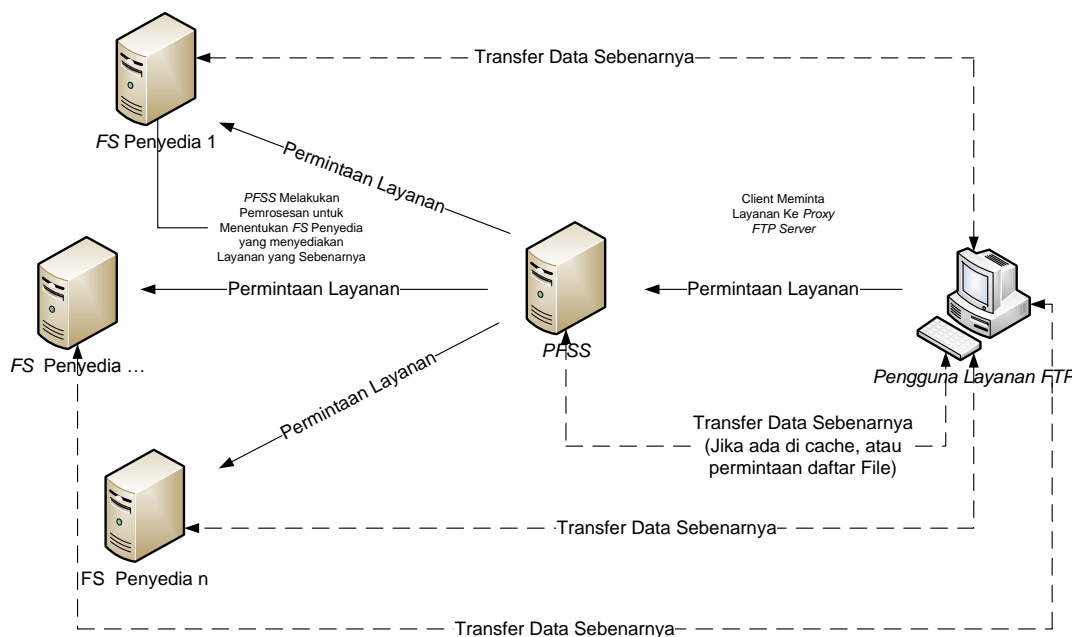


# Bab III

## ANALISIS

### 3.1 Deskripsi Umum Perangkat Lunak

Perangkat lunak *PFS* yang akan dikembangkan akan diberi nama *Proxy FTP Server Software*, yang kemudian akan diacu sebagai *PFSS*, yang merupakan kependekan dari nama perangkat lunak. Perangkat lunak *PFSS* ini berfungsi sebagai perantara antara sekumpulan *FTP server* penyedia dan pengguna layanan *FTP* dalam mengakses layanan *FTP*. Melalui *PFSS* ini, pengguna akan meminta layanan *FTP*, kemudian *FS* yang menyediakan layanan *FTP* yang sebenarnya akan memberi layanannya, dan data sebagai entitas layanan akan lewat.



Gambar III-1 Model Layanan

### 3.2 Spesifikasi Perangkat Lunak

Berikut adalah spesifikasi yang harus dipenuhi oleh perangkat lunak.

1. Dapat melakukan *download* dan *upload resource* pada *FS* penyedia melalui *PFSS*.

2. Dapat melihat daftar *resource* pada *FS* penyedia melalui *PFSS*.
3. Dapat melakukan administrasi *FS* penyedia, yaitu penambahan, penghapusan, dan pengubahan data *FS* penyedia secara manual.
4. Dapat melakukan fungsi *routing* untuk meminta layanan (*download, upload,* dan lihat daftar *resource*). Proses ini dilakuka *PFSS*, dan *routing* untuk memberikan layanan diantara sekumpulan *FS* penyedia yang terdaftar di *PFSS*.
5. Dapat menyediakan tambahan kehandalan dan ketersediaan layanan *FTP* pada *FS* penyedia melalui *FS* dengan adanya *caching* dan redundansi.
6. Dapat melakukan pembatasan akses dengan melakukan penyaringan (*filtering*) layanan berdasarkan alamat IP.
7. *FS* penyedia tidak perlu mengetahui bahwa layanan *FTP* yang digunakan oleh pengguna melalui *PFSS* atau secara langsung.
8. Menyediakan layanan multi-pengguna untuk layanan *FTP* di *PFSS*.

### **3.3 Batasan Perangkat Lunak**

Berikut adalah batasan perangkat lunak.

1. Pada pengguna *PFSS*, mode transfer yang didukung adalah hanya mode aktif. Hal dikarenakan protokol *FTP* tidak dapat melakukan *redirecting* alamat transfer data.
2. Representasi tipe data yang didukung adalah ASCII dan *image*, dengan kontrol format adalah *nonprint*, struktur adalah struktur *file*, dan mode transmisi adalah *stream*.
3. Tidak mendukung IPv6.
4. Layanan *FTP* di *PFSS* dapat langsung diakses oleh pengguna dan layanan *FTP* di *FS* yang dikelola oleh *PFSS* dapat langsung diakses oleh *PFSS*. Hal ini berarti tidak terdapat *proxy, firewall,* atau *middle application* yang sejenis yang menghubungkan antara *server* dan *client*.

### **3.4 Analisis Permasalahan**

Proses yang dilakukan dalam analisis untuk menyelesaikan permasalahan pengembangan *PFSS* meliputi , yaitu :

1. Analisis layanan *proxy* (penerusan *file* dan/atau direktori) dan transparansi layanan *FS*.
2. Analisis pengelolaan data yang disediakan *FS* penyedia.
3. Analisis pengelolaan *FS* Penyedia yang menyediakan layanan *FTP* melalui *PFSS*.
4. Analisis penanganan dinamika dan kehandalan layanan *FTP* melalui *PFSS*.
5. Analisis otentikasi.

Dari hasil analisis ini, kemudian akan menjadi acuan dalam pendefinisian arsitektur, implementasi *PFSS* dan teknologi yang digunakan dalam implementasi.

### **3.5 Analisis Layanan Proxy (Penerusan File dan/atau Direktori) dan Transparansi Layanan FS**

Pada perangkat lunak ini terdapat layanan *proxy*. Layanan *proxy* pada layanan *FTP* yang diberikan bersifat transparan bagi pengguna, dimana pengguna tidak mengetahui *FS* mana yang memberikan layanan, tetapi hanya mengetahui bahwa ia meminta dan menerima layanan melalui *PFSS*. Untuk memberikan tranparansi ini, digunakan *URI*. *URI* (*Universal Resource Identifier*) adalah penamaan *resource* unik yang dapat diakses melalui internet. *URI* berbentuk :

```
[nama-protokol]://[nama-host]/[direktori-atau-nama-resource-spesifik-host]
```

Dengan menggunakan *URI* ini, pengguna hanya perlu mengakses *URI* di layanan *PFSS*, kemudian *PFSS* akan melakukan pemrosesan, kemudian memutuskan *FS* yang akan melayani permintaan layanan pengguna, dan menangani proses pentransferan data ke pengguna, sehingga pengguna hanya mengetahui bahwa *PFSS* yang memberi layanan *FTP* ke pengguna.

Pada sisi penyedia layanan *FTP*, terdapat satu atau lebih *FS* yang menyediakan layanan *FTP* sebenarnya, layanan yang diberikan juga bersifat transparan, yang berarti layanan ini diberikan ke pengguna melalui *PFSS* dan pengguna *FTP* tidak tahu *FS* mana yang sebenarnya memberi layanan *FTP* kepadanya. Jadi *FS* hanya memberikan layanan ke *PFSS* yang kemudian akan meneruskannya ke pengguna.

Dalam menyediakan layanan *FTP* kepada pengguna, terdapat dua alternatif cara yang dapat digunakan *PFSS*, yaitu :

1. Dengan melewati semua *resource* yang ditransfer antara *client* dan *FS* penyedia, melalui *PFSS*.
2. Dengan melakukan *redirect* permintaan dari *client* ke *PFSS*, kemudian di *redirect* ke *FS* penyedia.

Pada pendekatan pertama, semua data yang ditransfer antara *FS* penyedia dan pengguna akan melewati *PFSS*, hal ini akan menjadi salah satu sumber *bottleneck* dalam transmisi data, hal ini disebabkan karena setiap transmisi tentu akan memakan *resource* di *PFSS*, sehingga dapat terjadi *PFSS* tidak dapat melayani lagi permintaan layanan pengguna, karena tidak sanggup, padahal *FS* penyedia sanggup memberikannya. Keuntungan dari pendekatan ini, adalah mudahnya mengontrol arus data.

Sedangkan pada pendekatan kedua, terdapat keuntungan bahwa selama *FS* penyedia sanggup menyediakan layanan, maka pengguna dapat menggunakan layanannya, berbeda dengan pendekatan pertama. *PFSS* hanya perlu melakukan proses pemetaan dan kemudian memberikan alamat data bagi pengguna, yang proses hal ini tidak disadari oleh pengguna secara langsung, tetapi telah ditangani oleh perangkat lunak *FC*. Sehingga hal ini tidak memakan *resource* yang besar pada *PFSS*, dan *PFSS* bisa fokus pada proses *mapping*. Selain itu, keuntungan yang didapat dari pendekatan pertama juga didapat pada pendekatan ini, yaitu dapat dengan mudah mengatur arus data.

Maka dari hasil perbandingan diatas, pendekatan kedua dipilih untuk digunakan dalam pengembangan perangkat lunak *PFSS*.

### 3.6 Analisis Pengelolaan Data yang Disediakan FS Penyedia

Pada perangkat lunak ini diperlukan proses pemetaan antara *resource* yang disediakan di *PFSS* dan *resource* di *FS* penyedia. Proses pemetaan dideskripsikan sebagai berikut :

1. *PFSS* mengambil daftar *resource* dari semua *FS* penyedia.
2. *PFSS* akan mengorganisasi presentasi struktur direktorinya sendiri (dengan menyimpan data pemetaan antara struktur direktori internal *PFSS* dengan struktur asli di *FS* penyedia).

Pada tahap kedua, dapat terjadi masalah dalam konflik penamaan *resource*. Dalam struktur penamaan direktori dan *file* diusahakan akan sama dengan direktori dan *file* penyedia. Jika terdapat kesamaan nama pada suatu level struktur direktori atau *file*, maka akan dibedakan secara otomatis oleh *PFSS*. Dalam melakukan perbedaan ini, terdapat dua cara, yaitu :

1. Perbedaan seawal mungkin, yang berarti pada tingkat direktori.

Hal ini berarti bahwa perbedaan dilakukan berdasarkan perbedaan yang terdapat pada struktur sistem *file* terluar.

Sebagai contoh :

Terdapat dua *FS* penyedia, dengan penamaan direktori yang sama pada tingkat *root* layanan *FTP* (/), yaitu direktori *incoming*, dan terdapat *file* yang bernama sama pada direktori tersebut

```
ftp://a.itb.ac.id/incoming/
```

```
ftp://b.itb.ac.id/incoming/
```

maka, sebagai contoh, oleh *PFS* akan diorganisasi menjadi:

```
ftp://GFS.itb.ac.id/incoming_a.itb.ac.id/
```

```
ftp://GFS.itb.ac.id/incoming_b.itb.ac.id/
```

2. Perbedaan seakhir mungkin, yang berarti pada tingkat *file*.

Hal ini berarti dari kesamaan struktur direktori atau *file*, yang terdalam (berarti bahwa penamaan ulang hanya dilakukan terhadap *file*).

Sebagai contoh, terdapat dua *FS* penyedia, dengan penamaan direktori yang sama pada tingkat root layanan *FTP* (/), yaitu direktori *incoming*, dan terdapat *file* yang bernama sama pada direktori tersebut

```
ftp://a.itb.ac.id/incoming/
```

```
ftp://b.itb.ac.id/incoming/
```

maka, sebagai contoh, oleh *PFSS* akan diorganisasi menjadi:

```
ftp://GFS.itb.ac.id/incoming/
```

dan,

```
ftp://a.itb.ac.id/incoming/data.txt
```

```
ftp://b.itb.ac.id/incoming/data.txt
```

maka, sebagai contoh, oleh *PFSS* akan diorganisasi menjadi:

```
ftp //GFS.itb.ac.id/incoming/data_a.itb.ac.id.txt
```

```
ftp //GFS.itb.ac.id/incoming/data_b.itb.ac.id.txt
```

Pada pendekatan pertama, terdapat kemudahan dalam melakukan pemetaan, karena tinggal melakukan perbedaan pada persamaan yang ditemukan sedini mungkin (dapat pada *file* atau direktori), sehingga, pengecekan dilakukan lebih sedikit dari pendekatan kedua. Selain itu proses pemetaan lebih sederhana. Tetapi sulit untuk mendukung *caching* dan penemuan *file* yang sama di *server* yang berbeda, karena redundansi cenderung dilakukan oleh pengguna, karena perbedaan dilakukan di awal, yang berarti perbedaan dilakukan sejak ada direktori yang sama.

Pada pendekatan kedua, lebih susah dalam melakukan proses pemetaan, karena perbedaan dilakukan selama mungkin, dan setiap persamaan awal (direktori), maka *file* yang terdapat dalam direktori tersebut akan digabungkan dalam satu direktori di *PFSS*. Tetapi dengan pendekatan kedua, lebih mudah untuk mendukung *caching* dan penemuan *file* yang sama di *server* yang berbeda, dan dianggap sebagai *file* yang sama, sehingga lebih dapat mendukung redundansi dan menambah kehandalan layanan.

Dengan pertimbangan bahwa *PFSS* juga menyediakan fitur *caching* dan redundansi, maka akan lebih baik untuk menggunakan pendekatan kedua dalam pengembangan perangkat lunak *PFSS* ini.

Selain itu, diperlukan juga suatu cara agar daftar *file* yang terdapat di *PFSS* sedapat mungkin sama dengan yang ada di *FS* pada setiap waktu. Untuk melakukan ini, terdapat dua cara yaitu :

1. Melakukan *update* berkala.

Pada pendekatan ini, *update* daftar *file* di *FS* penyedia dilakukan secara berkala untuk tiap *ftp server* penyedia.

2. Melakukan *update* tiap kali ada permintaan yang bersesuaian dengan *FS* penyedia.

Untuk melakukan proses *update file* yang tersedia di *FS*, maka *update* akan dilakukan setiap kali ada request dari pengguna untuk *FS* tersebut. Dan *update* yang dilakukan lokal terhadap *file* yang diminta oleh pengguna saja.

Pada pendekatan pertama, hal ini mudah dilakukan dan cenderung memakan *resource* yang sedikit. Namun jika selang *update* terlalu lama, maka kemungkinan terjadi ketidaksinkronan antara daftar *file* di *PFSS* dan *FS* lebih besar.

Pada pendekatan kedua, *resource* yang dimakan akan tergantung pada seberapa sering suatu data di *FS* penyedia diakses. Sehingga jika suatu *FS* penyedia sangat populer, maka akan memakan *resource* yang banyak. Namun dengan pendekatan ini, kesinkronan antara daftar *file* di *PFSS* dan *FS* sangat terjaga.

Kedua pendekatan ini memiliki keuntungan dan kelebihan tersendiri, dan dapat digabungkan. Pada pengembangan perangkat lunak *PFSS* ini, akan dilakukan pendekatan gabungan antara pendekatan pertama dan kedua, yaitu *update* dilakukan berdasarkan request terhadap *FS* yang mempunyai data yang bersesuaian, namun *update* hanya dilakukan maksimal dalam selang waktu tertentu. Sehingga walaupun suatu *FS* penyedia datanya sering diakses, namun tidak setiap kali ada data yang berkaitan dengannya, maka *FS* penyedia tersebut diakses, namun maksimal tiap suatu waktu tertentu, seperti dalam lima menit. Sehingga terjadi kompromi antara *resource* dan sinkronisasi.

Sehingga pada proses *download*, akan dilakukan pengecekan penamaan terhadap pemetaan yang berelasi dengan suatu *resource* di *PFSS* untuk mendapatkan *resource* sebenarnya di *FS* penyedia, sedangkan pada proses *upload*, akan dilakukan pengecekan penamaan terhadap pemetaan yang berelasi dengan suatu *resource* di *PFSS* untuk melihat apakah terjadi konflik, dan jika terjadi konflik, maka akan terjadi penimpaan *resource* dan dengan begitu ditentukan juga *resource* mana di *FS* penyedia mana yang akan ditimpa, dan pada proses melihat daftar *resource*, akan dilakukan pengambilan seluruh daftar *resource* di *FS* penyedia yang berkaitan, dan kemudian akan dipetakan ke *resource* yang direpresentasikan di *PFSS*, jika terjadi konflik penamaan maka akan diselesaikan dengan metode yang dipilih diatas, yaitu metode pembedaan nama *resource* seakhir mungkin, kemudian *update* daftar *resource* dilakukan dengan menggunakan pendekatan *update* yang dipilih di atas, yaitu pendekatan gabungan antara *update* berkala dan *update* setiap kali ada permintaan.

### **3.7 Analisis Pengelolaan FS Penyedia yang Menyediakan Layanan FTP melalui PFSS**

Pada *PFSS*, diperlukan pengelolaan data *FS* penyedia yang menyediakan layanan *FTP*. Untuk mengetahui *FS* mana saja yang akan menyediakan layanannya melalui *PFSS*, maka harus terdapat suatu mekanisme pencatatan *FS* yang menyediakan layanannya melalui *PFSS*. Kumpulan *FS* yang menyediakan layanannya melalui *PFSS* ini kemudian disebut *FS* penyedia. Ada dua kemungkinan cara yang dapat digunakan untuk melakukan pencatatan ini, yaitu:

1. Administrasi oleh *FS*. Pada cara ini, tiap *FS* yang ingin menyediakan layanannya melalui *PFSS* harus mendaftarkan diri ke *PFSS*. Hal ini dapat dilakukan dengan mendaftarkan diri melalui sebuah antarmuka untuk memasukkan data *manual* tentang informasi *FS* penyedia di *PFSS*. Antarmuka ini disediakan sebagai layanan terpisah dari *FS*, jadi mungkin diimplementasikan melalui perangkat lunak *web*, baik dalam bentuk halaman *web* atau melalui *web service*, ataupun melalui perintah dalam *shell FTP*.

2. Administrasi oleh sistem *PFSS*. Dengan cara ini, pencatatan akan dilakukan oleh *PFSS*. Terdapat dua cara untuk melakukan ini, yaitu:
  - a. Dengan menyediakan sebuah perangkat lunak yang akan melakukan *scanning* layanan *FS* pada tiap *host* di jaringan lokal organisasi pada waktu-waktu tertentu. Perangkat lunak ini dapat disebut sebagai perangkat lunak pencari layanan *FTP*. Jika perangkat lunak pencari layanan *FTP* menemukan *FS* baru, maka ia akan menambahkan ke daftar *FS* terdaftarnya dan dianggap sebagai *ftp server* yang menyediakan layanan *FTP* melalui *PFSS*. Begitu pula, jika perangkat lunak pencari layanan menemukan suatu *host* yang sebelumnya menyediakan layanan *FTP*, dan kemudian tidak lagi menyediakan layanan *FTP*, maka ia akan menghapusnya dari daftar *FS* terdaftarnya.
  - b. Pendaftaran dilakukan oleh *admin* dari *PFSS*. Pada *PFSS* disediakan sebuah *human interface*, yang dapat digunakan untuk administrasi *FS* penyedia.

Pada pendekatan pertama, *FS* yang ingin mendaftarkan diri harus tahu bahwa dia terdaftar, dan pada *FS* diperlukan sebuah perangkat lunak tambahan yang memantau layanan ini dan mendaftarkannya. Kelemahannya adalah pada diperlukan usaha tambahan untuk membuat program dan mengatur *FS* ini, dimana seharusnya *FS* penyedia tidak perlu berurusan dengan pendaftaran, ia dapat menyediakan layanan *FTP* melalui *PFSS* tanpa menyadarinya.

Pada pendekatan kedua, *FS* tidak perlu mengetahui keadaan bahwa layanannya diakses oleh pengguna menggunakan *PFSS*. Hal ini sangat memudahkan jika terjadi perubahan pada *FS* penyedia, karena perubahan yang terjadi di *PFSS* tidak perlu diurus juga oleh *FS* penyedia, seperti jika terjadi perubahan alamat IP *FS* penyedia, agar layanan *FS* penyedia tetap dapat diakses melalui *PFSS*, *FS* penyedia tidak perlu melakukan aksi tambahan di sistem *PFSS*. Pada pendekatan kedua ini, terdapat dua alternatif. Pada alternatif pertama, dimana perubahan *FS* penyedia dilakukan secara otomatis oleh program, terdapat kekurangan bahwa harus memakan *resource* yang cukup banyak jika jaringan di scan cukup besar. Sedangkan pada alternatif kedua, diperlukan tambahan aksi dari admin *PFSS* untuk mengadministrasi *FS* penyedia.

Pada pengembangan perangkat lunak *PFSS* ini, digunakan pendekatan kedua, dengan alternatif kedua. Selain itu pada pendekatan pertama, lebih sulit dari pendekatan yang dipilih, dalam hal penjagaan administrasi dan implementasi, sedangkan pendekatan kedua alternatif pertama, diperlukan sumber daya komputasi yang lebih besar. Pertimbangan lainnya dalam pemilihan pendekatan adalah berdasarkan kemudahan dalam administrasi *PFSS*, yaitu kemudahan admin *PFSS*, -bukannya *FS* penyedia-, dapat dengan baik memantau *FS* mana saja yang diberikan layanannya melalui *PFSS*, serta kemudahan dalam pengembangan perangkat lunak *FS*.

### **3.8 Analisis Penanganan Dinamika dan Keandalan Layanan FTP melalui PFSS**

Pada *PFSS*, dilakukan penanganan dinamika dan keandalan layanan *FTP*. Penanganan dinamika dan keandalan yang dimaksud berhubungan dengan tersedianya layanan *FTP* pada *FS* penyedia. Permasalahan dinamika dan keandalan dibedakan; Permasalahan dinamika berkaitan bahwa pengecekan tersedianya layanan *FTP* pada *FS* penyedia; Permasalahan keandalan berkaitan dengan apa yang akan dilakukan jika suatu layanan *FTP* tidak tersedia pada suatu *FS* penyedia.

Secara umum, permasalahan dinamika layanan *FTP* pada *FS* penyedia, berupa:

1. Layanan *FS* mati. Hal ini dapat dikarenakan jika *host FS* mati, mengalami crash, *host FS* hidup tetapi layanannya saja yang mati, jaringan komputer yang terhubung putus, atau perangkat lunak *FS* mati dan/atau mengalami crash.
2. Alamat *FS* berubah. Perubahan alamat meliputi perubahan alamat IP dan/atau nama *host*.

Pada penanganan masalah dinamika, maka jika terjadi perubahan pada suatu layanan *FTP* pada *FS* Penyedia, maka layanan *FTP* itu akan diubah statusnya menjadi aktif atau nonaktif, tetapi tidak dihilangkan dari *FS* penyedia terdaftar di *PFSS*. Hal ini dilakukan ketimbang dengan langsung menghapus, karena proses penambahan, pengubahan, atau penghapusan dilakukan pada manajemen *FS* penyedia yang dilakukan oleh *admin*, berdasarkan bahasan analisis sebelumnya.

### 3.8.1 Analisis Penanganan Dinamika FS Melalui PFSS

Untuk menangani permasalahan dinamika, terdapat dua cara :

1. Pengecekan berkala.
2. Pengecekan tiap kali ada permintaan yang berkaitan dengan suatu FS penyedia.

Pada pendekatan pertama, PFSS akan melakukan pengecekan tersedianya layanan FTP di setiap FS penyedia secara periodik, dan jika terjadi perubahan maka status layanan FTP yang berkaitan akan diubah menjadi aktif atau nonaktif.

Pada pendekatan kedua, PFSS ini akan mengetahui kondisi layanan FTP pada FS penyedia, dengan secara implisit melakukan pengecekan tiap kali ada permintaan layanan dari pengguna, baik mengambil *file*, menyimpan *file*, atau melihat daftar *resource*. Pengecekan yang dilakukan disatukan dengan permintaan layanan.

Pendekatan yang digunakan adalah pendekatan kedua. Hal ini digunakan karena untuk melakukan suatu aksi yang harus melibatkan FS penyedia secara aktif tentunya dibutuhkan pengecekan terhadap ketersediaan layanan FS pada saat aksi dilakukan.

### 3.8.2 Analisis Penanganan Keandalan Layanan FS Melalui PFSS

Pada penanganan permasalahan keandalan, terdapat beberapa cara dalam menyediakan tambahan keandalan dalam layanannya, antara lain:

1. Melakukan penyimpanan yang redundan pada beberapa FS penyedia terdaftar. Penentuan data yang disimpan secara redundan, mempunyai kriteria antara lain, tingkat 'kepopuleran' suatu *file*, dan jenis account yang digunakan untuk menyimpan *file* melalui FS. Penentuan nilai kriteria ini akan dijelaskan pada sub bab 3.8.3.
2. Melakukan *caching*, pada PFSS. penentuannya juga mempunyai kriteria yang sama dengan proses penyimpanan redundan, tetapi dengan cara perhitungan yang bisa berbeda.

Pada pendekatan pertama, akan dapat membantu meningkatkan tingkat ketersediaan data sehingga menyediakan layanan data dengan lebih baik, karena data tersimpan di lebih dari satu FS penyedia.

Pada pendekatan kedua, hal ini akan membantu mempercepat transfer dan meningkatkan tingkat ketersediaan data, karena data tersimpan di *FS*. Data *cache* ini disimpan selama perangkat lunak *PFSS* hidup dan akan dihapus begitu perangkat lunak dimatikan.

Pada *PFSS* ini, digunakan kedua pendekatan untuk meningkatkan kehandalan layanan pada *PFSS*, karena sifat kedua pendekatan ini berbeda dan saling melengkapi satu sama lain.

### 3.8.3 Analisis Kebijakan Penggantian pada *PFSS*

Pada proses *caching* dan/atau redundan terdapat penentuan kebijakan penggantian (*replacement policy*). *PFSS* menggunakan kombinasi pendekatan yang terdapat pada dasar teori. Hal ini dilakukan dengan menggunakan parameter yang bisa diubah untuk menentukan kebijakan penggantian tersebut. Nilai dari parameter ini dapat diubah oleh admin *PFSS*. Untuk menentukan parameter apa saja yang digunakan, dipertimbangkan keperluan dari *PFSS* yang bertindak sebagai *proxy* dari perangkat lunak jaringan. Berdasarkan jenis kebijakan penggantian yang ada pada sub bab 2.2.3, yaitu:

1. *First In First Out (FIFO)*
2. *Least Recently Used (LRU)*
3. *Most Recently Used (MRU)*
4. *Least Frequenty Used (LFU)*

Pendekatan yang digunakan adalah kebijakan LRU. Kebijakan LRU digunakan pada saat melakukan pertimbangan untuk memasukkan suatu entitas *resource* ke *cache* dan/atau membuat entitas menjadi redundan. Kebijakan LRU ini digunakan karena entitas *resource* ditransfer dalam perangkat lunak jaringan cenderung berubah seiring waktu. Namun batas waktu ini dapat berubah. Selain itu, antara suatu jaringan lokal universitas satu dan lainnya, mungkin memiliki batas waktu ideal yang berbeda. Seperti suatu universitas yang memiliki fokus di bidang teknik akan sering menggunakan file teknik, seperti CAD atau file source code, dan dalam suatu kurun waktu tertentu *file* tersebut akan sering diperlukan oleh banyak orang di universitas tersebut. Hal ini tentu berbeda dengan universitas yang memiliki fokus di bidang

manajemen. Adanya perbedaan ini, dapat diakomodasi dengan adanya parameter yang berkaitan dengan kebijakan LRU dan dapat diubah oleh admin *PFSS*. Berkaitan dengan contoh di atas, maka diperlukan juga adanya parameter yang menentukan jenis *file* apa saja yang akan dimasukkan ke dalam *cache* dan/atau dibuat redundan. Sehingga dihasilkan parameter yang dapat diubah, yaitu :

1. Nilai minimum seberapa sering suatu *file* diakses (nilai minimum selang waktu suatu *file* diakses).
2. Jenis *File*.
3. Nilai minimum waktu di *update* dari hari ini.

Nilai minimum dari tiga parameter diatas yang dibutuhkan agar suatu *file* bisa dimasukkan dalam *cache*/dibuat redundan adalah parameter yang bisa dimasukkan dan dapat diubah oleh admin *PFSS*.

#### **3.8.4 Analisis Kebijakan Penulisan pada *PFSS***

Sedangkan pada kebijakan penulisan, terdapat tiga pilihan, seperti yang diterangkan pada sub bab 2.2.3 yaitu :

1. *Write-through*
2. *Write-back*
3. *No-write*

Pada pendekatan pertama dan kedua, diizinkan untuk menuliskan data melalui *cache*, sedangkan pada *no-write* tidak diizinkan menuliskan data melalui *cache*. Karena *PFSS* yang sifatnya meneruskan dan melakukan fungsi *routing* saja, maka *PFSS* tidak mengizinkan untuk melakukan penulisan melalui *cache*, melainkan jika terjadi penulisan harus langsung di *FS* penyedia. Namun penulisan ini dapat dilakukan melalui *PFSS*, tetapi tidak ditulis di *cache*, melainkan langsung ditulis di *FS* penyedia.

### **3.9 Analisis Keamanan pada FS Penyedia dan Pengguna Layanan FTP di PFSS**

Untuk masalah keamanan, *PFSS* mempertimbangkan hal sebagai berikut.

1. Keamanan penyedia layanan *FTP* dengan *PFSS*.
2. Keamanan pengguna layanan *FTP* dengan *PFSS*.

Pada komunikasi antara *FS* penyedia sebagai penyedia layanan *FTP* yang sebenarnya dengan *PFSS*, tidak terdapat suatu penanganan khusus. Komunikasi ini terjadi seperti layaknya komunikasi untuk layanan *FTP* biasa, yaitu penggunaan *password*, seperti yang digunakan antara pengguna layanan *FTP* dengan *PFSS*.

Untuk keamanan pada pengguna layanan *FTP* di *PFSS*, kebutuhan keamanan diakomodasi dengan adanya autentifikasi pengguna dan penyaringan alamat IP yang dapat mengakses *PFSS*.

Untuk proses autentifikasi pengguna, *PFSS* mempunyai sistem pengaturan pengguna tersendiri. *PFSS* akan mendukung multi pengguna, seperti layanan *FTP* lainnya. Tiap pengguna pada *FS* penyedia dapat berasosiasi dengan lebih dari satu pengguna *PFSS*, dan pengguna *PFSS* berasosiasi dengan lebih dari satu pengguna *FTP*, masing-masing satu pada tiap *FS* penyedia. Untuk membuat asosiasi, maka kedua pengguna, yaitu pengguna di sistem *PFSS*, dan pengguna di suatu *FS* terdaftar, harus ada terlebih dahulu. Hal ini dilakukan agar mendukung sistem multi pengguna. Perhatikan bahwa tiap pengguna akan mempunyai struktur *file* tersendiri di *PFSS*.

Untuk proses penyaringan alamat IP, *PFSS* akan mempunyai dua daftar yang disimpan berkaitan dengan penyaringan alamat IP, yaitu:

1. *PFSS* akan menyimpan alamat IP yang boleh melakukan akses ke layanan *FTP* di *PFSS*.
2. *PFSS* akan menyimpan alamat IP yang tidak boleh melakukan akses ke layanan *FTP* di *PFSS*.

Pada pendekatan pertama akan efektif dan efisien jika *PFSS* merupakan perangkat lunak yang diakses oleh kalangan terbatas, yang jumlah sedikit dan dapat diidentifikasi dengan mudah oleh *admin PFSS*. Namun akan sangat buruk

performansinya jika *PFSS* merupakan perangkat lunak yang layanannya dapat diakses oleh publik.

Pendekatan kedua mempunyai karakteristik yang bertentangan dengan pendekatan pertama. Ia akan sangat efektif jika layanan perangkat lunak merupakan layanan publik dan dapat diakses oleh jumlah yang cukup besar.

Dengan memperbandingkan kedua pendekatan ini, dan melihat karakteristik *PFSS* yang merupakan perangkat lunak yang menyediakan layanan *FTP* pada suatu jaringan kampus universitas yang cenderung terbuka dan dengan jumlah pengakses yang cukup banyak, maka pendekatan kedua akan lebih efektif untuk digunakan.

Sehingga untuk melakukan proses penyaringan, *PFSS* melakukan pengecekan setiap alamat IP dari pengguna layanan *FTP* dan mencocokkan dengan daftar alamat IP yang tidak diperbolehkan untuk mengakses layanan *FTP* pada *PFSS*. Jika alamat IP tersebut cocok maka pengguna layanan *FTP* tersebut tidak dapat mengakses layanan tersebut, dan sebaliknya, pengguna layanan *FTP* akan dapat mengakses jika alamat IP yang digunakan tidak terdapat pada daftar tersebut.

### **3.10 Model Analisis Perangkat Lunak**

#### **3.10.1 Spesifikasi Pengguna Perangkat Lunak**

Pengguna perangkat lunak *PFSS* adalah sebagai berikut.

1. Pengguna *FTP*

Pengguna *FTP* adalah pengguna layanan *PFSS*.

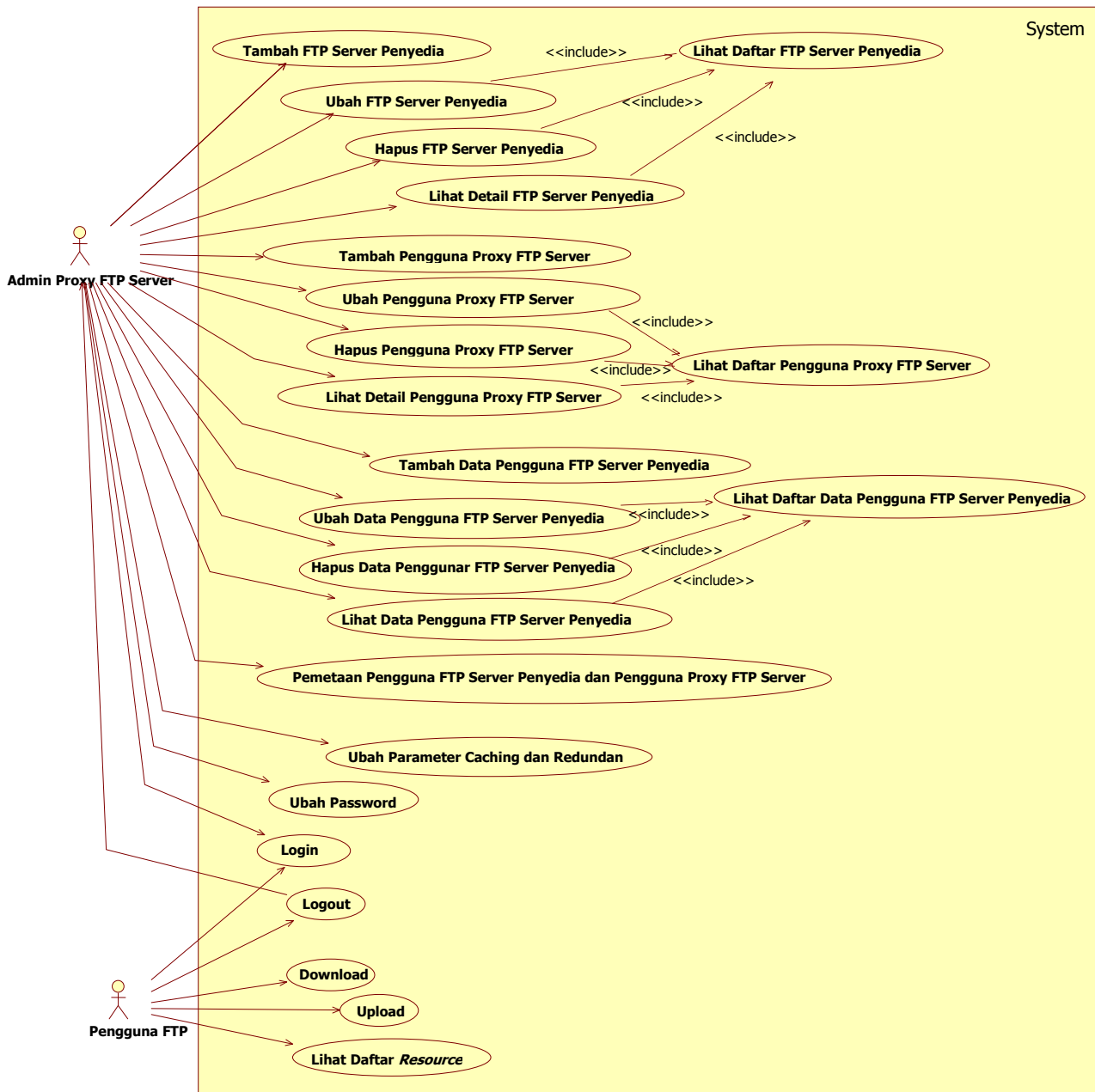
2. Admin *PFSS*

Admin *PFSS* adalah *admin* yang mengelola *PFSS*.

#### **3.10.2 Pemodelan Fungsionalitas**

Dalam pengembangan perangkat lunak ini, digunakan metode *Unified Model*, yang menggunakan UML sebagai bahasa pemodelannya. Berdasarkan spesifikasi perangkat lunak (Bab 3.2), dan analisis terhadap masalah yang terdapat dalam pengembangan

PFSS (Bab 3.5 – Bab 3.9), maka dibuat pemodelan fungsionalitas, akan digunakan diagram *use case*. Gambar III-2 adalah diagram *use case* untuk perangkat lunak ini.



Gambar III-2 Diagram *Use Case*

Semua spesifikasi perangkat lunak yang berhubungan penggunaan PFSS oleh pengguna layanan FTP, digabungkan menjadi tiga *use case*, yaitu *download*, *upload*, dan lihat daftar *resource*. Tiga *use case* ini merupakan kegunaan utama dari layanan FTP.

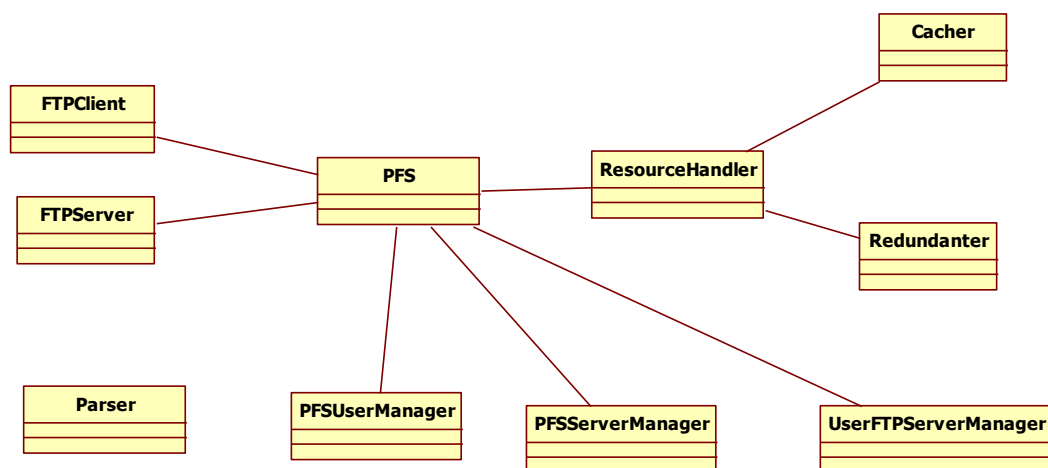
Sedangkan spesifikasi-spesifikasi yang berhubungan dengan administrasi *PFSS* oleh Admin *PFSS* digabungkan menjadi *use case* lainnya.

### 3.10.3 Pemodelan Skenario Kejadian

Perilaku dari *use case* yang digambarkan pada pemodelan fungsionalitas dijelaskan dan dideskripsikan pada skenario kejadian di lampiran A.

### 3.10.4 Pemodelan Kelas Analisis

Berdasarkan analisis yang telah dilakukan sebelumnya, maka dapat dihasilkan diagram kelas analisis seperti pada Gambar III-3. *PFSS* merupakan kelas utama yang mengatur semua proses dalam *PFSS*, baik secara langsung atau tidak langsung. Proses ini telah meliputi semua *use case* yang terdapat sub bab sebelumnya. Selain itu, terdapat satu kelas *utility* yang digunakan oleh semua kelas, yaitu kelas Parser. Kelas ini menyediakan layanannya dalam bentuk *static*, yaitu layanannya disediakan tanpa memerlukan instansiasi kelas.

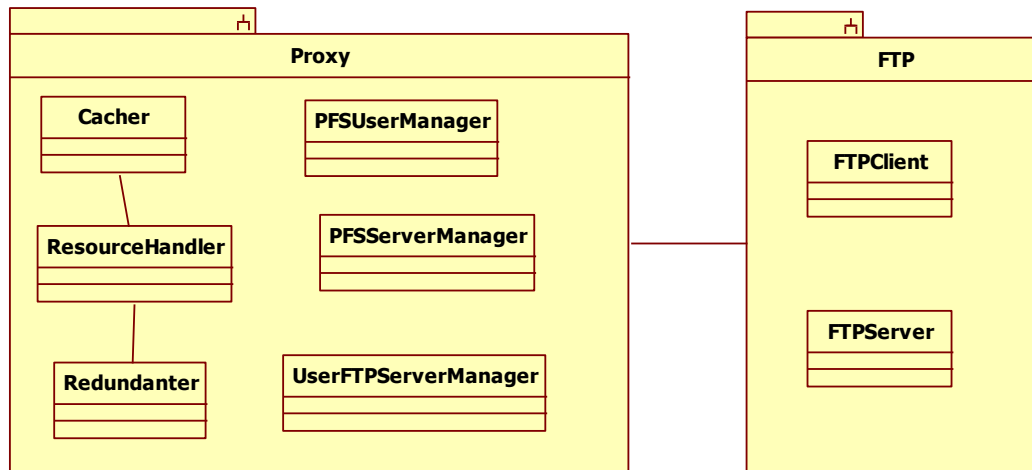


Gambar III-3 Diagram kelas analisis

### 3.10.5 Pemodelan Subsistem

Untuk memudahkan pengembangan selanjutnya, maka dari kelas analisis yang dihasilkan akan dibuat pembagian subsistem. Secara umum, terdapat 2 subsistem utama, yaitu subsistem yang menangani layanan *FTP*, dan subsistem yang melakukan

administrasi fungsi *proxy* yang dijelaskan pada spesifikasi perangkat lunak pada sub bab 3.2. Kelas *PFSS* merupakan kelas yang digunakan untuk mengontrol kedua subsistem ini. Pembagian subsistem terdapat pada gambar III-4.



**Gambar III-4** Subsistem kelas analisis