BAB 5
DISTRIBUTED OBJECT AND REMOTE INVOCATION

Ibnu Pradipta – 33237
Hanindito H.P – 33308
Firman Nanda – 33529
Jurusan Teknik Elektro FT UGM,
Yogyakarta

5.1 PENDAHULUAN

Bab ini berkaitan dengan model pemrograman untuk aplikasi terdistribusi yang merupakan aplikasi yang terdiri dari program-program bersama yang berjalan di beberapa proses yang berbeda. Program-program tersebut harus mampu untuk meminta operasi di lain proses, yang sering berjalan di komputer yang berbeda. Untuk mencapai hal ini, beberapa model pemrograman yang familiar telah diperluas untuk diterapkan ke dalam distributed program:

- paling awal dan mungkin paling terkenal adalah perpanjangan dari panggilan prosedur konvensional, yang memperbolehkan klien untuk memanggil prosedur dalam program yang berjalan di server terpisah dan umumnya di komputer yang berbeda dari klien.

- baru-baru ini, objek pemrograman berbasis Model ini telah ditambahkan untuk memungkinkan objek dalam proses yang berbeda untuk berkomunikasi dengan satu sama lain melalui Remote Method Invocation (RMI). RMI adalah perluasan dari local method invocation yang memungkinkan sebuah objek yang hidup dalam satu proses untuk memohon method objek yang berada di proses lain.

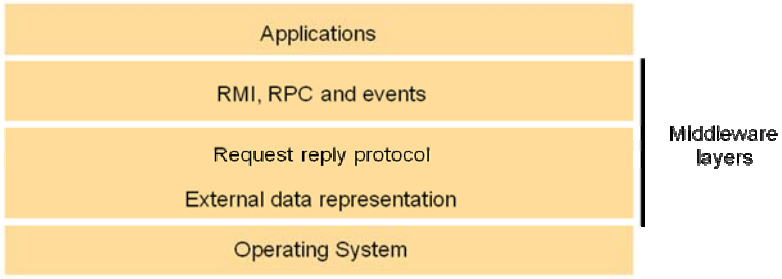
- pemrograman berbasis objek memungkinkan untuk menerima pemberitahuan dari kejadian di objek-objek lain di mana mereka telah terdaftar. Model ini telah ditambahkan untuk memungkinkan didistribusikan berdasarkan aktivitas program yang akan ditulis.

yang perlu diperhatikan bahwa kita menggunakan istilah 'RMI' untuk merujuk kepada RMI dalam cara yang umum - Ini tidak boleh dikacaukan dengan contoh-contoh khusus dari metode Remote Invocation seperti Java RMI. Kebanyakan perangkat lunak sistem terdistribusi saat ini ditulis dalam bahasa obyek- oriented. dan RPC dapat dipahami dalam kaitannya dengan RMI. Oleh karena ini bab berkonsentrasi pada acara RMI dan paradigma, yang masing-masing berlaku untuk didistribusikan objek. Komunikasi antara objek terdistribusi diperkenalkan dalam Bagian, yaitu :

Middleware o Software yang menyediakan model pemrograman diatas dari dasar blok dari proses dan melewatkan pesan yang biasa disebut middleware. The middleware layer menggunakan protokol yang didasarkan pada pesan antara proses untuk memberikan tingkat yang lebih tinggi.

abstraksi seperti remote invocation dan event, seperti yang diilustrasikan pada Gambar 5.1. Sebuah aspek penting dari middleware adalah penyediaan lokasi transparansi dan independen dari protokol komunikasi, sistem operasi dan perangkat keras komputer, Beberapa bentuk middleware emungkinkan komponen terpisah yang akan ditulis dalam bahasa pemrograman yang berbeda.

Location Tranparency : Dalam RPC, klien yang memanggil sebuah prosedur tidak dapat mengatakan apakah prosedur berjalan dalam proses yang sama atau dalam proses yang berbeda, mungkin yang berbeda komputer. Klien juga tidak perlu mengetahui lokasi dari server. objek-objek dapat menghasilkan Events dan objek-objek yang menerima pemberitahuan tentang peristiwa-peristiwa itu perlu tidak menyadari satu lokasi lainnya.



Gambar 5.1

Communication Protocol : protokol-protokol komunikasi yang mendukung protokol middleware abstraksi yang independen dari protokol transportasi yang mendasarinya. Sebagai contoh, permintaan-reply protokol dapat diimplementasikan di UDP atau TCP.

Computer Hardware : Dua standar yang disetujui untuk representasi data eksternal dijelaskan dalam Bagian 4.3. Ini digunakan ketika marshalling dan unmarshalling pesan. Mereka menyembunyikan perbedaan karena arsitektur perangkat keras, seperti byte pemesanan.

Sistem operasi : Semakin tinggi tingkat abstraksi yang disediakan oleh lapisan middleware yang independen dari sistem operasi yang mendasarinya.

Penggunaan beberapa bahasa pemrograman : Beberapa middleware dirancang untuk memungkinkan aplikasi didistribusikan menggunakan lebih dari satu bahasa pemrograman. Secara khusus, CORBA yang memungkinkan klien menulis dalam satu bahasa untuk memohon method berada di server program yang ditulis dalam bahasa lain. Hal ini dicapai dengan menggunakan Interface Definition Language atau IDL untuk mendefinisikan antarmuka. IDL ini dibahas dalam bagian berikutnya.

5.1.1 Interfaces

Kebanyakan bahasa pemrograman modern menyediakan cara untuk mengatur sebuah program sebagai kumpulan modul yang dapat berkomunikasi dengan satu sama lain. Komunikasi antara modul

dapat melalui prosedur panggilan antara modul atau dengan akses langsung ke variabel dalam modul lain. Dalam rangka untuk mengendalikan kemungkinan interaksi antara modul, sebuah antarmuka eksplisit didefinisikan untuk setiap modul. Antarmuka modul menetapkan prosedur dan variabel yang dapat diakses dari modul lain. Modul dilaksanakan sehingga dapat menyembunyikan semua informasi tentang mereka kecuali yang tersedia melalui antarmuka. selama antarmuka tetap sama, implementasi dapat dapat diubah tanpa mempengaruhi pengguna modul.

Antarmuka dalam sistem terdistribusi o Dalam program terdistribusi, modul-modul dapat berjalan dalam proses terpisah. Tidaklah mungkin untuk menjalankan modul dalam satu proses untuk mengakses variabel dalam sebuah modul proses lain. Oleh karena itu, antarmuka modul yang dimaksudkan untuk RPC atau RMI tidak dapat menetapkan akses langsung ke variabel. Perhatikan bahwa CORBA IDL interface dapat menetapkan atribut, yang tampaknya melanggar aturan ini. Namun, atribut tidak diakses secara langsung.

mekanisme Parameter-passing, misalnya panggilan dengan nilai dan panggilan dengan referensi, yang digunakan dalam prosedur lokal panggilan tidak cocok ketika pemanggil dan prosedur yang berada dalam proses yang berbeda. Spesifikasi prosedur atau metode dalam interface dari modul program terdistribusi menggambarkan parameter sebagai input atau output atau kadang-kadang keduanya. Input parameter dilewatkan ke modul remote dengan mengirimkan nilai-nilai argumen dalam pesan permintaan dan kemudian memasok mereka sebagai argumen ke operasi yang akan dieksekusi di server. Parameter output dikembalikan dalam pesan balasan dan digunakan sebagai hasil dari panggilan atau menggantikan nilai-nilai variabel yang terkait dalam pemanggil. Ketika parameter yang digunakan untuk kedua input dan output nilai harus ditransmisikan dalam pesan requet and reply.

Dua paragraf berikutnya membahas antarmuka yang digunakan dalam client-server model RPC dan dalam model objek terdistribusi RMI:

Layanan antarmuka : Dalam model client-server, setiap server menyediakan satu set prosedur yang tersedia untuk digunakan oleh klien. Sebagai contoh, sebuah file server akan memberikan prosedur untuk membaca dan menulis file. Antarmuka layanan istilah digunakan untuk merujuk pada spesifikasi prosedur yang ditawarkan oleh saluran pembuangan, menentukan jenis masukan dan argumen output dari masing-masing prosedur.

Remote interface : dalam model objek terdistribusi, remote interface menentukan metode objek yang tersedia untuk invokasi oleh objek dalam proses-proses lain, mendefinisikan jenis input dan output argumen dari masing-masing. Namun, besar Perbedaannya adalah bahwa metode dalam remote interface dapat melewati objek sebagai argumen dan hasil method.

Interface Definition Language o Sebuah mekanisme RMI yang dapat diintegrasikan dengan bahasa pemrograman tertentu, jika mencakup notasi yang memadai untuk mendefinisikan interface, sehingga input dan output parameter yang akan dipetakan ke bahasa normal parameter. Java RMI adalah contoh di mana suatu mekanisme RMI telah ditambahkan ke bahasa pemrograman berorientasi obyek. Pendekatan ini berguna ketika semua bagian dari sebuah aplikasi terdistribusi dapat ditulis dalam bahasa yang sama. Hal ini juga nyaman karena memungkinkan para programmer untuk menggunakan satu bahasa untuk lokal dan remote invocation.

Namun, banyak layanan yang biasa digunakan yang ada ditulis dalam C++ dan bahasa lainnya. Ini akan bermanfaat untuk memungkinkan program yang ditulis dalam berbagai bahasa, termasuk Java, untuk mengaksesnya dari jarak jauh. Interface Definition Language (atau IDLs) didesain untuk memungkinkan objek diimplementasikan dalam berbagai bahasa untuk memanggil satu sama lain. Sebuah IDL menyediakan notasi untuk mendefinisikan interface di mana masing-masing parameter dari sebuah metode mungkin bisa digambarkan sebagai input atau output.

```
// Infile Person.idl
struct Person {
string name;
string place;
long year;
};
interface PersonList {
readonly attribute string listname;
void addPerson(in Person p) ;
void getPerson(in string name, out Person p);
long number();
};
```

Gambar 5.2

Gambar 5.2 memperlihatkan contoh sederhana dari CORBA IDL. Struktur *person* yang sama dengan yang digunakan untuk menggambarkan marshalling. Interface bernama *PersonList* menentukan metode yang tersedia untuk RMI di remote obyek yang mengimplementasikan antarmuka tersebut. Sebagai contoh, metode *addPerson* menentukan argumen *in*, yang berarti bahwa itu adalah sebuah argumen input , dan method *getperson* yang mengambil sebuah contoh nama Person menentukan argumen kedua keluar, yang berarti bahwa itu adalah argumen output.

5.2 KOMUNIKASI ANTAR DISTRIBUTED OBJEK

Objek berbasis model untuk sistem terdistribusi mengekstensi model yang didukung oleh bahasa pemrograman berorientasi obyek untuk membuat itu berlaku untuk didistribusikan objek. Bagian ini akan dibahasa komunikasi antara objek terdistribusi dengan cara RMI. Materi yang disajikan yaitu :

- *Object model* : tinjauan singkat aspek-aspek yang relevan dari model object, cocok bagi pembaca dengan pengetahuan dasar yang berorientasi obyek bahasa pemrograman, misalnya Java atau C++.
- *Distributed Object* : Sebuah obyek presentasi berbasis sistem terdistribusi, yang berpendapat bahwa model objek sangat sesuai untuk sistem terdistribusi.
- *The Distributed Object Model* : Sebuah diskusi tentang perluasan model objek diperlukan untuk itu untuk mendukung objek terdistribusi.
- *Design issues* : Sebuah set argumen tentang desain alternatif:
    1. invokasi lokal dijalankan persis sekali, Tetapi apa semantik yang cocok dan mungkin bagi invokasi remote?
    2. Bagaimana semantik RMI dapat dibuat mirip dengan method lokal invokasi dan apa perbedaan yang tidak dapat dihilangkan?
- *Implementation* : Sebuah penjelasan mengenai bagaimana lapisan middleware atas permintaan - reply protokol dapat dirancang untuk mendukung aplikasi RMI antara tingkat-terdistribusi objek.
- *Distributed Garbage Collection* : Sebuah presentasi dari sebuah algoritma untuk didistribusikan pengumpulan sampah yang sesuai untuk digunakan dengan implementasi RMI.

5.2.1 Object Model

Sebuah program berorientasi objek, misalnya di Jawa atau C++, terdiri dari kumpulan objek yang saling berinteraksi, masing-masing terdiri dari satu set data dan satu set method. sebuah obyek berkomunikasi dengan obyek lain dengan menerapkan method mereka, umumnya lewat argumen dan menerima hasil. Obyek dapat mengenkapsulasi data mereka dan kode metode. Beberapa bahasa, misalnya Java dan C + +, memungkinkan programmer untuk mendefinisikan Misalnya variabel objek yang dapat diakses secara langsung. Tetapi untuk digunakan dalam distributed sistem objek, sebuah objek data harus dapat diakses hanya melalui method.

Object References o Object dapat diakses melalui obyek referensi. Sebagai contoh, di java, sebuah variabel yang muncul untuk mengadakan sebuah objek benar-benar memegang referensi ke sebuah objek. Untuk memanggil sebuah metode dalam suatu objek, obyek referensi dan nama metode yang diberikan, bersama-sama dengan argumen yang diperlukan. Objek metode yang dipanggil kadang-kadang disebut target dan kadang-kadang penerima.

Interface o interface memberikan definisi dari tanda tangan dari serangkaian metode (yang adalah, jenis argumen mereka, kembali nilai-nilai dan pengecualian) tanpa menentukan implementasi. Sebuah objek akan menyediakan antarmuka tertentu jika kelasnya berisi kode yang melaksanakan method interface tersebut. Di Java, sebuah kelas dapat mengimplementasikan beberapa antarmuka, dan metode antarmuka dapat dilaksanakan oleh setiap kelas. Sebuah interface juga mendefinisikan sebuah tipe yang dapat digunakan untuk menyatakan tipe variabel. Perhatikan bahwa interface tidak memiliki konstruktor.

Actions O Action dalam sebuah program berorientasi objek dimulai oleh sebuah objek memohon sebuah metode dalam objek lain. Sebuah invokasi dapat mencakup informasi tambahan (argumen) diperlukan untuk melaksanakan metode. Penerima menjalankan metode yang tepat dan kemudian mengembalikan control ke objek invokasi, yang kadang-kadang menyediakan hasilnya. Sebuah invokasi dari method memiliki dua efek, yaitu :

1. state penerima dapat berubah, dan
2. invokasi lebih lanjut mengenai metode dalam objek-objek lain mungkin terjadi.

Exceptions o Program dapat menemukan berbagai macam kesalahan dan kondisi-kondisi tak terduga . Selama pelaksanaan sebuah method, banyak masalah yang berbeda mungkin ditemukan: misalnya. konsisten nilai dalam variabel objek, atau kegagalan dalam mencoba untuk membaca atau menulis ke file atau socket jaringan. Exceptions menyediakan cara yang bersih untuk berurusan dengan kesalahan kondisi tanpa kode rumit. Sebuah block kode dapat didefinisikan untuk membuang exceptions kapan kondisi tertentu atau kesalahan tak terduga muncul. Ini berarti bahwa kontrol lolos ke blok kode yang lain yang bisa menarik pengecualian. Kontrol tidak kembali ke tempat dimana pengecualian terlempar.

Garbage Collection o hal ini diperlukan untuk menyediakan cara untuk membebaskan ruang yang ditempati oleh objek ketika mereka tidak lagi diperlukan. Suatu bahasa, misalnya Jawa, yang dapat mendeteksi secara otomatis ketika objek tidak lagi dapat diakses.

5.2.2 Distributed object

Sistem objek terdistribusi dapat mengadopsi arsitektur client-server. Dalam kasus ini, objek dikelola oleh server dan klien memanggil method mereka dengan menggunakan remote metode invokasi. Di RMI, permintaan klien untuk memohon sebuah metode objek dikirim dalam pesan ke server. invokasi dilakukan dengan melaksanakan metode objek di server dan hasilnya dikembalikan ke klien dalam pesan lain.

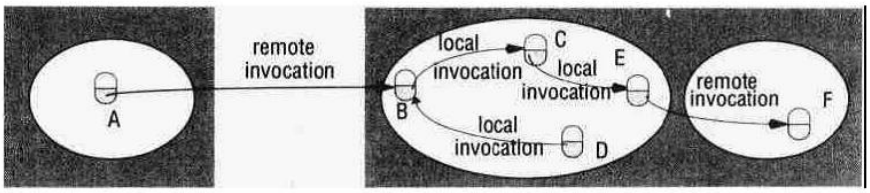
Distributed object dapat mengasumsikan model arsitektur lain. Sebagai contoh, objek dapat direplikasi dalam rangka memperoleh manfaat untuk toleransi kesalahan dan meningkatkan kinerja, dan objek dapat bermigrasi dengan maksud untuk meningkatkan kinerja mereka.

Setelah klien dan server objek berada dalam proses yang berbeda yang melaksanakan enkapsulasi. Artinya, keadaan suatu objek hanya dapat diakses oleh metode objek, yang berarti bahwa tidak mungkin bagi metode yang tidak sah untuk bertindak atas state. Misalnya kemungkinan RMls dari objek di komputer yang berbeda menunjukkan bahwa objek dapat diakses secara bersamaan. Oleh karena itu, kemungkinan akses yang saling bertentanganmuncul. Namun, fakta bahwa data dari suatu objek diakses hanya oleh metode sendiri memungkinkan objek untuk menyediakan metode untuk melindungi diri terhadap akses yang salah.

Keuntungan lain memperlakukan state bersama didistribusikan program sebagai koleksi object adalah bahwa object dapat diakses melalui RMI atau dapat disalin ke local cache dan diakses secara langsung dengan ketentuan bahwa implementasi kelas tersedia pada komputer lokal.

5.2.3 Distributed Object Model

Bagian ini membahas ekstensi untuk model obyek untuk membuatnya berlaku untuk mendistribusikan objek. Setiap proses berisi koleksi objek, beberapa di antaranya dapat menerima baik lokal dan remote invokasi, sedangkan yang lain hanya objek dapat menerima lokal invokasi, seperti yang ditunjukkan pada Gambar 5.3.

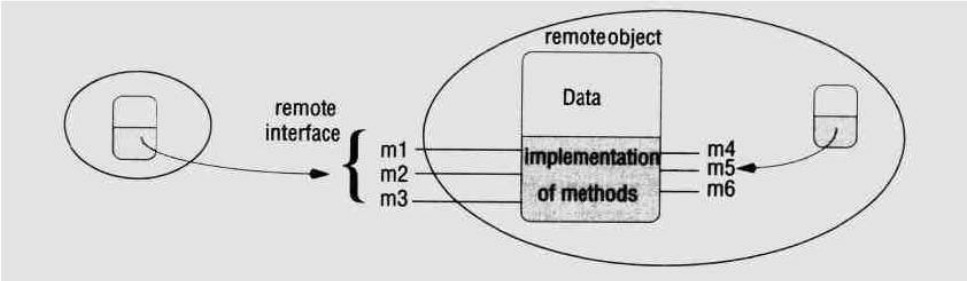


Gambar 5.3

Method invokasi antara obyek yang berada pada proses yang berbeda , apakah di komputer yang sama atau tidak, yang dikenal sebagai Remote Method Invocation. Method invocation antara objek dalam proses yang sama disebut Local Method Invocation.

merujuk kepada objek yang dapat menerima remote invocation sebagai remote objek. Di Gambar 5.3, objek B dan F adalah objek remote. Semua objek dapat menerima lokal invokasi, meskipun mereka dapat menerima mereka hanya dari objek lain yang memegang referensi mereka. Sebagai contoh, objek C harus memiliki referensi ke obyek E sehingga dapat meminta salah satu metode. Berikut dua konsep dasar yang merupakan inti dari distributed object model :

- *Remote Object Reference* : objek lain dapat meminta method objek jarak jauh jika mereka memiliki akses ke *Remote Object Reference* . Sebagai contoh, sebuah objek remote referensi untuk B pada Gambar 5.3 harus tersedia untuk A.
- *Remote interface* : Setiap remote obyek memiliki remote interface yang menentukan mana metode dapat dipanggil dari jarak jauh. Sebagai contoh, objek B dan F harus memiliki remote interface. Paragraf berikut ini membahas referensi objek remote, remote interface dan aspek model object terdistribusi.



Gambar 5.4

Remote Object References O Gagasan referensi obyek ditambahkan untuk memungkinkan setiap objek yang dapat menerima RMI untuk memiliki referensi objek remote. Sebuah objek remote referensi adalah sebuah identifier yang dapat digunakan di seluruh sistem terdistribusi untuk merujuk objek remote unik tertentu. Representasinya, yang umumnya berbeda dari yang referensi objek local. Remote objek referensi analog lokal dalam:

1. remote objek untuk menerima metode remote invokasi ditetapkan sebagai remote referensi obyek, dan
2. referensi objek remote mungkin akan dilewatkan sebagai argumen dan hasil dari jauh metode invokasi.

Remote Interface O remote Kelas objek remote mengimplementasikan metode dari remote interface, misalnya sebagai metode instan publik di Java. Objek dalam proses-proses lain hanya dapat memanggil metode-metode yang termasuk remote interface. Lokal obyek dapat meminta metode dalam remote interface serta metode lain dilaksanakan oleh objek remote. Perhatikan bahwa remote interface, seperti semua interface, tidak memiliki konstruktor.

Action in a distributed object system o Seperti dalam non-terdistribusi, suatu tindakan diprakarsai oleh sebuah metode invokasi, yang dapat berakibat lebih lanjut pada metode invokasi objek lain. Tetapi dalam kasus terdistribusi, benda-benda yang terlibat dalam rantai invokasi yang saling terkaitmungkin berlokasi di proses yang berbeda atau komputer yang berbeda. ketika sebuah invokasi melintasi batas dari sebuah proses atau komputer, RMI digunakan, dan remote referensi objek harus tersedia untuk membuat RMI mungkin. Dalam Gambar 5.3, yang objek A harus memegang referensi ke objek remote objek referensi objek B. Remote dapat diperoleh sebagai hasil dari metode remote invocation. Sebagai contoh, objek A dalam Gambar 5.3 bisa mendapatkan referensi ke objek remote F dari objek B.

Garbage Collection in a distributed-object system o Jika suatu bahasa, misalnya Java, mendukung pengumpulan sampah, maka sistem RM1 yang terkait harus mengizinkan pengumpulan sampah dari remote object. Pendistribusian pengumpulan sampah umumnya dicapai oleh

kerjasama antara sampah lokal yang ada kolektor dan modul tambahan yang melakukan suatu bentuk pengumpulan sampah yang terdistribusi.

Exception o Setiap remote invokasi mungkin gagal karena alasan-alasan yang terkait dengan objek invoked berada dalam proses yang berbeda atau komputer dari INVOKER. Sebagai contoh, proses berisi objek remote mungkin telah jatuh atau mungkin terlalu sibuk untuk menjawab, atau invokasi atau pesan Hasilnya mungkin akan hilang. Oleh karena itu. Method remote invocation harus mampu meningkatkan timeout seperti yang disebabkan oleh distribusi maupun yang diangkat selama pelaksanaan metode invoked.

5.2.4 Desain issues for RMI

Bagian sebelumnya menunjukkan bahwa RMI adalah perpanjangan method local invokasi alami . Dalam bagian ini, kita membahas dua masalah desain yang timbul dalam membuat extension:

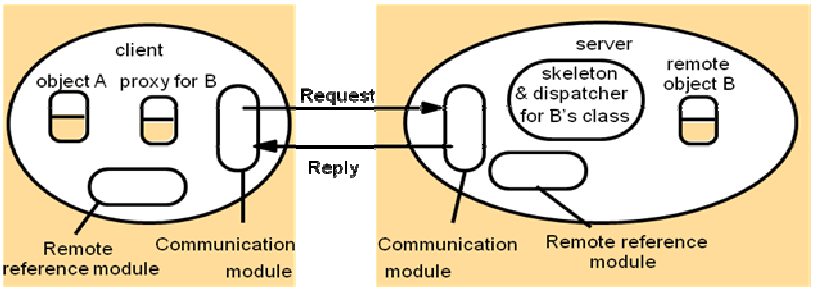
- Meskipun invokasi lokal dilaksanakan tepat sekali, hal ini tidak selalu menjadi kasus metode remote invokasi. Alternatif yang dibahas.
- Tingkat transparansi yang diinginkan untuk RMI.

RMI Invocation Semantics O di mana kita menunjukkan bahwa doOperation dapat diimplementasikan dengan cara yang berbeda untuk memberikan jaminan pengiriman berbeda. Pilihan utama adalah:

- *Retry Requet Message* : apakah permintaan Retransmit pesan sampai baik balasan diterima atau server diasumsikan telah gagal.
- *Duplicate Filtering* : ketika retransmisi digunakan, apakah untuk menyaring duplikat permintaan pada server.
- *Retransmission of Result* : apakah akan menyimpan history dari pesan hasil untuk mengaktifkan
  hasil hilang yang akan kembali tanpa mengeksekusi ulang operasi pada server.

Transparency O The originators dari RPC, Bitrell dan Nelson [1984], memiliki tujuan untuk membuat remote prosedur seperti panggilan seperti panggilan prosedur lokal mungkin, tanpa perbedaan dalam sintaks antara lokal dan remote prosedur panggilan. Semua panggilan yang penting untuk marshalling dan pesan melewati prosedur tersembunyi dari para programmer yang membuat membuat panggilan. Gagasan transparansi ini telah diperpanjang hingga berlaku untuk didistribusikan objek. tapi itu tidak hanya melibatkan bersembunyi marshalling dan pesan lewat, tetapi juga tugas untuk menemukan dan menghubungi remote objek. Sebagai contoh, Java RMI membuat remote metode invokasi yang sangat mirip lokal dengan mengizinkan mereka untuk menggunakan sintaks yang sama.

Namun, invokasi jauh lebih rentan terhadap kegagalan daripada yang lokal, karena mereka melibatkan jaringan. komputer lain dan proses lain. Mana saja dari invokasi semantic yang dipilih, selalu ada kemungkinan bahwa tidak ada hasilnya akan diterima dan dalam hal kegagalan. Ini membutuhkan sesuatu yang dapat membuat remote invokasi dapat pulih dari situasi seperti ini.



Gambar 5.5

5.2.5 Implementasi RMI

Beberapa objek terpisah dan modul terlibat dalam proses penerimaan permintaan metode remot objek. Ini ditunjukkan pada Gambar 5.6, di mana tingkat aplikasi objek A memanggil sebuah metode dalam aplikasi remote objek B dimana dia yang memegang remote referensi objek. Bagian ini membahas peran masing-masing komponen yang ditunjukkan pada gambar 5.6 tersebut, pertama berurusan dengan komunikasi dan referensi remot modul dan kemudian dengan perangkat lunak RMI yang menjalankannya.

Modul komunikasi Kedua modul komunikasi bekerja sama melaksanakan permintaan-reply protocol *request-reply*, yang mengirimkan permintaan dan membalas pesan antara klien dan server. Modul komunikasi hanya menggunakan tiga item pertama, yang menetapkan jenis pesan, request idnya, dan referensi remot objek yang diminta. MethodId dan semua yang menyusun dan menguraikan adalah perhatian dari perangkat lunak RMI yang dibahas di bawah ini. Modul komunikasi bersama-sama bertanggung jawab untuk menyediakan permohonan semantic yang spesifik, misalnya *at-most-once*. Modul komunikasi pada server memilih operator untuk kelas dari objek yang dipanggil, melewati pada referensi lokal, yang didapat dari remote modul referensi yang dikembalikan pada pengidentifikasi remot objek dalam pesan permintaan. Peran dari operator dibahas pada software RMI di bawah ini.

Remote Modul Reference Remot referensi modul bertanggung jawab untuk menerjemahkan antara local dan referensi remote objek dan untuk menciptakan referensi remot objek. Untuk mendukung tugasnya ini, referensi modul remote dalam setiap proses memiliki tabel remote objek yang mencatat korespondensi antara objek lokal referensi dalam proses dan referensi remote objek (dimana system-wide). Tabel meliputi:

Sebuah entri untuk semua objek remote dipegang oleh proses. Sebagai contoh, pada Gambar 5.6. objek remote B akan dicatat dalam tabel di server. • Sebuah entri untuk setiap proxy lokal. Sebagai contoh, pada Gambar 5.6 proxy untuk B akan tercatat dalam tabel di klien. Peran proxy dibahas pada software RMI di bawah ini. Tindakan remote referensi modul adalah sebagai berikut:

- Ketika remote objek untuk diteruskan sebagai argumen atau hasil untuk pertama kalinya, modul referensi remot modul diminta untuk membuat referensi objek remote, yang menambahkan tabel.
- Ketika referensi objek remote tiba dalam permintaan atau membalas pesan, remote modul referensi diminta untuk menyesuaikan referensi obyek lokal, yang mungkin mengacu baik pada proxy atau ke objek remot. Dalam hal objek remote referensi tidak ada dalam tabel, menciptakan perangkat lunak RMI proxy baru dan meminta remote referensi modul untuk menambahkannya ke tabel.

Software RMI ini terdiri dari suatu lapisan perangkat lunak antara application level objek dan komunikasi dan reeferensi remot modul. Peran middleware objek ditunjukkan pada Gambar 5.6 adalah sebagai berikut:

- Proxy: Peran proxy adalah untuk membuat permohonan metoderemot transparan untuk klien dengan bertingkah seperti objek lokal ke invoker, tetapi selain melaksanakan suatu permintaan ini akan diteruskan sebuah pesan ke objek remote. Itu menyembunyikan rincian remot objek referensi, yang menyusun argumen, menguraikan hasil dan pengiriman dan penerimaan pesan dari klien.
- Operator/Dispatcher : Sebuah server memiliki satu operator dan kerangka untuk masing-masing mewakili kelas remote objek. Dalam contoh kita, server memiliki operator dan kerangka untuk kelas remot objek B. operator menerima pesan permintaan dari modul komunikasi. Ia menggunakan methodId untuk memilih metode yang tepat dalam kerangka kemudian menyampaikan pesan permintaan. Operator dan proxy menggunakan sama alokasi *methodld* terhadap metode antarmuka remote.
- Skeleton: Kelas jauh objek memiliki kerangka, yang mengimplementasikan metode dalam antarmuka remote. Mereka dilaksanakan cukup berbeda dari metode-metode di objek remote. Sebuah metode menguraikan kerangka argumen dalam pesan permintaan dan memanggil metode yang sesuai dalam objek remote.

Generasi dari kelas proxy, operator dan kerangka Kelas-kelas untuk proxy, operator dan kerangka yang digunakan di RMI dihasilkan secara otomatis oleh sebuah antarmuka kompiler, Sebagai contoh, dalam pelaksanaan Orbix CORBA, interface dari remote objek didefinisikan dalam IDL CORBA, dan kompilator antarmuka dapat digunakan untuk menghasilkan kelas untuk proxy, dispatcher dan kerangka di C + +. Untuk Java RMI, himpunan metode yang ditawarkan oleh objek remote didefinisikan sebagai antarmuka Java yang diimplementasikan dalam kelas objek remote. Java RMI compiler menghasilkan proxy, operator dan kerangka kelas dari kelas dari remote objek,

Server dan Client Program server berisi kelas untuk operator/dispatcher dan kerangka, bersama dengan kelas-kelas implementasi dari semua remote objek yang mendukung. Yang terakhir kadang-kadang disebut pelayan kelas. Selain itu, program server berisi bagian initialization (misalnya dalam utama metode di Java atau C + +). Bagian inisialisasi bertanggung jawab untuk menciptakan danmenginisialisasi setidaknya satu dari objek jauh yang dibangun oleh server. Tambahan objek remote dapat dibuat sebagai tanggapan atas permintaan dari klien. Inisialisasi bagian mungkin juga mendaftar beberapa objek yang terpencil dengan sebuah map. Umumnya, ini akan mendaftarkan hanya satu remote objek, yang dapat digunakan untuk akses sisanya.

Binder pada umumnya program-program Klien memerlukan sarana untuk mendapatkan objek jauh referensi untuk setidaknya satu dari objek remote dipegang oleh server. Sebagai contoh, pada Gambar 5.3, objek A akan memerlukan referensi objek remote untuk objek B. pengikat dalam sistem terdistribusi adalah layanan terpisah yang menyimpan tabel yang berisi pemetaan dari tekstual nama untuk referensi objek remote. Hal ini digunakan oleh server untuk mendaftarkan remote objek dengan nama dan klien untuk mencari mereka.

Server Thread Setiap kali sebuah objek menjalankan permintaan ke remote, bahwa pelaksanaan dapat mengarah pada metode permintaan metod lebih lanjut dalam remot objek lain, yang mungkin memakan waktu untuk kembali. Untuk menghindari pelaksanaan permintaan satu remote menunda eksekusi lain, umumnya server mengalokasikan thread terpisah untuk pelaksanaan masing-masing permintaan dari jauh. Ketika hal ini terjadi, perancang pelaksanaan objek remote harus memungkinkan efek pada keadaan bersamaan eksekusi.

Aktivasi dari remot objek Beberapa aplikasi memerlukan informasi itu untuk bertahan waktu yang lama, Namun, tidak praktis untuk mewakili benda-benda seperti informasi untuk disimpan dalam menjalankan proses untuk periode terbatas, terutama karena mereka belum tentu digunakan sepanjang waktu. Untuk menghindari pemborosan potensi sumber daya karena untuk menjalankan semua server yang mengelola remot-remottersebut sepanjang waktu, server dapat dimulai kapan saja mereka dibutuhkan oleh klien, seperti yang dilakukan untuk set standar TCP layanan seperti FTP, yang dimulai pada permintaan oleh layanan yang disebut thread.

Persistent Object Stores Sebuah objek yang dijamin untuk hidup di antara aktivasi dari proses ini disebut objek terus-menerus. Objek persisten umumnya dikelola oleh penyimpan objek yang terus-menerus menyimpan keberadaanya dalam bentuk tersusun pada disk. Contoh termasuk layanan persistant objek CORBA dan Java Persistent.

Lokasi objek Bagian 4.3.3 menggambarkan suatu bentuk remot referensi obyek yang berisi Internet alamat dan nomor port dari proses yang menciptakan remote objek sebagai cara untuk menjamin keunikan. Bentuk referensi obyek terpencil juga dapat digunakan sebagai alamat untuk objek remote selama objek tetap dalam proses yang sama untuk sisa hidupnya. Tetapi beberapa remot

objek akan ada dalam serangkaian proses yang berbeda,mungkin pada komputer yang berbeda, sepanjang masa hidupnya

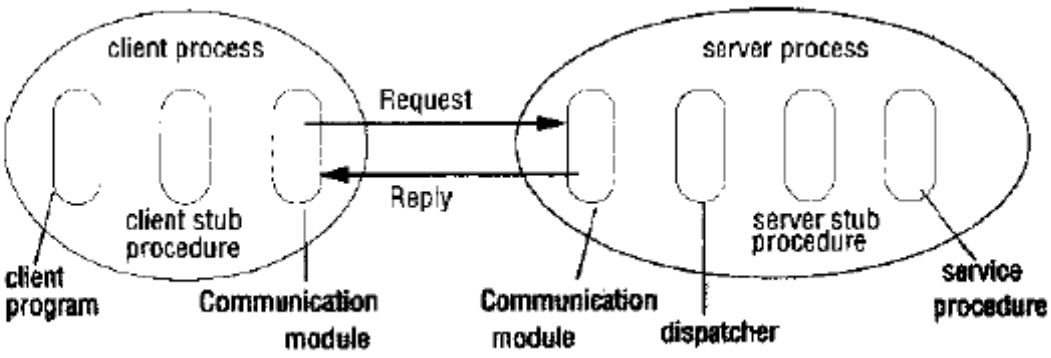
5.2.6 Distributed garbage collection

Tujuan dari sampah terdistribusi adalah untuk memastikan bahwa jika lokal atau jauh referensi ke objek masih diadakan di mana saja di satu set didistribusikan objek, maka objek secara sendirinya akan terus ada, tapi begitu tidak ada objek lagi memegang referensi untuk itu, objek akan dikumpulkan dan memori menggunakan dikembalikan seperti semula.

Leases in Jini Sistem distribusi Jini termasuk spesifikasi untuk penyewaan [Arnold et al. 1999] yang dapat digunakan dalam berbagai situasi ketika satu benda menawarkan sumber objek lain, seperti misalnya ketika remote objek referensi menawarkan alat-alat lain. Benda yang menawarkan sumber daya tersebut beresiko karena harus menjaga sumber daya ketika pengguna tidak lagi tertarik atau program mereka mungkin telah keluar. Untuk menghindari rumit protokol untuk mengetahui apakah pengguna sumberdaya masih tertarik, sumber daya yang ditawarkan untuk jangka waktu terbatas, Pemberian penggunaan sumber daya untuk jangka waktu waktu ini disebut sebagai sewa.

5.3 Prosedur Pemanggilan Remote

Sebuah panggilan prosedur remot sangat mirip dengan metode remote permintaan di klien panggilan program prosedur dalam menjalankan program lain dalam proses server. Server mungkin menjadi klien dari server lain untuk memungkinkan rantai RPCs, Seperti disebutkan dalam pendahuluan bab ini, proses server mendefinisikan dalam layanan antarmuka prosedur yang tersedia untuk panggilan jarak jauh, RPC, seperti RMI, dapat diterapkan untuk memiliki salah satu dari permintaan pilihan semantic.



Gambar 5.6 Prosedur Peran Klien dan server dalam RPC

Perangkat lunak yang mendukung RPC ditunjukkan pada Gambar di atas. Hal ini serupa dengan yang ditampilkan dalam Gambar 5.6 kecuali bahwa tidak ada referensi terpencil modul yang diperlukan, karena prosedur panggilan tidak peduli dengan benda-benda dan obyek referensi. Klien

yang mengakses layanan mencakup satu tulisan rintisan prosedur untuk masing-masing prosedur dalam layanan antarmuka. Peran prosedur tulisan rintisan mirip dengan proxy. Berperilaku seperti prosedur lokal kepada klien, tapi bukannya melaksanakan panggilan. itu marsekal prosedur pengenal dan argumen dalam pesan permintaan, yang mengirimkan komunikasi melalui modul-nya ke server.

5.3.1 Sun studi kasus RPC

RFC 1831 [Srinivasan tahun 1995] menggambarkan Sun RPC yang dirancang untuk client-server komunikasi di Sun File sistem jaringan atau NFS. Sun RPC kadang-kadang disebut ONC (Open Network Computing) RPC. Hal ini diberikan sebagai bagian dari berbagai Sun dan Sistem operasi UNIX lain dan juga tersedia dengan instalasi NFS lain, lmplementors memiliki pilihan menggunakan prosedur remote panggilan di kedua UDP atau TCP. Ketika Sun RPC digunakan dengan UDP, panjang dan membalas pesan permintaan dibatasi panjang - secara teoritis hingga 64 kilobyte, tetapi lebih sering dalam praktek untuk 8 atau 9 kilobyte

```
const MAX = 1000;
typedef int FileIdentifier;
typedef int FilePointer;
typedef int Length;
struct Data {
      int length;
      char buffer[MAX];
};
struct writeargs {
      FileIdentifier f;
      FilePointer position;
      Data data;
};
struct readargs {
      FileIdentifier f;
      FilePointer position;
      Length length;
};
program FILEREADWRITE {
   version VERSION {
      void WRITE(writeargs)=1;      1
      Data READ(readargs)=2;  2
   }=2;
} = 9999;
```

Sun RPC sistem menyediakan bahasa antarmuka yang disebut XDR dan sebuah antarmuka kompilator yang disebut rpcgen yang dimaksudkan untuk digunakan dengan bahasa C prograrnming.

Interface definition Language The Sun XDR bahasa, yang awalnya dirancang untuk menentukan data extemal representasi, diperluas untuk menjadi sebuah definisi bahasa antarmuka. Ini dapat digunakan untuk mendefinisikan antarmuka layanan bagi Sun RPC dengan menetapkan satu set

prosedur yang mendukung definisi bersama-sama dengan definisi tipe. Notasi ini agak bersifat sederhana dibandingkan dengan yang digunakan oleh CORBA IDL atau Java.

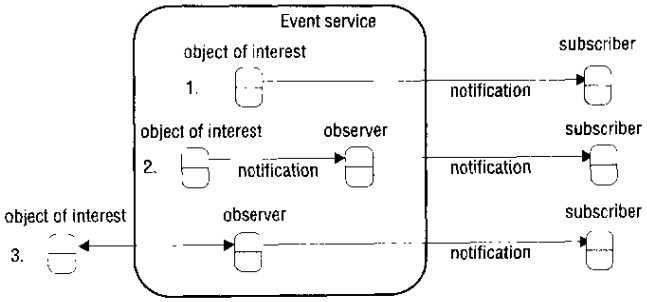
Binding Sun RPC menjalankani layanan mengikat setempat disebut mapper di port mapper nomor port pada setiap komputer. Setiap contoh dari sebuah port mapper program catatan nomor. nomor versi dan nomor port yang digunakan oleh masing-masing layanan berjalan di lokal. Kapan server dijalankan programnya itu nomor register, nomor versi dan nomor port dengan port mapper setempat.

Otentikasi pesan permintaan dan pembalasan Sun RPC menyediakan kolom tambahan yang memungkinkan informasi otentikasi akan disahkan antara klien dan server. Pesan permintaan berisi mandat dari pengguna menjalankan program klien. Sebagai contoh, dalam UNIX otentikasi termasuk kepercayaan uid dan gid dari pengguna. Akses mekanisme kontrol dapat dibangun di atas informasi otentikasi yang dibuat tersedia ke server melalui prosedur argumen kedua. Program server bertanggung jawab untuk menegakkan kontrol akses dengan memutuskan apakah akan mengeksekusi setiap prosedur panggilan menurut informasi otentikasi.

5.4 EVENT DAN NOTIFICATION

Ide dibelakang penggunaan event adalah bahwa sebuah objek bisa memberi reaksi kepada perubahan yang terjadi pada objek yang lain. Notifikasi dari event merupakan asinkronus dan tergantung penerimanya. Sistem terdistribusi berbasis event mengeksten model event local dengan mengizinkan multiple objek di lokasi berbeda untuk dinotifikasi akan event yang terjadi pada sebuah object. Ia menggunakan paradigm *publish-subscribe*, dimana sebuah objek yang menggenerate event, *publish* tipe event yang akan membuat objek lain bisa mengobservasinya. Objek yang menginginkan menerima notifikasi dari objek yang di-publish men –*subscribe* tipe even yang mereka inginkan.

Event dan notification bisa dipakai dalam aplikasi yang berbeda, sebagai contoh untuk mengkomunikasikan sebuah bentuk yang ditambahkan ke gambar, modifikasi dokumen, fakta bahwa seseorang telah masuk atau keluar dari room, atau sepotong peralatan atau buku tag elektronik pada sebuah lokasi baru.



Gambar 5.7

5.4.1 The Participants in distributed event notification

Gambar 5.7 menunjukkan arsitektur yang menentukan peran yang dimainkan oleh objek-objek yang berpartisipasi dalam sistem berbasis event yang terdistribusi. Arsitektur ini dirancang untuk men-decouple penerbit (publisher) dari pelanggan (subscriber), sehingga membuat penerbit dapat dikembangkan secara tersendiri dari pelanggan mereka, dan sejauh mungkin membatasi kerja yang dipaksakan pada penerbit oleh pelanggan. Komponen utamanya adalah sebuah layanan event yang memelihara sebuah database even yang diterbitkan dan keinginan pelanggan. Event pada sebuah objek yang diinginkan diterbitkan di layanan event. Pelanggan memberitahu layanan event (event service) tentang jenis kegiatan yang mereka ingin lakukan. Ketika suatu peristiwa atau event terjadi di objek yang diperlukan atau diinginkan, pemberitahuan dikirim ke pelanggan untuk jenis dari event. Peran objek yang berpartisipasi adalah sebagai berikut:

- *Objek yang diinginkan(object of interest):* Ini adalah sebuah objek yang mengalami perubahan keadaan, sebagai akibat dari operasi yang diajukan. Perubahan keadaan ini memungkinkan untuk dibutuhkan objek lainnya. Deskripsi ini memungkinkan untuk event-event seperti orang yang memakai lencana aktif memasuki ruangan, dalam hal ini ruangan adalah objek kepentingan dan operasi terdiri dari penambahan informasi tentang orang baru ke catatan yang ada di ruangan. Objek yang diinginkan dianggap sebagai bagian dari layanan event jika ia mentransmisikan suatu notifikasi.

- *Event:* Suatu kejadian terjadi pada sebuah objek yang menarik sebagai akibat dari penyelesaian metode eksekusi.
- *Notification* : Sebuah notification adalah obyek yang berisi informasi tentang suatu peristiwa. Biasanya, berisi jenis acara dan atributnya, yang umumnya meliputi identitas objek, methode yang diterapkan, dan waktu kejadian atau nomor urut.
- *Subscriber* : Pelanggan Sebuah pelanggan adalah obyek yang telah berlangganan beberapa jenis peristiwa di objek lain. Ini menerima pemberitahuan tentang kejadian tersebut.
- *Object Observer*: Tujuan utama dari seorang pengamat adalah objek decouple kepentingan dari para pelanggan atau subscriber . Sebuah objek menarik dapat memiliki banyak berbeda pelanggan dengan berbagai kepentingan. Sebagai contoh, pelanggan mungkin berbeda untuk yang jenis peristiwa yang mereka minati, atau yang berbagi persyaratan yang sama untuk jenis mungkin berbeda dalam nilai atribut yang menarik.
- *Publisher* : Ini adalah obyek yang menyatakan bahwa hal itu akan menghasilkan pemberitahuan jenis acara tertentu. Penerbit bisa menjadi objek kepentingan atau seorang pengamat atau observer. Gambar 5.7 menunjukkan tiga kasus:

1. Sebuah objek yang menarik di dalam layanan peristiwa tanpa seorang pengamat. lt mengirim pemberitahuan langsung kepada pelanggan.

2. Sebuah objek yang menarik di dalam pelayanan event dengan pengamat. Objek minat mengirimkan pemberitahuan melalui pengamat ke pelanggan.
3. Sebuah objek menarik di luar layanan event. Dalam kasus ini, seorang pengamat query objek yang menarik untuk menemukan ketika peristiwa terjadi. Pengamat mengirim pemberitahuan kepada pelanggan.

Delivery semantics o Berbagai jaminan pengiriman yang berbeda dapat disediakan untuk pemberitahuan - salah satu yang harus dipilih tergantung pada persyaratan aplikasi. Sebagai contoh, jika IP multicast yang digunakan untuk mengirimkan pemberitahuan kepada sekelompok penerima, yang kegagalan model akan berhubungan dengan yang dijelaskan untuk IP multicast, tetapi tidak menjamin bahwa penerima tertentu akan menerima pesan pemberitahuan tertentu. Ini cukup untuk beberapa aplikasi, misalnya untuk menyampaikan kondisi terakhir pemain dalam sebuah permainan internet.

Roles for Observers O Walaupun notifikasi bisa dikirim langsung dari objek kepada penerima, tugas pemberitahuan pengolahan dapat dibagi di antara proses pengamat memainkan berbagai peran yang berbeda. Ada beberapa contoh, diantaranya :

- *Forwarding* : Seorang pengamat forwarding dapat melaksanakan semua pekerjaan pengiriman notifications untuk subscriber atas nama satu atau lebih objek. Semua objek perlu lakukan adalah untuk mengirimkan pemberitahuan kepada pengamat forwarding, meninggalkan untuk lanjutkan dengan tugas normal, Untuk menggunakan forwarding pengamat, sebuah objek menyampaikan informasi tentang subscriber.
- *Filtering of notification* : Penyaring dapat diterapkan oleh pengamat sehingga mengurangi jumlah notifications yang diterima menurut beberapa predikat yang di isi setiap pemberitahuan. Sebagai contoh, suatu peristiwa yang mungkin berhubungan dengan penarikan dari rekening bank, namun penerima hanya tertarik pada mereka yang lebih besar dari $ 100. Pattern of events : Ketika sebuah objek subscribes peristiwa pada kepentingan object. mereka dapat menentukan pola peristiwa yang mereka tertarik menetapkan Sebuah pola hubungan antara beberapa peristiwa. Sebagai contoh, seorang pelanggan mungkin tertarik bila ada tiga tipe penarikan dari rekening bank tanpa campur tangan deposit. Persyaratan serupa adalah untuk mengkorelasikan kejadian di berbagai objek kepentingan.
- *Notifications mailboxes* : Dalam beberapa kasus, pemberitahuan harus ditunda sampai pelanggan potensial siap untuk menerima mereka, Sebagai contoh, pelanggan telah rusak sambungan atau ketika sebuah objek pasif dan telah diaktifkan kembali. An pengamat dapat mengambil peran kotak surat pemberitahuan, yang adalah untuk menerima pemberitahuan atas nama pelanggan, hanya melewati mereka di (dalam satu batch) ketika pelanggan siap untuk menerima mereka. Pelanggan harus dapat giliran pengiriman dan turun seperti yang diperlukan. Pelanggan mendirikan kotak surat pemberitahuan ketika register dengan objek dengan menentukan kotak surat pemberitahuan sebagai tempat untuk mengirim pemberitahuan.

5.4.2 Jini Distributed event specification

Jini Distributed event specification yang dijelaskan oleh Arnold et of. [1999] memungkinkan pelanggan potensial dalam satu Java Virtual Machine (JVM) untuk berlangganan ke dan menerima pemberitahuan peristiwa dalam sebuah objek yang menarik di JVM lain, biasanya di lain komputer. Objek utama yang terlibat dalam Jini Distributed event specification adalah:

- *Event generator*: Sebuah event generator adalah obyek benda-benda lain yang memungkinkan untuk berlangganan ke peristiwa dan menghasilkan pemberitahuan.
- *Remote event Listeners:* adalah obyek yang dapat menerima pemberitahuan.
- *Remote event* : adalah obyek yang melewati nilai rs ke remote event pendengar. Sebuah remote event adalah sama dengan apa yang kita disebut notifications.
- *Third Party Agent* : agen pihak ketiga mungkin sela antara objek interest dan pelanggan. Mereka adalah setara pengamat. Sebuah objek subscribes peristiwa dengan memberitahukan peristiwa generator tentang jenis acara dan menetapkan acara remote pendengar sebagai target untuk pemberitahuan.
- *RemoteEventListener*: ini interface memberikan sebuah metode yang disebut notify. Pelanggan dan third party agent mengimplementasikan antarmuka RemoteEventListener ehingga mereka dapat menerima pemberitahuan bila metode memberitahukan dipanggil. Turunan dari Mewakili kelas RemoteEvent pemberitahuan dan lulus sebagai argumen untuk notify method.
- *RemoteEvent* : Kelas ini memiliki instance variable yang berlaku:

    1. referensi ke acara generator di mana peristiwa itu terjadi.
    2. peristiwa identifier, yang menentukan jenis aktivitas pada acara generator.
    3. sebuah nomor urut, yang berlaku untuk peristiwa tipe. Nomor urutan harus meningkat dari waktu ke waktu peristiwa terjadi. Ini dapat digunakan untuk memungkinkan penerima untuk peristiwa urutan jenis tertentu dari sumber tertentu atau untuk menghindari menerapkan peristiwa yang sama dua kali.
    4. sebuah marshalled objek. Ini diberikan saat penerima tipe mengirimkan aktivitas dan dapat digunakan oleh penerima untuk tujuan apapun. Hal ini biasanya memegang apapun informasi yang diperlukan oleh penerima untuk mengidentifikasi acara dan bereaksi terhadap kejadian. Sebagai contoh, bisa mencakup penutupan yang akan dijalankan ketika diberitahu.
- *EventGenerator* : interface ini memberikan sebuah metode yang disebut register. Event generator mengimplementasikan antarmuka EventGeneraror, yang menggunakan method mendaftar untuk berlangganan untuk kejadian di event generator. Argumen untuk meregister ditentukan:

    1. event identifier, yang menentukan jenis aktivitas.
    2. marshalled objek yang akan diserahkan kembali dengan setiap pemberitahuan.
    3. remote referensi ke aktivitas pendengar objek, tempat untuk mengirim pemberitahuan.

4. periode penyewaan yang diminta. Masa sewa guna usaha menentukan durasi sewa guna usaha dibutuhkan oleh pelanggan.

Third Party Agent o pihak ketiga agen yang berada disela antara acara generator dan seorang pelanggan dapat memainkan berbagai peran yang berguna, termasuk semua orang dijelaskan di atas. Dalam kasus yang paling sederhana, sebuah register pelanggan kepentingan dalam jenis aktivitas tertentu di peristiwa generator dan menentukan dirinya sebagai pendengar remote event. Ini sesuai kasus diilustrasikan pada Gambar 5.7

I'hird pihak agen dapat diatur oleh sebuah event generator atau oleh pelanggan. Suatu event generator dapat menempatkan satu atau lebih pihak ketiga agen antara dirinya dan pelanggan. Sebagai contoh, event generator pada setiap komputer dapat memanfaatkan dari pihak ketiga. Pihak ketiga agen dapat mengambil tanggung jawab untuk memperbarui penyewaan.

5.5 JAVA RMI CASE STUDY

Java RMI Jawa memperluas model obyek untuk memberikan dukungan bagi distributed objek dalam Bahasa java. Secara khusus, hal itu memungkinkan objek untuk memanggil method pada objek remote dengan menggunakan sintaks yang sama seperti untuk local invocation. Selain itu, pemeriksaan pengetikan berlaku sama untuk remote invokasi untuk lokal. Namun, sebuah remote objek membuat invokasi harus mengerti bahwa ada target yang diremote, karena harus menangani RemoteExceptions dan pelaksana objek jauh harus mengerti bahwa itu jauh karena harus melaksanakan Remote interface. Meskipun didistribusikan model objek Java diintegrasikan ke dalam cara alami, yang semantik melewati parameter berbeda karena INVOKER dan sasaran yang jauh dari satu sama lain.

Pemrograman aplikasi terdistribusi di Jawa RMI harus relatif sederhana karena merupakan satu-sistem bahasa - remote interface didefinisikan di bahasa java. Jika sebuah Multiplelanguage sistem seperti CORBA digunakan, kebutuhan programmer untuk belajar suatu IDL dan untuk memahami bagaimana peta ke bahasa pemrograman. Akan tetapi, bahkan dalam satu-sistem bahasa, programmer objek jarak jauh harus mempertimbangkan perilaku dalam lingkungan bersamaan.

Dalam studi kasus ini , kita gunakan bersama papan tulis sebagai contoh. Ini adalah program yang didistribusikan sekelompok memungkinkan pengguna untuk berbagi pandangan umum dari permukaan gambar berisi objek grafis, seperti persegi panjang, garis dan lingkaran, masing-masing yang telah ditarik oleh salah satu pengguna. server mempertahankan keadaan saat ini gambar dengan menyediakan operasi bagi klien untuk menginformasikan hal tentang bentuk terbaru pengguna mereka telah diambil dan membuat catatan dari semua bentuk yang telah diterima. Server juga menyediakan operasi memungkinkan klien untuk mengambil bentuk terbaru yang diambil oleh pengguna lain dengan pemungutan suara server. Server memiliki versi nomor (integer) yang bertahap setiap kali

bentuk baru datang dan melekat ke bentuk baru. Operasi menyediakan server memungkinkan klien untuk menanyakan tentang versi nomor dan nomor versi dari setiap bentuk, sehingga mereka dapat menghindari mengambil bentuk yang sudah mereka miliki.

```
import java.rmi.*;
import java.util.Vector;
public interface Shape extends Remote {
        int getVersion() throws RemoteException;
        GraphicalObject  getAllState() throws RemoteException;    1
}
public interface ShapeList extends Remote {
        Shape newShape(GraphicalObject g) throws RemoteException;      2
        Vector allShapes() throws RemoteException;
        int getVersion() throws RemoteException;
}
```

Gambar 5.8

Remote Interface in java RMI o Remote interface didefinisikan dengan memperluas sebuah interface yang disebut Remote java.rmi disediakan dalam paket. Method ini harus membuang RemoteException, tapi aplikasi pengecualian khusus juga mungkin dilemparkan. Gambar 5.ll menunjukkan contoh dua antarmuka remote disebut Shape dan ShapeList. Dalam contoh ini, GraphicalObject adalah sebuah kelas yang memegang suatu state objek grafis, misalnya jenisnya, posisinya, menutupi persegi panjang, garis warna dan sampai warna, dan menyediakan operasi untuk mengakses dan meng-update dengan sebuah state. GraphicalObject harus melaksanakan Serializable interface. Pertimbangkan Shape antarmuka pertama: getVersion method yang mengembalikan sebuah integer,

Sedangkan metode getAllState mengembalikan sebuah instance dari kelas `Graphical Objek. Sekarang mempertimbangkan Shapelist antarmuka dengan method newShape melewati sebuah instance dari Graphica;Object sebagai argumen, tetapi kembali sebuah objek dengan remote interface (yaitu, sebuah remote object) sebagai hasilnya. Hal penting yang perlu dicatat adalah bahwa kedua objek biasa dan objek remote dapat muncul sebagai argumen dan hasil dalam antarmuka remote. Yang terakhir selalu dilambangkan dengan nama antarmuka remote mereka. Dalam poin berikutnya, kita bahas bagaimana benda biasa dan terpencil berlalu objek sebagai argumen dan hasil.

Parameter dan result passing o Di Java RM1, parameter metode diasumsikan menjadi masukan parameter dan hasil dari metode ini adalah satu parameter output. Bagian sebelumnya Java menggambarkan serialisasi, yang digunakan untuk rnarshalling argumen dan hasil di Java RMI.

- *Passing remote object* : Bila jenis parameter atau nilai hasil didefinisikan sebagai remote interface, argumen yang sesuai atau hasil selalu lulus sebagai remote referensi obyek. Sebagai contoh pada Gambar 5.8. baris 2, nilai pengembalian method didefinisikan sebagai newShape Shape - antarmuka remote. Ketika referensi objek remote diterima, dapat digunakan untuk membuat panggilan RMI pada remote objek yang dirujuknya.

- *Passing non-remote Object* : Semua Serializable non-remote objek yang disalin dan melewati nilai. Ketika sebuah objek ini dilalui oleh nilai, baru objek diciptakan dalam proses penerima. Metode objek baru ini dapat ia dipanggil secara lokal, dan dapat menyebabkan state yang berbeda dari keadaan objek asli dalam proses pengiriman.

Downloading of classess O Java dirancang untuk memungkinkan kelas untuk di-download dari satu mesin virtual yang lain. Hal ini terutama relevan untuk didistribusikan objek yang digunakan sebagai sarana berkomunikasi remote invokasi. Kita telah melihat bahwa non-remote objek melewati nilai dan objek jauh di pass by reference sebagai argumen dan hasil dari RMIs. Jika penerima tidak sudah memiliki kelas objek lewat nilai. kode akan didownload secara otomatis. Demikian pula, jika penerima objek jauh referensi tidak sudah memiliki kelas untuk proxy, kodenya di-download secara otomatis. Hal Ini memiliki dua keuntungan:

- Tidak perlu untuk setiap pengguna untuk menyimpan set yang sama kelas dalam kerja merekalingkungan.
- Kedua program client dan server dapat membuat transparan menggunakan contoh baru kelas setiap kali mereka ditambahkan.

RMI registry O RMIregistry adalah binder untuk Jawa RMI. Sebuah contoh dari RMl registry harus berjalan pada setiap server host komputer yang jauh objek. hal memelihara meja pemetaan tekstual, URL-gaya nama-nama untuk referensi ke objek remote host pada komputer. Hal ini diakses oleh Penamaan metode dari kelas, metode yang mengambil sebagai URL-argumen string diformat dalam bentuk:

//computerName : port/objectName

ComputerName dan port di mana merujuk ke lokasi RMIregistry. Jika mereka dihilangkan, komputer lokal dan port diasumsikan dalam kondisi default. Hal tersebut menawarkan antarmuka metode yang ditunjukkan pada Gambar 5. 9, di mana pengecualian tidak terdaftar - semua metode bisa melempar RemoteExt'eprion. Layanan ini bukan untuk seluruh sistem layanan mengikat. Klien harus mengarahkan pertanyaan lookup host tertentu.

void rebind (String name, Remote obj)
This method is used by a server to register the identifier of a remote object by name, as shown in Figure 15.13, line 3.
void bind (String name, Remote obj)
This method can alternatively be used by a server to register a remote object by name, but if the name is already bound to a remote object reference an exception is thrown.
void unbind (String name, Remote obj)
This method removes a binding.
Remote lookup(String name)
This method is used by clients to look up a remote object by name, as shown in Figure 15.15 line 1. A remote object reference is returned.
String [] list()
This method returns an array of Strings containing the names bound in the registry.

Gambar 5.9

5.5.1 Building Client and server programs

Bagian ini menguraikan langkah-langkah yang diperlukan untuk menghasilkan program klien dan server yang menggunakan Remote interfaces Shape dan ShapeList ditunjukkan pada Gambar 5.11. Program server merupakan sebuah versi sederhana dari whiteboard server yang mengimplementasikan dua antarmuka Shape dan ShapeList. Kita menggambarkan sebuah program klien polling sederhana dan kemudian memperkenalkan teknik panggil balik (callback) yang dapat digunakan untuk menghindari kebutuhan untuk polling server.

Program server. Server adalah sebuah whiteboard server: ia mewakili masing-masing bentuk sebagai sebuah remote obyek yang mengimplementasikan antarmuka Shape dan memegang keadaan suatu objek grafis sebaik nomor versinya; ia mewakili koleksi bentuknya oleh objek yang jauh yang mengimplementasikan antarmuka ShapeList dan memegang koleksi dalam bentuk vektor.

Server terdiri dari sebuah metode utama (main method) dan sebuah kelas pelayan (servant class) untuk melaksanakan masing-masing remote interface. Metode utama dari server menciptakan sebuah instance dari ShapeListServant dan mengikatnya ke sebuah nama dalam RMIregistry, seperti ditunjukkan pada Gambar 5.13 (baris 1 dan 2). Dua kelas servant atau pembantu adalah ShapeListServant, yang mengimplementasikan antarmuka ShapeList, dan ShapeServant, yang mengimplementasikan antarmuka Shape.

```
import java.rmi.*;
public class ShapeListServer{
        public static void main(String args[]){
                System.setSecurityManager(new RMISecurityManager());
                 try{
                        ShapeList aShapeList = new ShapeListServant();                  1
                                Naming.rebind("Shape List", aShapeList );               2
                        System.out.println("ShapeList server ready");
                }catch(Exception e) {
                        System.out.println("ShapeList server main " + e.getMessage());}
        }
}
```

Gambar 5.10

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.util.Vector;
public class ShapeListServant extends UnicastRemoteObject implements ShapeList {
         private Vector theList;                 // contains the list of Shapes        1
         private int version;
        public ShapeListServant()throws RemoteException{...}
        public Shape newShape(GraphicalObject g) throws RemoteException {      2
                version++;
                Shape s = new ShapeServant( g, version);                        3
                theList.addElement(s);
                return s;
        }
        public  Vector allShapes()throws RemoteException{...}
        public int getVersion() throws RemoteException { ... }
}
```

Gambar 5.11

Gambar 5.11 memberikan garis besar kelas ShapeListServant. Perhatikan bahwa ShapeListServant (baris 1), seperti banyak kelas pelayan, extends kelas bernama UnicastRemoteObject, yang menyediakan remote object yang hidup hanya sepanjang proses di mana mereka diciptakan.

Implementasi metode antarmuka remote di kelas pelayan sepenuhnya secara langsung karena mereka dapat dilakukan tanpa kepedulian untuk rincian komunikasi. Pertimbangkan metode newShape di Gambar 5.11 (baris 2), yang bisa disebut metode pabrik karena memungkinkan klien untuk meminta

pembentukan remote objek. Menggunakan constructor dari ShapeServant, yang menciptakan remote objek baru yang berisi GraphicalObject dan nomor versi yang lulus sebagai argumen. Jenis nilai kembali atau return dari newShape adalah Shape – antarmuka yang dilaksanakan oleh remote objek baru. Sebelum kembali, metode rrewShape menambahkan bentuk baru ke vektor yang berisi bentuk daftar (baris 3).

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.Vector;
public class ShapeListClient{
   public static void main(String args[]){
        System.setSecurityManager(new RMISecurityManager());
        ShapeList aShapeList = null;
        try{
                aShapeList  = (ShapeList) Naming.lookup("//bruno.ShapeList")     ;        1
                Vector sList = aShapeList.allShapes();                           2
        } catch(RemoteException e) {System.out.println(e.getMessage());
        }catch(Exception e) {System.out.println("Client: " + e.getMessage());}
   }
}
```

Gambar 5.12

Metode utama dari sebuah server memerlukan penciptaan manajer keamanan untuk mengaktifkan keamanan Java untuk menerapkan perlindungan yang sesuai untuk server RMI. Manajer keamanan default disebut RMISecurityManager telah disediakan. Melindungi sumber daya lokal untuk memastikan bahwa kelas-kelas yang diambil dari situs remote tidak dapat memiliki efek pada sumber daya seperti file, tetapi berbeda dalam program yang memungkinkan untuk memberikan kelas loader sendiri dan menggunakan refleksi. Jika server RMI tidak diset manajer keamanan, proxy dan kelas-kelas hanya dapat diambil dari classpath lokal, dalam rangka untuk melindungi program dari kode yang di-download sebagai hasil dari metode remote invocation.

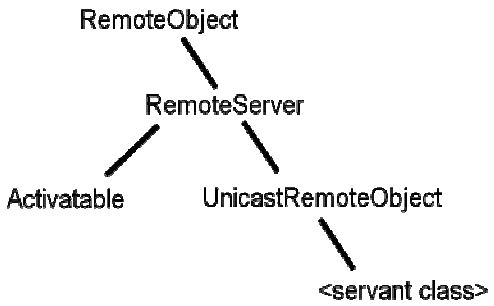
Program Klien. Sebuah klien yang disederhanakan untuk server ShapeList diilustrasikan pada Gambar 5.12. Setiap program klien perlu untuk memulai dengan menggunakan map untuk mencari referensi remote obyek. Klien menetapkan manajer keamanan dan lalu mendongak referensi remote objek untuk remote objek menggunakan operasi pencarian dari RMIregistry (baris l). Setelah memperoleh referensi objek remote awal, klien berlanjut dengan mengirimkan RMIs untuk objek yang jauh atau kepada orang lain yang ditemukan selama pelaksanaannya sesuai dengan kebutuhan penerapannya.

Callback. Ide umum di belakang callback adalah bahwa selain polling klien ke server untuk mengetahui apakah suatu peristiwa telah terjadi, server harus menginformasikan kliennya kapanpun event telah terjadi. Istilah callback digunakan untuk merujuk ke tindakan server untuk memberitahukan klien tentang sebuah event. Callback dapat diimplementasikan di RMI sebagai berikut:

- Klien menciptakan remote obyek yang mengimplementasikan antarmuka yang berisi metode untuk server untuk memanggil. Kita menyebutnya sebagai objek panggil balik (callback objecy).
- Server menyediakan sebuah operasi yang memungkinkan klien yang menginginkan untuk menginformasikan hal itu dari referensi objek remote callback mereka objek. Ia mencatatnya dalam sebuah daftar.
- Ketika suatu peristiwa yang diinginkan terjadi, server memanggil klien tertarik. Untuk contoh, server whiteboard akan memanggil kliennya setiap kali sebuah objek grafis ditambahkan.

Penggunaan callback menghindari kebutuhan klien untuk meminta atau poll objek yang diinginkan di server dan kerugiannya:

- Kinerja server dapat terdegradasi oleh polling yang konstan.
- Klien tidak dapat memberitahu pengguna tentang perbaruan pada waktu yang tepat.



Gambar 5.13

5.5.2 Desain dan Implementasi Java HMI

Sistem Java RMI yang asli menggunkan semua komponen yang ditunjukkan pada Gambar 5.6. Tetapi dalam Java 1.2, fasilitas refleksi digunakan untuk membuat operator umum dan untuk menghindari kebutuhan kerangka. Proxy klien dihasilkan oleh kompilator yang disebut *rmic* dari kelas server terkompilasi - bukan dari definisi dari antarmuka remote.

Penggunaan Refleksi. Refleksi digunakan untuk menyampaikan informasi dalam permintaan pesan tentang metode yang akan dipanggil. Hal ini dicapai dengan bantuan kelas Method dalam paket refleksi. Setiap instance Method mewakili karakteristik metode tertentu, termasuk kelas, jenis

argumen, niai return dan pengecualian. Fitur paling menarik dari kelas ini adalah sebuah instance dari Method dapat dipanggil pada objek kelas yang sesuai dengan menggunakan metode yang memanggil. Metode yang dipanggil memerlukan dua argumen: pertama menentukan objek yang menerima invokasi dan yang kedua adalah sebuah Obyek array yang berisi argumen. Hasilnya adalah kembali sebagai jenis Obyek.

Kelas Java mendukung RMI. Gambar 5.16 menunjukkan struktur turunan kelas mendukung Java RMI server. Satu-satunya kelas yang perlu programer sadari adalah UnicastRemoteObjecr, yang setiap kelas pelayan sederhana dibutuhkan untuk di-extend. Kelas UnicastRemoteObject mengeksten sebuah kelas abstrak yang disebut RernoteServer, yang menyediakan versi abstrak metode-metode yang diperlukan dengan remote server. UnicastRemoteObject ini contoh pertama dari RemoteServer yang akan diberikan. Yang lain disebut Activatable sekarang tersedia untuk menyediakan objek activatable. Lebih lanjut mungkin menyediakan alternatif direplikasi objek. Kelas RernoteServer adalah subclass dari RemoteObject yang memiliki variable instance memegang referensi remote obyek dan menyediakan metode berikut:

- *equals*, metode ini membandingkan referensi objek remote;
- *toString*: metode ini memberikan isi dari referensi objek remote sebagai sebuah String;
- *readObject, writeObject*: metode ini men-deserialize/serialize remote objek.

Selain itu, operator instanceOf dapat digunakan untuk menguji objek jauh.