Chapter 2

Data and Application Integration

Figures
In the rest of this section we discuss in more details this class of data integration requirements generally occurring in mashup development. For this reason, in particular sources and, every time a query is posed to the integrated system, the data integration system is characterized by "transient" data integration tasks, im-

Typical architectures for materialized and virtual data integration [3].

Fig. 2.1 Typical architectures for materialized and virtual data integration [3].
Fig. 2.2 Example of an integrated database storing shipment data extracted from different data sources. Each data source is characterized by a local schema. Data integration is performed according to a virtual global schema managed by the mediator.
(a) GAV Mapping for the global relation BILLING. The global relation is defined as a view on the local source relations.

(b) LAV Mapping for Source2 and Source3. The local source relations are defined as views over the global relations.

Fig. 2.3 Example of GAV and LAV schema mappings for the integrated order DB.
Fig. 2.4 The basic architecture of a distributed system and application.
Data and Application Integration

Operations may be plugged in the form of external, custom application logic, providing for extensibility. The configuration and operation of the whole message broker can be managed via a dedicated administration interface. Examples of message brokers are IBM’s WebSphere Message Broker and today’s enterprise service buses (ESBs).

2.4.2 Workflow management systems

Alonso et al. correctly point out that the key contribution of message brokers to application integration is their capability to hide the heterogeneity and distribution of the information systems to integrate and to provide a uniform view on their data and functionalities.

Workflow management systems (WfMSs) provide support for the last ingredient, i.e., the definition of integration logic. The initial application area of WfMSs was office automation, whose purpose is to automate the coordination of repetitive, administrative processes involving multiple human actors and electronic documents. Soon, it became evident that the automation support provided by WfMSs could be leveraged to coordinate also software systems, not only human actors. Nowadays, WfMSs (or business process management systems, as they are called more prominently today) are very flexible systems that enable the seamless coordination of software and human actors, typically starting from a graphical model that expresses the necessary integration logic. Most systems also provide support for advanced coordination features, such as transactions, exception handling, recovery and compensation, events, deadline management, and similar.

![Workflow Model](image)

**Fig. 2.5** A simplified workflow model of a possible integration logic for the logistics application integration scenario. The model shows how the integration logic coordinates the interaction with existing information systems.
2.4 Application Integration

ness processes and agile applications that span organizations and platforms caused a re-thinking of some of the traditional development practices and the emergence of a new paradigm. This new paradigm is called service-oriented computing (SOC), and it is a computing paradigm that uses Web services as building blocks for the engineering of composite, distributed applications out of the reusable application logic encapsulated by Web services.

SOC builds on two ingredients: an ecosystem of readily reusable Web services and a development paradigm based on composition as core abstraction. We discuss both in the following.

2.4.4.1 Service-oriented architecture (SOA)
The architecture of the Web service ecosystem is commonly called a service-oriented architecture (SOA), which is a logical architecture for the design of software systems that provide services to either end-user applications or to other services distributed in a network, via published and discoverable interfaces.

The SOA can be articulated into three roles and two artifacts (see Figure 2.6): A web service (the first artifact) is implemented and made publicly accessible by a so-called provider (or service provider). In order to advertise the web service and enable its potential clients to be aware of the existence of the service, the provider publishes a service descriptor (the second artifact), which describes the purpose and features of the service, where to access it (a URI), and how to access it (e.g., using which communication protocol). This advertisement is supported by a dedicated registry, which aggregates service descriptors from multiple providers and allows potential consumers to search for descriptors. If a consumer wants to use some external functionality, it can query the registry for suitable web services, obtaining as response a service descriptor (or a set thereof). Following the instructions on how to invoke the service contained in the retrieved descriptor, the consumer can bind to the concrete service and use it.

Fig. 2.6 The service-oriented architecture (SOA) with its roles and artifacts.
Fig. 2.7 The distributed computing environment enabled by web services as an instance of the generic architecture of distributed systems (see Figure 2.4).
The header of the process definition, among others, specifies the languages used to query XML data and define expressions (e.g., used by the `<condition> and <while>` activities).

Extensions enable the definition of new attributes, new elements, operation extensions, etc.

Enables declaring dependencies on external XML Schema or WSDL definitions.

Partner links model relationships between partner processes for peer-to-peer communications. Partner links have a type and an endpoint reference.

Message exchanges associate outbound messages with inbound messages.

Variables hold messages or data and capture the state of the process.

Correlation sets are sets of properties shared by all messages of a conversation.

Fault handlers allow the definition of fault handling activities.

Event handlers enable reacting to external events or alarms.

Activities specify the process logic.

**Fig. 2.8** The basic structure of a service composition (a process) in BPEL [163].
2.5 Cloud Computing comes for the cost of the resources consumed by the virtualization itself. Yet, practice has shown that the benefits largely outweigh this additional effort.

2.5.2 Cloud architectures

In line with the above virtualization strategies, cloud architectures can be structured into different logical layers, each providing a different type of resource as a service and, hence, different abstractions. We illustrate these layers in Figure 2.9 and discuss the three bottom-most layers, i.e., the technological ones (for an explanation of the HaaS layer, we refer the reader to [31] or [149]):

- **Infrastructure as a Service (IaaS)**: IaaS provides computing infrastructure as a service, that is, it provides an abstract view on hardware (e.g., computers, CPUs, storage devices, memory, etc.) and allows the user to run their own images of operating systems on top of the virtualized hardware. The management interface provided by IaaS providers typically allows their users to dynamically define the required hardware characteristics and to start and stop operating system instances (e.g., to save money, as running an instance costs money). The target user of IaaS is the system administrator.

- **Platform as a Service (PaaS)**: PaaS usually provides development environments or runtime environments for software development and software application services.

- **Software as a Service (SaaS)**: Applications

**Fig. 2.9** A simplified cloud architecture stack with IaaS, PaaS, SaaS, and HaaS (adapted from [172]).