LAPORAN PEMBUATAN *DOMAIN-SPECIFIC LANGUAGE* "SYSMOD" MENGGUNAKAN JETBRAINS MPS

Dosen Pengampu:

Pak Joe

Bu rahil

Disusun oleh:

Arkan Ramadhan Nugraha	241524033
Hizba Shafwatuddin	241524046

D4 TEKNIK INFORMATIKA

JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA

POLITEKNIK NEGERI BANDUNG

2025

DAFTAR ISI

BAB I. PENDAHULUAN

I.1. Latar Belakang

Dalam sistem operasi modern, pengaturan hak akses pengguna terhadap file dan folder merupakan hal yang sangat penting untuk menjaga keamanan serta keteraturan sistem. Namun, konfigurasi hak akses sering kali bersifat kompleks dan bervariasi antar sistem operasi. Untuk menyederhanakan proses ini, digunakan pendekatan Domain Specific Language (DSL), yaitu bahasa pemrograman khusus yang dirancang untuk domain terbatas dengan sintaks yang mudah dipahami oleh pengguna domain tersebut.

Proyek ini bertujuan untuk merancang dan mengimplementasikan DSL yang mampu merepresentasikan aturan manajemen hak akses pengguna terhadap sumber daya (file dan folder) pada suatu sistem operasi Linux. DSL ini memungkinkan pengguna untuk mendefinisikan pengguna (user), grup pengguna (group), serta izin akses (permission) dengan format yang lebih sederhana dan konsisten.

Dengan memanfaatkan JetBrains MPS, bahasa ini dapat dirancang secara modular, memiliki validasi sintaks, serta mendukung fitur seperti scope, constraints, dan behavior.

I.2. Rumusan Masalah

Berdasarkan latar belakang tersebut, rumusan masalah adalah:

1. Bagaimana cara merepresentasikan domain manajemen hak akses sistem operasi ke dalam bentuk DSL?
2. Bagaimana struktur grammar DSL yang sesuai untuk menggambarkan entitas user, group, file, dan permissions?
3. Bagaimana cara mengimplementasikan DSL ini menggunakan JetBrains MPS beserta fitur-fitur seperti structure, editor, constraints, dan behavior?
4. Bagaimana hasil evaluasi usabilitas DSL terhadap aspek kemudahan, konsistensi, dan ekspresivitas?

I.3. Tujuan

Tujuan dari praktikum ini adalah:

1. Merancang DSL untuk domain manajemen hak akses sistem operasi.
2. Mengimplementasikan DSL menggunakan platform JetBrains MPS.
3. Meningkatkan pemahaman mahasiswa terhadap konsep language engineering dan language workbench.

BAB II. TINJAUAN PUSTAKA

II.1. Domain Specific Language

DSL adalah bahasa pemrograman yang dirancang untuk menangani permasalahan dalam domain tertentu secara efisien. Berbeda dengan general-purpose language seperti Java atau Python, memiliki cakupan yang sempit namun sintaks yang lebih natural dan ringkas untuk domainnya.

II.2. JetBrains MPS

JetBrains Meta Programming System (MPS) adalah language workbench yang memungkinkan pembuatan DSL berbasis projectional editing, di mana struktur bahasa direpresentasikan langsung dalam bentuk pohon abstrak (AST), bukan teks mentah.

Fitur utama MPS meliputi:

- Structure: mendefinisikan konsep dan relasi antar elemen.
- Editor: mendesain tampilan dan input untuk setiap konsep.
- Constraints: menentukan batasan validasi antar elemen.
- Behavior: mendefinisikan logika perilaku konsep.
- Generator: mengubah model DSL menjadi output lain (misalnya file konfigurasi).

II.3. Usability Evaluation for DSLs

Evaluasi usability DSL dilakukan berdasarkan beberapa aspek seperti learnability, expressiveness, simplicity, feedback, consistency, error tolerance, dan overall satisfaction. Setiap aspek dapat diberi skor 1–5 untuk menilai tingkat kemudahan penggunaan DSL bagi pengguna domain.

BAB III. PERANCANGAN DSL

III.1. Domain

Domain yang dimodelkan oleh DSL ini adalah Administrasi Sistem (*Systems Administration*), yang secara spesifik difokuskan pada Manajemen Konfigurasi Kontrol Akses pada server berbasis Linux. Tujuan utamanya adalah untuk mengabstraksi proses konfigurasi yang kompleks dan imperatif (seperti bash) menjadi sebuah model yang deklaratif, aman (*type-safe*), dan mudah dibaca. Pengguna (administrator sistem) cukup *mendefinisikan* keadaan akhir yang diinginkan, seperti siapa saja pengguna yang ada, mereka anggota grup apa saja, dan bagaimana hak akses sumber daya diatur. DSL ini dirancang untuk menerjemahkannya menjadi skrip konfigurasi yang dapat dieksekusi di sistem operasi Debian Linux.

III.2. Entitas

Domain yang diangkat adalah pengelolaan hak akses pengguna terhadap sumber daya sistem operasi. Entitas utama dalam domain ini antara lain:

- SystemOp (Sistem Operasi): Konsep root yang menjadi kontainer utama untuk seluruh konfigurasi. Konsep ini merepresentasikan satu sistem yang akan dikelola (misalnya, "Debian13").
- Group (Grup): Merepresentasikan deklarasi sebuah grup pengguna. Didefinisikan di dalam blok DefGroup.
- User (Pengguna): Merepresentasikan deklarasi seorang pengguna sistem.
- File dan Folder: Merepresentasikan sumber daya sistem yang akan dikelola hak akses dan kepemilikannya.
- Root (Pengguna Spesial): Konsep singleton yang merepresentasikan pengguna root yang sudah ada (built-in) di sistem.
- Permission (Izin): Konsep data yang berisi tiga properti: read, write, dan execute.

III.3. Relasi dan Langkah-langkah Antar Entitas

- Program dimulai dengan mendefinisikan SystemOp (Sistem). Ini adalah *root node* yang menampung seluruh konfigurasi dan bertindak sebagai ScopeProvider (penyedia ruang lingkup).Satu User dapat tergabung dalam beberapa Group
- Di dalam Sistem, Group (Grup) dideklarasikan terlebih dahulu. Grup-grup ini didefinisikan di dalam blok Groups (DefGroup). Langkah ini harus dilakukan sebelum mendefinisikan User atau Permission, karena Group akan menjadi "bahan" yang direferensikan oleh konsep lain.
- Kemudian, User (Pengguna) didefinisikan. Definisi User *tergantung pada* Group yang telah dideklarasikan, karena User akan merujuk ke Group tersebut untuk menentukan keanggotaan grupnya (misalnya, groups [teachers hackers]).

- Setelah Group dan User ada, File atau Folder didefinisikan. Konfigurasi File dan Folder *tergantung pada* User dan Group yang sudah ada untuk mengisi properti owner dan group (misalnya, owner root, group baba).
- Terakhir, Permission (Hak Akses) ditentukan. Hak akses ini didefinisikan di dalam blok permissions { ... } pada File atau Folder. Langkah ini *tergantung pada* User dan Group yang ada untuk menentukan target dari aturan izin tersebut (misalnya, furry read allow... atau arney read deny...).

III.4. Aturan

1. Root adalah pengguna spesial dengan semua akses.
2. Permission dapat berupa allow atau deny dan deny mengoverride allow.
3. Folder/File wajib memiliki owner dan group yang valid.
4. Semua User dan Group harus didefinisikan dalam satu SystemOperation
5. Sebuah SystemOp (1) dapat berisi banyak deklarasi Group (N). (Relasi 1:N)
6. Sebuah SystemOp (1) dapat memiliki banyak deklarasi User (N). (Relasi 1:N)
7. Sebuah User (1) dapat memiliki banyak Group (N). Sebaliknya, sebuah Group (1) dapat memiliki banyak User (M). Ini merepresentasikan relasi M:N (banyak-ke-banyak) antara Pengguna dan Grup.
8. Properti read, write, dan execute pada Permission secara eksplisit menggunakan enumerasi Allowdeny (allow/deny). Ini adalah representasi langsung dari semantik ACL (Access Control List) Linux, di mana hak akses dapat diberikan (allow) atau diblokir secara eksplisit (deny).
9. Seluruh referensi (UserReference dan GroupReference) diatur oleh ScopeProvider pada SystemOp. Ini aturan utamanya: sebuah referensi hanya dapat merujuk ke User atau Group yang didefinisikan di dalam SystemOp yang sama.

III.5. Grammar DSL (BNF)

```
(* === Aturan Utama (Top-Level) === *)

PROGRAM         -> SYSTEM_OP


SYSTEM_OP       -> 'SystemOperation' ID '{'
                    GROUP_DEF
                    USER_DEF*
                    FOLDER_DEF*
                    FILE_DEF*
                  '}'
```

```
(* === Definisi Entitas === *)
GROUP_DEF       -> 'Groups' '{' ID* '}'

USER_DEF        -> 'User' ID '{'
                    ('home' STRING_LITERAL | '<no home>')
                    'groups' '[' ID* ']'
                  '}'

FOLDER_DEF      -> 'Folder' ID '{'
                    'dir' STRING_LITERAL
                    'recursive' BOOLEAN
                    'owner' OWNER_TARGET
                    'group' ANY_TARGET
                    'permissions' '{' PERMISSION_RULE* '}'
                  '}'

FILE_DEF        -> 'File' ID '{'
                    'dir' STRING_LITERAL
                    'owner' OWNER_TARGET
                    'group' ANY_TARGET
                    PERMISSION_RULE*
                  '}'

(* === Aturan Izin dan Target === *)
PERMISSION_RULE -> ANY_TARGET 'read' STATUS ',' 'write' STATUS ','
'execute' STATUS

(* ANY_TARGET merepresentasikan interface ITarget (UserReference,
GroupReference, atau Root)
```

Kita menggunakan ID untuk id_user dan id_group demi kesederhanaan.

```
*)
ANY_TARGET      -> ID | 'root'

(* OWNER_TARGET merepresentasikan interface UserLike (UserReference atau
Root)
*)
OWNER_TARGET    -> ID | 'root'



(* === Terminal === *)
STATUS          -> 'allow' | 'deny'
BOOLEAN         -> 'true' | 'false'

(* === Token Leksikal (Implisit) === *)
(* ID              -> [a-zA-Z_][a-zA-Z0-9_]* *)
(* STRING_LITERAL  -> "" (karakter apapun) "" *)
```

BAB IV. IMPLEMENTASI DSL DI JETBRAINS MPS

IV.1. Struktur (Stucture)

1. Konsep SystemOp
   Konsep SystemOp merupakan *root node* dari DSL, yang bertindak sebagai kontainer utama untuk seluruh konfigurasi sistem operasi.

```
concept SystemOp extends    BaseConcept
                 implements IMainClass
                            INamedConcept
                            ScopeProvider

  instance can be root: true
  alias: SystemOperation
  short description: <no short description>

  properties:
  << ... >>

  children:
  defgroup : DefGroup[0..1]
  users    : User[0..n]
  folders  : Folder[0..n]
  files    : File[0..n]

  references:
  << ... >>
```

   a. SystemOp mengimplementasikan INamedConcept agar setiap konfigurasi sistem memiliki properti name (nama).
   b. Konsep ini mengimplementasikan IMainClass (dari jetbrains.mps.execution.util) agar konfigurasi dapat dieksekusi atau dijalankan langsung dari dalam lingkungan MPS untuk proses generasi kode.
   c. SystemOp juga mengimplementasikan ScopeProvider, yang menjadikannya sebagai penyedia ruang lingkup (scope) untuk semua referensi (seperti User dan Group) yang didefinisikan di dalamnya.
   d. Sebagai root node, SystemOp memiliki children berupa defgroup (tipe DefGroup), users (tipe User [0..n]), folders (tipe Folder [0..n]), dan files (tipe File [0..n]).
2. Konsep Group dan DefGroup
   a. DefGroup: Konsep ini berfungsi sebagai blok khusus untuk deklarasi grup. Konsep ini memiliki alias "Groups" untuk meningkatkan keterbacaan. Di dalamnya, terdapat child groups

dengan kardinalitas [0..n], yang memungkinkan definisi satu atau banyak Group.

b. Group: Konsep ini merepresentasikan deklarasi sebuah grup di dalam sistem operasi. Group mengimplementasikan INamedConcept sehingga setiap grup wajib memiliki nama.

```
concept Group extends    BaseConcept
             implements INamedConcept

  instance can be root: false
  alias: Group
  short description: <no short description>

  properties:
  << ... >>

  children:
  << ... >>

  references:
  << ... >>
```

```
concept DefGroup extends    BaseConcept
                 implements <none>

  instance can be root: false
  alias: Groups
  short description: <no short description>

  properties:
  << ... >>

  children:
  groups : Group[0..n]

  references:
  << ... >>
```

3. Konsep User

Konsep User merepresentasikan deklarasi seorang pengguna sistem operasi. User juga mengimplementasikan INamedConcept untuk properti name.

a. Struktur: User memiliki child home (tipe StringLiteral [0..1]) yang bersifat opsional, untuk menentukan direktori home pengguna.

b. Referensi: User memiliki reference groups (tipe GroupReference [0..n]) untuk menampung satu atau lebih referensi ke grup yang telah dideklarasikan sebelumnya di DefGroup.

```
concept User extends    BaseConcept
            implements INamedConcept

  instance can be root: false
  alias: User
  short description: <no short description>

  properties:
  << ... >>

  children:
  home   : StringLiteral[0..1]
  groups : GroupReference[0..n]

  references:
  << ... >>
```

4. Konsep Folder dan File

   Dua konsep ini merepresentasikan sumber daya sistem yang akan dikonfigurasi izin akses dan kepemilikannya. Keduanya mengimplementasikan INamedConcept.

   a. Struktur: Folder dan File memiliki struktur serupa yang mencakup:
      i. dir (tipe StringLiteral [1]): Jalur (path) absolut dari file atau folder.
      ii. owner (tipe UserLike [1]): Referensi ke pemilik sumber daya.
      iii. group (tipe ITarget [1]): Referensi ke grup yang memiliki sumber daya.
      iv. userpermissions (tipe SetPermission [0..n]): Daftar aturan izin akses.

   b. Properti Khusus Folder: Konsep Folder memiliki child tambahan recursive (tipe BooleanConstant [1]) untuk menandakan apakah konfigurasi (seperti chown dan ACL) akan diterapkan secara rekursif.

```
concept Folder extends    BaseConcept
              implements INamedConcept

  instance can be root: false
  alias: Folder
  short description: <no short description>

  properties:
  << ... >>

  children:
  userpermissions : SetPermission[0..n]
  dir             : StringLiteral[1]
  recursive       : BooleanConstant[1]
  owner           : UserLike[1]
  group           : ITarget[1]

  references:
  << ... >>
```

```
concept File extends    BaseConcept
             implements INamedConcept

  instance can be root: false
  alias: File
  short description: <no short description>

  properties:
  << ... >>

  children:
  userpermissions : SetPermission[0..n]
  dir             : StringLiteral[1]
  owner           : UserLike[1]
  group           : ITarget[1]

  references:
  << ... >>
```

5. Enumerasi allowdeny
   Enumerasi ini mendefinisikan nilai-nilai diskrit untuk status izin. Enumerasi Allowdeny memiliki dua anggota (member): allow dan deny, yang merepresentasikan pemberian atau penolakan akses.

```
enumeration allowdeny

members:
  • allow   <default presentation>
  • deny    <default presentation>

default member: null
```

6. Konsep Permission
   Konsep Permission adalah kontainer data yang menampung tiga properti: read, write, dan execute. Masing-masing properti ini bertipe enumerasi Allowdeny, sehingga nilainya bisa allow atau deny.

```
concept Permission extends    BaseConcept
                   implements <none>

  instance can be root: false
  alias: <no alias>
  short description: <no short description>

  properties:
  read    : allowdeny
  write   : allowdeny
  execute : allowdeny

  children:
  << ... >>

  references:
  << ... >>
```

7. Konsep SetPermission
   Konsep SetPermission bertindak sebagai aturan pemetaan (mapping) dalam blok izin. Konsep ini menghubungkan target (siapa yang diberi izin) dengan izin itu sendiri (apa yang diizinkan).
   a. Struktur: SetPermission memiliki dua children:
   b. target (tipe ITarget [1]): Menentukan subjek aturan (bisa berupa UserReference, GroupReference, atau Root).
   c. permission (tipe Permission [1]): Mendefinisikan izin read, write, dan execute untuk target tersebut.

```
concept SetPermission extends    BaseConcept
                      implements <none>

  instance can be root: false
  alias: <no alias>
  short description: <no short description>

  properties:
  << ... >>

  children:
  target     : ITarget[1]
  permission : Permission[1]

  references:
  << ... >>
```

8. Konsep Referensi dan Interface
   Untuk memfasilitasi referensi yang aman dan polimorfisme, digunakan beberapa konsep referensi dan interface:
   a. GroupReference: Berfungsi sebagai pointer atau referensi ke sebuah Group yang telah dideklarasikan.

```
concept GroupReference extends    BaseConcept
                       implements ITarget

  instance can be root: false
  alias: <no alias>
  short description: <no short description>

  properties:
  << ... >>

  children:
  << ... >>

  references:
  group : Group[1]
```

   b. UserReference: Berfungsi sebagai referensi ke sebuah User yang telah dideklarasikan.

```
concept UserReference extends    BaseConcept
                      implements ITarget
                                 UserLike

  instance can be root: false
  alias: <no alias>
  short description: <no short description>

  properties:
  << ... >>

  children:
  << ... >>

  references:
  user : User[1]
```

c. Root: Sebuah konsep singleton yang secara khusus merepresentasikan pengguna root dari sistem.

```
concept Root extends    BaseConcept
             implements UserLike
                        ITarget

  instance can be root: false
  alias: <no alias>
  short description: <no short description>

  properties:
  << ... >>

  children:
  << ... >>

  references:
  << ... >>
```

d. UserLike (Interface): Interface ini diimplementasikan oleh UserReference dan Root. Tujuannya adalah untuk membatasi tipe pada reference owner di File dan Folder, sehingga memastikan bahwa owner hanya dapat berupa entitas yang bersifat seperti pengguna.

```
interface concept UserLike extends <none>

  properties:
  << ... >>

  children:
  << ... >>

  references:
  << ... >>
```

e. ITarget (Interface): Interface ini diimplementasikan oleh UserReference, GroupReference, dan Root. Penggunaan interface ini memungkinkan polimorfisme pada reference group (di File/Folder) dan target (di SetPermission), sehingga keduanya dapat merujuk ke pengguna ataupun grup.

```
interface concept ITarget extends <none>

  properties:
  << ... >>

  children:
  << ... >>

  references:
  << ... >>
```

IV.2. Editor

Aspek Editor dalam MPS mendefinisikan bagaimana struktur logis (konsep) dari DSL direpresentasikan secara visual dan bagaimana pengguna berinteraksi dengannya. Editor ini menentukan sintaksis konkret dari bahasa.

4.2.1. Proses Desain Editor

Implementasi editor memanfaatkan fungsionalitas auto-generation dari MPS. Dengan menggunakan intensi (Alt+Enter) pada cell kosong di EditorMPS menyediakan opsi untuk menggenerasi node cell bawaan (default). Untuk DSL ini, layout editor yang menyerupai pernyataan (statement-like) dipilih sebagai titik awal. Layout ini kemudian dimodifikasi secara manual untuk menciptakan sintaksis yang intuitif dan sesuai dengan domain permasalahan (konfigurasi sistem).

4.2.2. Legenda Notasi Node Cell di Editor MPS

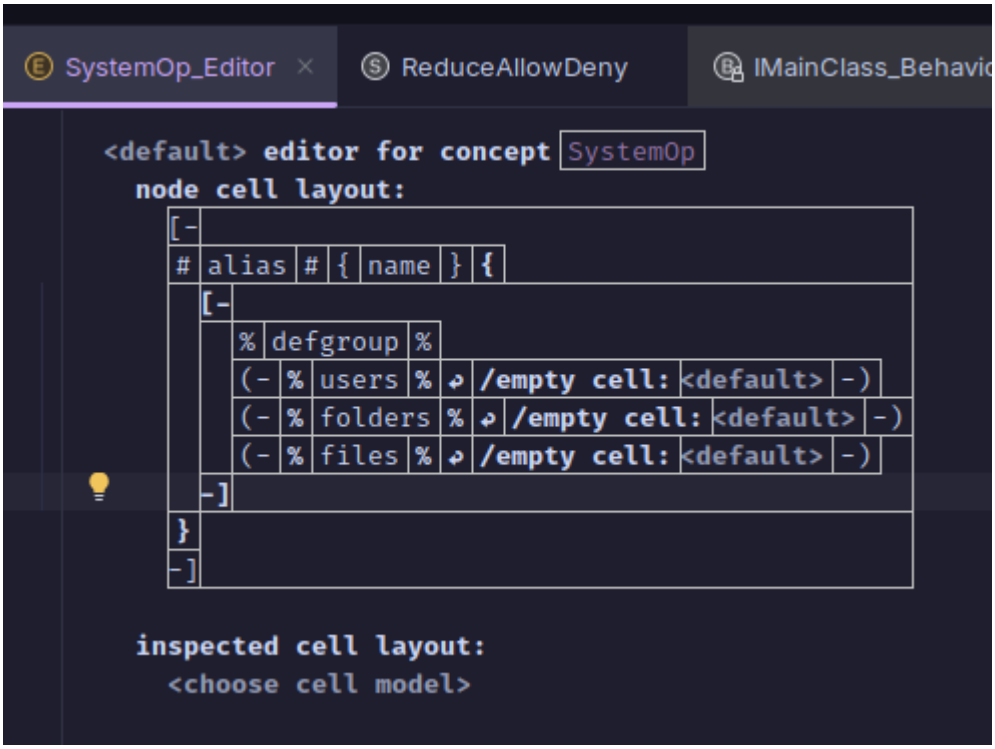
Struktur editor di MPS didefinisikan secara visual menggunakan komposisi sel (cells). Berikut adalah legenda notasi yang digunakan dalam definisi editor yang disediakan MPS:

1. [- ... -]: Mendefinisikan koleksi sel horizontal. Sel-sel di dalamnya akan disusun dari kiri ke kanan.
2. (- ... ↩ -): Mendefinisikan koleksi sel vertikal. Simbol ↩ (atau ↪) menandakan bahwa setiap elemen child dalam koleksi akan ditampilkan pada baris baru.
3. #alias#: Sel Alias. Sel ini secara otomatis menampilkan alias yang telah didefinisikan pada konsep (misalnya, SystemOperation untuk konsep SystemOp).
4. {namaProperti}: Sel Properti. Sel ini menampilkan dan mengizinkan modifikasi nilai dari properti yang sesuai (misalnya, {name} untuk properti name).
5. %namaChild%: Sel Child. Sel ini menyisipkan (me-render) editor dari child node yang ditentukan (misalnya, %owner% akan menyisipkan editor untuk child owner).
6. (%namaRef% -> {prop}): Sel Referensi. Sel ini menampilkan properti (misalnya {name}) dari node yang direferensikan (%namaRef%). Ini memungkinkan pengguna untuk melihat dan mengedit referensi dengan cara yang ringkas.

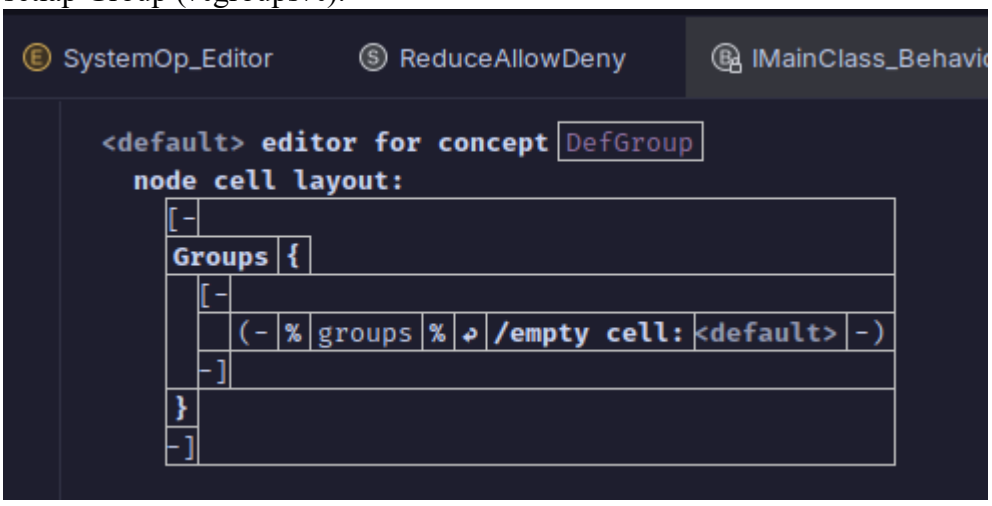
4.2.3. Analisis Editor per Konsep

Hubungan antara Struktur (IV.1) dan Editor (IV.2) sangat erat. Struktur mendefinisikan apa yang disimpan (data), sementara editor mendefinisikan bagaimana data tersebut ditampilkan dan diedit (visualisasi).

1. SystemOp: Editor untuk SystemOp (konsep root) menggunakan layout blok { ... } dengan alias dan nama di bagian atas. Di dalamnya, terdapat koleksi vertikal ((- ... ↩ -)) untuk menampilkan children defgroup, users, folders, dan files secara berurutan.



2. DefGroup: Editor ini menggunakan alias "Groups" sebagai teks konstan, diikuti oleh layout blok { ... }. Di dalamnya, terdapat koleksi vertikal untuk setiap Group (%groups%).

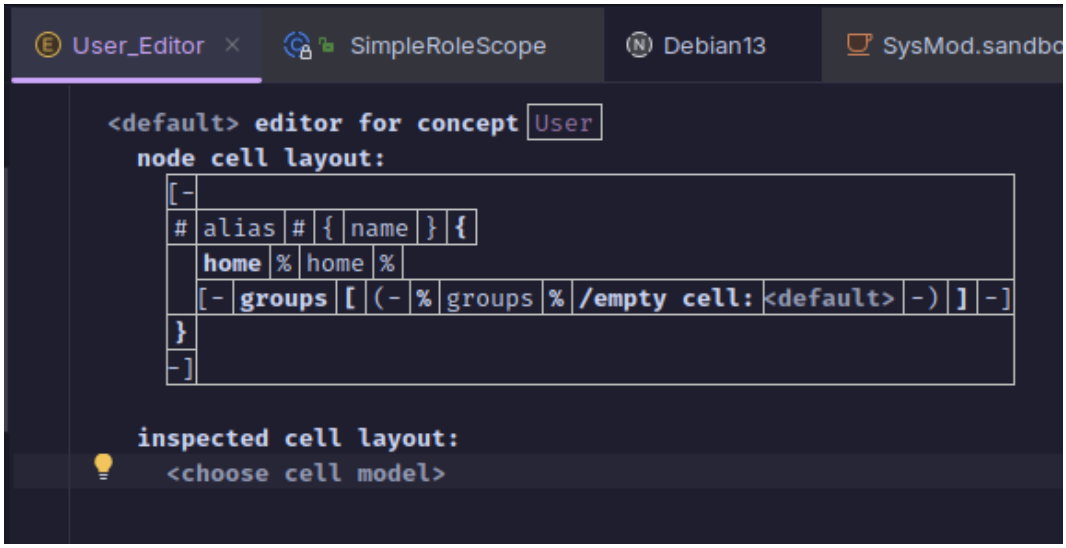


3. Folder dan File: Editor untuk Folder dan File didesain agar terlihat seperti blok deklarasi yang umum dalam bahasa skrip. Editor ini menggunakan teks konstan (misalnya, dir, owner, group, permissions) yang diikuti oleh sel child atau properti yang sesuai (%dir%, %owner%, %group%). Hal ini menciptakan sintaksis yang sangat mudah dibaca.

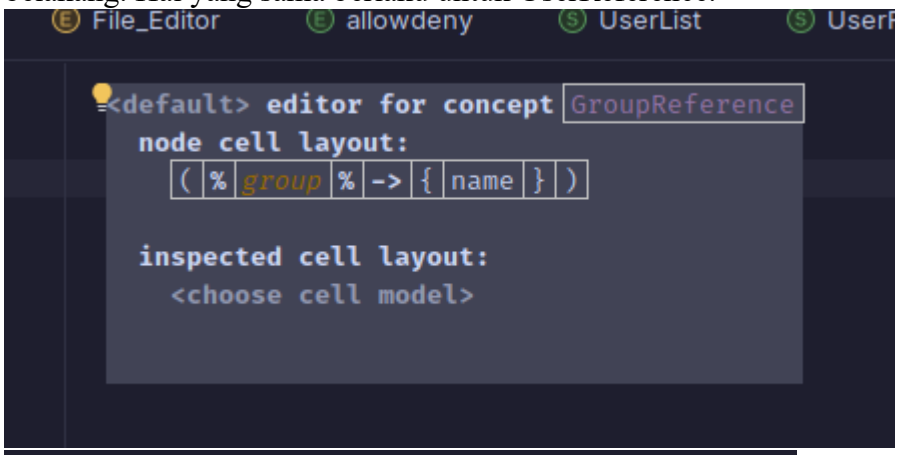
```
<default> editor for concept Folder
  node cell layout:
    [-
    # alias # { name } {
      dir % dir %
      recursive % recursive %
      owner % owner %
      group % group %
      permissions {
      [-
        (- % userpermissions % ↵ /empty cell: <default> -)
      -]
      }
    }
    -]

  inspected cell layout:
    <choose cell model>
```

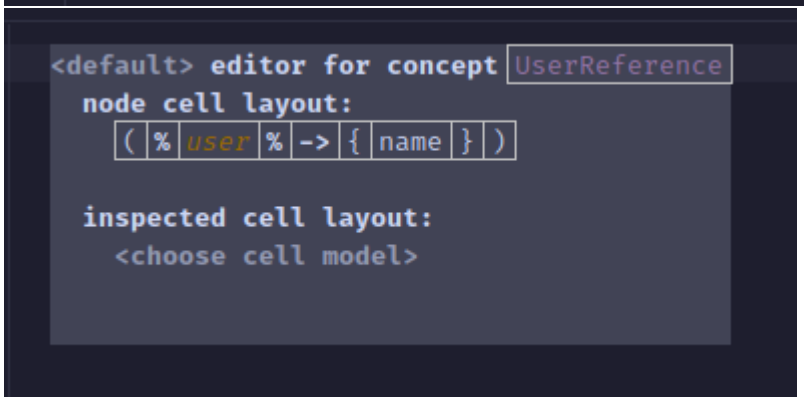
```
<default> editor for concept File
  node cell layout:
    [-
    # alias # { name } {
      dir % dir %
      owner % owner %
      group % group %
      [-
        (- % userpermissions % ↵ /empty cell: <default> -)
      -]
    }
    -]

  inspected cell layout:
    <choose cell model>
```

4. User: Serupa dengan Folder, editor User juga berbentuk blok. Editor ini menampilkan home sebagai child opsional dan groups sebagai koleksi horizontal ([- ... -]) dari referensi.



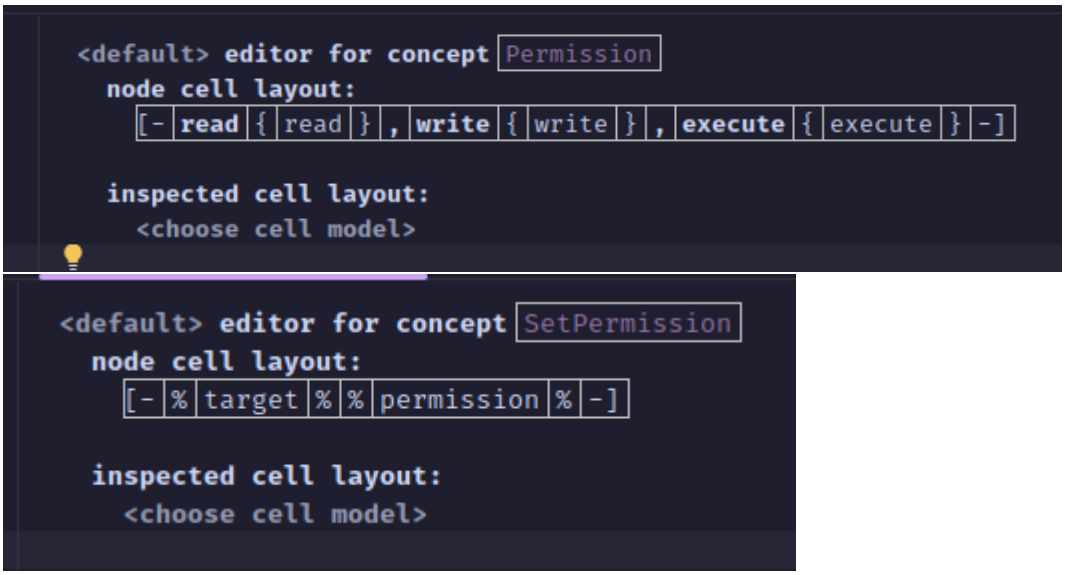
5. GroupReference dan UserReference: Editor untuk GroupReference ((% group % -> { name })) adalah contoh implementasi sel referensi yang ringkas. Bagi pengguna DSL, editor ini hanya menampilkan nama dari grup yang dituju. MPS menangani penyelesaian referensi dan validasi di latar belakang. Hal yang sama berlaku untuk UserReference.





6. Permission dan SetPermission: Editor Permission adalah koleksi horizontal sederhana ([- read {read} , write {write} , execute {execute} -]). Editor SetPermission menggabungkan %target% (yang bisa berupa UserReference, GroupReference, atau Root) dan %permission% secara horizontal, menghasilkan sintaksis yang padat dan jelas seperti

baba read allow, write deny, execute deny.



IV.3. Constraints

Aspek Constraints (Batasan) dalam MPS berfungsi untuk mendefinisikan aturan dan batasan-batasan semantik pada sebuah konsep. Batasan ini tidak terkait dengan tampilan (seperti Editor), melainkan dengan validitas model. Salah satu peran terpenting dari constraints adalah untuk mengatur scope dari sebuah referensi.

4.3.1. Batasan Referensi dan Scope

Dalam DSL ini, batasan referensi (*reference constraints*) digunakan pada UserReference dan GroupReference. *Scope* mendefinisikan himpunan *node* target yang valid yang dapat ditunjuk oleh sebuah referensi. Saat pengguna mencoba mengisi referensi (misalnya, dengan Ctrl+Space), *scope* inilah yang menentukan daftar pilihan yang muncul.

Pada implementasi ini, batasan untuk referensi group pada GroupReference dan referensi user pada UserReference diatur ke scope inherited.

- concepts constraints GroupReference { ... link {group} ... scope inherited for Group }
- concepts constraints UserReference { ... link {user} ... scope inherited for User }

```
concepts constraints GroupReference {
  can be child <none>

  can be parent <none>

  can be ancestor <none>

  instance icon <none>

  <<property constraints>>

  link {group}
    referent set handler <none>
    scope inherited for Group



  default scope <no default scope>

  additional methods

  << ... >>
}
```

Notasi scope inherited merupakan sebuah mekanisme delegasi. Ini menginstruksikan MPS bahwa GroupReference tidak mendefinisikan *scope*-nya sendiri. Sebaliknya, ia mendelegasikan pencarian *scope* ke *node* induknya dalam hierarki AST (*Abstract Syntax Tree*).

4.3.2. Sinergi dengan Behavior ScopeProvider

Mekanisme scope inherited ini bekerja secara sinergis dengan *Behavior* yang diimplementasikan pada konsep SystemOp. Seperti yang telah dijelaskan di IV.1, SystemOp mengimplementasikan ScopeProvider.

Alur kerjanya adalah sebagai berikut:

1. Pengguna mencoba mengisi referensi group pada GroupReference.
2. *Constraint* scope inherited pada GroupReference dieksekusi, yang kemudian "bertanya" kepada induknya (User) mengenai *scope* untuk Group.
3. User juga tidak mendefinisikan *scope*, sehingga pencarian terus naik ke SystemOp.
4. SystemOp, sebagai ScopeProvider, mengintersepsi permintaan ini.

5. Metode getScope() yang telah di-*override* pada *Behavior* SystemOp dieksekusi. Metode ini mengembalikan daftar Group yang didefinisikan *hanya* di dalam *child* defgroup milik SystemOp tersebut.

Hasil akhirnya adalah sebuah enkapsulasi yang kuat. Referensi GroupReference (dan UserReference) dipastikan tidak dapat "keluar" dan merujuk ke grup atau pengguna yang didefinisikan di dalam SystemOp lain. Setiap SystemOp menjadi unit konfigurasi yang mandiri dan terisolasi.

IV.4. Behavior

Aspek *Behavior* dalam MPS memungkinkan sebuah konsep untuk memiliki metode, baik *virtual*, *non-virtual*, *static*, maupun *constructor*. Ini adalah solusi yang lebih superior dibandingkan *utility methods* statis atau *node wrappers*, karena ia mengikat "logika bisnis" (*business logic*) atau operasi umum langsung ke konsep itu sendiri.

Dalam DSL ini, *Behavior* digunakan untuk dua tujuan utama:

1. Implementasi ScopeProvider: Menyediakan logika konkret untuk batasan *scope* yang dijelaskan di IV.3.

2. Abstraksi Logika Generator: Membuat metode *helper* untuk menyembunyikan detail implementasi dari generator, sehingga generator menjadi lebih bersih dan deklaratif.

4.4.1. Implementasi ScopeProvider

Seperti yang telah disebutkan, konsep SystemOp mengimplementasikan *interface* ScopeProvider. Logika dari *interface* ini didefinisikan dalam *Behavior* SystemOp dengan meng-*override* metode getScope().

```
concept behavior SystemOp {

  constructor {
    <no statements>
  }

  public virtual Scope getScope(concept<> kind, node<> child)
    overrides ScopeProvider.getScope {
    if (kind.isSubConceptOf(User)) {
      return SimpleRoleScope.forNamedElements(this, link/SystemOp : users/);

    }
    if (kind.isSubConceptOf(Group)) {
      return SimpleRoleScope.forNamedElements(this.defgroup, link/DefGroup : groups/);

    }

    return null;
  }

}
```

Metode getScope() ini dipanggil setiap kali *constraint* scope inherited (dari UserReference atau GroupReference) dieksekusi. Metode ini memeriksa "jenis" (kind) referensi yang sedang dicari:

- Jika kind adalah turunan dari User, metode ini mengembalikan *scope* yang berisi daftar *node* dari *child role* users milik SystemOp saat ini.
- Jika kind adalah turunan dari Group, metode ini mengembalikan *scope* yang berisi daftar *node* dari *child role* groups yang ada di dalam defgroup.

4.4.2. Abstraksi Logika Untuk Generator

Kasus penggunaan kedua (dan yang paling umum) adalah untuk menyederhanakan generator. Daripada menempatkan logika `if-then-else` yang kompleks di dalam templat generator, logika tersebut diekstraksi ke dalam metode *Behavior* pada konsep yang relevan.

1. Polimorfisme Itarget

   Masalah: Generator perlu menghasilkan prefiks yang berbeda untuk ACL (setfacl), yaitu u: untuk pengguna dan g: untuk grup.

   Solusi: Interface ITarget mendefinisikan "kontrak" (contract) metode virtual getTargetName() dan getType(). Setiap konsep yang mengimplementasikan ITarget (yaitu UserReference, GroupReference, dan Root) harus menyediakan implementasi konkret untuk metode-metode ini.

```
concept behavior UserReference {

  constructor {
    <no statements>
  }

  public virtual string getTargetName( )
    overrides ITarget.getTargetName {
    this.user.name;
  }

  public virtual string getType( )
    overrides ITarget.getType {
    "u";
  }

  public virtual string getTargetName( )
    overrides UserLike.getTargetName {
    this.user.name;
  }

}
```

```
concept behavior GroupReference {

  constructor {
    <no statements>
  }

  public virtual string getTargetName( )
    overrides ITarget.getTargetName {
    this.group.name;
  }

  public virtual string getType( )
    overrides ITarget.getType {
    "g";
  }

}
```

Hasilnya, Templat generator tidak perlu lagi memeriksa tipe dari target. Ia dapat memanggil node.target.getType() dan node.target.getTargetName() secara polimorfik. Logika untuk membedakan user dan group telah diabstraksi ke dalam *Behavior* konsep.

2. Helper Method Permission.getVal()

   Masalah: Generator perlu mengonversi tiga nilai enumerasi (read, write, execute) menjadi satu string format rwx (misalnya, rwx, r--, rw-).

   Solusi: Behavior pada konsep Permission mendefinisikan metode helper getVal().

```
public string getVal( ) {
  string conf = "";
  if (this.read.is(allow)) {
    conf += "r";
  } else {
    conf += "-";
  }
  if (this.write.is(allow)) {
    conf += "w";
  } else {
    conf += "-";
  }
  if (this.execute.is(allow)) {
    conf += "x";
  } else {
    conf += "-";
  }
  conf;
}
```

   Hasil: Logika if-then-else yang kompleks ini kini terenkapsulasi di dalam konsep Permission. Templat generator menjadi sangat bersih, hanya perlu memanggil permissionNode.getVal() untuk mendapatkan string rwx yang sudah diformat dengan benar.

IV.5. Generator

Aspek *Generator* mendefinisikan semantik denotasional dari DSL, yaitu bagaimana model abstrak yang ditulis dalam DSL diterjemahkan (ditransformasi) menjadi model konkret dalam bahasa lain yang sudah ada. Dalam kasus ini, DSL `SystemOp` ditransformasikan menjadi `BaseLanguage` (Java) yang, ketika dieksekusi, akan menghasilkan skrip `bash` sebagai keluaran akhir.

4.5.1. Konfigurasi Pemetaan

Pusat dari generator adalah Mapping Configuration. File ini mendefinisikan aturan-aturan utama untuk transformasi. Aturan yang paling fundamental adalah Root Mapping Rule.

- root mapping rules: concept SystemOp --> SystemImpl

Aturan ini menginstruksikan generator bahwa untuk setiap *node* SystemOp (yang merupakan *root* dari model DSL), sebuah *node* ClassConcept (konsep BaseLanguage untuk kelas Java) bernama SystemImpl harus dibuat. Ini adalah titik masuk utama dari proses transformasi, yang mengubah satu model SystemOp menjadi satu berkas .java.

```
mapping configuration main
top-priority group    false


mapping labels:
  << ... >>

parameters:
  << ... >>

is applicable:
  <always>

conditional root rules:
  << ... >>

root mapping rules:
  concept          SystemOp --> SystemImpl
  inheritors       false
  condition        <always>
  keep input root  default


weaving rules:
```

4.5.1. Templat Utama (Root Template)

Templat SystemImpl adalah berkas BaseLanguage (Java) yang berisi makro-makro generator. Templat ini mendefinisikan kerangka kode Java yang akan dihasilkan.

Strategi Generasi: Java Menghasilkan Skrip Bash

Tujuan akhir dari generator ini adalah untuk menghasilkan sebuah skrip .sh (Bash) yang dapat dieksekusi. Strategi yang digunakan adalah dengan menghasilkan sebuah kelas Java (SystemImpl) yang memiliki metode main().

Ketika dieksekusi, metode main() ini melakukan trik cerdas:

1. Pengalihan *Output*: Ia mengalihkan System.out standar Java ke sebuah FileOutputStream yang menunjuk ke sebuah berkas baru (misalnya, Debian13_sysmod.sh).

```
public static void main(string[] args) {
  string fn = "$[test]" + "_sysmod" + ".sh";
  File outputFile = new File(fn);
  PrintStream originalOut = System.out;

  try (FileOutputStream fos = new FileOutputStream(outputFile, false)) {

    PrintStream ps = new PrintStream(fos);
    System.setOut(ps);
```

2. Pencetakan Skrip: Setiap panggilan System.out.println() setelah pengalihan ini tidak akan mencetak ke konsol, melainkan menulis baris perintah bash langsung ke dalam berkas .sh tersebut.

```
System.out.println("#!/bin/bash");
System.out.println("set -e");
System.out.println("set -x");
System.out.println("SUDO=''");
System.out.println("if [ \"$EUID\" -ne 0 ]; then");
System.out.println("    SUDO='sudo'");
System.out.println("fi");
System.out.println("if ! command -v setfacl >/dev/null 2>&1; then");
System.out.println("    echo \"Installing ACL tools\"");
System.out.println("    $SUDO apt update -y\n    $SUDO apt install -y acl\nfi");
System.out.println("echo 'System Operation " + "$[test]" + " ....'");
System.out.println("echo 'Groups'");
$MAP_SRC$[$LOOP$[printGroupScriptBlock("$[test]"); ]]
System.out.println("echo '--------------------'");
System.out.println("echo 'Users'");
```

3. Metode *Helper*: Logika bash yang kompleks (seperti useradd dengan flag opsional atau setfacl) diabstraksi ke dalam metode-metode *helper* Java *static private* (contoh: printUserScriptBlock, printFolderConfig) untuk menjaga kebersihan dan modularitas kode di dalam main().

```
private static void printUserScriptBlock(string username, string homeDir, list<string> groups) {
  string groupOptions = "";
  if (groups != null && !groups.isEmpty) {
```

```
private static void printGroupScriptBlock(string group) {
  System.out.println("$SUDO groupadd -f " + group);
}
private static void printFolderConfig(string path, string owner, string group, boolean recursive,
    map<string, list<string>> permissions) {
```

4.5.2. Analisis Makro dan Alur Logika

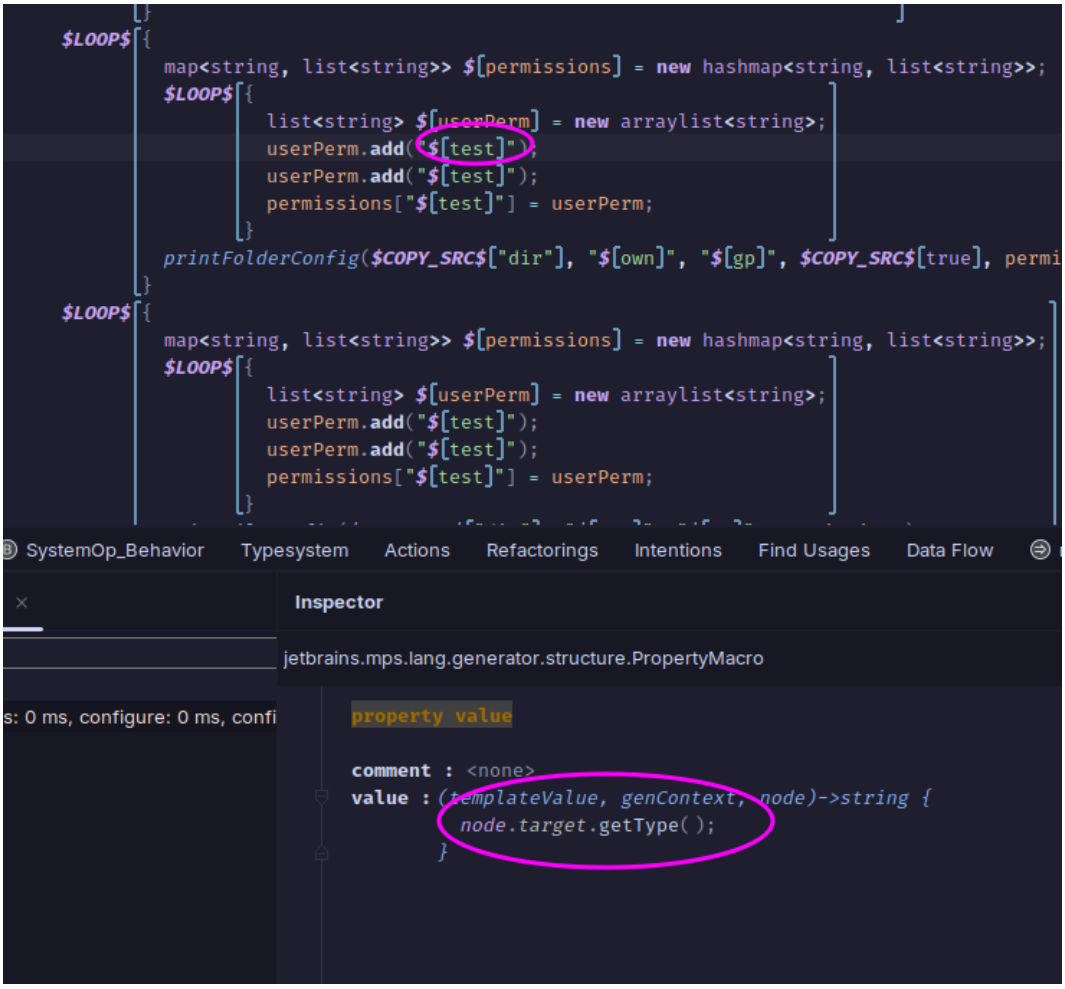
Templat generator diisi secara dinamis menggunakan makro-makro yang merujuk kembali ke *node* input DSL (SystemOp).

- $LOOP$ (Makro Perulangan): Makro ini digunakan untuk mengiterasi koleksi *children* dari *node* input.
    - SystemOp.defgroup.groups: Iterasi ini memanggil printGroupScriptBlock() untuk setiap Group yang didefinisikan.
    - SystemOp.users: Iterasi ini mengumpulkan grup-grup untuk setiap User dan memanggil printUserScriptBlock().
    - SystemOp.folders: Iterasi ini mengumpulkan semua SetPermission dan memanggil printFolderConfig() untuk setiap Folder.
- $var (Makro Properti): Makro ini menyisipkan nilai properti dari *node* DSL ke dalam kode Java. Contohnya, nama kelas Java diambil dari properti name milik SystemOp.
- $COPY_SRC$ (Makro Salin): Makro ini digunakan untuk menyalin *node* yang sudah kompatibel (seperti StringLiteral untuk dir atau BooleanConstant untuk recursive) langsung dari model input ke model output Java.
- genContext.uniqueName(...): Utilitas generator ini digunakan untuk membuat nama variabel Java yang unik di dalam *loop* (misalnya usergroups_a, permissions_b), yang sangat penting untuk mencegah konflik nama variabel di *scope* main().

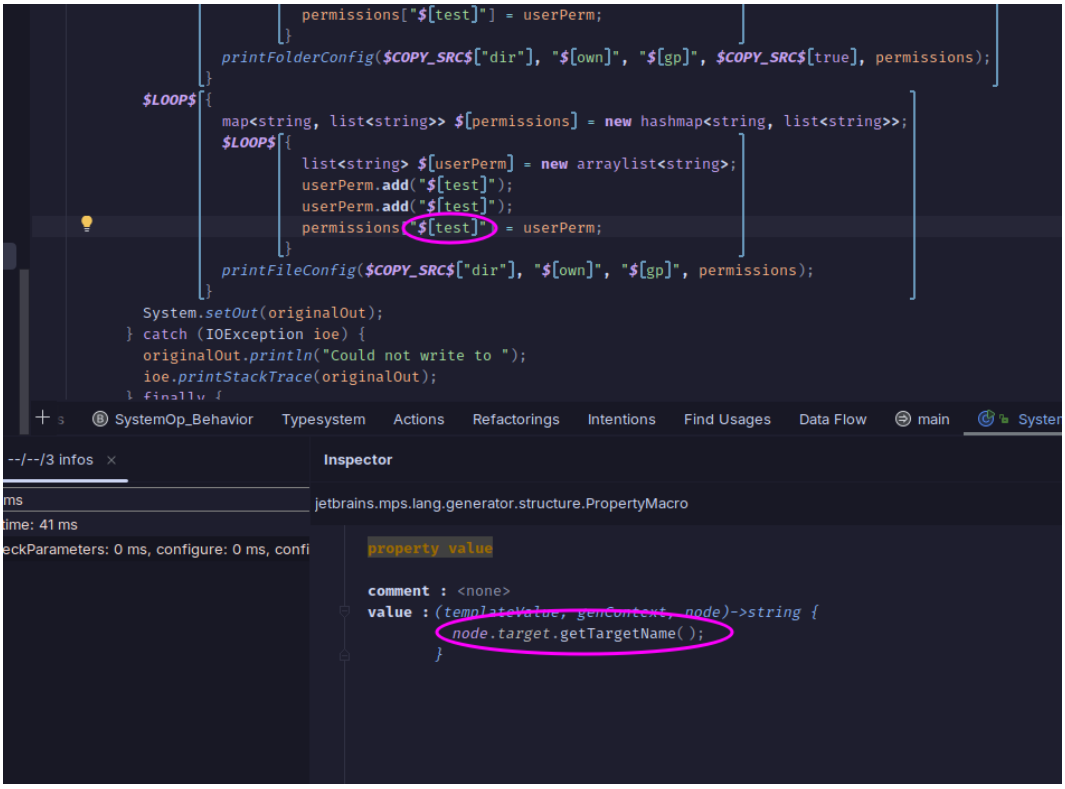
4.5.3. Sinergi dengan Aspek Behavior

Bagian paling elegan dari generator ini adalah bagaimana ia memanfaatkan metode *Behavior* yang telah didefinisikan sebelumnya (IV.4) untuk menyembunyikan kompleksitas. Templat generator tidak lagi berisi logika if-then-else untuk membedakan *user* dan *group*, atau untuk memformat string rwx.

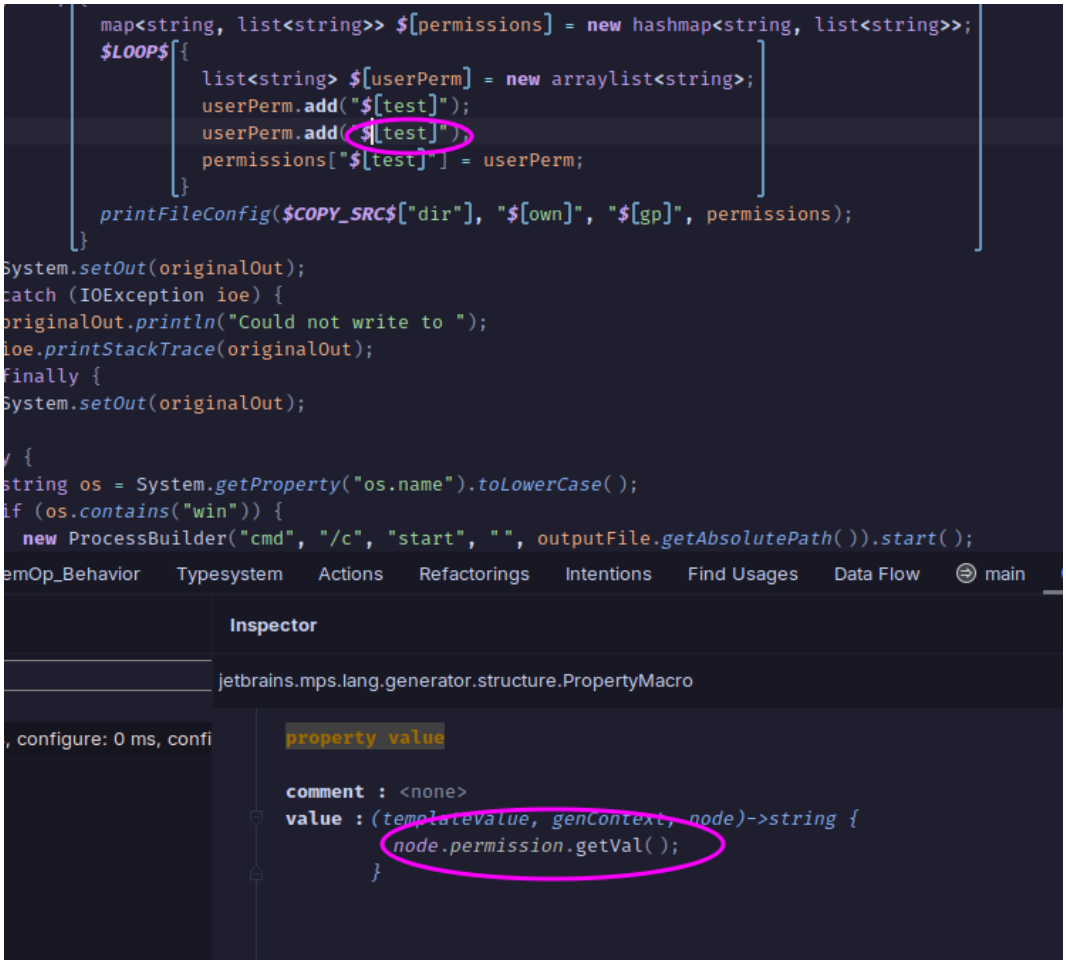
- node.target.getType(): Dipanggil di dalam *loop* userpermissions untuk mendapatkan string u atau g.



- node.target.getTargetName(): Dipanggil untuk mendapatkan nama baba atau furry.



- node.permission.getVal(): Dipanggil untuk mendapatkan string rwx, r--, --- , dst.



Dengan memanggil metode-metode *helper* dari *Behavior* ini, templat generator tetap bersih, deklaratif, dan mudah dipahami. Ia hanya fokus pada *apa* yang harus digenerasi (struktur Java), sementara *bagaimana* cara mendapatkan data spesifik (logika rwx atau u/g) telah diabstraksi ke dalam *Behavior*.

Hasil Akhir: Eksekusi dari kelas SystemImpl.java yang digenerasi adalah sebuah berkas skrip bash yang lengkap, idempoten (aman dijalankan berulang kali), dan siap untuk mengonfigurasi sistem Debian sesuai dengan model DSL yang ditulis oleh pengguna.

IV.6. Hasil Akhir

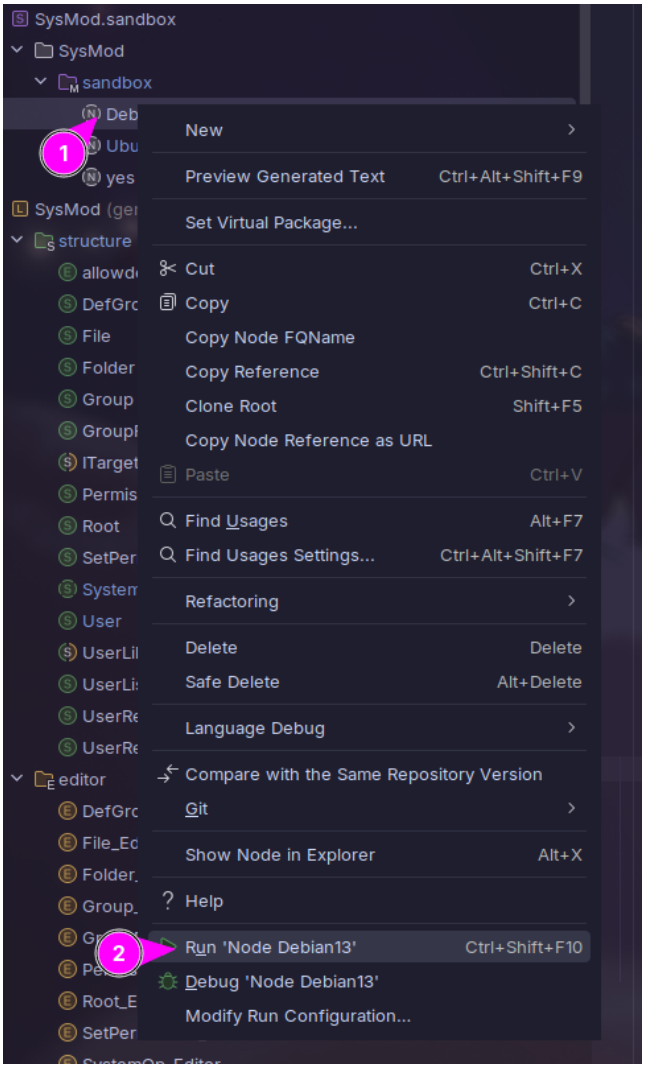
4.6.1. DSL

Contoh program:

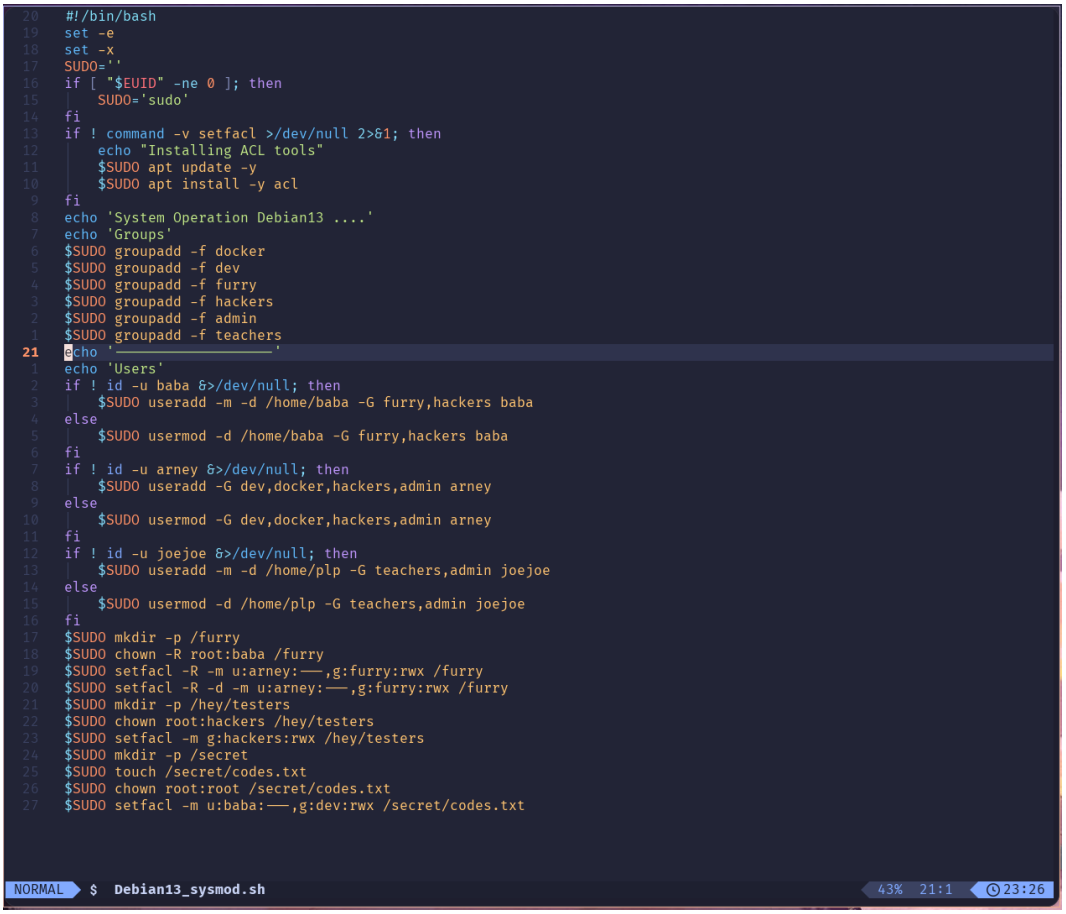
```
SystemOperation Debian13 {
    Groups {
        docker
        dev
        furry
        hackers
        admin
        teachers
    }
    User baba {
      home "/home/baba"
      groups [ furry hackers ]
    }
    User arney {
      home <no home>
      groups [ dev docker hackers admin ]
    }
    User joejoe {
      home "/home/plp"
      groups [ teachers admin ]
    }
    Folder frry {
      dir "/furry"
      recursive true
      owner root
      group baba
      permissions {
        furry read allow , write allow , execute allow
        arney read deny , write deny , execute deny
      }
    }
    Folder test {
      dir "/hey/testers"
      recursive false
      owner root
      group hackers
      permissions {
        hackers read allow , write allow , execute allow
      }
    }
    File secret_codes {
      dir "/secret/codes.txt"
      owner root
      group root
        dev read allow , write allow , execute allow
        baba read deny , write deny , execute deny
    }
}
```

4.6.2. Output

Sudah dikonfigurasikan supaya bisa di-run langsung di MPS



Akan membuka .sh dengan aplikasi pembuka defaultnya dari Sistem Operasi



```
#!/bin/bash
set -e
set -x
SUDO=''
if [ "$EUID" -ne 0 ]; then
    SUDO='sudo'
fi
if ! command -v setfacl >/dev/null 2>&1; then
    echo "Installing ACL tools"
    $SUDO apt update -y
    $SUDO apt install -y acl
fi
echo 'System Operation Debian13 ....'
echo 'Groups'
$SUDO groupadd -f docker
$SUDO groupadd -f dev
$SUDO groupadd -f furry
$SUDO groupadd -f hackers
$SUDO groupadd -f admin
$SUDO groupadd -f teachers
echo '____________________'
echo 'Users'
if ! id -u baba &>/dev/null; then
    $SUDO useradd -m -d /home/baba -G furry,hackers baba
else
    $SUDO usermod -d /home/baba -G furry,hackers baba
fi
if ! id -u arney &>/dev/null; then
    $SUDO useradd -G dev,docker,hackers,admin arney
else
    $SUDO usermod -G dev,docker,hackers,admin arney
fi
if ! id -u joejoe &>/dev/null; then
    $SUDO useradd -m -d /home/plp -G teachers,admin joejoe
else
    $SUDO usermod -d /home/plp -G teachers,admin joejoe
fi
$SUDO mkdir -p /furry
$SUDO chown -R root:baba /furry
$SUDO setfacl -R -m u:arney:——,g:furry:rwx /furry
$SUDO setfacl -R -d -m u:arney:——,g:furry:rwx /furry
$SUDO mkdir -p /hey/testers
$SUDO chown root:hackers /hey/testers
$SUDO setfacl -m g:hackers:rwx /hey/testers
$SUDO mkdir -p /secret
$SUDO touch /secret/codes.txt
$SUDO chown root:root /secret/codes.txt
$SUDO setfacl -m u:baba:——,g:dev:rwx /secret/codes.txt

NORMAL  $  Debian13_sysmod.sh                    43%  21:1  23:26
```

Sudah di coba di Debian melalui Docker Image

```
root@f062fb6231e7:/# ls
bin  boot  dev  etc  furry  hey  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  secret  srv  sys  tmp  usr  var
```

```
root@f062fb6231e7:/# cat etc/group
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:
tty:x:5:
disk:x:6:
lp:x:7:
mail:x:8:
news:x:9:
uucp:x:10:
man:x:12:
proxy:x:13:
kmem:x:15:
dialout:x:20:
fax:x:21:
voice:x:22:
cdrom:x:24:
floppy:x:25:
tape:x:26:
sudo:x:27:
audio:x:29:
dip:x:30:
www-data:x:33:
backup:x:34:
operator:x:37:
list:x:38:
irc:x:39:
src:x:40:
shadow:x:42:
utmp:x:43:
video:x:44:
sasl:x:45:
plugdev:x:46:
staff:x:50:
games:x:60:
users:x:100:
nogroup:x:65534:
docker:x:1000:arney
dev:x:1001:arney
furry:x:1002:baba
hackers:x:1003:baba,arney
admin:x:1004:arney,joejoe
teachers:x:1005:joejoe
baba:x:1006:
arney:x:1007:
joejoe:x:1008:
root@f062fb6231e7:/#
```

BAB V. PENUTUP

V.1. Evaluasi Usability DSL

Aspek Usability DSL	Deskripsi	Skor	Catatan
Learnability	LearnabilitySeberapa mudah memahami struktur DSL pertama kali	4	Sangat intuitif. Strukturnya (Groups, User, Folder) logis dan mengikuti alur konfigurasi yang umum.
Expressiveness	Dapat mengekspresikan aturan domain dengan jelas	5	Sangat ekspresif. Mampu menangani *ownership* (owner/group), ACLs (permissions), referensi ke *user* dan *group* dalam satu blok, dan bahkan properti opsional (seperti home <no home>) dengan sangat baik.
Simplicity	tySintaks tidak berlebihan dan mudah diingat	4	Sintaksnya bersih dan minimal. Blok permissions memang agak verbose (panjang), namun ini adalah pilihan desain yang baik untuk mengutamakan kejelasan (clarity) di atas keringkasan (brevity).
Feedback	Pesan kesalahan dan petunjuk dari editor jelas	5	Ini adalah keunggulan utama desain Anda. Penggunaan ScopeProvider (IV.3, IV.4) memberikan *feedback* instan dan *type-safe* yang luar biasa, memastikan referensi user dan group *hanya* bisa merujuk pada entitas yang ada di dalam SystemOperation yang sama.
Consistency	Istilah dan format DSL konsisten	5	Sangat konsisten. Semua deklarasi level atas (User, Folder, Groups) menggunakan pola keyword { ... } yang seragam. Aturan properti (dir "...", owner root) juga konsisten.
Error Tolerance	Kesalahan kecil tidak langsung menyebabkan gagal	4	Editor MPS (projectional editor) pada dasarnya mencegah *semua* kesalahan sintaksis murni (typo). Kesalahan struktural

			(misalnya, lupa mengisi dir) langsung ditandai sebagai eror, yang merupakan *feedback* yang baik, bukan *tolerance* yang buruk.
Overall Satisfaction	Seberapa puas pengguna terhadap DSL ini	5	Ini adalah desain DSL yang sangat matang. Anda berhasil menggunakan fitur-fitur terbaik MPS (*Scope*, *Behavior*, *Interfaces*, *Generator*) untuk menciptakan alat yang aman, terekspresi, dan menghasilkan skrip bash yang idempoten dan benar secara semantik. *Excellent work.*

V.2. Kesimpulan

Dari hasil perancangan dan implementasi, DSL untuk manajemen hak akses sistem operasi ini:

- Berhasil merepresentasikan domain secara jelas dengan struktur bahasa yang mudah dipahami.
- Dapat diimplementasikan dengan baik menggunakan JetBrains MPS.
- Mampu menghasilkan model yang valid dan mudah diekspor ke bentuk konfigurasi.
- Berdasarkan evaluasi, DSL memiliki tingkat usability yang baik, dengan kekuatan utama pada expressiveness dan consistency.

LAMPIRAN

Link Github

https://github.com/PLP-uwu-team/sysmod-language-mps