Politiche amministrative – 1

Definiscono a chi spetta il compito di concedere e/o revocare le autorizzazioni all'accesso.

- Amministrazione centralizzata: un singolo amministratore (o gruppo di amministratori) ha il controllo sugli accessi di tutto il sistema
- Amministrazione gerarchica: un singolo amministratore è responsabile per assegnare responsabilità amministrative ad altri amministratori.
- Amministrazione cooperativa: le autorizzazioni non possono essere specificate da un singolo amministratore, ma richiedono la cooperazione di più amministratori.

Politiche amministrative – 2

- Ownership: Il proprietario di un oggetto può amministrare gli accessi all'oggetto.
- Amministrazione decentralizzata: l'amministrazione delle autorizzazioni è suddivisa fra più utenti (o gruppi di utenti).
  - spesso associata all'ownership. Il proprietario di un oggetto può delegare i privilegi amministrativi ad altri.

  L'amministrazione decentralizzata introduce flessibilità nella gestione, nascono però complicazioni.

Amministrazione decentralizzata

Ci sono diverse politiche di amministrazione decentralizzata, che differiscono nel modo in cui rispondiamo alle seguenti domande.

- quale deve essere la granularità delle autorizzazioni amministrative?
- può la concessione essere ristretta?
- chi può revocare le autorizzazioni?
- cosa ne facciamo delle autorizzazioni che colui a cui revochiamo il privilegio amministrativo ha garantito?

Amministrazione decentralizzata

Ci sono diverse politiche di amministrazione decentralizzata, che differiscono nel modo in cui rispondiamo alle seguenti domande.

- quale deve essere la granularità delle autorizzazioni amministrative?
  può essere al fine livello di granularità del singolo accesso (azione, oggetto)
- può la concessione essere ristretta?
  generalmente no. però potrebbe essere utile
- chi può revocare le autorizzazioni?
  chi le ha concesse; il proprietario; qualsiasi amministratore
- cosa ne facciamo delle autorizzazioni che colui a cui revochiamo il privilegio amministrativo ha garantito?
  le cancelliamo? (non sempre bello)
  le lasciamo? (e se rimangono “pendenti”)?

Modello originale di System R

L'utente che crea una relazione (di base) ne è il proprietario e può concedere ad altri autorizzazioni su questa.

Le autorizzazioni possono essere concesse con la grant option

Un utente che ha la autorizzazione per un accesso a su una relazione T con la grant option può concedere ad altri autorizzazioni per l'accesso a su T

Modello originale di System R – revoca 1

Quando ad un utente è revocata la grant option su un accesso tutte le autorizzazioni che non sarebbero esistite se l'utente non avesse mai ricevuto la autorizzazione revocata devono essere ricorsivamente cancellate.

- $AUTH$ lo stato di autorizzazione iniziale
- $G_1, \ldots, G_n$ sequenza (storia) di operazioni di grant
- $AUTH'$ stato risultante dopo le concessioni $G_1, \ldots, G_n$
- La revoca di una concessione G_k deve risultare nello stato di autorizzazione $AUTH''$ che sarebbe risultato dalla esecuzione su $AUTH$ della sequenza di concessioni $G_1, \ldots, G_{k-1}, G_{k+1}, \ldots, G_n$

Modello originale di System R – revoca 2

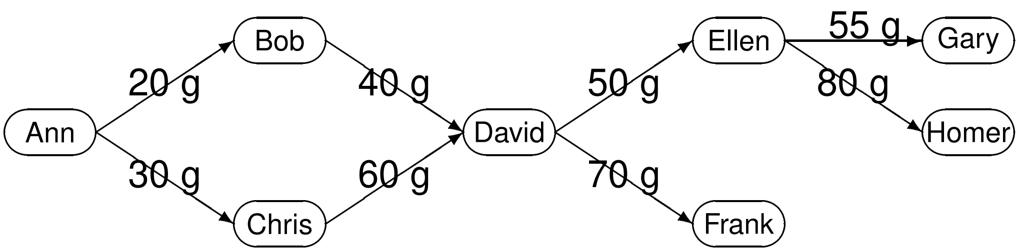
Per implemetare la revoca con la semantica descritta è necessario tenere traccia, per ogni autorizzazione:

- se ha la grant option
- chi la ha concessa
- il tempo al quale è stata concessa

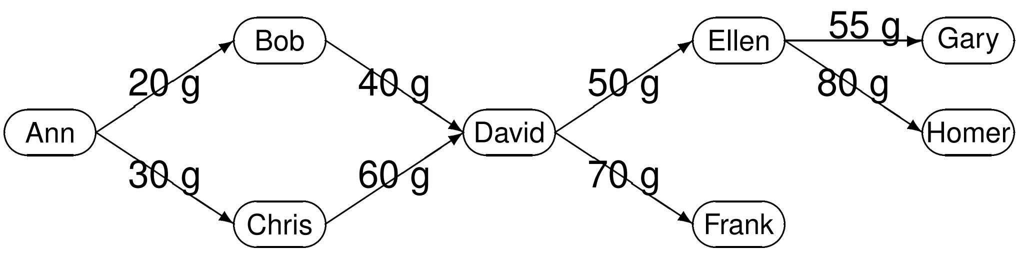
Sia (u,a,T,go,grantor,tempo) una autorizzazione revocata:

1. Determina il tempo minimo t_m delle autorizzazioni di u per l'accesso a alla tabella T con la grant option
2. Revoca ricorsivamente tutte le autorizzazioni per a su o garantite da u prima del tempo t_m

Modello originale di System R – esempio di revoca

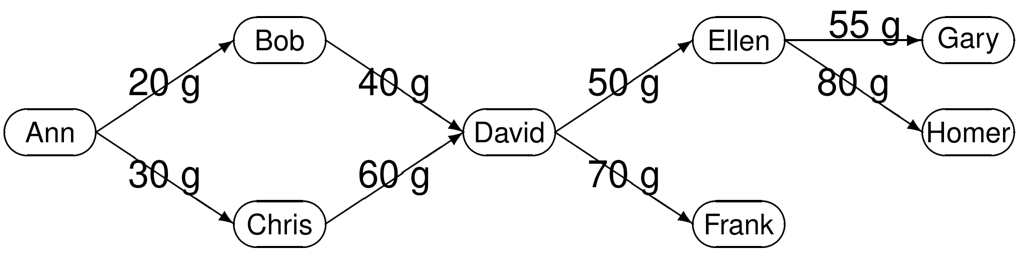


Modello originale di System R – esempio di revoca

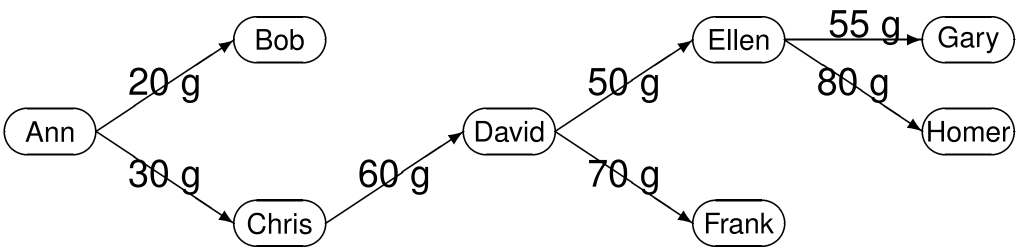


Bob revoca la autorizzazione a David

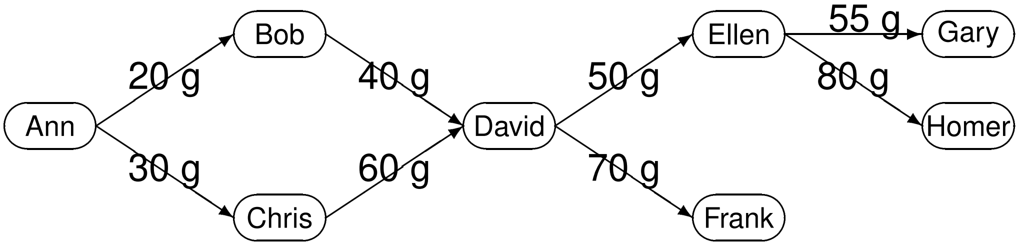
Modello originale di System R – esempio di revoca



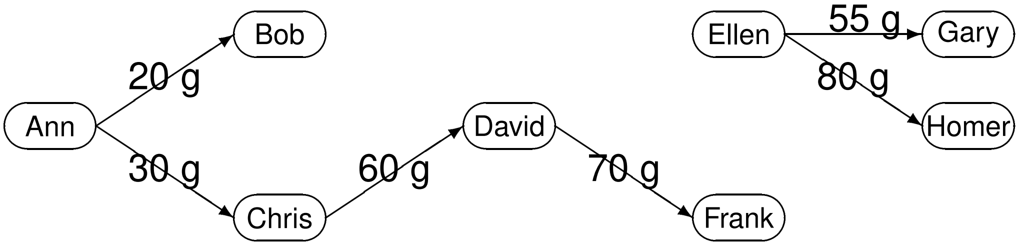
Bob revoca la autorizzazione a David



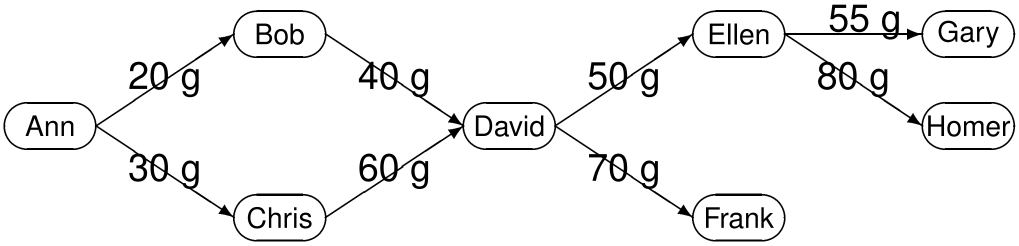
Modello originale di System R – esempio di revoca



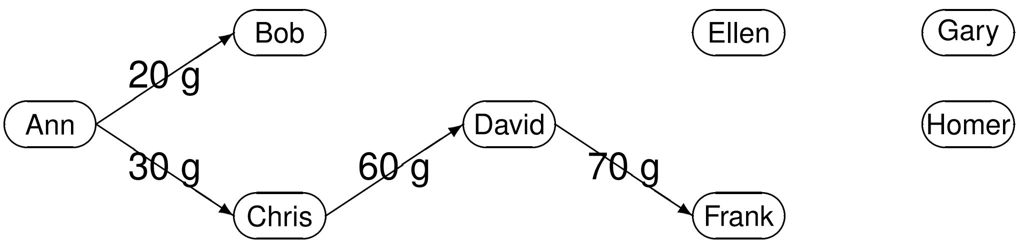
Bob revoca la autorizzazione a David



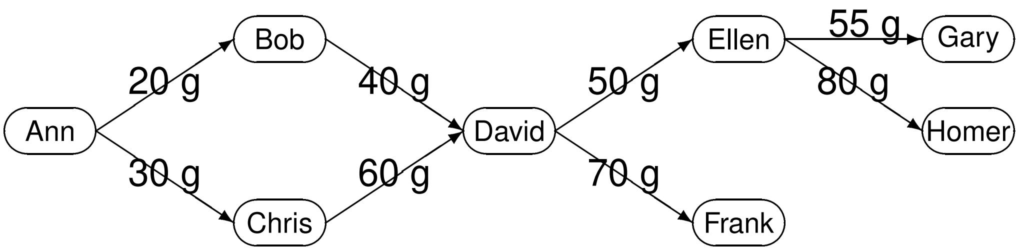
Modello originale di System R – esempio di revoca



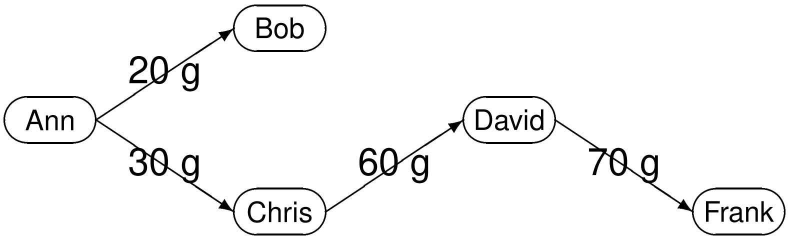
Bob revoca la autorizzazione a David



Modello originale di System R – esempio di revoca



Bob revoca la autorizzazione a David



Revoca in System R

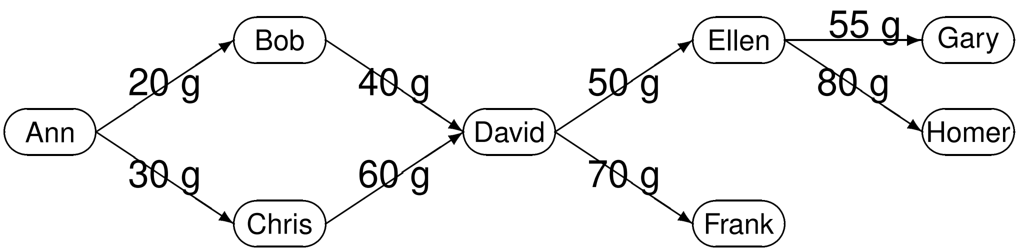
La revoca ricorsiva può non essere sempre voluta

- Non necessariamente tutto quello che ha concesso un utente va calcellato (es., utente cambia lavoro per promozione)
- Tutti i programmi dipendenti dalle autorizzazioni revocate vengono invalidati

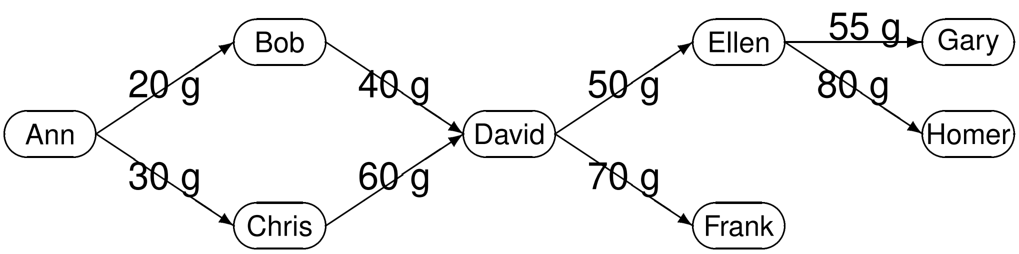
Possibile alternativa

- invece di revocare le autorizzazioni rispecificarle per l'utente che ha chiesto la revoca

Revoca non ricorsiva – esempio

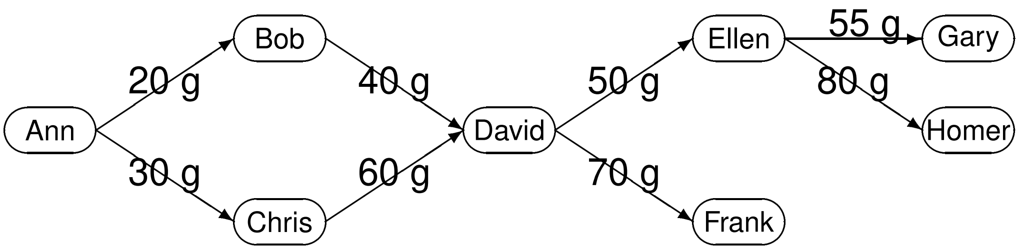


Revoca non ricorsiva – esempio

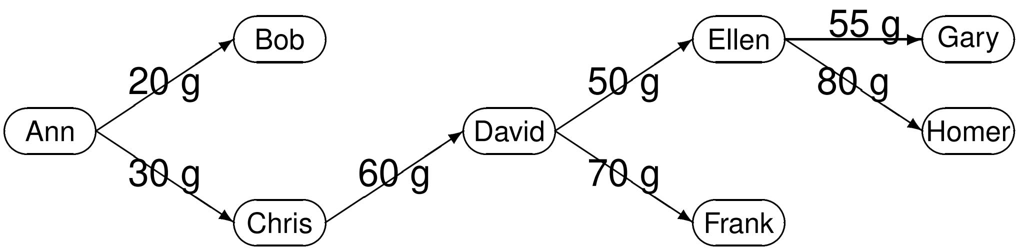


Bob revoca la autorizzazione a David

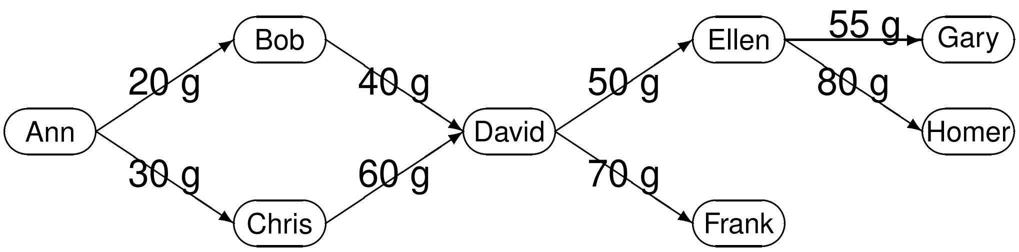
Revoca non ricorsiva – esempio



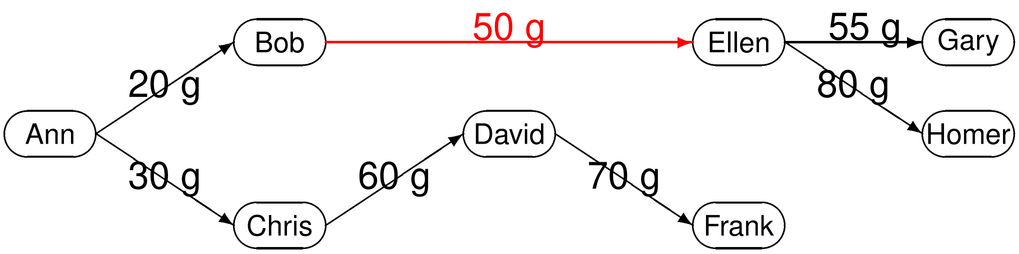
Bob revoca la autorizzazione a David



Revoca non ricorsiva – esempio



Bob revoca la autorizzazione a David



Revoca in SQL

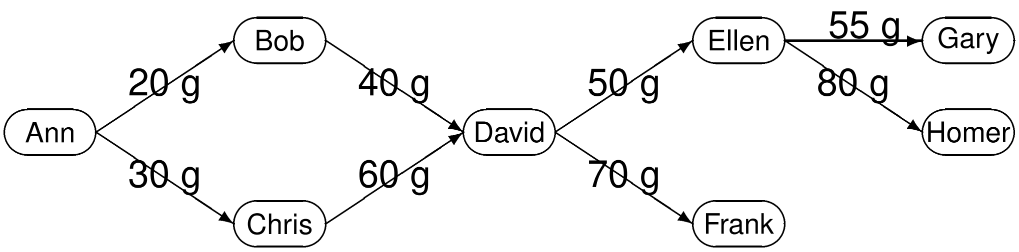
La revoca può essere richiesta cascade (ricorsiva) o no cascade (non ricorsiva

Ricorsiva È stata modificata togliendo la considerazione del tempo

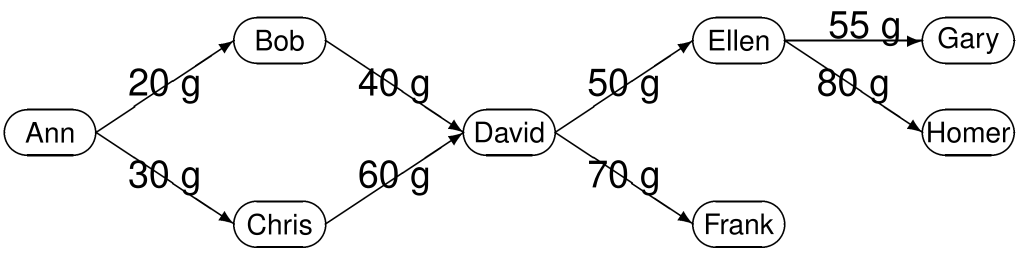
- Le autorizzazioni garantite da un utente vengono cancellate solo se l'utente non ha più la grant option per l'accesso (a prescindere dal tempo)
  - attenzione ai cicli

Non ricorsiva Se la cancellazione della autorizzazione che viene revocata comporta cancellazioni ricorsive, la revoca non viene effettuata

Revoca ricorsiva in SQL – esempio

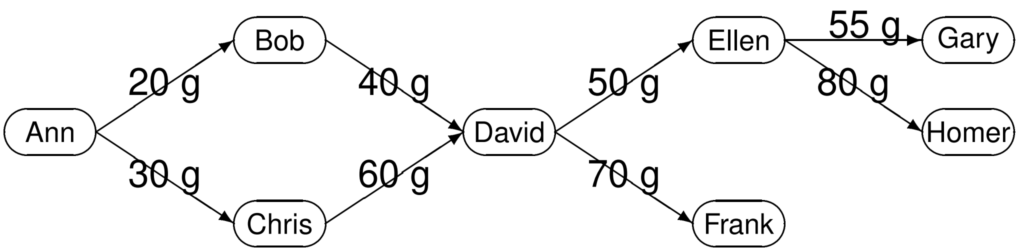


Revoca ricorsiva in SQL – esempio

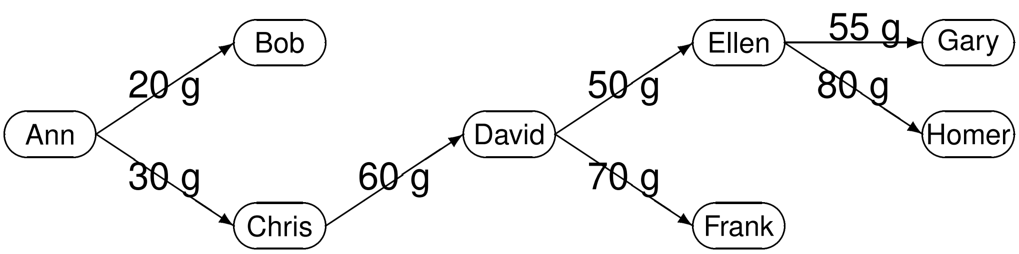


Bob revoca la autorizzazione a David

Revoca ricorsiva in SQL – esempio



Bob revoca la autorizzazione a David



Sicurezza e Privatezza – Lezione 6 12 © Pierangela Samarati

DAC – autorizzazioni con condizioni

Le autorizzazioni possono contenere condizioni che ne limitano la validità

system-dependent valutano il soddisfacimento di predicati di sistema

- locazione (es., solo da macchine all'interno della rete locale).
- tempo (es., solo dalle 9:00am alle 5:00pm)

history dependent dipendono dalla storia delle richieste

content-dependent dipendono dal valore dei dati, possono:

- restringere l'insieme di oggetti a cui si può accedere (es., si può accedere solo a file il cui contenuto soddisfa certe condizioni)
- restringere l'accesso a porzioni di oggetti (es. solo a tuple di una relazione per cui i valori degli attributi soddisfano certe condizioni) $\Longrightarrow$ query modification

Autorizzazioni su/per gruppi

Autorizzazioni riferite singole entità (utenti, file, ...) troppo pesanti da gestire

- supporto di astrazione (raggruppano le entità). Generalmente basate su relazioni gerarchiche: utenti/gruppi; oggetti/classi; file/directory; pattern di indirizzi IP numerici e simbolici....

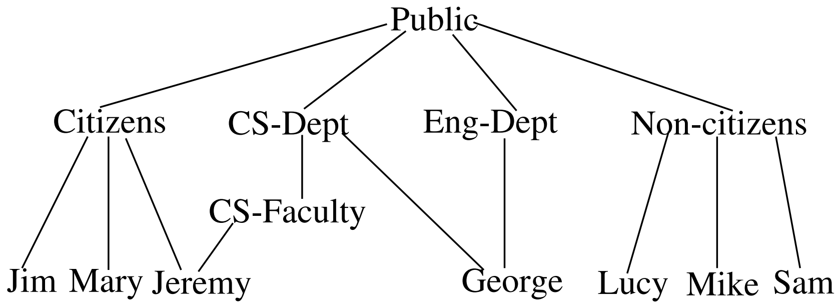
  Gerarchia: DAG o albero

  Le autorizzazioni possono essere specificate su astrazioni e propagarsi lungo la gerarchia.

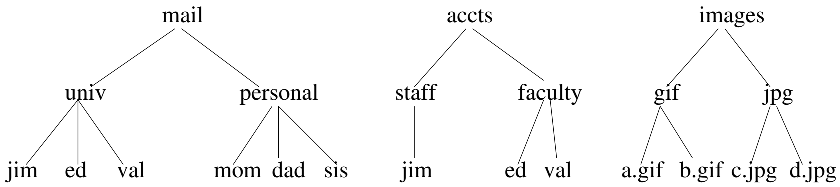
  $\Longrightarrow$ Autorizzazioni concesse a un gruppo di utenti si applicano a tutti i membri del gruppo

  $\Longrightarrow$ Autorizzazioni garantite per una astrazione degli oggetti si applicano a tutti i suoi membri

Esempio di gerarchia utenti-gruppi



Esempio di gerarchia di oggetti



Autorizzazioni su/per gruppi

Non sempre le autorizzazioni specificate su un elemento non minimale di gerarchia si propagano.

Certe autorizzazioni si riferiscono a modi di accesso esercitabili sull'elemento (non minimale)

Es., in Unix i privilegi su directory non si propagano ai file in essa contenuti; definiscono privilegi su directory

- possono essere utilizzati per definire precondizioni sull'accesso ai suoi componenti (es., per accedere ad un file in Unix è necessario avere privilegio di esecuzione (x) sulle directory nel suo pathname assoluto)

Supporto di eccezioni

Vantaggio delle astrazioni: semplificano la gestione di autorizzazioni

L'utilità delle astrazioni è limitata se è possibile supportare eccezioni.

Es., "tutti gli impiegati tranne Alice possono leggere il file letteraA"

- devo specificare una autorizzazione per ogni impiegato tranne che per Alice

Soluzione: supporto di autorizzazioni negative

- (Impiegati, read, file, +)
- (Alice, read, file, -)

La presenza sia di autorizzazioni positive sia negative può portare inconsistenze (o ambiguità). Cosa facciamo?

Permessi e negazioni – 1

Le autorizzazioni negative sono un facile modo di supportare eccezioni

Le autorizzazioni negative sono state inizialmente introdotte, separatamente dalle autorizzazioni positive

politica aperta: autorizzazioni (negative) specificano negazioni all'accesso. Tutti gli accessi non esplicitamente negati sono permessi; in contrapposizione a

politica chiusa: autorizzazioni positive specificano permessi di accesso. Solo gli accessi esplicitamente autorizzati possono essere eseguiti.

Politiche più recenti supportano entrambe, ma

- cosa facciamo se per un accesso abbiamo sia + sia -? (inconsistenza)
- cosa facciamo se per un accesso non abbiamo nè + nè -? (non completezza)

Permessi e negazioni – 2

Non completezza può essere risolta

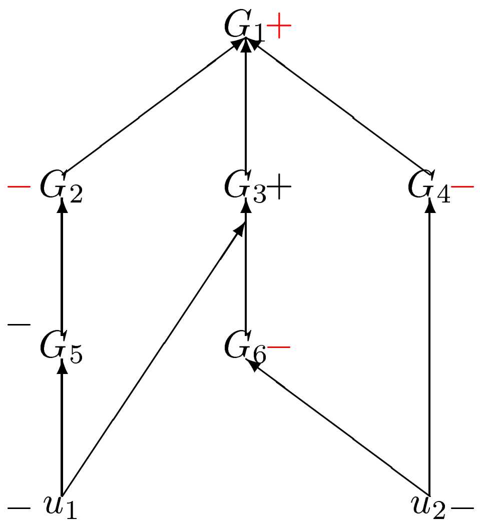
- assumendo completezza (es. Orion): per ogni accesso almeno una autorizzazione o positiva o negativa deve esistere $\Longrightarrow$ troppo pesante
- assumendo come base o la politica aperta o la politica chiusa politica di default

Permessi e negazioni – 3

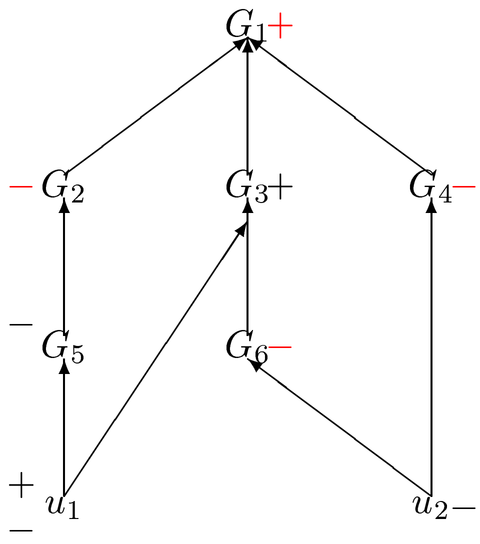
Diverse politiche per la risoluzione dei conflitti sono possibili

- denials-take-precedence le autorizzazioni negative vincono (principio fail safe)
- most-specific-takes-precedence l'autorizzazione "più specifica" vince
- most-specific-along-a-path-takes-precedence l'autorizzazione che è "più specifica" vince solo sui cammini che passano dall'autorizzazione
  $\Rightarrow$ le autorizzazioni si propagano lungo la gerarchia fino a che trovano autorizzazioni più specifiche

Esempi di risoluzione dei conflitti



più specifica



più specifica lungo un cammino

Most specific takes precedence – 1

La politica che fa vincere la autorizzazione più specifica è intuitiva e naturale ...... ma

- può non risolvere tutti i conflitti

  `(CS-Dept, read, letteraA, +)`

  `(Eng-Dept, read, letteraA, -)`

  cosa facciamo per `George` che appartiene ad entrambi?

- cosa facciamo nel caso di gerarchie diverse?

  `(Impiegati, read, letteraA, +)`

  `(Alice, read, Lettere, -)`

Most specific takes precedence – 2

- in alcuni casi può non essere voluta.

  Es., autorizzazioni che non permettono eccezioni

  – `(Impiegati, read, bulletin-board, +)`

    Voglio che nessuno sia in grado di specificare eccezioni

  – `(Impiegati, read, budget, +)`

    `(Impiegati_temporanei, read, budget, -)`

    Non voglio che la restrizione specificata per gli Impiegati_temporanei possa essere bypassata.

Altre politiche per la risoluzione dei conflitti – 1

Strong vs weak (e.g., Orion) Le autorizzazioni sono classificate come forti o deboli

- Le autorizzazioni forti non possono essere sovrascritte
  $\Longrightarrow$ non ammettono eccezioni
- Le autorizzazioni deboli possono essere sovrascritte.
  - le autorizzazioni forti sovrascrivono sempre le autorizzazioni deboli (indipendentemente dalla loro specificità o segno)
  - le autorizzazioni deboli si possono sovrascrivere in accordo alla politica della autorizzazione più specifica lungo un cammino

Alcune limitazioni/complicazioni:

- Supporta solo due livelli di priorità, può non bastare
- Le autorizzazioni forti devono essere consistenti
  Non è facile quando i gruppi sono "dinamici"

Altre politiche per la risoluzione dei conflitti – 2

Esplicità priorità le autorizzazioni hanno delle priorità specifiche associate

- difficile da gestire

Posizionale la autorizzazione delle autorizzazioni dipende dal loro ordine nella lista di autorizzazione

- scarica la responsabilità di risolvere i conflitti a chi specifica le autorizzazioni
- difficile da applicare in caso di amministrazione decentralizzata

Grantor-dependent la priorità delle autorizzazioni dipende da chi le ha garantite

Time-dependent la priorità delle autorizzazioni dipende dal tempo al quale sono state concesse (es., la più recente vince)

- applicabilità limitata

Politiche per la risoluzione dei conflitti

Le politiche non sono in alternativa. Es., si può applicare prima "most specific" e poi "denials-take-precedence" sui conflitti rimasti

Non esiste una politica migliore di un'altra.

Diverse politiche corrispondono a diverse scelte che possono essere prese per la risoluzione dei conflitti.

Cercare di supportare tutte le differenti semantiche che la negazione può avere (negazione forte, eccezione....) porta a modelli non gestibili.

Per questo motivo spesso le autorizzazioni negative non vengono usate.

Però sono utili.

$\Longrightarrow$ Sistemi che supportano autorizzazioni negative adottano una specifica politica per la risoluzione di conflitti

Autorizzazioni positive e negative in Apache

Le autorizzazioni possono essere positive e negative.

È specificato un ordine che descrive come interpretarle. Due scelte:

deny,allow sono valutate prima le autorizzazioni negative e l'accesso è permesso per default. Un richiedente ha accesso se non ha alcuna autorizzazione negativa *oppure* ha una autorizzazione positiva.

allow,deny sono valutate prima le autorizzazioni positive e l'accesso è negato per default. Un richiedente non ha accesso se non ha alcuna autorizzazione positiva *oppure* se ha una autorizzazione negativa.

Esempio

Order Deny,Allow
Deny from all
Allow from .crema.unimi.it