INTRODUCTION

One of the driving factors in IT development is the availability of cheap and efficient network technologies. The Internet is no longer used only as a medium for personal communication. Organizations start utilizing this technology to link existing applications and to create new ones. While traditionally systems were designed to respond to interactive user requests, they are now more and more aiming at autonomous, distributed data processing. Systems are connected and instantly react to changes to improve their functionality and utility (cf. zero latency enterprises). Mobile systems and other volatile configurations demand reactions to continuous changes; finance applications must be notified of price fluctuations; supply chain management must observe stock level changes; and information retrieval applications must forward new content (Banavar, Chandra, Strom, & Sturman, 1999; Gray, 2004).

The focus when dealing with the delivery of data and services is changing, moving from a stationary world to one that is in a state of flux. Traditionally, data and services have been viewed as being stationary in a collection of objects or databases, with inquiries directed to them in a request/reply mode of interaction. This concept has led to client/server architectures that emphasize explicit delegation of functionality, where processes access remote functionality to accomplish their own goal. Remote procedure calls (RPC) and derivative techniques are classic examples (Birrell & Nelson, 1984; Mullender, 1993); even the incipient Web services mainly rely on sending requests with the Simple Object Access Protocol (SOAP; Alonso, Casati, Kuno, & Machiraju, 2003). These techniques deliberately draw from a successful history of engineering experience, their principles are well understood, and they have been an appropriate choice for many well-defined problems.

In the context of dynamic or large-scale applications, however, request/reply has serious restrictions. The direct and often synchronous communication between clients and servers enforces a tight coupling of the communicating parties and impairs scalability (Franklin & Zdonik, 1998). Clients poll remote data sources, and they have to trade resource usage for data accuracy, especially in chains of dependent servers. Unnecessary requests due to short polling intervals waste resources, whereas long intervals increase update latency. The obvious need for asynchronous and decoupled operation has led to various extensions of existing middleware standards. For instance, CORBA and Java 2 Enterprise Edition (J2EE) were extended with asynchronous invocation methods and notification services (Object Management Group [OMG], 1999; Schmidt & Vinoski, 1999; Sun Microsystems, 2002). Database research, software engineering, and coordination theory corroborate the advantages of loosely coupled interaction (Cilia, Bornhövd, & Buchmann, 2001; Papadopoulos & Arbab, 1998; Sullivan & Notkin, 1992). The following presents a classification of communication paradigms and technologies to outline their fundamental characteristics.

COMMUNICATION PARADIGMS

Irrespective of the actual technology used for communication, different modes of interaction can be distinguished that determine the way interdependencies between processes are established. Four models are differentiated by two dimensions (see Table 1; Fiege, Mühl, & Gärtner, 2003). The first attribute, initiator, describes whether the consumer or the provider of data initiates the interaction, where the former depends on data or functionality provided by the latter. The second attribute, addressing, distinguishes whether the addressee of the interaction is known or unknown, i.e., whether peer(s) are addressed directly or indirectly. Any interaction between a set of processes can be classified according to these models. Even though interaction may show more nuances in practice, the models are complete in the sense that they essentially cover all major paradigms.

Request/reply is the most widely used interaction model. Any kind of remote procedure call or client/server interaction belongs to this class. The initiator is the consumer (i.e., client) that requests data and/or function-

<table>
<thead>
<tr>
<th>Initiator</th>
<th>Consumer</th>
<th>Provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>Request/Reply</td>
<td>Callback</td>
</tr>
<tr>
<td>Indirect</td>
<td>Anonymous Request/Reply</td>
<td>Event-based</td>
</tr>
</tbody>
</table>
ality from the provider (i.e., server). It expects data to be delivered back or relies on a specific task to be done. The provider is directly addressed, its identity is known, and the caller is able to incorporate information about the callee into his own state and processing, resulting in a tight coupling of the cooperating entities. Replies are mandatory in this model unless the system and failure model excludes errors.

Anonymous request/reply does not specify the provider that should process the request. Instead, requests are delivered to an arbitrary, possibly dynamically determined set of providers. The consumer does not know the identity of the recipient(s) a priori, yet it expects at least one reply. This model is eligible when multiple providers are available that are selected at runtime by the communication subsystem, e.g., according to the specific parameters contained in a request or according to load balancing policies.

In the callback mode consumers register at a specific, known provider their interest to be notified whenever an observed condition becomes true. The provider repeatedly evaluates the condition and if necessary calls back the registered process. This kind of interaction is employed in the well-known observer design pattern (Gamma, Helm, Johnson, & Vlissides, 1995). The provider is responsible for maintaining a list of registered consumers. If multiple callback providers are of interest, a consumer must register at each of them separately. The identity of peers is known and must be managed by the application.

The event-based interaction model has characteristics inverse to the request/reply model. The initiator of communication is the provider of data, that is, the producer of notifications. Notifications are not addressed to any specific set of recipients; consumers rather issue subscriptions that describe what kind of data they are interested in. A notification is delivered to a consumer if it matches one of its subscriptions. Providers are not aware of consumers and vice versa. An intermediary messaging service conveys the notifications. The essential characteristic of this model is that without knowing consumers of notifications, producers send information only about their own state, precluding any assumptions on consumer functionality. The overall functionality is not encoded in the processes but determined implicitly by their composition and interaction.

Request/reply and event-based interaction are characterized by the simplicity of the former and the flexibility of the latter. Request/reply is easy to handle, implement, and understand. It corresponds to the imperative nature of the client/server paradigm and of common programming languages. However, this model has three principal drawbacks:

1. Architecture lock.
2. Point-to-point communication limits scalability.
3. Polling limits accuracy in automated computations.

The architecture of the system is locked up in application code, and changes to an existing configuration are difficult to deploy at runtime (Garlan & Shaw, 1993; Papadopoulos & Arbab, 1998). The explicit point-to-point communication relies on the presence of specific servers and requires that peers are online simultaneously. This is a severe restriction considering the increasing interconnection of IT systems which reside in different organizational domains. Obviously, bandwidth consumption is another limiting factor of point-to-point communication. Finally, when replacing interactive with automated data processing, computation becomes information-driven. It is no longer initiated on request, but when new input data is available.

Databases are affected in two ways. First, data stores are no longer the important central aspect of future architectures. Traditional database functionality fulfills helper functions. Second, and more importantly, database research contributes to the ongoing work on data dissemination systems.

**TECHNIQUES**

Generally, there is no best implementation technique for a certain model of interaction. The technique must be chosen in view of the deployment environment, the required quality of service, and the need for flexibility and scalability. A classification of data dissemination techniques was presented by Franklin and Zdonik in 1998 (see Figure 1). They distinguish client pull vs. server push, periodic vs. aperiodic, and unicast vs. 1-to-N (multicast) delivery. These alternatives may even be mixed in one

![Figure 1. Classification of data delivery](image-url)