45

Одноранговые сети

В ЭТОЙ ГЛАВЕ...

- Обзор технологии P2P
- Платформа Microsoft Windows Peer-to-Peer Networking с ее важными компонентами PNRP и PNM
- Построение приложений P2P с помощью .NET Framework

Технология организации одноранговых сетей (peer-to-peer networking), часто называемая технологией P2P, является одной из самых полезных и при этом часто неправильно понимаемых среди средств, появившихся в последние несколько лет. Когда люди думают о P2P, им на ум, как правило, приходит лишь одна вещь: возможность обмена музыкальными файлами, зачастую незаконным образом. Это связано с тем, что приложения для обмена файлами наподобие BitTorrent стали очень популярными, а в них для работы используется именно технология P2P.

Однако, хотя технология P2P применяется в приложениях для обмена файлами, это вовсе не означает, что она не может использоваться в других приложениях. На самом деле, как будет показано в главе, эта технология может применяться в целом ряде других приложений, и она становится все более и более важной в современном мире повсеместных коммуникаций. В этом можно будет убедиться в первой части настоящей главы при кратком рассмотрении технологии P2P.

В Microsoft тоже не обошли стороной появление технологии P2P и стали разрабатывать собственные инструменты и средства для ее применения. Так появилась платформа Microsoft Windows Peer-to-Peer Networking, исполняющая роль своего рода каркаса для коммуникаций в приложениях P2P. В состав этой платформы входят такие важные компоненты, как PNRP (Peer Name Resolution Protocol — протокол преобразования имен членов) и PNM (People Near Me — соседние пользователи). Кроме того, в версию .NET Framework 3.5 было включено новое пространство имен `System.Net.PeerToPeer` и несколько новых типов и средств, позволяющих создавать приложения P2P с минимальными усилиями.

Обзор технологии P2P

Технология P2P представляет собой альтернативный подход к организации сетевых коммуникаций. Для того чтобы понять, чем P2P отличается от "стандартного" подхода к обеспечению коммуникаций, не помешает сделать шаг назад и вспомнить, что собой представляет связь типа "клиент-сервер". Коммуникации такого типа очень часто применяется в современных сетевых приложениях.

Архитектура типа "клиент-сервер"

Традиционно взаимодействие с приложениями по сети (в том числе Интернет) организуется с использованием архитектуры типа "клиент-сервер". Прекрасным примером могут служить веб-сайты. При просмотре веб-сайта происходит отправка по Интернет соответствующего запроса веб-серверу, который затем возвращает требуемую информацию. Если необходимо загрузить какой-то файл, это делается напрямую с веб-сервера.

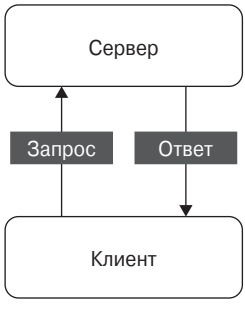

Рис. 45.1. *Простой вариант архитектуры "клиент-сервер"*

Аналогично, настольные приложения, имеющие возможность подключения к локальной или глобальной сети, обычно устанавливают соединение с каким-то одним сервером, например, сервером баз данных или сервером, предоставляющим набор служб.

На рис. 45.1 показан простой вариант архитектуры типа "клиент-сервер".

Ничего по сути неправильного в такой архитектуре нет, и на самом деле во многих случаях она будет оказываться именно тем, что нужно. Однако ей присуща проблема с масштабируемостью. На рис. 45.2 показано, как она будет масштабироваться при добавлении дополнительных клиентов.

С добавлением каждого клиента нагрузка на сервер, который должен взаимодействовать с каждым клиентом, будет увеличиваться. Если снова взять пример с веб-сайтом, то такое увеличение

нагрузки может стать причиной выхода веб-сайта из строя. При слишком большом трафике сервер просто перестанет реагировать на запросы.

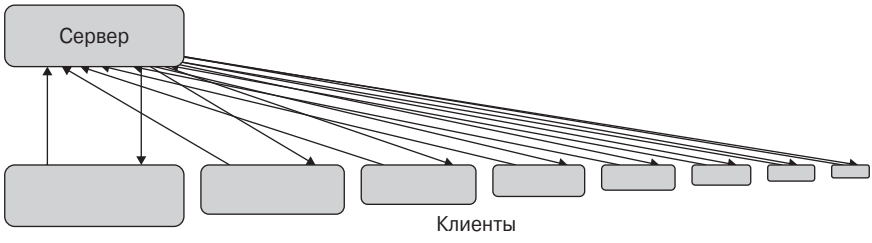

Рис. 45.2. *Масштабирование архитектуры "клиент-сервер" при добавлении дополнительных клиентов*

Конечно, существуют варианты масштабирования, с помощью которых можно смягчить подобную ситуацию. Один из них предусматривает масштабирование "вверх" за счет увеличения мощи и ресурсов сервера, а другой — масштабирование "вширь" путем добавления дополнительных серверов. Первый способ, естественно, ограничивается доступными технологиями и стоимостью более мощного оборудования. Второй способ потенциально более гибкий, но требует добавления дополнительного уровня в инфраструктуру для обеспечения клиентов возможностью либо взаимодействовать с отдельными серверами, либо поддерживать состояние сеанса независимо от сервера, с которым осуществляется взаимодействие. Для этого доступна масса решений, таких как продукты, позволяющие создавать веб-фермы или фермы серверов.

Архитектура типа P2P

Одноранговый (peer-to-peer) подход полностью отличается от подхода с масштабированием "вверх" или "вширь". В случае применения P2P вместо того, чтобы сосредоточить усилия на попытках улучшить коммуникации между сервером и его клиентами, все внимание уделяется поиску способов, которыми клиенты могут взаимодействовать между собой. Давайте для примера представим, что веб-сайтом, с которым взаимодействуют клиенты, является `www.wrox.com`, а издательство Wrox объявило о выходе новой версии данной книги на этом сайте и предоставлении его для бесплатной загрузки всем желающим, но лишь на протяжении одного дня. Не трудно догадаться, что при таком положении дел накануне появления книги веб-сайт начнет просматривать масса людей, которые будут постоянно обновлять его содержимое в своих браузерах и ожидать появления файла. Как только файл станет доступным, все они одновременно начнут пытаться загрузить его и, скорее всего, веб-сервер, который обслуживает веб-сайт `wrox.com`, не выдержит такого натиска и выйдет из строя.

Чтобы предотвратить выход веб-сервера из строя, можно воспользоваться технологией P2P. Вместо отправки файла прямо с сервера сразу всем клиентам он может быть отправлен только определенному числу клиентов. Несколько остальных клиентов могут далее загрузить его у тех клиентов, у которых он уже есть. После этого еще несколько клиентов могут загрузить его у клиентов, получивших его вторыми, и т.д. По сути, этот процесс может происходить даже быстрее благодаря разбиению файла на куски и распределению этих кусков среди клиентов, одни из которых будут загружать их прямо с сервера, а другие — из других клиентов. Именно так и работают технологии файлообменных систем вроде BitTorrent, как показано на рис. 45.3.

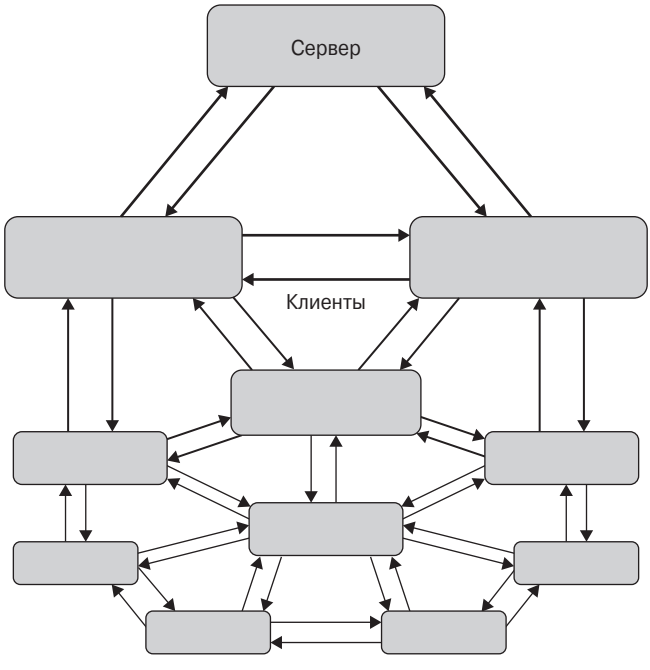

Рис. 45.3. *Пример архитектуры P2P*

Особенности архитектуры P2P

Тем не менее, в описанной здесь архитектуре обмена файлами все равно остались кое-какие проблемы, которые должны быть решены. Для начала, каким образом клиенты узнают о том, что существуют другие клиенты, и как они будут обнаруживать фрагменты файла, которые, возможно, имеются у других клиентов? Кроме того, каким образом гарантировать оптимальное взаимодействие между клиентами, если их могут отделять друг от друга континенты?

Каждый клиент, участвующий в работе сетевого приложения P2P, для преодоления этих проблем должен быть способен выполнять следующие операции:

- *обнаруживать* других клиентов;
- *подключаться* к другим клиентам;
- *взаимодействовать* с другими клиентами.

В том, что касается способности обнаруживать других клиентов, возможны два очевидных решения: поддержка списка клиентов на сервере, чтобы клиенты могли получать его и связываться с другими клиентами (называемыми *peers* — равноправными участниками), либо использование инфраструктуры (например, PNRP, о которой речь пойдет в следующем разделе), которая позволяет клиентам обнаруживать друг друга напрямую. В большинстве файлообменных систем применяется решение с поддержкой списка на сервере и используются серверы, называемые “трекерами” (trackers). В файлообменных системах в роли сервера может также выступать и любой клиент, как показано на рис. 45.3, объявляя, что у него имеется доступный файл, и регистрируя его на сервере-трекере. На самом деле в чистой сети P2P вообще не нужны никакие серверы, а лишь равноправные участники.

Проблема подключения к другим клиентам является более тонкой и распространяется на всю структуру используемой приложением P2P сети. При наличии одной группы клиентов, в которой все должны иметь возможность взаимодействовать друг с другом, топология соединений между этими клиентами может приобретать чрезвычайно сложный вид. Зачастую производительность удается улучшать за счет создания нескольких групп клиентов с возможностью установки подключения между клиентами в каждой из них, но не с клиентами в других группах. В случае создания этих групп по принципу локальности можно добиться дополнительного повышения производительности, поскольку в таком случае клиенты получают возможность взаимодействовать друг с другом по более коротким (с меньшим числом прыжков) сетевым путям между машинами.

Способность взаимодействовать с другими клиентами, пожалуй, не так важна, поскольку существуют хорошо зарекомендовавшие себя протоколы вроде TCP/IP, которые вполне могут применяться и здесь. Конечно, допускается привносить свои улучшения, как в высокоуровневые технологии (например, использовать службы WCF, получая в распоряжение все предлагаемые ими функциональные возможности), так и в низкоуровневые протоколы (например, применять протоколы многоадресной рассылки и тем самым обеспечивать отправку данных во множество конечных точек одновременно).

Обеспечение клиентов возможностью обнаруживать, подключаться и взаимодействовать друг с другом играет центральную роль в любой реализации P2P. Реализация, рассматриваемая в настоящей главе, предполагает применение типов `System.Net.PeerToPeer` вместе с PNM для обеспечения возможности обнаружения других клиентов и PNRP для обеспечения возможности подключения к другим клиентам. Как будет показано в последующих разделах, эти технологии позволяют обеспечивать возможность выполнения всех трех необходимых операций.

Терминология P2P

В предыдущих разделах уже было представлено понятие *равноправного участника* (peer) — именно так называют клиентов в сети P2P. Слово "клиент" в сети P2P не имеет никакого смысла, потому что здесь нет обязательного сервера, клиентом которого нужно быть.

Группы равноправных участников, которые соединяются друг с другом, называются *ячейками* (meshes), *облаками* (clouds) или *графами* (graphs). Каждая отдельная группа считается *хорошо соединенной*, если соблюдено хотя бы какое-то одно из следующих условий.

- Между каждой парой равноправных участников существует путь соединения, позволяющий каждому участнику подключаться к другому равноправному участнику требуемым образом.
- Между каждой парой равноправных участников существует относительно небольшое количество соединений, по которым они могут связываться.
- Удаление одного равноправного участника из группы не лишает остальных равноправных участников возможности взаимодействия друг с другом.

Обратите внимание, что это вовсе не означает, что каждый равноправный участник должен обязательно иметь возможность подключаться к каждому другому равноправному участнику напрямую. На самом деле, если проанализировать сеть с математической точки зрения, то можно обнаружить, что для соблюдения упомянутых выше условий равноправным участникам необходимо иметь возможность подключаться к относительно небольшому количеству других равноправных участников.

Еще одним понятием в технологии P2P, о котором следует знать, является *волновое распространение* (flooding). Под волновым распространением подразумевается способ, которым один фрагмент данных может передаваться по сети всем равноправным участникам и которым может производиться опрос других узлов в сети для обнаружения конкретного

фрагмента данных. В неструктурированных сетях P2P этот процесс протекает довольно произвольно; при этом сначала устанавливается связь с ближайшими соседними равноправными участниками, которые затем, в свою очередь, связываются со своими ближайшими соседями, и т.д. до тех пор, пока не будет охвачен каждый равноправный участник в сети. Также допускается создавать и структурированные сети P2P с четко определенными путями, по которым должно происходить распространение запросов и данных среди равноправных участников.

Решения P2P

При наличии подходящей инфраструктуры для P2P можно начинать разрабатывать не просто улучшенные версии клиент-серверных приложений, но и совершенно новые приложения. Технология P2P особенно подходит для приложений следующих классов:

- приложения, предназначенные для распространения содержимого, в том числе упоминавшиеся ранее приложения обмена файлами;
- приложения, предназначенные для совместной работы, такие как приложения, которые позволяющие открывать общий доступ к рабочему столу и "белой доске" (whiteboard);
- приложения, предназначенные для обеспечения многопользовательской связи и позволяющие пользователям общаться и обмениваться данными напрямую, а не через сервер;
- приложения, предназначенные для распределения обработки, как альтернатива приложениям для суперкомпьютеров, которые обрабатывают огромные объемы данных;
- приложения Web 2.0, объединяющие в себе некоторые или все перечисленные выше приложения и превращающие их в динамические веб-приложения следующего поколения.

Платформа Microsoft Windows Peer-to-Peer Networking

Платформа Microsoft Windows Peer-to-Peer Networking (Платформа для создания одноранговых сетей Windows производства Microsoft) представляет собой предлагаемую Microsoft реализацию технологии P2P. Она поставляется в составе операционных систем Windows XP SP2, Windows Vista и Windows 7, а также доступна в виде дополнения для Windows XP SP1. Она включает в себя две технологии, которые можно использовать для создания приложений P2P в .NET:

- протокол Peer Name Resolution Protocol (PNRP), который можно применять для публикации и преобразования адресов равноправных участников сети;
- сервер People Near Me (Соседние пользователи), который можно применять для обнаружения локальных равноправных участников (и который в настоящее время доступен только в Windows Vista и Windows 7).

Обе эти технологии более подробно рассматриваются далее в разделе.

Протокол PNRP

Естественно, для реализации приложения P2P можно использовать любой имеющийся в распоряжении протокол, но при работе в среде Microsoft Windows имеет смысл хотя бы рассмотреть вариант применения протокола PNRP. К настоящему времени успело вый-

ти две версии этого протокола. Первая версия предлагалась в составе Windows XP SP2, Windows XP Professional x64 Edition и Windows XP SP1 с пакетом Advanced Networking Pack for Windows XP. Вторая версия была выпущена вместе Windows Vista и сделана доступной для пользователей Windows XP SP2 в виде отдельного загружаемого компонента (см. KB920342 по адресу `support.microsoft.com/kb/920342`). Вторая версия PNRP также предлагается в Windows 7. Первая и вторая версии PNRP не совместимы между собой, и потому в настоящей главе рассматривается только вторая версия.

Сам по себе протокол PNRP не предоставляет всего, что необходимо для создания приложения P2P. Скорее, он является лишь одной из основополагающих технологий, которые применяются для преобразования адресов равноправных участников сети P2P. Протокол PNRP позволяет клиенту регистрировать конечную точку (называемую *именем равноправного участника* (peer name)), которая автоматически распространяется между остальными равноправными участниками в группе (облаке). Это имя инкапсулируется в идентификатор PNRP (PNRP ID). Любой равноправный участник, который обнаруживает идентификатор PNRP, может использовать PNRP для его преобразования в имя самого равноправного участника и затем начинать взаимодействовать с соответствующим клиентом напрямую.

Например, можно определить имя равноправного участника, представляющее конечную точку службы WCF, и воспользоваться PNRP для его регистрации в группе (облаке) в виде идентификатора PNRP ID. Равноправный участник, выполняющий подходящее клиентское приложение, которое применяет механизм обнаружения, способный распознавать имена равноправных участников для предоставляемой службы, тогда сможет обнаружить этот идентификатор PNRP ID. После обнаружения участник с помощью протокола PNRP установит местонахождение конечной точки службы WCF и начнет пользоваться этой службой.

Важным моментом является то, что PNRP не делает никаких предположений относительно того, что на самом деле скрывается за именем равноправного участника. Право решать, как использовать это имя после обнаружения оставляется за самими равноправными участниками. Информация, которую равноправный участник получает от PNRP при преобразовании идентификатора PNRP, включает в себя адрес IPv6, а также обычно и адрес IPv4 участника, который опубликовал этот идентификатор, вместе с номером порта и, необязательно, небольшим количеством дополнительных данных. Если равноправный участник не знает, что означает имя другого равноправного участника, он вряд ли сможет сделать с этой информацией что-нибудь полезное.

Идентификаторы PNRP

Идентификаторы PNRP ID являются 256-битными значениями. Младшие 128 бит используются для обозначения уникальным образом индивидуального равноправного участника сети, а старшие 128 бит – для представления его имени. Старшие 128 бит представляют собой хеш-комбинацию, состоящую из хешированного значения открытого ключа, которое получается от осуществляющего публикацию равноправного участника, и строки длиной до 149 символов, которая представляет имя этого участника. Хешированный открытый ключ (называемый *авторитетным источником* (authority)) в сочетании с этой строкой (называемой *классификатором* (classifier)) называются идентификатором P2P. Вместо хешированного открытого ключа также может использоваться значение 0, в случае чего имя равноправного участника считается незащищенным (если применяется открытый ключ, то имя считается защищенным).

На рис. 45.4 схематично показана структура идентификатора PNRP.

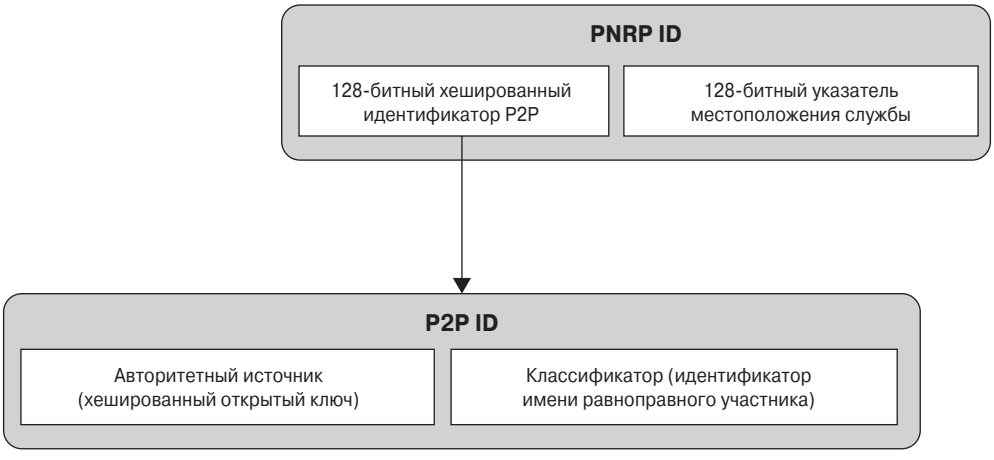

Рис. 45.4. *Структура идентификатора PNRP*

Служба PNRP на стороне равноправного участника отвечает за поддержание списка идентификаторов PNRP, как тех, которые публикует сама, так и тех, которые она получает в виде кэшированного списка от экземпляров служб PNRP, находящихся в других местах облака. Когда равноправный участник пытается преобразовать идентификатор PNRP, служба PNRP либо использует кэшированную копию конечной точки для выяснения адреса того равноправного узла, который опубликовал данный идентификатор PNRP, либо спрашивает его соседей, не могут ли они сделать это. В конечном итоге с узлом, опубликовавшим идентификатор PNRP, устанавливается соединение, и служба PNRP получает возможность преобразовать его.

Обратите внимание, что все это не требует участия администратора. Все, что понадобится сделать — позаботиться о том, чтобы равноправные участники знали, что следует делать с именами после их преобразования с помощью своей локальной службы PNRP.

Равноправные участники сети могут применять PNRP для нахождения идентификаторов PNRP, совпадающих с определенным идентификатором P2P. Эту возможность можно использовать для реализации простейшего механизма, позволяющего обнаруживать незащищенные имена равноправных участников. Дело в том, что в случае отображения несколькими равноправными участниками незащищенного имени с одинаковым классификатором, их идентификатор P2P ID будет совпадать. Конечно, из-за того, что любой равноправный участник может использовать незащищенное имя, нет никакой гарантии, что конечная точка, с которой будет устанавливаться соединение, будет именно той, которая ожидается, поэтому такое решение подходит лишь для реализации механизма обнаружения по локальной сети.

Облака PNRP

Выше было рассказано, как PNRP осуществляет регистрацию и преобразование имен равноправных участников в облаке (группе). Облако (или группа) поддерживается *seed-сервером*, которым может быть любой сервер с запущенной службой PNRP, которая поддерживает запись хотя бы об одном равноправном участнике. Для службы PNRP доступны облака двух типов.

- **Облака локальных соединений (Link local).** В такие облака входят компьютеры с подключением к локальной сети. Каждый ПК может подключаться к более чем одному облаку такого типа при условии наличия у него нескольких сетевых адаптеров.

- **Глобальные облака (Global).** В такие облака по умолчанию входят компьютеры с подключением к Интернету, хотя также можно определять приватное глобальное облако. Разница состоит в том, что для глобального облака с подключением к Интернету Microsoft поддерживает seed-сервер, а для определяемого самостоятельно приватного глобального облака должен использоваться собственный seed-сервер. В последнем случае необходимо позаботиться о том, чтобы все равноправные участники могли подключаться к нему, настроив соответствующим образом параметры политики.

В прошлых выпусках PNRP существовали облака еще и третьего типа, которые назывались облаками локальных сайтов (Site local). Они больше уже не применяются и потому в главе не рассматриваются.

Для выяснения, в какие облака входит данный компьютер, служит следующая команда:

```
netsh p2p pnrp cloud show list
```

На рис. 45.5 показано, как обычно выглядит результат запуска этой команды.

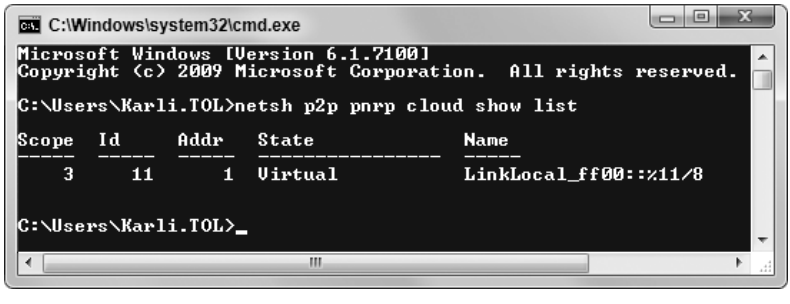

Рис. 45.5. *Типичный вывод команды* `netsh`

В данном случае результат указывает, что доступно всего лишь одно облако локальных соединений (Link local). Понять это можно как по значению в столбце `Name` (Имя), так и по значению в столбце `Scope` (Область), в котором для облаков локальных соединений всегда отображается значение `3`, а для глобальных облаков — значение `1`. Для подключения к глобальному облаку необходимо иметь глобальный адрес IPv6. Компьютер, на котором запускалась команда `netsh` с результатами, показанными рис. 45.5, таковым не обладал, потому для него оказалось доступным только локальное облако.

Облака могут находиться в одном из перечисленных ниже состояний.

- `Active` (Активное). Если облако находится в состоянии `Active`, оно может применяться для публикации и преобразования имен равноправных участников.
- `Alone` (Одинокое). Если к облаку, из которого запрашивается равноправный участник, больше никаких равноправных участников не подключено, оно будет находиться в состоянии `Alone`.
- `No Net` (Нет сети). Если равноправный участник потерял соединение с сетью, состояние облака может измениться с `Active` на `No Net`.
- `Synchronizing` (Синхронизация). Облака будут переходить в состояние `Synchronizing` при подключении к ним какого-то равноправного участника. Это состояние будет сменяться другим очень быстро, поскольку подключение не занимает много времени, поэтому увидеть облако в таком состоянии удается редко.

- `Virtual` (Виртуальное). Служба PNRP подключается к облакам только при возникновении потребности в регистрации или преобразовании имени какого-то равноправного участника. Если соединение с облаком находится в неактивном состоянии на протяжении более 15 минут, облако может перейти в состояние `Virtual`.

При наличии проблем с сетевой связью следует проверить, не препятствует ли брандмауэр передаче локального сетевого трафика через порты UDP 3540 или 1900. UDP-порт 3540 используется протоколом PNRP, а UDP-порт 1900 — протоколом SSDP (Simple Service Discovery Protocol — простой протокол обнаружения служб), который, в свою очередь, используется службой PNRP (а также устройствами UPnP).

Особенности PNRP в Windows 7

В Windows 7 протокол PNRP предусматривает использование нового компонента под названием *DRT* (Distributed Routing Table — таблица распределенной маршрутизации). Этот компонент отвечает за определение структуры используемых PNRP ключей, роль которой в реализации по умолчанию исполняет описанный выше идентификатор PNRP. С помощью API-интерфейса DRT можно определить альтернативную схему ключей при условии, что они представляют собой 256-битные целочисленные значения (подобно идентификаторам PNRP ID). Это означает возможность использования любой схемы, но при этом, разумеется, нужно самостоятельно заботиться о генерации и защите ключей. За счет применения этого компонента можно создавать совершенно новые топологии облаков, выходящие за рамки действия PNRP и, следовательно, за рамки контекста настоящей главы, поскольку этот прием относится к числу сложных.

В Windows 7 также появилась новая опция для подключения к другим пользователям в приложении Remote Assistance, которая называется Easy Connect. Эта опция использует PNRP для обнаружения пользователей, к которым необходимо подключиться. После создания сеанса с помощью Easy Connect или других средств (например, отправки приглашения по электронной почте) пользователи могут открыть общий доступ к своим рабочим столам и оказывать помощь друг другу через интерфейс Remote Assistance.

Служба People Near Me

Служба PNRP, как было показано в предыдущем разделе, применяется для определения местонахождения равноправных участников сети P2P. Очевидно, что она является важной действующей технологией при продумывании процесса обнаружения, подключения и взаимодействия в приложении P2P, но сама по себе она ни одного из этих этапов полностью не реализует. В реализации этапа обнаружения ей помогает служба People Near Me (Соседние пользователи), которая позволяет находить равноправных участников, которые зарегистрировались в локальной сети (т.е. подключены к тому же локальному облаку, что и данный компьютер).

Эта служба наверняка уже попадалась на глаза, поскольку она является встроенным компонентом Windows Vista и Windows 7, а также используется в приложении Windows Meeting Space, которое позволяет открывать равноправным участникам общий доступ к различным приложениям. Для конфигурирования этой службы применяется элемент Change People Near Me (Изменить соседних пользователей) панели управления, быстро перейти к которому можно путем ввода в поле для поиска внутри меню Start (Пуск) слова `people` (“соседние”). Этот элемент позволяет получить доступ к диалоговому окну, показанному на рис. 45.6.

После регистрации эта служба становится доступной в любом приложении, которое построено с учетом возможности ее использования.

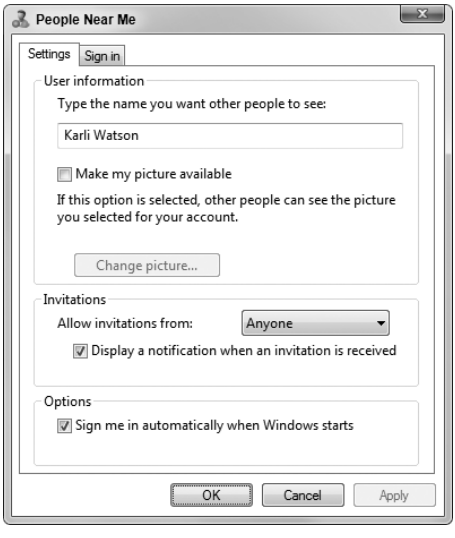

Рис. 45.6. *Конфигурирование службы People Near Me*

На момент написания книги служба PNM было доступна только в семействе операционных систем Windows Vista и Windows 7. Возможно, в будущем появятся пакеты обновлений или дополнительные загружаемые модули, которые позволят сделать ее доступной в Windows XP.

Создание приложений P2P

Теперь, когда известно, что собой представляет технология P2P, и какие технологии доступны разработчикам приложений .NET для реализации приложений P2P, пришла пора посмотреть, как создавать такие приложения. Из приведенного выше материала понятно, что в любом приложении PNRP должен применяться протокол PNRP для публикации, распространения и преобразования имен равноправных участников. Поэтому в первую очередь здесь будет показано, как достичь этого в .NET, а затем — каким образом использовать службу PNM в качестве каркаса для приложения P2P. Последнее может быть выгодно, поскольку в случае применения PNM реализовать собственные механизмы обнаружения не понадобится.

Перед изучением этих тем сначала необходимо ознакомится с классами, которые предлагаются в следующих пространствах имен:

- `System.Net.PeerToPeer`
- `System.Net.PeerToPeer.Collaboration`

Для работы с этими классами нужно обязательно ссылаться на сборку `System.Net.dll`.

Пространство имен `System.Net.PeerToPeer`

Классы в пространстве имен `System.Net.PeerToPeer` инкапсулируют API-интерфейс для PNRP и позволяют взаимодействовать со службой PNRP. Их можно применять для решения двух основных задач:

- регистрация имен равноправных участников;
- преобразование имен равноправных участников.

В последующих разделах все упоминаемые типы относятся к пространству имен `System.Net.PeerToPeer`, если только явно не указано другое.

Регистрация имен равноправных участников

Для регистрации имени равноправного участника потребуется выполнить перечисленные ниже шаги.

1. Создайте защищенное или незащищенное имя равноправного участника с определенным классификатором.
2. Настройте процесс регистрации этого имени, предоставив следующие сведения.
   - Номер порта TCP.
   - Облако или облака, в которых должно быть зарегистрировано имя участника (если не указано, PNRP будет регистрировать имя равноправного участника во всех доступных облаках).
   - Комментарий длиной до 39 символов.
   - Дополнительные данные объемом до 4096 байт.
   - Информация о том, должны ли для имени равноправного участники автоматически генерироваться конечные точки (поведение по умолчанию, при котором конечные точки автоматически генерируются на основе IP-адреса или адресов равноправного члена и номера порта, если таковой указан).
   - Коллекция конечных точек.
3. Зарегистрируйте имя равноправного участника в локальной службе PNRP.

После выполнения третьего шага имя равноправного члена становится доступным для всех соседей в выбранном облаке (или облаках). Регистрация равноправного члена продолжается то тех пор, пока не будет прекращена явным образом, или до тех пор, пока не будет завершен процесс, зарегистрировавший имя равноправного участника.

Для создания имени равноправного участника применяется класс `PeerName`. Экземпляр этого класса создается из строкового представления идентификатора P2P ID в форме `авторитетный_источник.классификатор` либо из строки классификатора и типа `PeerNameType`. Можно использовать как тип `PeerNameType.Secured`, так и тип `PeerNameType.Unsecured`. Например:

```
PeerName pn = new PeerName("Peer classifier", PeerNameType.Secured);
```

Поскольку в незащищенном имени равноправного участника значением авторитетного источника является `0`, следующие строки кода эквивалентны:

```
PeerName pn = new PeerName("Peer classifier", PeerNameType.Unsecured);
PeerName pn = new PeerName("0.Peer classifier");
```

После создания экземпляр `PeerName` можно использовать вместе с номером порта для инициализации объекта `PeerNameRegistration`:

```
PeerNameRegistration pnr = new PeerNameRegistration(pn, 8080);
```

В качестве альтернативного варианта, можно установить для объекта `PeerNameRegistration`, создаваемого с использованием его параметра по умолчанию, свойство `PeerName` и (необязательно) свойство `Port`. Кроме того, желаемый экземпляр `Cloud` можно передать либо в третьем параметре конструктору `PeerNameRegistration`, либо указать с помощью свойства `Cloud`. Экземпляр `Cloud` можно получить либо из имени облака, либо с применением одного из следующих статических членов класса `Cloud`.

- `Cloud.Global`. Это статическое свойство позволяет получить ссылку на глобальное облако, которое в зависимости от конфигурации политики равноправных участников может быть и приватным глобальным облаком.
- `Cloud.AllLinkLocal`. Это статическое поле позволяет получить облако, содержащее все облака локальных соединений, доступные для данного равноправного участника.
- `Cloud.Available`. Это статическое поле позволяет получить облако, содержащее все облака, доступные для данного равноправного участника, т.е. локальные и глобальные (если таковые имеются).

После создания экземпляра `PeerNameRegistration` можно при желании установить его свойства `Comment` и `Data`. При этом следует помнить об ограничениях этих свойств. При попытке установить для свойства `Comment` строку длиной более 39 символов Unicode, будет сгенерировано исключение `PeerToPeerException`, а при попытке установить для свойства `Data` байтовый массив размером более 4096 байт — исключение `ArgumentOutOfRangeException`. Можно также с помощью свойства `EndPointCollection` добавить конечные точки. Это свойство представляет собой коллекцию типа `System.Net.IPEndPointCollection` с объектами `System.Net.IPEndPoint`. В случае применения свойства `EndPointCollection` может понадобиться установить свойство `UseAutoEndPointSelection` в `false`, чтобы предотвратить автоматическую генерацию конечных точек.

После того как все готово к регистрации имени равноправного участника, можно вызвать метод `PeerNameRegistration.Start()`. Для удаления записи о регистрации имени равноправного члена из службы PNRP служит метод `PeerNameRegistration.Stop()`.

Ниже приведен пример регистрации защищенного имени равноправного участника вместе с комментарием:

```
PeerName pn = new PeerName("Peer classifier", PeerNameType.Unsecured);
PeerNameRegistration pnr = new PeerNameRegistration(pn, 8080);
pnr.Comment = "Get pizza here";
pnr.Start();
```

Преобразование имен равноправных участников

Для преобразования имени равноправного участника должны быть выполнены следующие шаги.

1. Сгенерируйте имя равноправного члена на основе известного идентификатора P2P либо идентификатора P2P ID, полученного посредством операции обнаружения.
2. Воспользуйтесь распознавателем (resolver) для преобразования имени равноправного участника и получения коллекции записей с именами равноправных членов. Распознаватель можно ограничить отдельным облаком и/или максимальным количеством возвращаемых результатов.
3. Из любой полученной записи с именем равноправного участника извлеките само имя, а также необходимую информацию о конечной точке, комментарии и дополнительные данные, если таковые присутствуют.

Подобно процессу регистрации имени равноправного участника, этот процесс начинается с создания объекта `PeerName`. Отличие состоит в том, что здесь используется имя равноправного участника, которое уже было зарегистрировано одним или более удаленными участниками. Простейшим способом для получения списка активных членов в локальном облаке является регистрация незащищенного имени для каждого равноправного участника с одним и тем же классификатором и применение этого имени на стадии преобразования.

Однако для глобальных облаков пользоваться такой стратегией не рекомендуется, поскольку незащищенные имена могут быть легко подделаны.

Для преобразования имен равноправных участников используется класс `PeerNameResolver`. Получив в распоряжение экземпляр этого класса, можно выбрать, как должно выполняться преобразование — синхронно с применением метода `Resolve()` или же асинхронно с помощью метода `ResolveAsync()`.

Метод `Resolve()` можно вызывать с указанием как лишь одного параметра `PeerName`, так и дополнительно экземпляра `Cloud`, в котором должно производиться преобразование, максимального количества возвращаемых записей с именами равноправных участников в виде целого числа, или того и другого вместе. Этот метод возвращает экземпляр `PeerNameRecordCollection`, представляющий собой коллекцию объектов `PeerNameRecord`.

Например, ниже показан пример преобразования незащищенного имени равноправного участника во всех локальных облаках с возвратом максимум 5 результатов:

```
PeerName pn = new PeerName("0.Peer classifier");
PeerNameResolver pnres = new PeerNameResolver();
PeerNameRecordCollection pnrc = pnres.Resolve(pn, Cloud.AllLinkLocal, 5);
```

Метод `ResolveAsync()` предусматривает использование стандартной схемы вызова асинхронного метода, а именно — передачу уникального объекта `userState`, после чего ожидание поступления событий `ResolveProgressChanged`, свидетельствующих об обнаружении равноправных участников и выполнении преобразования, и события `ResolveCompleted`, свидетельствующего об окончании процесса обнаружения участников и преобразования. Метод `ResolveAsyncCancel()` позволяет отменить любой незавершенный асинхронный запрос.

В обработчиках события `ResolveProgressChanged` используется параметр аргументов события `ResolveProgressChangedEventArgs`, унаследованный от стандартного класса `System.ComponentModel.ProgressChangedEventArgs`.

Свойство `PeerNameRecord` объекта аргумента события, получаемого в обработчике событий, можно применять для получения ссылки на запись с именем равноправного участника, которая была обнаружена.

Аналогичным образом для события `ResolveCompleted` требуется обработчик, принимающий параметр типа `ResolveCompletedEventArgs`, который наследуется от `AsyncCompletedEventArgs`. Этот тип имеет параметр `PeerNameRecordCollection`, который можно использовать для получения полного списка записей с именами равноправных членов, которые были обнаружены.

В следующем коде показан пример реализации обработчиков для этих событий:

```
private pnres_ResolveProgressChanged(object sender,
   ResolveProgressChangedEventArgs e)
{
  // Использование e.ProgressPercentage (унаследованного от базовых аргументов события)
  // Обработка PeerNameRecord из e.PeerNameRecord
}
private pnres_ResolveCompleted(object sender, ResolveCompletedEventArgs e)
{
  // Проверка наличия e.IsCancelled и e.Error (унаследованных
  // от базовых аргументов события)
  // Обработка PeerNameRecordCollection из e.PeerNameRecordCollection
}
```

После получения одного или более объектов `PeerNameRecord` можно переходить к их обработке. Этот класс `PeerNameRecord` предоставляет в распоряжение свойства `Comment` и `Data` для изучения комментариев и дополнительных данных, которые были добавлены при регистрации имени равноправного участника (если были), свойство `PeerName` для из-

влечения объекта PeerName и записи с именем участника и, что наиболее важно, свойство EndPointCollection. Как и в случае с PeerNameRegistration, здесь это свойство тоже представляет собой коллекцию типа System.Net.IPEndPointCollection, содержащую объекты System.Net.IPEndPoint. Эти объекты можно использовать для подключения к предоставляемым равноправным участником конечным точкам любым желаемым образом.

Безопасный доступ кода в System.Net.PeerToPeer

Пространство имен System.Net.PeerToPeer также включает два следующих класса, которые можно использовать вместе с моделью CAS (Code Access Security — безопасный доступ кода), рассматриваемой в главе 21:

- PnrpPermission, унаследованный от класса CodeAccessPermission;
- PnrpPermissionAttribute, унаследованный от класса CodeAccessSecurity Attribute.

Эти классы удобно применять для обеспечения возможности настройки доступа PNRP с помощью полномочий обычным для CAS образом.

Пример приложения

В примерах кода для этой главы имеется пример приложения P2P, которое называется P2PSample и иллюстрирует применение описанного в настоящем разделе пространства имен и связанных с ним концепций. Это приложение WPF, в котором для конечной точки равноправного участника используется служба WCF.

Конфигурационные настройки этого приложения содержатся в конфигурационном файле и указывают, как должно выглядеть имя равноправного участника и какой порт должен прослушиваться:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="username" value="Karli" />
    <add key="port" value="8731" />
  </appSettings>
</configuration>
```

Фрагмент кода ***App.config***

После сборки этого приложения можно приступить к его тестированию, либо скопировав его на другие компьютеры в локальной сети и запустив все его экземпляры, либо запустив множество его экземпляров на одном компьютере. В последнем варианте нужно будет поменять используемый для каждого экземпляра порт, внеся соответствующие изменения в отдельные конфигурационные файлы (понадобится скопировать содержимое каталога Debug на свой локальный компьютер и по очереди отредактировать каждый конфигурационный файл). В обоих вариантах результаты будут выглядеть яснее, если изменить также имя пользователя в каждом экземпляре.

После запуска всех экземпляров приложения можно щелкнуть на кнопке Refresh (Обновить), чтобы получить список доступных равноправных участников асинхронным образом. Обнаружив в этом списке нужного участника, можно с помощью щелчка на кнопке Message (Сообщение) отправить ему стандартное сообщение.

На рис. 45.7 показан пример того, как приложение выглядит в действии при запуске трех его экземпляров на одной машине. На рисунке видно, что один равноправный участник только что отправил сообщение другому равноправному участнику, и оно отобразилось в диалоговом окне.

Бо́льшая часть работы в этом приложении происходит в обработчике события Window_Loaded() окна Window1.

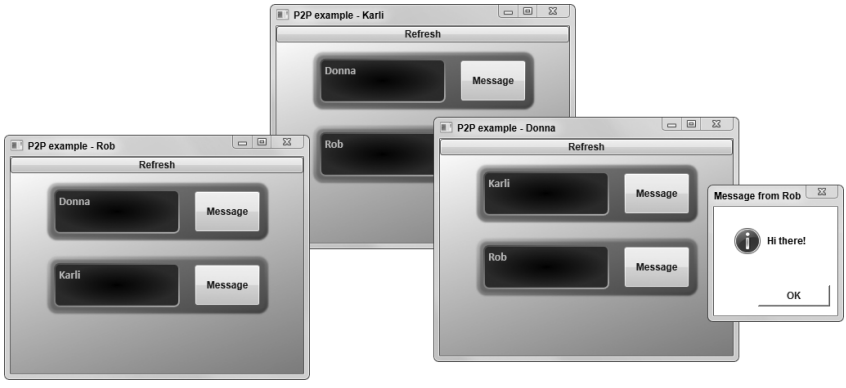

Рис. 45.7. *Приложение* `P2PSample` *в действии*

В этом методе сначала производится загрузка конфигурационной информации и установка в заголовке окна имени пользователя.

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
  // Получение конфигурационной информации из app.config
  string port = ConfigurationManager.AppSettings["port"];
  string username = ConfigurationManager.AppSettings["username"];
  string machineName = Environment.MachineName;
  string serviceUrl = null;
  // Установка заголовка окна
  this.Title = string.Format("P2P example - {0}", username);
```

Фрагмент кода **`Window1.xaml.cs`**

Далее с использованием адреса хоста равноправного участника и сконфигурированного порта определяется конечная точка, в которой должна предоставляться служба WCF. Привязкой этой службы будет `NetTcpBinding`, поэтому в URL-адресе конечной точки применяется протокол `net.tcp`.

```
// Получение URL-адреса службы с использованием адреса IPv4
// и порта из конфигурационного файла
foreach (IPAddress address in Dns.GetHostAddresses(Dns.GetHostName()))
{
  if (address.AddressFamily == System.Net.Sockets.AddressFamily.InterNetwork)
  {
    serviceUrl = string.Format("net.tcp://{0}:{1}/P2PService", address, port);
    break;
  }
}
```

Затем производится проверка полученного URL-адреса конечной точки, после чего выполняется регистрация и запуск службы WCF:

```
// Выполнение проверки, не является ли адрес нулевым
if (serviceUrl == null)
{
  // Отображение ошибки и завершение работы приложения
  MessageBox.Show(this, "Unable to determine WCF endpoint.",
                        // Не удается определить адрес конечной точки WCF
    "Networking Error", MessageBoxButton.OK, MessageBoxImage.Stop);
    // Ошибка сетевой связи
  Application.Current.Shutdown();
}
```

```
// Регистрация и запуск службы WCF
localService = new P2PService(this, username);
host = new ServiceHost(localService, new Uri(serviceUrl));
NetTcpBinding binding = new NetTcpBinding();
binding.Security.Mode = SecurityMode.None;
host.AddServiceEndpoint(typeof(IP2PService), binding, serviceUrl);
try
{
  host.Open();
}
catch (AddressAlreadyInUseException)
{
  // Отображение ошибки и завершение работы приложения
  MessageBox.Show(this, "Cannot start listening, port in use.",
                        // Не удается начать прослушивание, порт занят
    "WCF Error", MessageBoxButton.OK, MessageBoxImage.Stop);
   // Ошибка WCF
  Application.Current.Shutdown();
}
```

Обратите внимание на использование одиночного (singleton) экземпляра класса службы для налаживания простых коммуникаций между приложением-хостом и службой (для отправки и получения сообщений). Кроме того, для простоты режим безопасности в конфигурации привязки отключен.

Регистрация имени равноправного участника производится с помощью классов из пространства имен `System.Net.PeerToPeer`:

```
// Создание имени равноправного участника
peerName = new PeerName("P2P Sample", PeerNameType.Unsecured);

// Подготовка процесса регистрации имени равноправного
// участника в локальном облаке
peerNameRegistration = new PeerNameRegistration(peerName, int.Parse(port));
peerNameRegistration.Cloud = Cloud.AllLinkLocal;

// Запуск процесса регистрации
peerNameRegistration.Start();
}
```

После щелчка на кнопке Refresh в обработчике события `RefreshButton_Click()` вызывается метод `PeerNameResolver.ResolveAsync()` для обнаружения соседних равноправных участников асинхронным образом:

```
private void RefreshButton_Click(object sender, RoutedEventArgs e)
{
  // Создание распознавателя и добавление обработчиков событий
  PeerNameResolver resolver = new PeerNameResolver();
  resolver.ResolveProgressChanged +=
    new EventHandler<ResolveProgressChangedEventArgs>(
      resolver_ResolveProgressChanged);
  resolver.ResolveCompleted +=
    new EventHandler<ResolveCompletedEventArgs>(
      resolver_ResolveCompleted);

  // Подготовка к добавлению новых равноправных участников
  PeerList.Items.Clear();
  RefreshButton.IsEnabled = false;

  // Преобразование незащищенных имен равноправных участников асинхронным образом
  resolver.ResolveAsync(new PeerName("0.P2P Sample"), 1);
}
```

Остальная часть кода отвечает за отображение и взаимодействие с соседними равноправными участниками; ее можно изучить самостоятельно.

Предоставление конечных точек WCF через облака P2P является замечательным способом для обеспечения возможности установки местонахождения служб внутри предприятия, а также налаживания связи между равноправными участниками сети подобно тому, как было сделано в рассмотренном примере.

Пространство имен `System.Net.PeerToPeer.Collaboration`

Классы в пространстве имен `System.Net.PeerToPeer.Collaboration` предоставляют каркас, который можно использовать для создания приложений, предусматривающих работу со службой People Neat Me и API-интерфейсом совместной работы посредством P2P (P2P Collaboration API). Как упоминалось ранее, на момент написания настоящей книги это было возможно только в ОС Windows Vista или Windows 7.

Классы, определенные в этом пространстве имен, можно использовать для взаимодействия с соседними равноправными участниками и приложениями, включая выполнение следующих функций:

- осуществление входа (sign in) и выхода (sign out);
- обнаружение других равноправных участников;
- управление контактами и выявление присутствия других равноправных участников в сети.

Классы из этого пространства имен можно также применять для приглашения других пользователей присоединиться к работе в каком-то приложении или для обеспечения возможности обмена данными между пользователями и приложениями. Однако для того чтобы делать это, необходимо создавать собственные приложения, поддерживающие PNM, рассмотрение которых выходит за рамки настоящей главы.

Все упоминаемые в последующих разделах типы относятся к пространству имен `System.Net.PeerToPeer.Collaboration`, если только не указано другое.

Осуществление входа и выхода

Одним из наиболее важных классов в пространстве имен `System.Net.PeerToPeer.Collaboration` является `PeerCollaboration`. Это статический класс с множеством статических методов, которые можно применять для самых различных целей, как будет показано в этом и следующем разделах. В частности, для осуществления входа и выхода из службы People Near Me предназначены его методы `SignIn()` и `SignOut()`. Оба метода принимают единственный параметр типа `PeerScope`, который может иметь одно из перечисленных ниже значений.

- `PeerScope.None`. В случае использования этого значения методы `SignIn()` и `SignOut()` не будут иметь никакого эффекта.
- `PeerScope.NearMe`. Указание этого значения позволяет осуществлять вход и выход из локальных облаков.
- `PeerScope.Internet`. Применение этого значения позволяет осуществлять вход и выход из глобального облака (это может понадобится для установки связи с контактным лицом, которого в текущий момент нет в локальной подсети).
- `PeerScope.All`. Указание этого значения позволяет осуществлять вход и выход из всех доступных облаков.

При необходимости вызов `SignIn()` будет приводить к отображению диалогового окна конфигурирования People Near Me.

После входа в службу можно для свойства `PeerCollaboration.LocalPresenceInfo` можно установить значение типа `PeerPresenceInfo`. Это сделает доступной стандартную

функциональность мгновенного обмена сообщениями (IM), например, возможность установки состояния в "отошел". Для свойства `PeerPresenceInfo.DescriptiveText` можно установить строку Unicode длиной до 255 символов, а для свойства `PeerPresenceInfo.PresenceStatus` — значение перечисления `PeerPresenceStatus`. Значения этого перечисления описаны ниже.

- `PeerPresenceStatus.Away`. Равноправного участника нет на месте.
- `PeerPresenceStatus.BeRightBack`. Равноправного участника нет на месте, но он скоро вернется.
- `PeerPresenceStatus.Busy`. Равноправный участник занят.
- `PeerPresenceStatus.Idle`. Равноправный участник неактивен.
- `PeerPresenceStatus.Offline`. Равноправного участника нет в сети.
- `PeerPresenceStatus.Online`. Равноправный участник находится в сети и доступен для связи.
- `PeerPresenceStatus.OnThePhone`. Равноправный участник разговаривает по телефону.
- `PeerPresenceStatus.OutToLunch`. Равноправного участника нет на месте, но он вернется после обеда.

Обнаружение соседей

После подключения к локальному облаку можно получить список всех равноправных участников сети, которые находятся по соседству. Это делается с помощью метода `PeerCollaboration.GetPeersNearMe()`, который возвращает объект `PeerNearMeCollection`, содержащий в себе объекты `PeerNearMe`.

Свойство `Nickname` объекта `PeerName` позволяет получить имя обнаруженного соседа, свойство `IsOnline` — определить, находится ли он в текущий момент в сети, а (в случае низкоуровневых операций) свойство `PeerEndPoints` — определить имеющие к нему отношение конечные точки. С помощью свойства `PeerEndPoints` также можно выяснить статус объекта `PeerNearMe` в сети. Для получения объекта `PeerPresenceInfo` необходимо передать методу `GetPresenceInfo()` информацию о конечной точке, как было описано в предыдущем разделе.

Управление контактами и определение присутствия соседей в сети

Контакты представляют собой способ, которым можно запоминать других равноправных участников сети. Обнаруженных посредством службы People Near Me пользователей сети можно добавить в свои контакты и после этого связываться с ними в любой момент, когда они будут появляться в сети. Связываться с контактами можно как через локальные, так и через глобальные облака (при условии наличия IPv6-подключения к Интернету).

Чтобы добавить обнаруженного пользователя в контакты, необходимо вызвать метод `PeerNearMe.AddToContactManager()`. При вызове этого метода можно закрепить за контактом отображаемое имя, псевдоним (nickname) и адрес электронной почты. Однако обычно управление контактами осуществляется с помощью класса `ContactManager`.

В любом случае при манипулировании контактами приходится иметь дело с объектами `PeerContact`. Класс `PeerContact`, как и `PeerNearMe`, унаследован от абстрактного базового класса `Peer`, но имеет больше свойств и методов, чем `PeerNearMe`. Так, например, он включает в себя свойства `DisplayName` и `EmailAddress`, которые позволяют более подробно описывать обнаруженного PNM пользователя. Другое отличие между этими двумя классами связано с тем, что `PeerContact` имеет более явные отношения с классом `System.Net.PeerToPeer.PeerName`. С помощью свойства `PeerContact.PeerName` можно извлечь объект `PeerName`, после чего приступить к взаимодействию с любыми конечными точками, которые этот объект `PeerName` предоставляет, используя описанные ранее приемы.

Информация о локальном соседе также доступна в статическом свойстве `ContactManager.LocalContact`. Это позволит далее иметь дело со свойством `PeerContact` и деталями о локальном соседе.

Для добавления объектов `PeerNearMe` в локальный список контактов применяется либо метод `ContactManager.CreateContact()`, либо метод `CreateContactAsync()`, а объектов `PeerName` — метод `GetContact()`. Удаление контактов, представляемых объектами `PeerNearMe` или `PeerName`, производится с помощью метода `DeleteContact()`.

И, наконец, существуют события, которые можно обрабатывать в ответ на изменения в контактах. Например, событие `PresenceChanged` можно обрабатывать в ответ на изменения в информации о присутствии любого из известных `ContactManager` контактов.

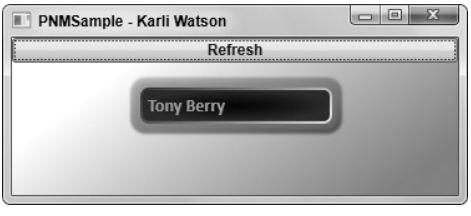

Рис. 45.8. *Пример приложения `PNMSample` в действии*

Пример приложения

В состав примеров кода для этой главы входит приложение, которое называется `PNMSample` и иллюстрирует использование классов из пространства имен `System.Net.PeerToPeer.Collaboration`. Это приложение похоже на предыдущее, но выглядит гораздо проще. Чтобы испробовать его в действии, необходимо иметь два компьютера, способных осуществлять вход в службу PNM, поскольку оно предусматривает построение и отображение списка соседей PNM из локальной подсети.

Если после запуска этого приложения есть хотя бы один доступный для обнаружения сосед, окно будет выглядеть примерно так, как показано на рис. 45.8.

Код приложения имеет точно такую же структуру, как в предыдущем примере. Правда, на этот раз в обработчике события `Window_Label()` выполняется не особо много работы кроме осуществления входа в PNM, потому что здесь нет службы, к которой требовалось бы получать доступ для запуска или выполнения процесса регистрации имени равноправного участника:

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
  // Осуществление входа в службу PNM
  PeerCollaboration.SignIn(PeerScope.NearMe);
```

Чтобы все выглядело немного красивее, для форматирования заголовка окна используется `ContactManager.LocalContact.Nickname`:

```
  // Получение имени локального равноправного участника, подлежащего отображению
  this.Title = string.Format("PNMSample - {0}",
  ContactManager.LocalContact.Nickname);
}
```

В `Window_Closing()` автоматически производится выход локального равноправного участника из PNM:

```
private void Window_Closing(object sender,
  System.ComponentModel.CancelEventArgs e)
{
  // Осуществление выхода из PNM
  PeerCollaboration.SignOut(PeerScope.NearMe);
}
```

Бо́льшая часть работы выполняется в обработчике события `RefreshButton_Click()`.

В частности, здесь используется метод PeerCollaboration.GetPeersNearMe() для получения и отображения списка соседних равноправных участников с применением класса PeerEntry, определенного в проекте. Если ни одного участника не обнаружено выводится соответствующее сообщение.

```
private void RefreshButton_Click(object sender, RoutedEventArgs e)
{
  // Получение списка локальных равноправных участников сети
  PeerNearMeCollection peersNearMe = PeerCollaboration.GetPeersNearMe();

  // Подготовка к добавлению новых участников
  PeerList.Items.Clear();

  // Изучение обнаруженных равноправных участников
  foreach (PeerNearMe peerNearMe in peersNearMe)
  {
    PeerList.Items.Add(
      new PeerEntry
      {
        PeerNearMe = peerNearMe,
        PresenceStatus = peerNearMe.GetPresenceInfo(
        peerNearMe.PeerEndPoints[0]).PresenceStatus,
        DisplayString = peerNearMe.Nickname
      });
  }

  // При необходимости — вывод сообщения о неудаче
  if (PeerList.Items.Count == 0)
  {
     PeerList.Items.Add(
       new PeerEntry
       {
         DisplayString = "No peers found."
                       // Равноправные участники не обнаружены
       });
  }
}
```

Как видно по этому примеру, классы, о которых было рассказано, существенно упрощают взаимодействие со службой PNM.

Резюме

В этой главе было показано, как реализовать в приложениях функциональность одноранговых сетей (P2P) с применением классов P2P, предлагаемых в .NET 4.

В частности, здесь были показаны различные типы решений, которые позволяет создавать технология P2P, и то, как эти решения можно структурировать, использовать в них протокол PNRP и службу PNM, а также типы из пространств имен System.Net.PeerToPeer и System.Net.PeerToPeer.Collaboration. Кроме того, демонстрировался очень полезный прием предоставления служб WCF как конечных точек P2P.

Тем, кто заинтересовался разработкой приложений P2P, стоит продолжить изучение PNM дальше, а также узнать о том, как применять канал одноранговой связи (peer channel), с помощью которого WCF-службы могут обеспечивать широковещательные коммуникации между множеством клиентов одновременно.

В следующей главе рассматривается технология Message Queuing.