Peer-to-Peer for Collaborative Applications

Gianpaolo Cugola and Gian Pietro Picco
Dipartimento di Elettronica e Informazione—Politecnico di Milano
Piazza Leonardo da Vinci, 32—20133 Milano, Italy
{cugola, picco}@elet.polimi.it

Abstract

Peer-to-peer systems recently captured the attention of practitioners and researchers as they provide an attractive alternative to client-server architectures. Peer-to-peer enables the creation of massively distributed networks of data repositories that can be setup and discarded easily according to applicative needs. Nevertheless, the current popularity of these systems is due mostly to their use for file sharing over the Internet.

In this paper, we argue that the advantages of a peer-to-peer architecture reach well beyond the realm of Internet file sharing. In particular, they become key in supporting enterprise processes and especially collaborative work involving mobile users. To support our view, in this paper we report about the design of an architecture and a core communication middleware in the EU project MOTION, where a peer-to-peer architecture is exploited to support collaboration among mobile and geographically distributed users.

1 Introduction and Motivation

The technological advances in computing and networking that characterized recent years determined an increasing use of information technology for enabling cooperative work among the members of a team. Computer supported cooperative work (CSCW) has become a prominent research area, dealing with issues spanning from coordination protocols and ergonomy of user interface down to middleware and architectures supporting distributed cooperation. This latter aspect is the one we focus on in this paper.

Client-server vs. peer-to-peer collaboration. Nowadays, teamwork is often supported using commonplace tools, like e-mail and Web-based tools, which serve the basic need of enabling communication. In some cases, more integrated applications like Microsoft Exchange and Lotus Notes, or special-purpose CSCW tools are employed, which typically provide users with a shared workspace that contains also the information and documents relevant to the task at hand. By and large, however, all these tools exploit a rigid client-server architecture, relying on a server for enabling communication and sharing application data.

On the other hand, collaboration is intrinsically peer-to-peer in nature. In our everyday life, human interaction is such that the members of a team typically interact directly with each other, without the need of an intermediary—like a server. Moreover, each member typically carries along the documents relevant for discussion, without the need of a centralized repository, that is instead typically exploited only for stable artifacts.

Pros and cons of peer-to-peer. These differences result in an "architectural mismatch" between the external view provided by the application and its internal software architecture. The effect of such a mismatch is to closely couple the interaction with the enabling architectural element, i.e., the server. The resulting drawback is a lack of flexibility in carrying out the interactions, which must all be funneled through the server. Thus, for instance, the only way to make a document available to other parties is by uploading it on the workspace server. These limitation becomes even more evident when mobility becomes part of the picture. People need to communicate and collaborate even while in movement, and independently from the place where they are. Sometimes, they need to cooperate by setting up an impromptu meeting by using only their computing devices and even in absence of a fixed networking infrastructure, as enabled by wireless ad hoc networking. Nevertheless, in this situations server access is often prevented by technical or administrative barriers.

In this paper, we argue that a peer-to-peer approach holds significant advantages over traditional client-

server architectures, by fostering an architecture that maps more naturally on the application requirements, and that is intrinsically more flexible and tolerant to reconfiguration.

When a peer-to-peer architecture is adopted, information and services are no longer gathered in a single point of accumulation. Instead, they are spread across all the nodes of the distributed system. Thus, for instance, users directly host the resources they want to share with others, with no need to publish them on some server. In a sense, the server becomes now collectively represented by the whole set of available peers.

Often, peer-to-peer architectures bring this notion to an extreme by fostering a location transparent perspective and dropping the conventional assumption that users know a priori which node is hosting the service desired. A peer-to-peer interaction is then associated with a mechanism that enables users to query the whole space of information provided by the currently connected peers, and find those with the relevant information or service.

The uncoupling of the users' clients from the server supporting interaction better suits the requirements coming from mobility, in that it frees clients from the coupling with servers, and at the same time allows an arbitrarily large and continuously changing set of nodes to be accessed at once.

Hence, not only these architectural characteristics, i.e., the absence of a centralized server, its replacement through the illusion of a single, global access point to the system, and the fluid configuration of the peer network, map more naturally onto the application view of a network of collaborating people. They also provide an architectural framework that is intrinsically more akin to scalability and reconfiguration. Interestingly, these features are relevant not only in mobile scenarios but also in fixed ones, where peers can be transparently added and removed as required by the current business priorities. Moreover, the decentralized nature of a peer-to-peer architecture encompasses more naturally than centralized ones the case of multisite or multicompany projects, where the cooperation infrastructure must span administrative boundaries, and is subject to security concerns.

Cooperation vs. Internet file sharing. At this point, it is natural to ask what is the relationship between what discussed so far and the many applications (e.g., Napster, Gnutella, and Freenet to name a few) developed in the last years that exploit a peer-to-peer architecture. Unfortunately, it turns out that these applications start from premises that are rather different from those outlined thus far. They target the Internet and aim at providing peer-to-peer computing over millions of nodes, with file sharing as their main application concern. By simplifying a little, we could say that what most of these applications essentially provide is a new generation of file transfer (FTP) service. The difference in perspective from the domain of collaborative work is made evident by their search capabilities, that typically do not guarantee to capture information about all matching files. In the domain of Internet file sharing, it is reasonable to adopt this best-effort approach, that may return an empty result set even if some of the searched files actually exist in the system. However, this is usually not acceptable in an enterprise environment, where completeness of searches is usually a key requirement.

Peer-to-peer is no silver bullet. As a further consideration, we may observe that currently available peer-to-peer applications bring peer-to-peer to an extreme, where the logical network of peer is totally fluid: every peer plays exactly the same role of the others, and none of them can be assumed to be fixed and contributing to the definition of a permanent infrastructure. This radical view, albeit intellectually stimulating, prevent access to the resources exported by non-connected peers. This is not acceptable in the enterprise domain we consider, where critical data is often required to be always available, independently from its owner. Moreover, the enterprise environment is much more structured than the Internet one. Leveraging this structure whenever possible is likely to provide improvements and optimizations.

The MOTION approach. Based on all these considerations, in the MOTION project we developed an innovative platform for collaborative applications, which adopts a peer-to-peer architecture specifically geared towards the enterprise domain. The next section describes the architectural principles defining our peer-to-peer approach, while Section 3 gives a concise overview of the middleware support that embodies these principles and that lies at the core of the platform.

2 Architectural Principles

The peer-to-peer approach taken in MOTION is inspired by previous work on the LIME [3] coordination model, and its generalization into the notion of *global virtual data structure* (GVDS) [2]. The notion of GVDS is a meta-model of communication, originally conceived for mobile environments, centered around the idea of enabling coordination among mobile units through a

data space that is transiently shared and dynamically built out of the data spaces provided by each unit in range.

This data space is global because it includes all the data contained in all the local data spaces of the connected units, and it is virtual since it does not exist physically as a single entity, like the local ones. Instead, it is an "illusion" dynamically generated by the underlying middleware upon invocation of the local access primitives, and according to changes in connectivity. Hence, the GVDS paradigm naturally enables a *context-aware* style of coordination where, in every moment, the context is described by the content of the GVDS, which reflects the state of accessible units.

The architecture of the MOTION platform enriches the notion of GVDS with some abstractions appropriate to the application domain, that deal mostly with users, data, and a way to group them according to some relationship, as described in the following.

Abstractions. The execution of an instance of the MOTION platform involves a set of individuals, which constitute what we call a *virtual population.* A *virtual community* is instead a subset of a virtual population, as defined by some membership relation. Typically, this relation is the interest in a given topic. A given individual can be a member of zero, one, or more virtual communities. At a given point in time, only a subset of the virtual community can be reached directly through network connectivity. We call this subset a *connected community.*

Thus, for instance, the set of people working at company ACME constitute a virtual population, while the set of people working at ACME and interested in the design of the latest model of toaster constitute a virtual community. The connected community is the one a given individual running a MOTION client can access, based on network connectivity. If no connectivity is available to a member, she will have access to a community temporarily containing only herself.

Communities define a scoping mechanism that can be leveraged off for grouping not only people, but also data. The *resource space* is the set of artifacts (e.g., documents) owned by each individual. It provides a minimal set of primitives needed to store, retrieve, and query the elements of the set. The owner of a resource space has access to all of its content. Instead, the *member space* is the subset of documents contained in an individual's resource space that pertain to a given community. An individual can be member of multiple communities, and she can choose which documents pertain to each community. The same document may be made available to several communities. This means that member spaces can have intersections. A member space can be seen as a "community view" on an individual's resource space. Finally, the *community space* is the union of all the member spaces belonging to the members of a community, and is effectively a GVDS spanning several nodes.

According to the notion of GVDS, the community space defines a transient data space, where information is provided only by the connected members. Nevertheless, the ability to communicate and share artifacts asynchronously was a key requirement for the development of the MOTION platform, due to its intended use in an enterprise environment. Supporting this requirement implies relying on the persistence of artifacts, independent from changes in connectivity.

For this reason, we associate to each community space a *community cabinet*, whose contents are not subjected to transient sharing, rather they are persistent. Since it is part of the community space, the cabinet is available to all community members. Nevertheless, the content of the cabinet is permanently available rather than transiently built and dynamically changing, i.e., the content is available to community members independently of whether the individual owning a given resource in the cabinet is currently connected. At this level of abstraction, we are not concerned about *how* cabinets will be implemented. In principle, several alternatives exist, with various degrees of decentralization (and complexity). We will detail this issue in the next section.

Architecture overview. The conceptual model described thus far is implemented using a peer-to-peer architecture whose main components are described in Figure 1.

At the top, the presentation layer takes care of visualizing the service interface to the user in a way that is adapted to the device the user is currently using. While the presentation layer may vary according to the device, the underlying business logic is coded only once, and contained in the TeamWork services. Services can be general-purpose (i.e., needed by users in any situation, like searching and browsing community spaces) or application-specific (e.g., those developed for the MOTION end-users). Moreover, services can be installed and removed dynamically using service configuration facilities. All services rely on a set of TeamWork Service Components, providing the building blocks for service programmers.

Below the TeamWork service layer, the Advanced Communication Middleware provides the core communication facilities needed to retrieve and disseminate information in a peer-to-peer network. The next sec-

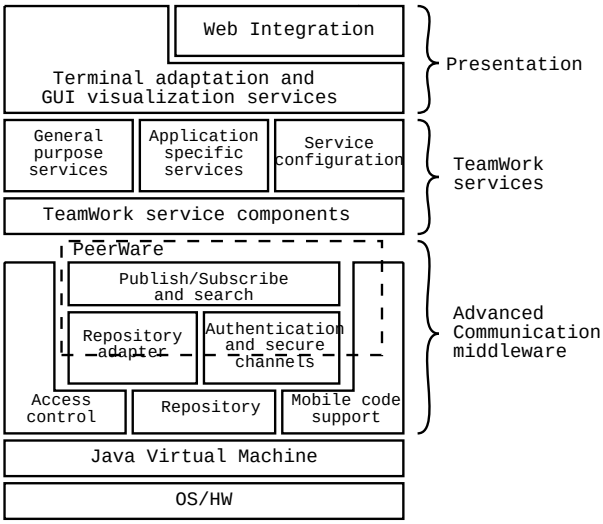

Figure 1: The architecture of the MOTION platform.

tion illustrates its key features.

3 Middleware Support

The Advanced Communication Middleware (ACM) provides the core communication facilities the MOTION platform builds upon. These facilities enable a peer[1] to make information contained in its repository available to other peers in the system and, at the same time, perform distributed queries towards other repositories, subscribe to events and disseminate the corresponding event notifications efficiently, push information from a source to a number of peers, and exploit mobile code to relocate application code across the peer network—all in a secure way.

Middleware Components. The data sitting on a machine and owned by a peer is stored in the *Repository*. These data are made available to the rest of the system through the main component of the ACM, the *Publish/Subscribe and Search* module. This module is actually implemented by PEERWARE [1], a peer-to-peer middleware that enables the GVDS notion by providing access to the union of all the repositories of all the connected peers.

The data structure provided by PEERWARE is a hierarchy of nodes containing documents, where a document may actually be accessible from multiple nodes,

[1]For simplicity, we assume a one-to-one mapping between users and peers, and use the two interchangeably. In the MOTION architecture there are ways to accommodate more complex schemes.

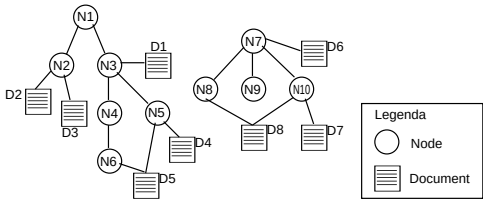

Figure 2: The data structure provided by PEERWARE.

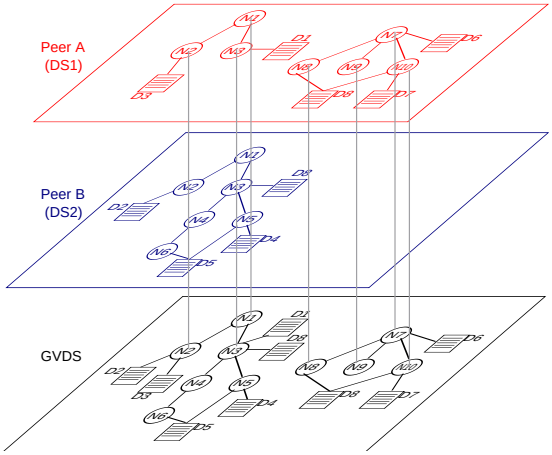

Figure 3: Building the GVDS in PEERWARE.

as shown in Figure 2. Hence, this data structure is similar in structure to the directory tree of a Unix file system. When a peer is isolated, it is given access only to its own tree. However, when connectivity with other peers is established, the peer has access to the virtual tree constructed by superimposing the trees contributed by all the peers in the system, as illustrated by Figure 3. This data structure is naturally exploited in MOTION to map the notion of community onto the hierarchy nodes.

By exploiting the primitives provided by PEERWARE, peers can query the global repository as if it was local, as well as subscribe for events occurring in the repository and receive the corresponding notifications. The hierarchy provides a natural mechanism to provide scoping, thus leading to an efficient implementation of searches, and to the definition of a security protection domain.

Security is clearly a relevant issue in the scenario we are targetting, where a user is empowered with the ability to query a global space of information at once. Security concerns are addressed in the ACM by two separate modules. The *Authentication and Secure Channels* module provides mechanisms for ensuring privacy and integrity of communication, by establishing encrypted

channels, and for handling the security information necessary to authenticate a peer. Essentially, this module creates a secure communication path among peers, so that access to the GVDS can be limited based on the identity of the peer requesting it. The actual security policy that determines the capabilities of a given peer is embedded by the *Access Control* module. It intercepts requests of operations on the Repository and checks whether they are valid according to the access control table and to the security information associated with the peer and evaluated during authentication, e.g., certificates.

It is important to note that the functionality provided by the security modules and by the repository are sharply decoupled from the specific implementation provided for these functionalities. Thus, the security protocols, as well as the format of the security information used to perform authentication, and the repository effectively used can be changed easily, e.g., to adapt them to the common practice of a specific business environment.

Finally, the *Mobile Code Support* module provides the ability to relocate dynamically application code across the various peers. This capability, albeit available directly to the TeamWork services, is also used directly by PeerWare to enable the dissemination of application-specific ways to access a remote repository, e.g., by providing application-dependent filtering capabilities.

Deployment and Prototype. As mentioned in Section 1, in collaborative applications there is a tension between the need of contributing documents only as long as the owner is involved in an interaction, and the need for maintaining other documents permanently available, independently from the set of users currently connected. This requirement is already captured by the architectural principles described in Section 2, by introducing the concept of cabinet. At the level of the run-time architecture, the same requirement led to the distinction between a set of permanently available *backbone peers*, and a fringe of *mobile peers* which are allowed to connect and disconnect as required. Given the way PeerWare operates and routes requests for data and notifications to events, all these peers are connected to form a tree in which the mobile peers represent the leaves, as shown in Figure 4.

Observe that, to keep intact the advantages of a peer-to-peer architecture in terms of flexibility, the distinction between backbone and mobile peers is only a deployment one. In other terms, both run the same MOTION components, which allows them to locally store data and to freely access the data shared by each peer

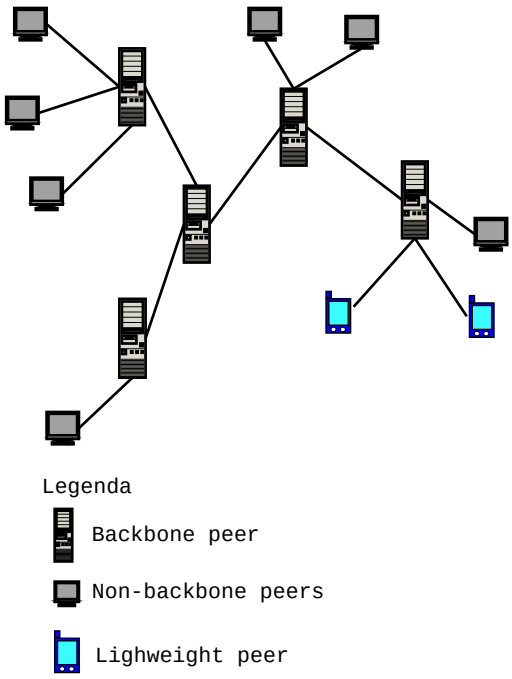

Figure 4: The run-time architecture of the MOTION platform.

and the services realized by the MOTION platform. In particular, backbone peers may act as cabinets for one or more communities, thus allowing data to be permanently accessible independent from the set of currently connected peers.

A third set of peers is that of lightweight peers. They either run a lightweight version of the MOTION components, which does not include the repository (e.g., in case of PDAs), or access the MOTION platform through a standard Web browser. In both cases they connect to one of the backbone peers, which act as an access point to the MOTION platform.

4 Conclusions

In this paper, we argued that peer-to-peer architectures, currently made popular by file sharing over the Internet, may bring considerable advantages when exploited for supporting collaboration among geographically distributed and mobile users. Collaboration defines a scenario where interaction is intrinsically peer-to-peer; exploiting a peer-to-peer architecture reduces the architectural mismatch between the application and its implementation. Moreover, the flexibility and scalability properties of this architecture opens up several opportunities, and become a natural choice when

mobility is part of the requirements.

This paper provided a concise overview of the middleware at the core of the MOTION platform. Preliminary experience with the platform and the middleware confirm the usability of the prototype and the relevance of the benefits. Further experimentation, including the delivery of a industrial-strength prototype and its deployment at the premises of the end-users involved in the MOTION project, will give us the opportunity to validate our ideas on the field.

Acknowledgments

We thank the members of the MOTION consortium for providing the context where the ideas described in this paper were born, discussed, and put in practice.

References

[1] G. Cugola and G.P. Picco. PEERWARE: Core Middleware Support for Peer-To-Peer and Mobile System. Technical report, Politecnico di Milano, November 2001. Submitted for publication. Available at `www.elet.polimi.it/~picco`.

[2] A.L. Murphy. *Enabling the Rapid Development of Dependable Applications in the Mobile Environment.* PhD thesis, Washington University in St. Louis, MO, USA, August 2000.

[3] A.L. Murphy, G.P. Picco, and G.-C. Roman. LIME: A Middleware for Physical and Logical Mobility. In *Proc. of the 21st Int. Conf. on Distributed Computing Systems (ICDCS-21)*, May 2001. To appear.