation can facilitate service composition and improve the performance of composed application.

5.1 TYPE MEDIATION

Data generated by a service can be directly used by other services if they share the same data types, structure, formats, and granularities, etc. However, such homogeneity cannot be assumed

within a large-scale service composition infrastructure. Data exist in various types and will continue to appear in different types that suit different applications. Many potential services never took the need for composition into account, assuming their results were what the consumer desired. Data type conversion is inevitable in supporting service composition.

Traditionally, a service serving as type broker or a distributed network of type brokers is used to mediate the difference among data in various formats [19]. The type brokers can convert data from one format to another acceptable format for the information client. The type brokers serve as proxies connecting client requests with appropriate source services. A type graph is used to figure out a chain of necessary conversions. There are two issues associated with using type brokers: efficiency and availability. First, the use of type brokers for type mediation can be inefficient. Large amount of data are forwarded to and from the brokers. The problem is exacerbated when a chain of conversions is involved. Second, the necessary type brokers may not exist for the desired data type. Since there are a large number of data formats, it is impractical to prepare a comprehensive set of type brokers covering all existing and future data types. New type brokers need to be created and maintained to conduct the type conversions needed by an application or a common intermediate format must be adopted, leading to a high conversion cost.

In our approach, mobile classes are used in place of type brokers to handle data conversion and integration of multiple input streams. The mobile classes are created by the application programmers as part of the specification for the composed applications. Rather than forwarding data among the type brokers, the composed application loads the mobile classes on the services to provide the type mediation functions. Similar to the chain of type brokers, multiple mobile classes for type mediation can be utilized together. Since the mobile classes are invoked on the source service, the multiple interim data transfers can be eliminated and the data traffic is limited to essential transmissions. The application of the mobile classes successfully addresses the efficiency and availability issues associated with type mediation.

5.2 RESULT EXTRACTION

Services can produce a wide variety of data suitable for extraction and reporting. A composed application has upstream services generating data that is consumed by downstream services. Selective result extraction is required when the output model of an upstream service is incompatible with the input model of a downstream service. For instance, an upstream service delivers an SQL cursor, but the downstream service expects a relation. In this example, an upstream service produces data progressively, while the downstream service consumes the data as a whole. Such a mismatch of how data is produced and consumed will stymie the downstream recipient. This is when mobile classes become valuable. A mobile class can be constructed to scroll the SQL cursor to fill a complete relation, and return the relation as the output. The mobile class is loaded onto the upstream service to mediate the output data for the downstream service. As the result, both services can collaborate despite the difference in how one generates data and how the other consumes the data.

Services are invariably built with the expected clients in mind. However, if their services are valuable, they will acquire more clients and be composed into more complex scenarios. Changing their interfaces would frustrate their original, intended clients. But ignoring new opportunities would cause the services to fall into disuse and be replaced by newer services with similar functionalities [29].

The difference in how an upstream service produces data and how a downstream service consumes data can present a seemingly insurmountable block to effective composition of services, particularly when the number of collaborating services and their applications grow. The use of active mediators, implemented through mobile classes, resolves the problems stemming from such incompatibilities.

5.3 ENGINEERING APPLICATION

FICAS is also applicable to integrate engineering services. In this section, we illustrate the implementation of a civil engineering information service infrastructure [16]. For a typical construction project scenario, different construction applications can reside at different locations, such as site offices or the company headquarters. Furthermore, these applications may have different interfaces and require different data formats. Project information is not shared and accessible among all project participants at all time. It is difficult and time-consuming for project managers on the construction sites to get the latest project information from the company headquarter and other places. If there are some changes on the construction site, it is also hard for project managers to evaluate the impacts of the changes on the whole project immediately. Project managers cannot reschedule the project on the construction site right away without the latest information. To improve the process, a ubiquitous environment is desirable, so that project participants can access and manage the latest project information from various engineering services, using software applications at different locations, including sites without computing services.

The infrastructure shown in Figure 12 has been developed using FICAS model. There are five types of clients and applications involved in the environment. Palm PDA devices are used to access project information via wireless modems, and web browsers provide project information to users who usually have access to high-speed Internet connections and more powerful computing devices. Three engineering software applications are included to manage the design and scheduling aspects of the project. An Oracle 8i relational database serves as the backbone information storage for this distributed service infrastructure. The active mediator acts as an intelligent bridge that connects various applications and devices with the database. It captures the client requests in the form of active objects from devices such as Palm and desktop browsers. Source queries are constructed and sent to the Oracle 8i database. The active mediator retrieves the information from the Oracle 8i database and conduct client-specific information processing by invoking the mobile classes incorporated in the client requests. The processed information with desired abstraction and suitable format is returned to the clients for displaying.

In this application, the engineering software applications all have different proprietary data models for describing project schedule information. By applying a common data model and utilizing FICAS, different applications are able to communicate with each other. The data converters are implemented as mobile classes, which act as bridges to map between the propriety data models and the relational data model, enabling the Oracle 8i database to serve as the backbone information store [16].

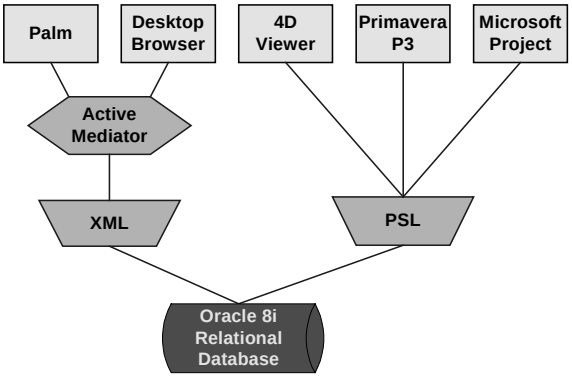

Figure 12: Integration of engineering services

6. CONCLUSIONS

This paper investigates the integration of services that communicate large volumes of data. Traditionally, a composed application is the central hub for all the data traffic, while each web services process data supplied by the composed application. This centralized data flow is shown to be inefficient when data are substantial. To improve effective use, the distributed data flow approach is introduced which allows direct data exchange among the web services.

The FICAS architecture is defined to enable smooth adoption of distributed data-flow and active mediation in services composition. Active mediation increases the customizability and flexibility of web services. Specifically, it enables interoperation of web services without requiring that heterogeneous data be transmitted via central nodes. It utilizes code mobility to facilitate dynamic information processing in service composition. Delegating the maintenance of software that has not been written by oneself is an important benefit of the services model [29]. Active mediation allows data-processing tasks to be specified for composed applications, at the same time separating computation from composition. Through some application scenarios, we presented the effectiveness and flexibility of active mediation in facilitating service composition.

An algorithm for planning distributed data-flows is presented. It enables the construction of a web services invocation sequence to minimize aggregate network traffic. A second algorithm that determines the optimal placement of mobile classes is introduced, and the applicability of the algorithm is discussed. Used appropriately, active mediation will greatly facilitate service composition, both in functionality and in performance. In the example engineering applications, the controlling node can run on a low bandwidth device and thus has tremendous effects on performance. Typically, mobile devices are attractive to manage complex scenarios in dealing with governmental regulation [14], engineering [16], healthcare [4], and military situations [21]. In those cases the benefits of distributed dataflow, enabled by active mediation, are striking.

7. ACKNOWLEDGMENTS

This work is partially sponsored by the Center for Integrated Facility Engineering at Stanford University, the Air Force (Grants F49620-97-1-0339 and F30602-00-2-0594), and the Product Engineering Program headed by Dr. Ram D. Sriram at NIST (National Institute of Standards and Technology). The authors would also like to acknowledge an equipment grant from Intel Corporation for the support of this research.

8. REFERENCES

[1] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., and Weerawarana, S. BPEL4WS Specification: Business Process Execution Language for Web Services Version 1.1, 2003, http://www-106.ibm.com/developerworks/library/ws-bpel/.

[2] Ankolekar, A., Burstein, M., Hobbs, J.R., Lassila, O., Martin, D.L., McIlraith, S.A., Narayanan, S., Paolucci, M., Payne, T., Sycara, K., and Zeng, H. DAML-S: Semantic Markup for Web Services. In Proceedings of the International Semantic Web Working Symposium (Stanford, CA, 2001), 2001.

[3] Banerji, A., Bartolini, C., Beringer, D., Chopella, V., Govindarajan, K., Karp, A., Kuno, H., Lemon, M., Pogossiants, G., Sharma, S., and Williams, S. Web Services Conversation Language (WSCL) 1.0, 2002, http://www.w3.org/TR/2002/NOTE-wscl10-20020314/.

[4] Berners-Lee, T., Hendler, J., and Lassila, O. The Semantic Web. Scientific American, 284, 5 (2001), 34-43.

[5] Blake, M.B. Coordinating Multiple Agents for Workflow-Oriented Process Orchestration. Information Systems and e-Business Management, 1, (2003), 1-18.

[6] Boehm, B., and Scherlis, B. Megaprogramming. In Proceedings of DARPA Software Technology Conference (Los Angeles, CA, 1992), 1992, 68-82.

[7] Brewington, B., Gray, R., Moizumi, K., Kotz, D., Cybenko, G., and Rus, D. Mobile Agents in Distributed Information Retrieval. in Klusch, M. ed. Intelligent Information Agents, Springer-Verlag, 1999, 355-395.

[8] Cardelli, L. A Language with Distributed Scope. Computing Systems, 8, 1 (1995), 27-59.

[9] Ceri, S., Pernici, B., and Wiederhold, G. Distributed Database Design Methodologies. Proceedings of the IEEE, 75, 5 (1987), 533-546.

[10] Curbera, F., Khalaf, R., Mukhi, N., Tai, S., and Weerawarana, S. The Next Step in Web Services. Communications of the ACM, 46, 10 (2003), 29-34.

[11] Douglis, F., and Ousterhout, J. Transparent Process Migration: Design alternative and the Sprite Implementation. Software: Practice and Experience, 21, 8 (1991), 757-785.

[12] Fuggetta, A., Picco, G.P., and Vigna, G. Understanding Code Mobility. IEEE Transactions on Software Engineering, 24, 5 (1998), 342-361.

[13] Kaiser, G., Stone, A., and Dossick, S. A Mobile Agent Approach to Lightweight Process Workflow. In Proceedings of International Process Technology Workshop (Villard de Lans, France, 1999), 1999.

[14] Lau, G.T., Kerrigan, S., Law, K.H., and Wiederhold, G. An E-Government Information Architecture for Regulation Analysis and Compliance Assistance. In Proceedings of

ICEC'04: Sixth International Conference on Electronic Commerce (Delft, The Netherlands, 2004), 2004.

[15] Liu, D. A Distributed Data Flow Model for Composing Software Services. Ph.D. Thesis, Stanford University, Stanford, CA, 2003.

[16] Liu, D., Cheng, J., Law, K.H., Wiederhold, G., and Sriram, R.D. Engineering Information Service Infrastructure for Ubiquitous Computing. Journal of Computing in Civil Engineering, 17, 4 (2003), 219-229.

[17] Liu, D., Sample, N., Peng, J., Law, K.H., and Wiederhold, G. Active Mediation Technology for Service Composition. In Proceedings of Workshop on Component-Based Business Information Systems Engineering (CBBISE'03) (Geneva, Switzerland, 2003), 2003.

[18] Moizumi, K. Implementing Distributed Services with Mobile Code: The Case of the Messenger Environment. In Proceedings of the IASTED International Conference on Parallel and Distributed Systems (Austria, 1998), 1998.

[19] Ockerbloom, J. Mediating Among Diverse Data Formats. Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, PA, 1998.

[20] Sample, N., Keyani, P., and Wiederhold, G. Scheduling Under Uncertainty: Planning for the Ubiquitous Grid. In Proceedings of 5th International Conference on Coordination Models and Languages (Coord2002) (York, UK, 2002), 2002.

[21] Sheng, S., Chandrakasan, A., and Brodersen, R.W. A Portable Multimedia Terminal. IEEE Communications Magazine, 30, 12 (1992), 64-75.

[22] Sheth, A., Cardoso, J., Miller, J., Kochut, K., and Kang, M. QoS for Service-Oriented Middleware. In Proceedings of the 2002 Conference on Systemics, Cybernetics and Informatics (SCI'02) (Orlando, FL, 2002), 2002.

[23] Sheth, A.P., Singhal, A., and Liu, M.T. An Analysis of the Effect of Network Parameters on the Performance of Distributed Database Systems. IEEE Transactions on Software Engineering, 11, 10 (1985), 1174-1184.

[24] Sycara, K. Multi-Agent Infrastructure, Agent Discovery, Middle Agents for Web Services and Interoperation. in Mutli-Agents Systems and Applications, Springer-Verlag New York, Inc., New York, USA, 2001, 17 - 49.

[25] Valetto, G., Kaiser, G., and Kc, G.S. A Mobile Agent Approach to Process-based Dynamic Adaptation of Complex Software Systems. In Proceedings of the 8th European Workshop on Software Process Technology (Vienna, Austria, 2001), 2001, 102-116.

[26] White, J.E. Mobile Agents. in Bradshaw, J.M. ed. Software Agent, MIT Press, 1997, 437-472.

[27] Wiederhold, G. Mediators in the Architecture of Future Information Systems. IEEE Computer, (1992), 38-49.

[28] Wiederhold, G. Information Systems that also Project into the Future. In Proceedings of Databases in Networked Information Systems (DNIS 2002) (Aizu, Japan, 2002), 2002, 1-14.

[29] Wiederhold, G. The Product Flow Model. In Proceedings of 15th Conference on Software Engineering and Knowledge Engineering (SEKE) (Skokie, IL, 2003), 2003, 183-186.

[30] Wiederhold, G., and Genesereth, M. The Conceptual Basis for Mediation Services. IEEE Expert, Intelligent Systems and Their Applications, 12, 5 (1997), 38-47.

[31] Wiederhold, G., Wegner, P., and Ceri, S. Towards Megaprogramming: A Paradigm for Component-Based Programming. Communications of the ACM, 35, 11 (1992), 89-99.

[32] Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., and Sheng, Q.Z. Quality Driven Web Services Composition. In Proceedings of the Twelfth International World Wide Web Conference (WWW2003) (Budapest, Hungary, 2003), 2003, 411-421.