3. SISTEMA OPERATIVO

Introduzione

Viene naturale pensare che lo stato dell'arte in un certo ambito tecnologico sia di quanto meglio possibile fare con gli strumenti a disposizione. I sistemi evolvono, e durante un'evoluzione solo piccole cose si trasformano, le basi e la struttura rimangono inalterate. A distanza di quasi 40 anni dai primi sistemi unix, 25 dalle prime interfacce grafica, non sarebbe forse necessario una *rivoluzione*? I principi che sono alla base dei sistemi operativi moderni, sono stati concepiti quando le macchine erano dotate di scarsissime risorse (qualche KB di ram, qualche MB di disco, e processori a meno di 1 MHZ), ed erano dedite perlopiù a scopi scientifici e industriali. Se oggi un progettista dovesse pensare un sistema operativo da zero, senza alcuna influenza, lo penserebbe di nuovo in questo modo?

Evoluzione dei sistemi operativi

I primi terminali non avevano ancora interfacce grafiche (GUI), l'interazione con l'utente avveniva principalmente tramite console a riga di comando: sembra abbastanza naturale che il primo pensiero per la comunicazione tra due processi, sia stata quello di trasformare l'output testuale di un processo A, in input testuale per il processo B. Ogni emissione testuale di un processo nasce per essere leggibile all'utente, e non alla macchina. La codifica e decodifica in testo introducono un costo e una latenza inutile, e vincolano i processi a mantenere invariato il modo con cui forniscono l'output (pena la riscrittura di tutte le applicazioni baste su tale output).

Dal punto di vista dell'interfaccia utente, il modello a finestra era quanto di più vicino possibile alla realtà del tempo: se una singola applicazione esponeva la propria interfaccia in un'area rettangolare (il monitor), più applicazioni dovevano esporre contemporaneamente la propria interfaccia in più rettangoli indipendenti, chiamate finestre. Le finestre rimangono oggetti distinti, e non integrabili: molto spesso è necessario includere parti di interfaccia fornita da una applicazione, all'interno di un altra, (un browser all'interno di un client di posta per l'e-mail di tipo HTML, oppure uno strumento per l'image processing all'interno di un word processor). Microsoft con la tecnologia OLE / Automation (basato su COM) ha tentato di risolvere questo problema, permettendo di integrare e scambiare oggetti tra applicazioni

Lo sviluppo delle rete, e l'esigenza sempre più frequente di comunicare tra sistemi remoti ed eterogenei, ha portato alla nascita di decine di protocolli, ognuno pensato per una servizio specifico. Ogni protocollo, sebbene con sintassi e regole diverse affronta lo stesso problema di codifica di messaggi con eventuali parametri, valori di ritorno e gestione degli errori. Il concetto di messaggio e molto simile a quello di funzione: non sarebbe stato forse migliore adottare un sistema universale per l'invocazione di procedure remote, senza studiare decine di protocolli diversi? Oggi giorno esistono molti sistemi per l'invocazione di procedure remote, come CORBA, DCOM, RPC, RMI, SOAP, ma non esiste ancora una standard o regola che imponga ai progettisti di basarsi su tali sistemi per la comunicazione remota, e continuano a persistere protocolli vecchi e non sicuri, come HTTP, FTP, SMTP, POP3, etc.

Il file nasce negli anni 50, il nome deriva dal raccoglitore che conteneva le schede perforate che costituivano un programma. La tecnologia si è evoluta, ma il concetto che rappresentava allora è rimasto pressoché immutato: il file è tutt'ora solamente un insieme contiguo di byte. Quando le memorie erano di piccole dimensioni, la quantità di file che un sistema poteva memorizzare era molto modesta, il semplice nome era sufficiente come sistema di catalogazione e ricerca. Un primo passo verso un sistema più organizzazione si è fatto con i file system gerarchici: l'organizzazione dei file in cartelle è quanto di più simile a quello che era la realtà dove *lo stesso documento può esistere in una sola cartella*, collocata ad esempio nell' edificio A, stanza B, Armadietto C, Scaffale D. Le cartelle permettono una organizzazione statica, e forniscono un'unica vista dei file, quando in molti casi servirebbero viste personalizzare a seconda del contesto, e del tipo di ricerca che vogliamo effettuare:

Nella realtà le canzoni sono organizzate in album. I nostri file musicali quindi potrebbero riprendere la stesa struttura di Artista -> Album -> Canzone all'interno del file system. Se un giorno volessimo ascoltare tutte le canzoni di un certo periodo, o genere ? A quel punto una struttura più comoda sarebbe Genere -> Canzone. Non è quindi possibile definire in modo statico un sistema di organizzazione migliore in quanto è dipendente dal contesto e dalle esigenze del momento.

I file sono elementi molto più vicini alla macchina che all'utente il quale non è interessa a come il sistema codifichi le informazioni su disco, ma solo a manipolare i contenuti nel momento in cui gli necessitano.

Oggetti e contenitori

Un sistema di organizzazione alternativo al file system è un sistema orientato agli *oggetti* che permetta di manipolare direttamente i contenuti, indipendentemente dalla loro provenienza, formato o codifica. Un oggetto può essere un elemento astratto come una brano musicale in formato MP3 e un immagine JPEG, oppure un elemento del mondo reale come mouse e tastiera. Un oggetto è descritto da una serie di attributi, che sono propri della classe di appartenenza (detta anche *tipo*).

Un brano musicale è descritto dagli attributi titolo, autore, album, etc.
Un contatto è descritto dagli attributi nome, cognome, numero di telefono, etc.
Un messaggio è descritto dagli attributi mittente, destinatario, oggetto, data, etc.

Due oggetti sono dello stesso tipo se forniscono lo stesso servizio o astrazione (ad esempio un mouse, un touchpad, o un tablet appartengono tutti alla tipologia "Dispositivi di puntamento"). Ogni oggetto ha un nome con il quale si presenta all'utente. Il nome è normalmente una combinazione degli attributi dell'oggetto stesso, (es. il nome di un brano musicale può essere "<Autore> – <Titolo>"), e non deve essere necessariamente univoco.

Gli oggetti sono organizzati in *container*. Un *container* è la generalizzazione della cartella, un oggetto in grado di contenere altri oggetti

Un album musicale è un oggetto, contenitore di brani musicali
Una rubrica è un oggetto, contenitore di contatti

Ogni container può scegliere liberamente dove e come leggere (o scrivere) gli oggetti in lui contenuti, e questa scelta sarà sempre mascherate all'utente. L'esistenza di oggetti con il ruolo di contenitori (che possono a loro volta contenere altri contenitori), conferisce al sistema una struttura gerarchica ad albero, simile al file system, ma a differenza del file system, lo stesso oggetto può comparire in più container contemporaneamente, e la sua collocazione fisica è completamente irrilevante[1]. In modo del tutto analogo ai file, un oggetto è identificato globalmente tramite un path, ovvero una stringa che contiene tutta la sequenza necessaria per raggiungere l'oggetto partendo dal contenitore radice.

Esempi di path:
Contatti / Tutti / Andrea Guerrieri[45]
Media / Audio / Tutti gli album / American Idiot / Green day – Jesus of superbia

Oggetti

Un oggetto è composto da una parte descritta, e da una funzionale. La parte descrittiva, è costituita dagli attributi, che come è stato accennato in precedenza, sono propri del tipo. La parte funzionale da una serie di interfacce, attraverso le quali l'utente invoca e usa i servizi che l'oggetto fornisce. Mentre quello che un oggetto è (i suoi attributi) non può variare con il tempo, in quanto sono proprio quegli attributi a fornirgli una particolare identità, come l'oggetto viene utilizzato, i servizio che fornisce possono cambiare (si può trovare sempre un modo nuovo per usare un oggetto). Per questo motivo la parte funzionale non deve essere dichiarata in modo statico durante la definizione del tipo, ma può essere estesa in qualsiasi momento.

Nel corso della storia, l'ipotetico oggetto di tipo casa ha subito variazioni strutturali: non è possibile oggi descrivere una casa con gli stessi attributi usati per le case delle preistoria. Questa affermazione sembrerebbe essere in contrasto con il concetto di immutabilità degli attributi di un oggetto. In questo caso è stato commesso un errore nella costruzione del modello. La casa non è un oggetto, ma è un servizio: è casa tutto ciò che può contenere un essere vivente. Gli oggetti sono ad esempio caverna, costruzione con frasche, costruzione in mattoni e tutti quanti forniscono il servizio di casa. Con il tempo si sono semplicemente aggiunti nuovi oggetti in grado di fornire tale servizio.

A seconda di come viene sviluppata la parte funzionale e quella descrittiva, l'oggetto può assumere il ruolo di *data object* (detto anche *entity*) o *service object* (detto anche *component*)

I *service object* hanno la parte funzionale che predomina su quella descrittiva (spesso assente). Solitamente ne esiste una sola istanza per tipo, creata durante l'inizializzazione del sistema. Il loro ruolo è di fornire servizi ad altri oggetti e componenti, durante lo svolgimento di una attività. A questa categoria appartengono ad esempio gli oggetti che

[1] Nel file system moderni, è possibile avere lo stesso file in più cartelle diverse tramite hard / soft link.

manipolano i componenti hardware. L'utente non entrerà mai in contatto diretto con questo tipo di oggetti, la cui conoscenza è più utile a programmatori e sistemisti.

I *data object* al contrario hanno la parte descrittiva che predomina su quella funzionale (comunque presente e importante). Il loro ruolo principale è quello di memorizzate dati di un particolare tipo, e risiedono generalmente su *storage*. Uno *storage* è l'astrazione di "tabella", un componente che ha il compito di memorizzare i *data object* dello stesso tipo, permettendone una rapida ricerca. Gli *storage* saranno il meccanismo principale di memorizzazione che di fatto sostituirà il file system. Questo tipo di oggetti saranno quelli con il quale l'utente entrerà in contatto più spesso.

Oggetti contenibili

Solo gli oggetti che possiedono un attributo chiave (univoco e non nullo) e implementano il servizio *data object* (`IDataObject`) possono essere figli di un *container*. La chiave ha il ruolo di fornire un meccanismo rapido di identificazione e riconoscimento per l'oggetto. Purtroppo non tutti i tipo possiedono nativamente un attributo che possa essere chiave, e in questi casi dovrà essere aggiunto un attributo fittizio nella definizione del tipo con il ruolo di chiave, chiamato identificatore (o id).

---

il CAP è un attributo chiave per gli oggetti di tipo"città" (non esistono due città con lo stesso cap), come la partita iva lo è per gli oggetti di tipo"Azienda". Il tipo"brano musicale" non ha alcun attributo univoco (non esiste alcun sistema di identificazione a valenza universale)

L'id è un attributo solitamente numerico che prende il nome del tipo, seguito dalle lettere "Id" (es. *DocumentId* per il tipo *Document*) Quando si crea un nuovo oggetto, l'id può essere assegnato manualmente dall'utente, oppure automaticamente dal sistema (ad esempio tramite un contatore auto-incrementale o un timestamp). Poiché nella maggior parte dei casi non esiste un attributo chiave nativo, per rendere tutti i tipi omogenei è buna norma dichiarare l'id anche se il tipo possiede già una chiave.

Gli attributi possono essere di tipo primitivo (numeri interi, reali, booleani, date, stringhe, etc), oppure oggetti. Un oggetto può essere assegnato ad un attributo in due modi distinti:

- **Per valore:** l'attributo contiene l'intera copia dell'oggetto.
- **Per riferimento**: l'attributo contiene solo il valore di alcuni campi che permettano l'identificazione dell'oggetto a cui si riferisce

La scelta tra queste due modalità deve essere effettuata durante la definizione dell'attributo, e non potrà essere modificata successivamente. Quando si assegna un oggetto per riferimento, viene effettuato un collegamento implicito con l'attributo chiave dell'oggetto referenziato e opzionalmente un collegamento con altri attributi scelti del progettista (magari più semplici da reperire o ricordare rispetto alla chiave)

Nell'oggetto di tipo "brano musicale" l'autore del brano solitamente è referenziato tramite l'attributo "nome" che non è chiave nell'oggetto "artista musicale".
Un oggetto di tipo "libro" viene referenziato solitamente tramite gli attributi " titolo" / "autore", e non la sua chiave (il codice ISBN)

Si ha un riferimento forte, quando il riferimento è attraverso almeno un attributo chiave, altrimenti un riferimento debole. E' sempre preferibile un riferimento forte ma non sempre possibile, la chiave è spesso sconosciuta soprattutto quando ai tratta di un id: essendo un attributo esterno al tipo (e spesso assegnato dal sistema) è possibile che lo stesso oggetto sia presente in due sistemi diversi, con identificativi diversi.

Un contatto che rappresenta fisicamente la stessa persona (stesso oggetto) può essere presente in due rubriche con id diverso!
Ogni magazzino usa un sistema personale per assegnare codici agli articoli (stesso oggetto, diverso id)

Per questa ragione è solito usare nel riferimento attributi nativi del tipo (quindi non id), anche a rischio di avere solo un riferimento debole. Un riferimento debole non potrà essere sempre risolto in modo univoco (potrebbero esister più oggetti che corrispondono alla relazione). Per questo motivo il solo riferimento debole non è sufficiente. Nel momento in cui si accede all'oggetto referenziato, il sistema tenta di creare un riferimento forte, cercando un oggetto conforme al riferimento debole: se viene trovata una sola corrispondenza, il riferimento è automaticamente risolto con la chiave dell'oggetto trovato, se vengono trovate più corrispondenze l'utente deve risolvere il conflitto manualmente. Una volta risolto il riferimento forte, questo valore viene memorizzato permanentemente nell'oggetto.

L'azienda A invia un ordine all'azienda B. L'ordine è composto da un insieme di articoli, identificati dal nome. Per verificare la disponibilità in magazzino, è necessario trovare esattamente a quale oggetto di tipo 'articolo' il

documento fa riferimento. Purtroppo il nome non è univoco, e quindi si ha solo un riferimento debole. Il sistema tenterà in modo automatico di creare un riferimento forte per ogni articolo del documento, con gli articoli presenti in magazzino. Nel caso ci siano più articoli con lo stesso nome, tramite un apposito pannello, l'utente manualmente sceglierà l'articolo giusto, basandosi ad esempio su marca, codice, etc.

Assegnare un oggetto per valore introduce una ridondanza, poiché lo stesso oggetto è presente in più copie diverse nel sistema. Nella maggior parte dei casi è quindi preferibile assegnare per riferimento, ma in altri è obbligatorio assegnare per valore.

Un documento fiscale, una volta archiviato non può più essere modificato. Se l'anagrafica del cliente a cui fa riferimento subisse variazione, queste non dovranno essere riportate nel documento. Se si assegnasse l'attributo "cliente" dell'oggetto "documento fiscale" per riferimento, il documento visualizzerebbe la versione attuale che potrebbe aver subito modifiche, rispetto a quella storica. Se si assegna invece per valore, nel documento c'è una copia dell'oggetto cliente ed eventuali modifiche successive non saranno visibili a meno di una sincronizzazione esplicita da parte dell'utente.

Parte funzionale e interfacce

Un interfaccia è un insieme di metodi (operazioni) tramite i quali si controlla e usa un servizio. Un oggetto si dice che implementa un'interfaccia (o espone un servizio), se è in grado di compiere tutte le operazioni che l'interfaccia prevede. L'implementazione può essere fatta all'interno dell'oggetto stesso, o attraverso un componente esterno.

Un hotel offre un servizio di trasporto che permettere ai suoi clienti di spostarsi da una parte all'altra della città. Il cliente per usufruire di tale servizio, deve recarsi alla reception, e su un apposito modulo specificare luogo di destinazione e durata del viaggio. Una volta effettuata la richiesta l'hotel assegna un mezzo (auto o bici), tramite il quale il cliente può raggiungere il luogo indicato.

- *Il trasporto è un **servizio** che permette ad una persona di spostare tra due luoghi*
- *L'hotel è un oggetto che espone il servizio di trasporto*
- *Il modulo alla reception è **l'interfaccia** attraverso il quale un cliente usa il servizio di trasporto*
- *Automobile e bicicletta sono gli oggetti che **implementano** il servizio di trasporto per l'oggetto hotel*

Qui sotto sono riportare alcune interfacce / servizi tipicamente esposte dai *data object*.

- **Formatter (`IObjectFormatter`)**: Trasforma l'oggetto in una stringa di testo. La formattazione non deve necessariamente essere reversibile, e quindi può contenere solo gli attributi significativi dell'oggetto.
- **Parser (`IObjectParser`):** Fornisce il servizio inverso al formatter, converte una stringa di testo in oggetto. Anche in questo caso, la stringa può contenere solo alcuni attributi dell'oggetto.
- **Converter (`IObjectConverter`)**: Converte un oggetto di un tipo, in un oggetto di un altro tipo.
- **Encoder (`IObjectEncoder`**): Trasforma l'oggetto in uno insieme contiguo di byte, al fine della serializzazione su un canale. In caso di elementi multimediali, l'encoder può applicare anche una compressione con perdita. La codifica deve essere totalmente reversibile, quindi deve contenere tutti gli attributi dell'oggetto.
- **Decoder (`IObjectDecoder`**): Fornisce il servizio inverso dell'encoder, trasforma l'insieme di byte deserializzato da un canale in un oggetto.
- **Interfaccia utente (`IObjectUserInterface`)**: Restituisce un pannello attraverso il quale l'utente può manipolare l'oggetto. I pannelli possono variare in base al tipo di operazione richiesta (visualizzazione, modifica, inserimento, etc)

La maggior parte di questi servizi ha una implementazione standard valida per qualunque oggetto, solo in rari casi il programmatore deve fornirà una implementazione particolare

Contenitori

Un *container* non è altro che un oggetto che implementa i servizi di *contenitore* (interfaccia `IContainer`). Compito di questo servizio è:

- Restituire informazioni degli oggetti contenuti (id / tipo / nome visualizzato) che corrispondono ad un filtro di ricerca
- Restituire un oggetto contenuto, partendo dal suo id / tipo

- Restituire la tipologia di oggetti che può contenere

Se il *container* è di lettura / scrittura (interfaccia `IWriteableContainer`) deve anche permettere di:

- Inserire un oggetto nel container
- Eliminare un oggetto dal container
- Creare una nuova istanza del tipo specificato
- Svuotare il container
- Spostare / copiare un oggetto contenuto in altro container

A seconda di come vengono gestite gli oggetti, delle possibilità di uso, i container possono essere raggruppati in sei categorie:

- **Memorizzazione**
  Contenitori direttamente collegati ad uno *storage* di un particolare tipo. Gli oggetti creati o aggiunti rimangono persistenti, e le operazioni di ricerca sono indicizzate. Sono di lettura / scrittura. (es. il container degli utenti del sistema)
- **Statici**
  Contenitori che contengono un elenco statico di oggetti, stabilito dal progettista, o da un'impostazione di sistema. Sono di sola lettura per l'utente, e possono essere di lettura / scrittura per il programmatore. (es. un oggetto *account di posta elettronica*, ha staticamente come figli posta in ingresso, posta in uscita, posta eliminata)
- **Attributi**
  Oggetti che espongono alcuni dei propri attributi sottoforma di oggetti figli. Possono essere di sola lettura, o lettura / scrittura. (es. l'attributo *sito web* di un oggetto di tipo *contatto*, potrebbe essere esposto anche come figlio del *contatto*)
- **Elaborazione**
  Contenitori nei quali vengono accodati oggetti che dovranno subire una trasformazione o elaborazione. Gli oggetti in questi contenitori risiedono generalmente in memoria, e vengono eliminati in automatico ad elaborazione completata. (es. il container della *posta in uscita*, trasmette tutte gli oggetti che vengono aggiunti, per poi spostarli nel container *posta inviata*)
- **Filtro/ricerca**
  Contenitori che mostrano solo gli oggetti i cui attributi corrispondono ad un particolare criterio di ricerca. Vengono usati per filtrarti oggetti forniti generalmente da altri container. Sono di sola lettura. (es. tutte i *messaggi di posta elettronica* non letti)
- **Raggruppamento**
  Contenitori simili a quelli filtro, raggruppano gli oggetti che hanno un attributo con lo stesso valore. Generalmente di lettura e scrittura. (es. tutti i *contatti* di una particolare categoria)

Questa suddivisione non è vincolante, ma rappresenta soltanto alcuni degli usi più frequenti.

Attributi e relazioni

Nei paragrafi precedenti si è affermato che gli attributi di un oggetto non possono essere estesi dopo la sua definizione. Questa affermazione è valida solo per gli attributi di tipo primitivo. Quando si dichiara un attributo di tipo oggetto, implicitamente si crea una relazione tra l'oggetto a cui l'attributo fa riferimento, e l'oggetto in cui l'attributo è stato dichiarato. L'attributo in questo caso è solo un mezzo per accedere alla relazione, e di fatto, non definisce un nuova proprietà dell'oggetto, ma solo un nuovo legame. Poiché una relazione coinvolge due elementi, sarebbe concettualmente scorretto che solo uno dei due ne tenesse traccia sottoforma di attributo, per questa ragione, quando si definisce un attributo di tipo oggetto si definisce implicitamente un attributo equivalente nell'oggetto a cui l'attributo fa riferimento.

Si può affermare che un brano musicale è contenuto in un album. Dal punto di vista del brano, l'album potrebbe essere un attributo che fa riferimento ad un oggetto di tipo "album musicale". Ruotando le parti, si può affermare che un album musicale contiene un elenco di brani. In quest'ottica potrebbe essere l'album ad avere un attributo di tipo"lista di brani musicale".

Nel modo classico di pensare, si fa spesso confusione tra cosa sia "attributo" in senso stretto (ovvero quelli di tipo primitivo), e cosa invece sia relazione. Gli unici elementi che possono davvero essere definiti come attributi sono quelli che descrivono proprietà fisiche dell'oggetto, sottoforma di numeri / unità di misura e nomi / definizioni.

La data di scadenza di una "attività", è normalmente definita come attributo primitivo. La data in realtà è la chiave attraverso la quale si associa l'oggetto di tipo "attività" con l'oggetto di tipo "giorno dell'anno".
L'indirizzo di una"persona" è normalmente definito con uno o più attributi di tipo primitivo (testo). Un indirizzo in realtà è solo una chiave tramite la quale si identifica un'unità abitativa. Il ruolo dell'indirizzo è quello di associare una persona ad una particolare unità abitativa!

Formalizzare ogni tipo di relazione tra oggetti, significherebbe definire di centinaia di tipi. Per questa ragione, è solito dichiarare solo i tipi che sono direttamente coinvolti nella descrizione del flusso.

Se lo scopo della nostra applicazione fosse gestire messaggi di posta elettronica, la casella di posta dovrebbe essere definita come oggetto. Ma in caso si volesse solo mantenere una rubrica di contatti, la casella di posta potrebbe essere dichiarata come attributo primitivo (una stringa di testa con l'indirizzo della casella)

Un attributo può diventare di tipo primitivo quando l'entità a cui fa riferimento non ha alcun tipo associato ed è riconducibile ad un solo attributo di tipo primitivo.

Un oggetto di tipo "azienda" è riconducibile ad un solo attributo di tipo testo (la sua ragione sociale, o partita IVA)
Un oggetto di tipo "città" è riconducibile ad un solo attributo di tipo testo (il nome della città)
L'attributo che lega un oggetto di tipo "persona" alla sua città di nascita può essere trasformato in attributo primitivo (testo) se e solo se non è stato dichiarato il tipo "città"

Conclusioni

Tramite la sola relazione di *contenenza* sarà possibile modellare l'intero sistema , dando all'utente una visione omogenea ed uniforme, attraverso un unico concetto semplice ed intuitivo.

Alcuni oggetti saranno memorizzati nel disco locale, altri saranno mantenuti in memoria e potranno essere ad esempio il risultato di una ricerca internet. Per l'utente saranno trasparenti tutti i meccanismi di comunicazione e memorizzazione, e manipolerà soltanto gli oggetti, spostandoli da un contenitore all'altro, apportando loro modifiche o inserendone di nuovi.

Gli oggetti necessari a descrivere il flusso aziendale saranno trattati allo stesso modo degli oggetti multimediali e degli oggetti che rappresentano componenti hardware. Molte interazioni che, a causa delle mancanza di informazioni e standard, il progettista doveva descrivere manualmente, saranno automatizzate dal sistema.

Quando si tenterà di inserire un oggetto in un container, se il container non sarà grado di manipolare quel tipo, il sistema lo convertirà in automatico in un tipo compatibile. Cosi ad esempio sarà possibile trascinare un oggetto di tipo "immagine", all'interno dell'oggetto "casella di posta" di un contatto, e questo sarà convertito in un oggetto "messaggio di posta" con l'immagine in allegato, e spedito tramite l'account predefinito dell'utente.

Partendo da un oggetto, sarà possibile navigare tra le relazione e raggiungere rapidamente tutti gli elementi a lui correlati. Ad esempio, entrando in una fattura, si potrà vedere l'oggetto contatto che rappresenta il cliente destinatario. Il cliente a sua volta conterrà un oggetto indirizzo (sede legale), che conterrà un oggetto città.. L'oggetto città potrebbe contenere un container di ricerca con ad esempio tutti gli esercizi commerciali facenti parte di quella città, che saranno a sua volta oggetti di tipo contatto.

Se si collegherà un disco formattato con un file system classico, ogni cartella sarà un container, e i file di formato noto (per i quali esiste un modulo in grado di convertirli in oggetti) saranno visti direttamente come oggetti. Quando si copierà un oggetto su container di tipo "cartella file system", questi saranno automaticamente convertiti in file e codificati nel formato opportuno.

Quando verrà collegato un telefono cellulare alla porta USB, i contatti presenti nella SIM, saranno trattato come un qualsiasi oggetto contatto e visibili nell'albero su un apposito container. A quel punto sarà sufficiente trascinare un contatto dal telefono per copiarlo nella banca dati locale, e viceversa.

L' utente non dovrà installare decine di programmi, o conoscere protocolli remoti come http, ftp, scp, nfs, etc: gli oggetti remoti saranno trattati allo stesso modo degli oggetti locali, e i protocolli necessari per la loro trasmissione mascherati da appositi container. Se un sito internet esporrà la propria mappa, sarà possibile ad esempio vedere le pagine del sito come figli dell'oggetto "sito internet", e addirittura entrare dentro ad un oggetto "pagina HTML"ed esplorare il dom.

Servizi

Nella costruzione di ogni progetto software, si possono individuare queste quattro fasi (non necessariamente in questo ordine):

- Vengono individuate e definite tutte le entità (*oggetti*) necessarie per descrivere il flusso e l'operazione che si vuole realizzare.
- Vengono cercate tutte le dipendenze da librerie (*servizi*), che non realizzano direttamente le specifiche, ma sono necessarie come strumenti di supporto. Questi strumenti possono essere implementati direttamente dentro all'applicazione, o si possono usare implementazioni esterne, fornite o dal sistema operativo o da terzi. (es. in una applicazione gestionale, la parte di I/O su database è un servizio tramite il quale l'applicazione rende persistenti i dati)
- Vengono progettate le interfacce grafiche (*pannelli*), tramite le quali gli utenti possono interagire con l'applicazione.
- Vengono coordinati tutti gli elementi precedentemente definiti in una serie di attività (*task*) che vengono invocate su richiesta dell'utente o del sistema (es. premendo il tasto salva, si avvia l'attività che memorizza i dati su disco)

Nell'edilizia si fa uso di materiali da costruzione, che, aggregati in modo particolare, danno vita ad altri elementi (ad esempio, con cemento, ferro e rena, si può modellare una trave in cemento armato, da usare nella costrizione del tetto). Si deve costruire un edificio: in fase di progettazione, si decide che dovranno essere innalzate delle pareti. Il progettista può affidarsi a pareti prefabbricate, oppure costruire le proprie usando i mattoni. E' inoltre previsto che sarà necessario sollevare carichi, e avere sempre a disposizione cemento fresco. Ad occuparsi di questi compiti, potranno essere gli operai, oppure dei macchinari specializzati, come gru o betoniere. Sarà comunque la ditta appaltatrice, e non il progettista ad effettuare tale scelta. Al via dei lavori gli operai, seguendo uno scrupoloso piano di lavoro, utilizzeranno materiale e macchinari per realizzare la costruzione.

- ***Gli oggetti** sono i materiali da costruzione e i macchinari (mattoni, cemento, gru, etc)*
- ***I servizi** sono "sollevamento carichi" "fornitura cemento" che sono implementati rispettivamente dall'oggetto "operaio" e dagli oggetti "betoniera" e "gru"*
- ***Il task** è il piano di lavoro*
- ***La cpu** è l'operaio che esegue il lavoro*

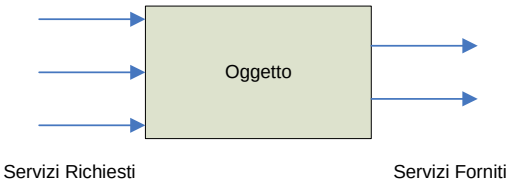
Nell'esempio precedente si nota come la scrittura del piano di lavoro sia spesso indipendente da chi lo debba attuare in concreto o attraverso quali macchinari. Lo stesso dovrebbe valere per un progetto software, che, una volta descritto, dovrebbe essere portabile in qualunque piattaforma. Questo in linea teorica è sempre possibile a patto che vengano tenuti separati i *servizi*, dalle rispettive implementazioni. Attualmente, i pochi progetti che sono sviluppati con questa filosofia, possono mantenere tale separazione solo a livello di codice sorgente: dopo la compilazione, l'applicazione si trasforma in un blocco monolitico di cui il sistema operativo non consoce più nulla[2]. L'unico modo per poter apportare delle modifiche o riutilizzare parte del progetto è accedere al sorgente. I gruppi a sostegno dell'open source hanno la profonda convinzione che il modo migliore per avere il controllo su un prodotto software sia quello di distribuirne il sorgente. Quando si permette ad un insieme di sviluppatori non coordinati, con background e metodi di lavoro differenti, di mettere mano allo stesso progetto, si rischia di disperdere la forza lavoro e incentivare l'individualismo di chi cerca solo popolarità. Qualora il sorgente fosse utilizzato solo a scopo consultivo / didattico per capirne e studiarne il funzionamento, non ci sarebbe nulla di negativo. Ma realizzare decine di copie dello stesso sistema che differiscono solo in particolari insignificanti, non è utile all'utente, che si trova confuso nello scegliere "la versione giusta", ne aiuta gli sviluppatori che devono decidere a quale corrente aderire (con una seria probabilità che aprano a sua volta un loro branch perché nessuna alternativa li soddisfa) Ogni tipo di modifica o di estensione ad un progetto andrebbe proposta e discussa da una autorità **centrale e competente** con il compito di rilasciare aggiornamenti periodici, o studiare una architettura tale che ogni trasformazione avvenga solo attraverso **moduli indipendenti** ed isolati.

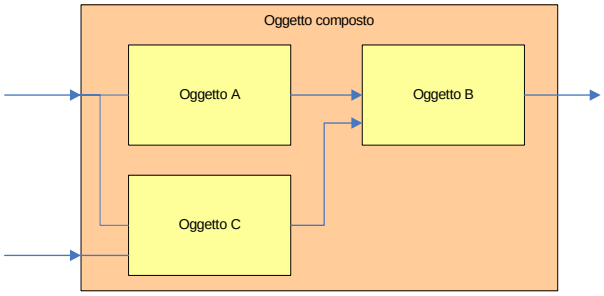
Quando estendiamo il nostro pc (ad esempio sostituendo la scheda video, o aggiungendo un banco di RAM), non richiediamo ai produttori hardware lo schema elettrico dei componenti, e non applichiamo l'estensione andando a smembrare la nostra scheda madre con fil di stagno e saldatore. Adottando uno comune interfaccia (bus PCI, AGP) è possibile fare comunicare due mondi sconosciuti, e permettere estensioni attraverso blocchi indipendenti ed interconnessi. Rimane ancora sconosciuto il perché nel mondo del software non sia possibile applicare gli stessi concetti, e sempre più persone si battano per qualcosa che, oltre ad essere inutile, è anche dannoso.

---

[2] Alcuni moduli compilati (librerie statiche / dinamiche) portano con se alcune informazioni sulla struttura del codice

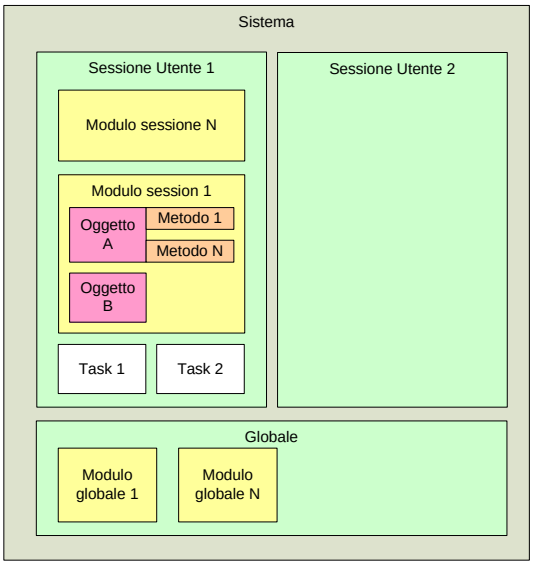
L’architettura a servizi ambisce a scindere definitivamente il “cosa si fa” dal “come si fa”, permettendo a chiunque di modificare quest’ultimo. Infatti, se l’applicazione esterna le proprie necessità, non collegandosi staticamente (o dinamicamente) ad una particolare libreria, il sistema e l’utente possono scegliere l’oggetto migliore in grado di soddisfare tali necessità. In questo modo, quando è disponibile un nuovo componente, che realizza in modo migliore un servizio, l’applicazione ne può fare uso, senza essere ricompilata, e senza che l’utente conosca la sua struttura interna.





Binding dei servizi

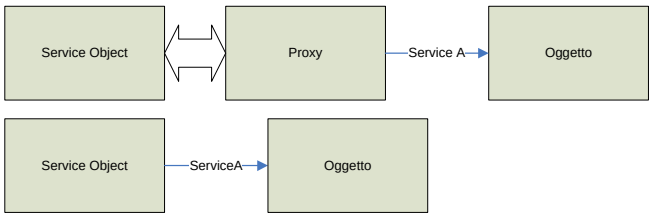
I servizi vengono solitamente implementati nei *service object*, e più oggetti possono implementare lo stesso servizio. Com’è possibile allora scegliere l’oggetto giusto, dato che le richieste sono mirate al servizio, e non ad una particolare implementazione? I *service object* non possono essere creati dall’utente / programmatore, la selezione è effettuata dal sistema in base al contesto in cui la richiesta viene effettuata. I contesti sono organizzati in livelli gerarchici, partendo da quello più specifico di *metodo* e risalendo a quello di *oggetto*, *task*, *modulo*, *sessione* e *macchina*. Ognuna di queste entità può indicare sia da quali servizi è dipendente, sia quali sono gli oggetti “predefiniti” che implementano tali servizi. Queste preferenze, tramite apposte regole, possono essere cambiate dall’utente, a qualunque livello nella gerarchia dei contesti (nei limiti delle sue autorizzazioni). Quando un metodo richiede un interfaccia per un servizio, la richiesta passa all’oggetto che contiene il metodo. Se l’oggetto non implementa l’interfaccia la richiesta passa allora al modulo che contiene l’oggetto. A meno di regole diverse, il modulo cerca tra i propri oggetti, il primo che implementi l’interfaccia. Se non viene trovato alcun oggetto, la richiesta sale ancora, e passa alla sessione. Anche qui, a meno di regole diverse, viene chiesto a ciascun modulo caricato in sessione un oggetto che implementi l’interfaccia. Se anche in questo caso non ci sono risultati, la richiesta passa al sistema, che fa lo stesso con i moduli globali (condivisi tra tutte le sessioni). Questo processo di ricerca viene chiamato *binding dei servizi*, ed è simile a quello che avviene con le librerie a collegamento dinamico, con l’unica differenza che il collegamento è verso un servizio, e non ad una particolare libreria/modulo.



Durante svolgimento di un task, è richiesta la conferma dell’utente prima di eseguire un’istruzione potenzialmente pericolosa. Con i sistemi attuali, ci potremmo affidare ad una particolare funzione di sistema, che ad esempio mostra una finestra di dialogo con un messaggio, ed i pulsanti “si” “no”.
Se volessimo lanciare lo stesso task in una console testuale? Dovremmo riscrivere il programma, sostituendo questa funzione con una più appropriata, che non faccia uso di finestre. In un sistema basato sui servizi, il task avrebbe richiesto al servizio “interfaccia utente” la conferma attraverso un ipotetico metodo “confirm”. In un ambiente grafico, l’implementazione di tale metodo avrebbe mostrato la classica finestra di dialogo con i due pulsante, ma in un ambiente testuale, un’altra implementazione avrebbe mostrato un messaggio di testo, e atteso la pressione dei tasti S (Si) o N (No).
Il task quindi non indica al sistema come la conferma debba essere data, ma delega al servizio” interfaccia utente” tale compito. Ad esempio, in un futuro prossimo dove i sistemi di riconoscimento vocale saranno più affidabili, questa conferma potrebbe avvenire anche a voce, ed il task potrà usufruirne senza dover essere ricompilato!!

Proxy

Spesso è necessario limitare l'interazione con un servizio, permettendo di usare solo un sottoinsieme delle sue funzioni. In questi casi, non si ha accesso diretto al servizio, ma ogni interazione avviene attraverso un proxy. Un *proxy* è un oggetto che si antepone ad un servizio, e ne espone la medesima interfaccia.



Quando vogliamo comunicare con una persona che ricopre una carica importante (es. il presidente di una società), quasi sempre lo si fa attraverso terzi. Ogni richiesta, passa da una segreteria, che ne valuta la rilevanza. Se è giudicata accettabile, la richiesta viene e messe in coda, ed inoltrata all'autorità nel momento opportuno. Una eventuale risposta verrà comunicata alla segreteria, che si occuperà di girarla a sua volta al richiedente. La segreteria ha il ruolo di proxy tra il l'autorità e il richiedente.

In un sistema orientato ai servizi, e non ad una particolare implementazione, è indifferente se l'accesso ad un servizio è tramite proxy. L'uso dei proxy può essere richiesto in molteplici scenari e in base alla funzioni fornite si possono raggruppati nelle seguenti categorie:

- **Filtro**
  Filtra le richieste ad un servizio: quelle che soddisfano determinati requisiti, vengono inoltrate al servizio reale, le altre scartate. Questo genere di proxy viene usato anche per il monitoraggio e la raccolta di dati statistici sull'uso di un servizio. Ad esempio, con i proxy filtro si possono implementare firewall per i servizi di rete, o di input/output su disco.
- **Trasmissione / Ricezione**
  Codifica le richieste ad un servizio su un canale seriale, per essere trasmesse ed eseguite da un oggetto remoto. Nel sistema remoto, un proxy analogo in ascolto decodifica il messaggio trasmesso, ed invoca il servizio locale. La serializzazione avviene usando uno dei protocolli standard per l'invocazione di metodi remoti (es. SOAP). Questi proxy possono essere generati automaticamente dal sistema, e costituiscono il principale meccanismo per la comunicazione in rete,
- **Sincronizzazione**
  Gestisce l'accesso in concorrenza per i servizi che non possono essere usati simultaneamente da più task. Solo le richieste del task che detiene l'accesso esclusivo, sono inoltrare al servizio reale, le altre vengono mantenute in code prioritarie, e i task associati sospesi. Ad esecuzione completata, il proxy risveglia il task in coda a priorità più alta, concedendogli l'accesso esclusivo, ed inoltra la sua precedente richiesta. Ad esempio, l'accesso ad ogni oggetto che rappresenta un componente hardware, dovrà essere effettuato tramite questi proxy.
- **Smistamento**
  Smista le richieste, inoltrandole ai *service object* opportuni. Questo tipo di proxy è usato quando più oggetti implementano lo stesso servizio, ed il programma non ne ha richiesto nessuno in particolare. I criteri di smistamento variano a seconda della tipologia di servizio e per questo generalmente non possono essere autogenerati. Ad esempio, ogni pacchetto IP può passare ad un proxy di smistamento, che in base all'indirizzo di destinazione e al traffico di rete, lo inoltri all'interfaccia di rete opportuna.
- **Broadcast**
  Inoltra le richieste ad un gruppo di *service object* simultaneamente. Eventuali valori di ritorno vengono persi, a meno che tutti i *service object* non abbiano restituito lo stesso valore. Ad esempio, il servizio di interfaccia grafica, chiede ad un qualsiasi dispositivo di puntamento di ricevere notifica quando cambia la posizione corrente, in modo da aggiornare il cursore. Questa richiesta viene fatta al proxy di broadcast, che la invierà a tutti gli oggetti caricati che forniscono un servizio di puntamento (come mouse e touchpad).

Servizi base

Il binding di un servizio, può diventare un'operazione molto lenta e costosa, soprattutto quando si deve risalire tutta la gerarchia dei contesti, partendo da un metodo. Per questo, in ogni contesto è mantenuto un catalogo dei servizi (`IServiceCatalog`), che tiene conto non solo del contesto corrente, ma anche di quelli superiori, per permettere una rapida associazione oggetto-servizio. Il catalogo dei servizi di sistema è il primo oggetto creato, e l'unico oggetto realmente indispensabile. Ogni modulo, in fase di installazione, pubblica in una apposita area, tutti i servizi esposti da ogni suo oggetto. Alla prima richiesta del servizio, se l'oggetto associato non è stato ancora creato, viene caricato in memoria il modulo a lui relativo, creata un'istanza dell'oggetto, e registrata nel catalogo di sistema. Per avere un sistema minimo, oltre al catalogo di sistema, devono essere implementati anche i seguenti servizi:

- **Registro:** Organizza le impostazioni di sistema e dei moduli in una struttura gerarchica formata da contenitori di coppie chiave / valore. Il valore può essere un qualsiasi tipo primitivo (testo, numero intero, numero reale, booleano, data, buffer)
- **Moduli:** Installa, rimuove e carica i moduli di sistema. Un modulo è un'immagine binaria che può contenere oggetti, servizi e task
- **Catalogo Storage**: Mantiene il catalogo che associa ad ogni *data object*, uno *storage* predefinito. Questo servizio normalmente dipende dai servizi disco, o dai servizi database.
- **Directory:** Organizza i *data object* in un sistema gerarchico partendo da un *container* radice, permettendone l'identificazione tramite *path*;

Conclusioni

Un sistema cosi descritto comporta un grande lavoro di progettazione per individuare e formalizzare tutti i servizi più comuni utilizzati dalle applicazioni oggi giorni, cercando di rendere i sistemi standard ed interoperabili tra loro. La creazioni di uno standard, non limita la concorrenza o la libertà delle software house, in quanto lo standard stabilisce solo il guscio esterno, come un oggetto appare al programmatore / utente. Viene lasciato libero arbitrio per quello che riguarda la realizzazione interna, in modo che ognuno possa raggiungere risultati diversi, permettendo al consumatore di scegliere il prodotto qualitativamente migliore.

La condivisione del lavoro, permette a chi sviluppa software di concentrarsi sul vero obiettivo, senza preoccuparsi più di infrastrutture, o parti accessorie, come un'interfaccia utente funzionale che abbia quanto di meglio si possa desiderare. Ne guadagna l'utente, che si trova un sistema omogeneo, e ne guadagna il programmatore che può scegliere di acquistare servizi esistenti, o implementare i propri, senza alterare il funzionamento interno dell'applicazione.

Oltre al vantaggio di poter scegliere come ottenere una data funzionalità, l'uso dei servizi permette di poter monitorare come un modulo usa componenti del sistema, e tramite proxy di limitarne l'accesso, senza perdita di prestazioni. Gli attuali firewall e antivirus, si frappongono a priori tra "il servizio" e l'applicazione, quando il controllo sarebbe utile solo a determinate categorie di processi ai quali non abbiamo concesso la nostra fiducia.

L'uso dei servizi forniti da sistemi remoti è totalmente trasparente e automatico, non sarà necessario progettare un protocollo specifico per ognuno di essi. Ogni servizio potrà essere remotizzato, senza alcun calo di prestazioni quando viene eseguito in locale: l'accesso ad un servizio locale è diretto, mentre quello ad un servizio remoto tramite proxy,

I servizi condivisi ottimizzano le risorse di sistema, in quanto più moduli possono utilizzare la medesima implementazione. Inoltre solo i servizi a bassissimo livello (che rappresentano la minoranza) rimangono vincolati alla piattaforma e l'hardware, tutto il resto può essere portato in più piattaforme, senza nemmeno bisogno di essere ricompilato, se il passaggio è tra piattaforme basata sulla stessa architettura.

Gestione dei processi

Limiti dei processi tradizionali

Quando l'utente avvia un'applicazione, il sistema operativo genera un processo ed inizia ad eseguire le istruzioni in esso contenute. L'utente, nel compiere questa azione, non è quasi mai interessato all'applicazione in quanto tale, ma piuttosto ai servizi che essa fornisce: la generazione di un processo è soltanto il mezzo per ottenerli. Normalmente un processo spende la maggior parte del suo tempo sospeso in attesa di qualche evento, eventuali esecuzione di codice avvengono solo ed esclusivamente a fronte di un cambiamento di stato, provocato o dall'utente (tramite dispositivi di input come mouse o tastiera), o da componenti interni del sistema (timer, etc). Perché avere allora tante "unità di esecuzione" attive, quando l'attività principale è attendere eventi? Il ruolo di un processo non è soltanto portare avanti una attività (eseguire codice), ma è sopratutto creare una infrastruttura che permetta all'utente di disporre di determinati servizi. I sistemi multiprocesso basati su scheduler preemptive causano una frammentazione delle attività, con anche dei rallentamenti visibili nel caso più processi tentino di accedere contemporaneamente a risorse hardware meccaniche, come le unità disco. A causa dei lunghi tempi di accesso, i continui spostamenti tra settori non contigui provoca una drastica diminuzione del throughput, rispetto ad accessi sequenziali continuative. Se si avessero informazioni sul fine di tali accessi si potrebbero applicare delle ottimizzazioni, non concedendo ad esempio a nessun altro processo la risorsa disco, fin quando una macro (ma breve) operazione di IO è in corso. Inoltre le molte interruzioni necessarie allo scheduler per assegnare la cpu ad un altro processo, si rivelano spesso inutile poiché nulla è cambiato e tutti i processi devono rimanere ancora sospesi, in attesa di qualche evento. Sebbene il multiprocesso abbia migliorato l'uso della risorsa cpu, e consentito il parallelismo, l'entità processo porta in se molti limiti, tra i quali:

- Un processo è un blocco monolito di bytecode, il sistema si limita ad eseguire meccanicamente le istruzioni in esso contenute, senza avere alcuna consapevolezza del loro fine. Non essendoci alcuna struttura che descriva le funzioni o i servizi offerti[3], non è possibile estrarre delle parti da utilizzarle in altri contesti.
- Un processo è un sistema isolato, la cooperazione con altri processi è possibile solo attraverso scambio asincrono di messaggi, su lenti canali seriali.
- Un processo, anche per eseguire un'attività minima, è costretto a caricare decine di librerie, occupando enormi quantità di ram[4]. Il caricamento può essere un'operazione lenta, soprattutto nel casi di uso massiccio di librerie a collegamento dinamico.
- Ogni ottimizzazione avviene a livello di processo: eventuali dati raccolti (cache) al fine di migliorare le prestazioni vengono persi dopo la sua terminazione

In un sistema orientato ai servizi, si tenta di superrare questi limiti, sostituendo il concetto di processo con quello di *modulo* e *task*.

Le applicazioni come msn messenger, skype, icq, sebbene con interfacce o protocolli diversi, forniscono tutte un servizio di messaggistica istantanea. Se un programmatore volesse modificare alcuni aspetti dell'interfaccia utente, per adattarla alle proprie esigenze, a meno che non disponga dei sorgenti, sarebbe costretto a riscrivere da zero l'intera applicazione[5] (sia la parte dei servizi, che quella di interfaccia). In un sistema orientato ai servizi, da un lato ci sarebbe la parte funzionale dell'applicazione, e dall'altro uno o più pannelli attraverso i quali controllare tali funzioni. Chiunque potrebbe far uso solo della parte funzionale all'interno della propria interfaccia utente, oppure tramite script che eseguono operazioni in batch (es, inviare un messaggio di auguri a tutti i contatti) senza avere sorgenti.

Moduli

Un modulo è un aggregato strutturato di tipi e servizi. In esso sono contenuti, per ogni tipo, sia i metadati della parte descrittiva sia il codice della parte funzionale. I metadati consentono di mantenere la definizione dei tipi anche all'interno di un modulo compilato. Per usare un tipo del modulo, l'intero modulo deve essere caricato in memoria. **Mentre ad ogni eseguibile caricato corrisponde un processo, ad un modulo caricato non è associata alcuna unità di esecuzione**. Il modulo si limita a rendere disponibili al sistema dei servizi, la dove venissero richiesti.

Ogni modulo contiene una procedura di inizializzazione, invocata al momento del caricamento. In questa procedura, vengono solitamente creati i *service object*, alcuni dei quali in ascolto su qualche *evento*. Un evento è un struttura esposta da un oggetto, usata per notificare ad altri oggetti (detti ascoltatori) il verificarsi di determinati cambiamenti. La notifica avviene invocando un metodo, specificato dall'ascoltatore durante la registrazione all'evento. Se più oggetti sono in ascolto per lo stesso evento, i loro metodi saranno invocati in modo sequenziale uno ad uno.

---

[3] Alcune tipologie di moduli, hanno una interfaccia funzionale (dll native), altri contengono database di tipi (dll com / .net, classi java)

[4] Solo la parte codice e dati statici di una libreria può essere condivisa tra più processi.

[5] Alcune applicazioni sono sviluppati con sistemi a componenti, che possono essere riutilizzati e inclusi in altre applicazioni

Un movimento del mouse provoca una trasmissione di dati sulla porta alla quale è connesso. Quando ci sono nuovi dati disponibili su una porta, viene generato un interrupt per permettere alla cpu di elaborarli. L'oggetto che fa le veci di "driver" del mouse, è in ascolto sull'evento "nuovi dati" dell'oggetto porta (es. porta USB). Una volta letti, e decodificati dati trasmessi, l'oggetto mouse genera a sua volta un evento, per informare altri oggetti del cambiamento avvenuto (click, scroll, spostamento, etc). Il gestore della GUI , in ascolto sull'eventi "posizione cambiata" del oggetto mouse, reagisce a tale notifica aggiornando la posizione del cursore.

Alcuni moduli possono essere registrati come moduli di sistema, e quindi caricati automaticamente durante il boot. Un modulo di sistema contiene gli oggetti e i servizi utilizzati più spesso o necessari per avere un ambiente di esecuzione minimo. Ad ogni modulo, in fase di compilazione, è associato staticamente un indirizzo di memoria base. Per evitare conflitti (più moduli allo stesso indirizzo), ed avere tutti i moduli contigui nello spazio di indirizzamento (lineare o virtuale), è applicata una rilocazione statica durante l'installazione.

FORNIRE ESEMPIO GRAFICO

Task

Il task è un'unità in grado di eseguire una porzione di codice (o procedura), al verificarsi di un evento. Il task ha normalmente vita breve, e termina non appena la procedura ad esso associata è stata eseguita. Tipicamente si utilizzano task per eseguire metodi esposti da un *service object.* Esistono tre tipi di task:

- **Task non periodici:** sono task eseguiti una sola volta, solitamente al seguito di qualche evento generato da una periferica hardware (click del mouse, pressione di un tasto, ricezione pacchetto di rete, etc).
- **Task periodici temporali**: sono task eseguiti con cadenza periodica, come ad esempio il playback di un contenuto multimediale (in un video a 25fps, ogni 40 millisecondi è necessario decodificare e disegnare un frame), o una operazione di manutenzione pianificata.
- **Task continuativi**: sono task eseguiti in modo continuativo, come ad esempio il rendering di una scene 3d di un videogioco.

Quando è necessario attendere il risultato di una operazione asincrona, o attendere che una risorsa occupata sia nuovamente disponibile, invece di impegnare la cpu con inutili cicli di polling, il task può essere sospeso. La sospensione provoca il rilascio immediato della cpu, permettendo cosi ad altri task di prenderne possesso. Non appena le condizioni che hanno portato alla sospensione vengono a meno (risorsa disponibile o operazione asincrona completata) il task viene risvegliato e l'esecuzione ripresa esattamente nel punto dove era stata interrotta.

Un utente clicca su un pulsante, generando cosi un task per la gestione dell'evento. Il metodo registrato all'evento "click" fa richiesta ad uno storage di istanziare un oggetto con un determinato id, provocando una lettura su un'unità disco. Nell'attesa che i dati vengano letti dal disco e depositati su un buffer, il task può sospendersi e liberare la cpu.

Un task non ha risorse associate: eventuali oggetti usati o creati durante l'esecuzione, rimangono in memoria anche dopo la sua terminazione. Un task deve essere pensato come un thread, in un sistema monoprocesso: tutti i task eseguiti all'interno della stessa sessione utente, condividono lo stesso spazio di indirizzi, ognuno con un area stack privata. L'unica informazioni che un task deve mantenere, è una copia dello stato della cpu al momento della sospensione.

Un task non viene eseguito immediatamente dopo la sua creazione: il nuovo task si posiziona in una coda prioritaria, e resta in attesa che una cpu sia disponibile. Task originati da interrupt hanno solitamente priorità più alta, rispetto a task utente / periodici. Per la sua natura, la creazione di un nuovo task ha circa lo stesso costo di una chiamata a funzione, di conseguenza non deve risultare eccessivo che ad ogni evento corrisponda un nuovo task.

L'esecuzione parallela di più task, è possibile solo se sono presenti più cpu (sistemi multi-core o multi-processore), oppure se la stessa cpu viene utilizzata a momenti alterni. La cpu può essere revocata ad un task, solo quando questo termina, o si sospende. L'oggetto che implementa i servizi di scheduling (`ITaskScheduler`), ha il compito di decidere con quale tecnica gestire l'esecuzione dei task, in particolare se adottare un sistema preemptive oppure no. In uno scheduler preemptive, scaduto un quanto di tempo stabilito (fisso o variabile) il task attualmente in esecuzione subisce una sospensione forzata, favorendo l'avanzamento di altri task in coda. In ogni caso, un task ad alta priorità e breve durata può richiedere allo scheduler di non essere sospeso forzatamente, e di attendere la propria terminazione prima di eseguire altri task.

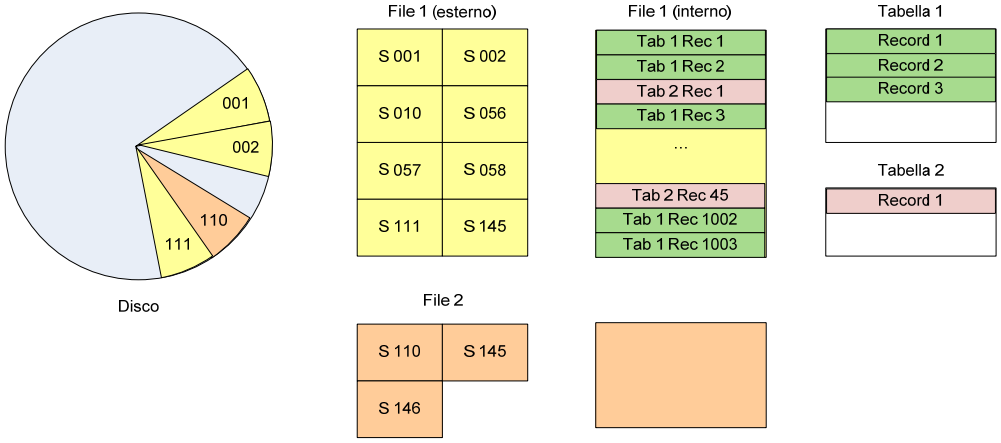
Task vs messaggi

L'organizzazione task permette di avere a disposizione unità di esecuzione leggere, che possono essere eseguite in parallelo, e che terminano quando non sono più necessarie, anziché sospendersi all'interno di cicli infiniti. La gestione di eventi attraverso task, si contrappone ai sistemi attuali basati su messaggi, dove un processo all'interno di un loop infinito processa di volta i volta i messaggi che vengono lasciati in coda. A meno di qualche sistema di filtraggio, il

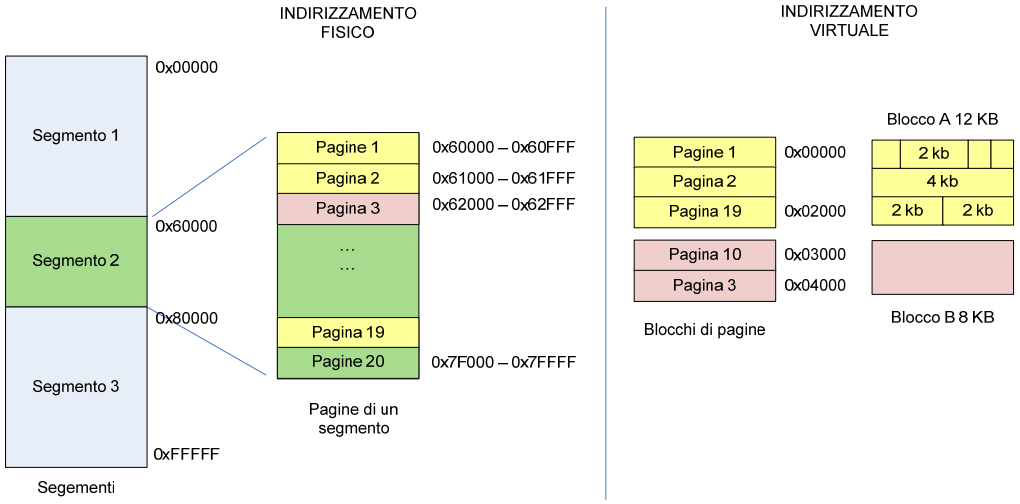
processo è costretto a leggere e decodificare ogni messaggio che gli viene inviato, per cui anche quelli non richiesti o non utili in un determinato contesto. Se l'elaborazione di un messaggio, non termina in tempi brevi, l'applicazione va in stallo, poiché non è possibile leggere il messaggio successivo, fin quando non si è processato quello in corso. Questo comportamento è particolarmente fastidio, quando tramite messaggi vengono comunicati eventi che riguardano l'interfaccia utente. Se da un lato la sequenzializzazione degli eventi comporta meno problemi, in quanto non si deve gestire programmazione concorrente, e accesso simultaneo a risorse condivise, dall'altro limita il parallelismo e costringe l'applicazione ad un carico di lavoro spesso non necessario. Tramite task l'esecuzione di codice avviene solo ed esclusivamente quando è realmente necessaria, invocando direttamente le procedure fini a gestire un evento, ed evitando di appesantire lo scheduler, e limitando i context switch.

Gestione della memoria

Iniziamo con illustrare come vengono usate attualmente due principali gruppi di memoria: le memorie RAM, e le memorie di memorizzazione di massa. (alle quali appartengono le unità disco),



Un'unità disco rigido è una superficie di memoria. L'unità minima di allocazione è il settore, che misura 512 byte. L'intero disco viene suddiviso in partizioni, in modo da formare un gruppo di unità logiche, ognuna con il proprio file system. Solitamente per diminuire la frammentazione, un file system raggruppa i settori in cluster, portando l'unità minima di allocazione tipicamente a 4 kb (8 settori). Un file è un gruppo di cluster, che logicamente rappresenta una superficie di memoria contigua (i cluster che lo compongono possono essere sparsi), che l'utente può espandere e allocare a sua volta. Un file che rappresenta un database, è diviso in tabelle. Ogni tabella ha bisogno di allocare delle porzioni di spazio all'interno del file per memorizzare i record. Anche la tabella è una superficie di memoria logicamente contigua che contiene record.



Una ram logicamente è un superficie contigua di memoria. Se l'architettura lo permette, è possibile dividere tale memoria in segmenti di dimensioni variabili. La suddivisione in segmenti permette di trattare aree di memoria diverse in modo diverso, in particolare assegnando a ciascuna i propri permessi di lettura, scrittura ed esecuzioni. In molte architetture, un segmento può essere ulteriormente suddiviso in piccole sezione di dimensioni fisse (solitamente 4 kb), chiamate pagine. La pagina è l'unità minima di allocazione all'interno del segmento ed possibile è riservare un gruppo di pagine su richiesta, creando un area contiguo in uno spazio di indirizzamento virtuale. Tale area può essere ulteriormente suddiviso dal gestore della memoria dell'applicazione, per allocare anche piccole sezioni di dimensioni inferiori alle dimensioni di una pagina.

Dagli esempi precedenti si nota come ogni tipologia di memoria presenti caratteristiche in comune, indipendentemente che si tratti di memoria ram, o unità disco. In particolare si nota l'esistenza di una *superficie di memoria* dalla quale, attraverso alcuni algoritmi di allocazione, ricavare superfici di memoria secondarie sulle quale memorizzare dati.

Superfici di memoria

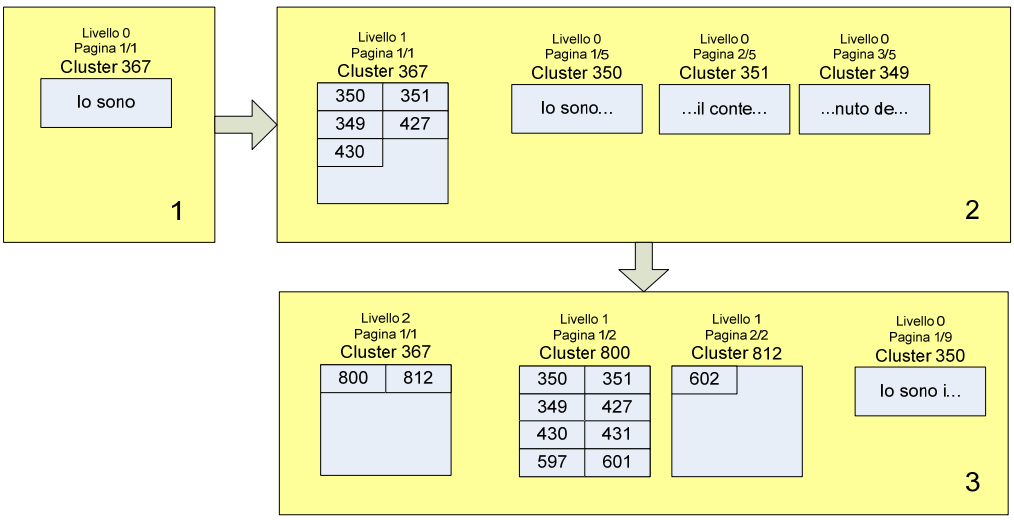
Una superficie di memoria (`IMemorySurface`) è un'area logicamente contigua di dimensioni conosciute, nella quale poter leggere / scrivere dati. A seconda della tipologia di memoria, i dati possono essere persistenti o volatili, di sola lettura o di lettura / scrittura. Le operazioni di lettura, scrittura, eliminazione e accesso non possono in genere essere fatte in posizioni / dimensioni arbitrarie: ogni superficie di memoria è divisa in blocchi (chiamati anche settori, o pagine), la cui dimensione solitamente è una potenza di due. Un blocco è l'unità minima di informazioni all'interno della memoria, di conseguenza non è possibile scrivere / leggere una quantità di dati che non sia multipla della dimensione del blocco. Attraverso un allocatore (`IMemoryAllocator`) è possibile raggruppare un insieme di blocchi di una superficie di memoria, per formare una superficie di memoria secondaria. Compito dell'allocatore (`IMemoryAllocator`) è:

- Allocare un'area di dimensioni date, logicamente contigua all'interno della superficie di memoria
- Riallocare un'area allocato precedentemente, modificando la sua dimensione
- Liberare un'area precedentemente allocata
- Mantenere informazioni sullo spazio disponibile nella memoria
- Mantenere una lista delle aree di memoria allocate
- Restituire un riferimento alla superficie di memoria, partendo dal suo identificativo

In caso di memoria RAM, l'identificativo può corrispondere proprio all'indirizzo fisico di memoria dove inizia il primo blocco, mentre nel caso di memoria di massa, può essere l'indice del primo cluster. E' possibile in linea teorica applicare un algoritmo di allocazione solitamente utilizzato in una memoria di massa su una memoria ram (es. suddividere la memoria RAM in file) Per semplicità di linguaggio si fa riferimento a superfici di memoria generiche (non necessariamente su disco) con il nome di file o blocco.

Allocazione su disco

Si assuma di avere una partizione da 160 GB con cluster da 4 kb (per un totale di 49.315.840 cluster) e che si voglia suddividere questa partizione in file di dimensioni variabili. Se i file sono di dimensioni inferiori o uguali a 4 kb, non sono necessarie strutture intermedie poiché file e cluster diventano sinonimi. Se i file sono maggiori di 4 kb, un cluster non è più sufficiente. Nel caso ottimistico è possibile espandere il file utilizzando i cluster immediatamente adiacenti al primo, ma al diminuire dello spazio libero, diminuiscono anche le probabilità di trovare zone del disco con un sufficiente numero di cluster liberi contigui. Se si utilizza una allocazione sparsa è necessaria una struttura che tenga traccia degli indici dei cluster di cui sono composti i file. Per minimizzare il numero delle letture durante gli accessi non sequenziali, è utile condensare tali indice in una tabelle, che per gli stessi ragionamenti di poco fa, non può superare il limite delle dimensioni del cluster. Se per indice viene utilizzato un intero a 32 bit (con il quale è possibile indicizzare 4.294.967.296 cluster diversi) la massima capienza della tabella è di 1024 indici (4.096 / 4), per file di massimo 4 mb (1024 x 4 kb). Se le dimensioni dovessero essere maggiori, è possibile aggiungere un secondo livello di indirezione: anziché indicizzare i cluster che contengono i dati, vengono indicizzati i cluster che contengono le tabelle degli indici. Con il secondo livello è possibile indicizzare 1024 tabelle, che a sua volta indicizzano 1024 cluster da 4 kb ciascuno, per un massimo di 4 gb a file (1024 x 1024 x 4 kb). Procedendo con questa tecnica, è possibile avere file di dimensioni arbitrarie, permettendo cosi al sistema di essere all'avanguardia rispetto alla capacità di memorizzazioni future, in cui ci potranno essere file di decine di gb.



Nel blocco 1 è presente un file nel cluster 367 di dimensioni inferiori a 4 kb che contiene il testo "io sono". Quando il testo aumenta è necessario allocare ulteriori cluster. Il cluster 367 diventa la tabella degli indici, e il vecchio contenuto viene copiato nel primo cluster libero (il 350). Come è possibile vedere nel blocco 2, il file ora è composto da 5 cluster, ognuno contenente una parte del testo"io sono il contenuto del file". Le dimensioni aumentano ulteriormente, e una tabella di indici non è più sufficiente. Il cluster 367 diventa la tabella degli indici di secondo livello, il vecchio contenuto viene copiato in un cluster libero (l'800). L'indice di secondo livello contiene indici di tabelle di indici che compongono il file.

Non tutto il cluster è utilizzato per memorizzare dati, la prima parte è riservata per l'intestazione. L'intestazione minima indica solo la tipologia di informazioni che il cluster contiene, attraverso un numero a 4 bit. Questo numero assume il valore 0 se il cluster contiene dati, maggior di 0 se il cluster contiene indici all'ennesimo livello di in direzione (1 primo livello, 2 secondo, etc). Rinunciando al 2.5 % della partizione è possibile avere un'intestazione più robusta che permetta la ricostruzione della sequenza dei cluster che compongono il file, qualora la tabella degli indici venisse danneggiata.

Il primo file all'interno di una partizione è la mappa di bit con lo stato di allocazione di ogni cluster. L'ennesimo bit della mappa con valore 0 indica che l'ennesimo cluster della partizione è libero (non allocato). Questo file ha dimensioni fisse, proporzionali alle dimensioni della partizione. (con cluster di 4 kb la dimensione della mappa è lo 0,00003% della dimensione della partizione) Questo è l'unico file a non avere una tabella di indici associata, poiché le sue dimensioni sono fisse e lo spazio necessario a contenere tutta la mappa è riservato in fase di formattazione, in testa alla partizione.

Il secondo file è la tabella che con l'elenco di tutti i file attualmente allocati, ogni riga comprende l'indice del primo cluster, e le dimensioni in byte del file. Questa tabella è superflua, ed è a scopo puramente informativo e diagnostico, poiché come già spiegato l'utente non avrà mai accesso diretto ai file.

Registro di configurazione

Il registro è un database gerarchico nel quali il sistema e i moduli possono scrivere impostazioni e configurazioni. E' suddiviso in contenitori (chiavi) identificati da un nome, che possono contenere a sua volta sotto-chiavi e attributi. In maniera analoga ad un oggetto, un attributo è una coppia nome – valore. Il valore può essere solo di tipo primitivo (non oggetti!) e deve avere lunghezza fissa. I figli di una chiave (attributi o sotto-chiavi), hanno una struttura irregolare, che difficilmente si ripete, per questo non necessitano di una definizione formale, e possono essere aggiunti o rimossi a piacimento. Gli elementi del registro sono dati che vengono principalmente letti, o al più modificati, ma molto raramente aggiunti o eliminati. Per questa ragione è conveniente adottare una struttura che renda ottima la ricerca / lettura / modifica a discapito magari dell'inserimento / eliminazione. E' stato scelto un registro centralizzato, rispetto all'uso di tanti piccoli file testuali sparsi per queste ragioni:

- La presenza di tanti piccoli file aumenta in modo sensibile la frammentazione interna del disco, molto spazio va perso in quanto raramente viene riempito un intero cluster.
- Nei file di testo non è possibile un accesso casuale o indicizzato, per recuperare il valore di una impostazione il file deve essere letto e interpretato in modo sequenziale fin quando non si trova l'elemento desiderato.
- Ogni operazione di inserimento, provoca uno scorrimento verso il basso di tutto il contenuto del file, provocando svariati accessi in scrittura in caso di file medio-lunghi.
- Ogni impostazione che non è di tipo testo deve essere ogni codificata / decodificata in stringa. Tali codifiche possono essere anche relativamente costose in caso di buffer binari.
- Lo spazio occupato da un dato in un file di testo è maggiore rispetto ad uno binario. Ad esempio un numero a n° bit in binario ha dimensione fisse indipendentemente dal valore che assume, mentre in formato testo la dimensione varia in base al numero di cifre decimali di cui è composto.

L'uso di file di testo, tra cui file xml, è utile solo quando il contenuto deve essere manipolato direttamente dall'utente o dal programmatore: l'accesso al registro è mascherato da un servizio, nessuno entrerà mai in contatto diretto con la struttura interna.

Dal punto di vista fisico, il registro è composto da tre file. Un file contiene l'elenco delle stringhe che rappresentano il nome delle chiavi / attributi, un altro contiene l'indice dei valori, e un altro infine i dati.

File

Nei sistemi tradizionali l'utente crea i file, associando a queste entità un particolare contenuto informativo (documento di testo, foglio di calcolo, fotografia, etc). Per l'utente informazione è sinonimo di file, e viceversa. Di nuovo c'è confusione tra il cosa si vuole ottenere e come ottenerlo: il file è **un modo** con il quale è possibile memorizzare e ricercare informazioni, non l'informazione in se.

Una bicicletta non rappresenta il concetto di spostamento. E' solo un mezzo attraverso il quale ci spostiamo.

Nel sistema le informazioni sono rappresentate da oggetti, e **gli oggetti non sono file**. Il file potrebbe essere uno dei modi con i quali si rende persistente un oggetto, ma non il solo. L'utente manipola gli oggetti attraverso *container*, che sono contenitori logici. Un container può facoltativamente utilizzare *storage* per i servizi di persistenza degli oggetti. Gli *storage* sono *service object* che operano nella più totale autonomia, e trasparenza rispetto all'utente. Alcuni *storage*, sono collegati a superfici di memoria, e utilizzano i file per memorizzare gli oggetti. Altri possono essere collegati in rete ad un dbms esterno, o un web service.

Spesso si crede che un meccanismo automatico di memorizzazione introduca un limite alla libertà dell'utente, che apparentemente non ha più il controllo e si domanda "dove sono i mie file?". In molti altri contesti, facciamo uso di sistemi chiusi dei quali non abbiamo il controllo, che nonostante tutto risultano molto più usabili di quanto lo possa essere un PC.

Per un utente che acquista una videocamera, tralasciando le differenze qualitative, non c'è differenza che il video sia analogico, mini-dv, MPEG2, su hard disk, cassetta, o DVD, l'importante è che alla pressione di un tasto"rec" il video venga memorizzato, per poi "riapparire" in un secondo momento con un tasto "play". Lo stesso vale per il telefono cellulare: immaginate se all'arrivo di un SMS il telefono proponesse la classica finestra "salva con nome"? Questa nuova funzionalità (la possibilità di scegliere dove salvare SMS) regalerebbe soltanto una inutile complicazione: all'utente non interessa come o dove il telefono memorizzi gli SMS, ma solo di poterli ricevere, leggere e ricercare con facilità.

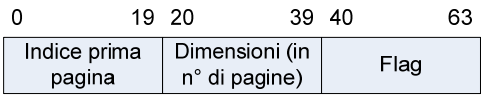
Per tutti i *data object* presenti in quantità massicce, sui quali sono necessarie operazioni di ricerca, inserimento, cancellazione, ci sono *storage* collegati a superfici di memoria. Questi *storage* sono costituiti da un insieme di file, dei quali uno contiene i dati (record o oggetti), e gli altri indici o altre strutture fini all'ottimizzazione (es. lista degli spazi liberi sul file dati).

In conclusione, i file su disco sono utilizzati solo come sede per strutture di sistema e moduli e per questa ragione non è stato adottato un file system di tipo gerarchico, ne un sistema di identificazione comprensibili all'uomo

Allocazione su RAM

A patto che sia supportata dall'architettura, sarà sempre utilizzata la memoria virtuale / paginazione Questo sistema permette innanzitutto di avere blocchi di memoria logicamente contigue, anche se non lo sono sul piano fisico, e infine di utilizzare lo spazio di indirizzamento per accedere a pagine che non risiedono fisicamente in memoria RAM[6].

La prima fase della gestione della memoria, consiste nell'organizzare lo spazio di indirizzamento virtuale che deve essere suddiviso in gruppi di pagine, ognuno con la propria funzione. La suddivisione non è statica, ma avviene dinamicamente in base alle richieste di ogni modulo. Difficilmente un modulo chiama direttamente i servizi di allocazione virtuale (`IVirtualAllocator`), in quanto sono presenti servizi ad alto livello in grado di gestire e ottimizzare le richieste di allocazione dinamiche più comuni. Lo stato di allocazione, è mantenuto su una apposita tabella, le cui righe hanno la seguente struttura:



La tabella deve essere sufficientemente ampia per contenere il caso limite, di un blocco distinto per ogni pagina (1.048.576 pagine nello spazio di indirizzamento di 4 GB, 64 bit per riga, per un totale di 8 MB).

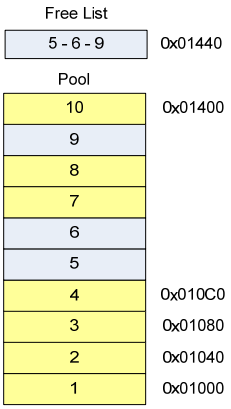
La maggior parte delle richieste di allocazione dinamica, riguarda blocchi di piccole dimensioni, che vengono allocati / deallocati molto frequentemente. Per questo tipo di richieste viene riservato un gruppo di pagine, suddiviso logicamente in blocchi di dimensione fissa, che prende il nome di *pool*. Nel sistema, ci possono essere più pool, ognuno con blocchi di dimensioni diverse. Ogni pool ha associata una lista contenente gli indici dei blocchi attualmente non allocati (free list). Ad ogni richiesta di allocazione, il sistema cerca il pool di dimensioni appropriate, in particolare quello con il blocco più piccolo in grado di soddisfare la richiesta. Se nel pool individuato esiste almeno un blocco libero, tale blocco

[6] L'accesso ad una pagina che non si trova un RAM causa un'eccezione (page fault), durante la quale il sistema deve provvedere al suo caricamento.

viene assegnato al richiedente ed eliminato dalla free list. In caso contrario, si avanza al pool successivo di pari dimensioni, fin quando non viene trovato uno pool con almeno un blocco libero. Se non viene trovato alcun pool, verrà allocato un nuovo pool.

TABELLA DEI POOL

Dimensione blocco	Totale blocchi	Totale blocchi liberi	Indirizzo primo blocco (base)	Indirizzo Free List
64	10	3	0x01000	0x01440
128	5	..	..	..
32	20	..	..	..
64	10	0	..	..



Le dimensioni del pool (il numero di blocchi da cui è composto) variano in base alle statistiche di uso, e alla disponibilità di ram, come variano la quantità di pool e le dimensione dei blocchi (tipicamente di 32 byte, 64 byte, 128 byte, 256 byte, 512 byte) L'indirizzo di ogni blocco del pool è dato dalla semplice formula:

*Indirizzo del blocco = Indirizzo base del pool + (indice del blocco * dimensioni del blocco)*

In caso di scarsa quantità di RAM le dimensioni dei pool possono essere ridotte dinamicamente, e alcuni pool (quelli che non hanno alcun blocco allocato) totalmente eliminati. L'uso dei pool permette di limitare la frammentazione della memoria, di velocizzare la ricerca di un blocco libero in caso di richieste massicce, e di diminuire il numero delle strutture necessarie a mantenere lo stato di allocazione.

Un pool di memoria è utilizzato anche per mantenere lo stack di ogni task. Poiché tutti i task condividono lo stesso spazio di memoria, non c'è possibilità di limitare l'area stack di un singolo task, che involontariamente potrebbe eccedere e sovrascrivere una zona di memoria riservata. Utilizzando la memoria virtuale, è possibile inserire una pagine di memoria vuota (non assegnata) che funga da separatore tra l'area stack, e il blocco sovrastante. In questo modo, appena viene superata la barriera tentando di leggere / scrivere nella pagine vuota, il processore genera un page fault, con il conseguente stack overflow.

Parte del pool (a seconda della disponibilità) è utilizzata come cache per i *data object* a cui si accede più frequentemente. Gli oggetti in questa cache, in base alle statistiche di uso, possono essere allocato e deallocati a discrezione del sistema in quanto per ogni *data object* esiste uno *storage*, che ne possiede una copia persistente. Quando un modulo fa richiesta di un oggetto ad uno *storage*, la richiesta passa attraverso un proxy che controlla prima in cache: se l'oggetto è presente, viene restituita la copia in cache, altrimenti la richiesta viene inoltrata allo *storage* di destinazione, che effettua l'estrazione. In caso di scrittura, le modifiche coinvolgeranno sia la cache, sia la copia persistente all'interno dello *storage*.

Per altre categorie di oggetti, come quelli che compongono l'interfaccia grafica, viene adottato un sistema a *reference count*, senza dimenticarne i limiti. Il *reference count* non deve essere applicato a prescindere su tutti gli oggetti, ma solo nei casi in cui è realmente necessario, evitando riferimenti circolari superflui. I garbage collector generazionali, oltre a doversi appoggiare su strutture molto complesse, portano il programmatore a dimenticare il costo dell'allocazione dinamica, della quale spesso viene abusato. E' si compito del sistema operativo / linguaggio di programmazione astrarre il più possibile dalla macchina fisica, ma tale astrazione non deve nascondere il significato di certe azioni: non c'è alcun vantaggio nel semplificare lo sviluppo del software, a discapito delle prestazioni del sistema.

Come sarà illustrato con maggior dettaglio nel capitolo successivo, nel sistema possono coesistere più sessioni utenti simultaneamente e ogni sessione utilizza il proprio spazio di indirizzamento virtuale. Solo le pagine che contengono dati in sola lettura, come ad esempio la parte di codice / dati statici di un modulo, possono essere condivise nella memoria fisica tra i più spazi di indirizzamento virtuale. Ad eccezione dei moduli di sistema, che sono allocati staticamente nella parte bassa della memoria, tutti gli altri saranno caricati di volta in volta nel primo gruppo di pagine disponibile. Poiché non esiste collegamento statico tra moduli, e le dipendenze sono verso i servizi, non ha alcuna rilevanza la posizione che assume un modulo in memoria.

Lettura e scrittura su file

L'uso della memoria virtuale non è utile solo ad estendere le capacità di memoria del sistema, a dispetto della RAM fisica a disposizione. L'uso di uno spazio di indirizzamento virtuale può essere utilizzato per uniformare tutte le operazione di IO, trattando ogni richiesta come fosse una operazione di lettura / scrittura su RAM.

L’accesso ad una superficie di memoria (ad eccezione della RAM) non è libero. Prima di ogni lettura o scrittura è necessario bloccare (*lock*) la porzione di memoria interessata, specificando offset e dimensione. Questa operazione riserva un gruppo di pagine nello spazio di indirizzamento virtuale, attraverso le quali è possibile (virtualmente) leggere e scrivere sulla superficie di memoria. Il trasferimento da superficie di memoria a RAM può essere effettuato in un'unica soluzione al momento del *lock*, o di volta in volta al primo accesso ad ogni pagina. Il *lock* può essere in sola lettura, o in lettura / scrittura. Nel caso sia anche in scrittura, nessun altro task può accedere all’area riservata (o ad un suo sottoinsieme), fin quando il *lock* non viene rilasciato, o il task terminato. Eventuali modifiche apportate in RAM, non vengono applicate alla superficie di memoria automaticamente: il programmatore deve invocare esplicitamente un commit per confermare la scrittura, o un rollback per annullare le modifiche fatte. Ogni scrittura è di tipo incrementale, quindi solo le pagine effettivamente modificate vengono trasferite.

Vogliamo leggere un file di testo, composto da un insieme di righe a lunghezza variabile. Poiché la lettura su file coinvolge generalmente piccolissime parti alla volta (nell’ordine dei byte), i sistemi operativi attuali[7] per minimizzare il numero di accessi al disco, eseguono una lettura bufferizzata, che coinvolge anche i dati adiacenti a quelli richiesti. La funzione o classe, che permette di leggere una linea dal file di testo, deve a sua volta fare una lettura bufferizzata, non conoscendo a priori la posizione carattere di fine riga. Per terzo, l’applicazione che invoca la procedura / metodo di lettura di una linea, deve fornire e allocare a sua volta un buffer, sufficientemente ampio a contenere l’intera riga (la cui dimensione è spesso sconosciuta). La semplice lettura di un file di testo, comporta l’allocazione di 3 buffer da parte dei tre componenti coinvolti. Utilizzando le superfici di memoria e i lock la lettura di una linea si trasforma in una semplice ricerca del carattere di fine riga in RAM, e può essere effettuata usando una qualsiasi funzioni che opera su stringhe (es. la strchr in c). Il programmatore non deve preoccuparsi di allocare e deallocare buffer temporanei.

In caso di necessità le pagine in RAM associate ad una operazione di *lock* che non hanno subito modifiche, possono essere liberate automaticamente, e utilizzate per altri scopi (le pagine eventualmente liberate, saranno nuovamente riallocate in RAM al primo accesso). Grazie a questo sistema è possibile effettuare il *lock*, anche di aree molto grandi, senza preoccuparsi della quantità di RAM disponibile, e senza togliere risorse al sistema.

Il *lock* su superficie di memoria permette di eliminare anche file / partizioni di swap: in caso siano necessarie grandi quantità di memoria, è sempre possibile allocare una superficie di memoria su disco di dimensioni arbitrarie, e tramite l’operazione di *lock*, leggere / e scrivere sulla superficie come se si stesse leggendo / scrivendo su ram.

[7] Si intende i sistemi unix-based (linux, mac OS X) e i sistemi windows

Hardware e Periferiche

Nei sistemi operativi attuali, è stato scelto di separare la parte software che gestisce l'hardware e i servizi a basso livello (detto *kernel*), dalla parte "ad alto livello" dedicata alle applicazioni e ai servizi non direttamente collegati ad una periferica. Questa scelta è stata effettuata principalmente per ragioni di sicurezza. Alcuni microprocessori supportano molteplici livelli di esecuzione, e in ogni livello è possibile eseguire solo un sottoinsieme di istruzioni. Solo il *kernel* si trova al livello in cui è permesso eseguire le *istruzioni speciali* in grado di comunicare direttamente con l'hardware o di assegnare privilegi a segmenti di memoria. Il livello di esecuzione può essere alterato solo da fattori esterni, come gli interrupt. Senza una protezione hardware un qualunque software potrebbe accedere direttamente al disco, saltando la verifica di eventuali autorizzazioni, oppure scrivere su particolari zone di memoria, cambiando alcuni flag per concedersi permessi che altrimenti non avrebbe.

Questo genere di protezioni sono indispensabili per alcune categorie di sistemi, per altri sono superflue. Per questo è possibile scegliere se affidarsi ad un unico livello di esecuzione (come avveniva per sistemi Windows non NT), o mantenere due livelli, uno di supervisore e uno di utente. L'eliminazione di un livello permette di avere un sistema più veloce e leggero in quanto l'esecuzione di procedure che accedono all'hardware avviene senza alcun cambio di contesto. Inoltre semplifica lo sviluppo driver, in quanto un modulo a livello supervisore (con accesso all'hardware) dispone dello stesso ambiente di esecuzione (servizi, oggetti, memoria etc) di un modulo a livello utente, e viceversa

Nei sistemi mono-utente, ad uso domestico e in particolare nei sistemi embedded e realtime è possibile usare un unico livello di esecuzione. Questo genere di sistemi nascono normalmente con un ruolo preciso e funzioni statiche, e non subiscono modifiche software a meno di migliorie o correzione di bug. Ogni eventuale aggiornamento avviene riprogrammando l'intero sistema, attraverso dispositivi normalmente non in dotazione all'utente, ma solo a centri autorizzati dal produttore.[8] Una volta garantita la sicurezza dei moduli di cui è composto il sistema, è tolta la possibilità di installarne di nuovi, il sistema rimane sicuro

Nei sistemi invece in cui è possibile aggiungere moduli, ai quali hanno accesso più utenti contemporaneamente, in cui è necessario limitare per ragioni di sicurezza alcune interazione, è possibile mantenere i due livelli di esecuzione, anche se questo non garantisce in assoluto la sicurezza. L'unica modo per avere un sistema sicuro è uno scrupoloso controllo da parte dell'utente, che non dovrebbe mai installare moduli sprovvisti di certificazione che ne garantisca la fonte e autenticità

La maggior parte delle installazioni di nuovi programmi, richiedono l'accesso tramite utente con privilegio di amministratore (o root). Con questo privilegio è possibile fare qualsiasi cosa al sistema, anche aggiungere moduli che operano a livello kernel in cui è possibile eludere qualunque sistema di sicurezza software, come acl, antivirus, firewall, etc.

Kernel e driver

Indipendentemente che si usi uno o due livelli di esecuzione, nel sistema non è presente uno strato che possa essere definito kernel, ma bensì solo alcuni servizi la cui implementazione necessita di un accesso diretto all'hardware. Alcuni dei servizi che operano a basso livello, assumo il ruolo che hanno attualmente i *driver*. In un certo senso il concetto di driver scompare fondendosi con quello di servizio. Se il driver è un elemento che permette di pilotare dispositivi hardware di natura diversa, attraverso un'interfaccia esterna comune, il servizio la generalizzazione di questo concetto estesa anche a componenti astratti.

L'accesso ai dati di un cd è normalmente fornito da un driver che comanda un componente hardware (lettore cd). E' possibile disporre di una copia del contenuto di un cd attraverso un file di immagine, che è un componente astratto. Il driver del lettore cd, e il file di immagine forniscono entrambi il servizio di superficie di memoria, al quale può collegarsi il servizio"file system cd-fs" per l'estrazione di file e cartelle.

Se è presente il livello supervisore l'uso di *service object* che accedono a componenti hardware è effettuato tramite *proxy*. Compito di questi *proxy* è fare da ponte tra i due livelli, effettuando i cambi di contesto, e mappando gli indirizzi di memoria di eventuali parametri / valori di ritorno, tra i due spazi di indirizzamento virtuale, normalmente diversi. Questi *proxy* possono essere generati in modo totalmente automatico.

Gestione ad oggetti

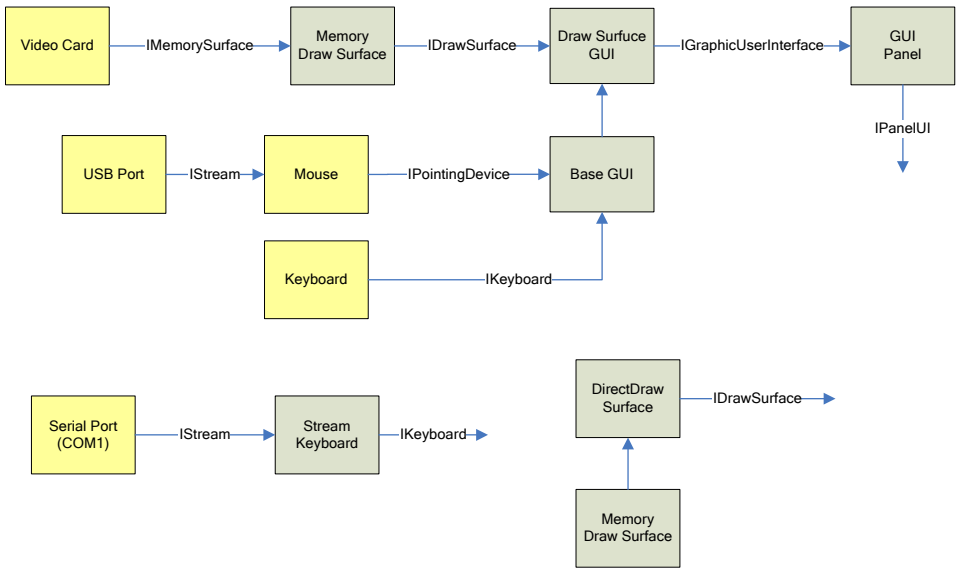
Ogni componente hardware è gestito da un oggetto, che ne espone i servizi attraverso interfacce standard. Come più volte ripetuto, l'oggetto che accede all'hardware non ha alcun trattamento particolare, ed è allo stesso livello di un qualunque altro oggetto. Come ogni *service object*, anche gli oggetti-driver possono avere delle dipendenze da altri servizi, tipicamente forniti da altri oggetti-driver di periferiche adiacenti.

---

[8] Non è più cosi per i telefoni cellulari moderni (smartphone) e palmari, nei quali l'utente può installare e rimuovere programmi.

L'oggetto-driver associato ad una chiavetta USB necessita un servizio canale seriale (fornito dalla porta USB) per la comunicazione con la chiave, e fornisce un servizio di superficie di memoria / unità disco. Tale servizio può essere utilizzato da un secondo oggetto, in grado di decodificare un file system (es FAT32) su superficie di memoria.

L'esempio sottostante mostra un possibile schema a blocchi del modello ad oggetti che gestisce la parte di visualizzazione grafica:



I blocchi in giallo sono service object collegati a componenti hardware, mentre le frecce rappresentano le interfaccie che forniscono. L'oggetto "Video Card", comunica direttamente con la scheda grafica, esponendo la superficie di memoria primaria (quella visibile) della scheda grafica attraverso l'interfaccia IMemorySurface. L'oggetto"Memory Draw Surface" fornisce i servizi di disegno 2D IDrawSurface, andando a scrivere direttamente su una superficie di memoria, senza sfruttare eventuali accelerazioni hardware. L'oggetto "Draw surface GUI" fornisce servizio di interfaccia grafica, disegnando i widget attraverso le primitive fornite da IDrawSurface, ed eredita alcuni comportamenti dalla classe base "Base GUI". L'oggetto "Draw surface GUI" ha inoltre bisogno di un servizio tastiera, e di un servizio di puntamento L'oggetto "Mouse", fornisce il servizio di puntamento, decodificando i dati inviati attraverso un canale dal mouse. Questo canale può essere fornito dall'oggetto che gestisce la porta USB, al quale è connesso fisicamente il mouse. Il servizio tastiera può essere dall'oggetto "Tastiera" che gestisce una normale tastiera hardware, oppure dall'oggetto "Strema Keyboard" che decodifica uno strema seriale simulando la pressione dei tasti. Infine l'oggetto "GUI Panel" utilizza i servizi forniti da "IGraphicUserInterface" per fornire a sua volta un servizio di interfaccia utenti basato su pannelli e finestre. In un sistema windows, dove non si ha accesso diretto all'hardware, è possibile fornire comunque il servizio IDrawSurface utilizzando DirectDraw per comunicare direttamente con la scheda grafica

Interrupt ed eventi

Gli interrupt sono il nucleo del funzionamento di un sistema hardware multi-task basato su eventi. Grazie a questi segnali, è possibile ricevere notifiche su alcuni cambiamenti di stato dei componenti hardware, come ad esempio il completamento di una operazione di IO su disco, lo svuotamento del buffer audio della scheda sonora o l'arrivo di un pacchetto sulla scheda di rete. Un interrupt deve essere servito più rapidamente possibile, per questo i microprocessori a fronte di tali segnali, interrompono immediatamente l'esecuzione, passando il controllo alla routine di servizio associata all'interrupt.

Nel sistema, ad ogni interrupt può essere associata una procedura di gestione, tipicamente un metodo del *service object* correlato alla periferica. L'esecuzione della procedura non avviene in maniera sincrona. Le uniche entità in grado di seguire codice sono i task, di conseguenza a fronte di un interrupt viene generato un nuovo task, che ha come procedura di ingresso proprio quella specifica per quell'interrupt. Questo task, ha priorità elevata, ma non viene eseguito immediatamente. La routine di servizio dell'interrupt, una volta generato il task, passa il controllo al servizio di scheduling. Questo servizio deve decidere se sospendere il task attualmente in corso, ed avviare il task associato all'interrupt o riprendere il task corrente, lasciando il task dell'interrupt in coda.

Sessioni e sicurezza

Utenti e i gruppi di utenti sono normalmente abbinati ad entità organizzativa del mondo reale, cosi persone che ricoprono un determinato ruolo nell'organizzazione, o afferiscono ad una particolare struttura / reparto possiedono determinati privilegi. Nei sistemi attuale si tende a separare la parte gestionale e di banca dati che descrivere e regole i flussi dell'organizzazione, da quella che invece regola i permessi e le autorizzazioni nel sistema. Inoltre, i gruppi sono entità che non possiedono attributi, ma mettono semplicemente in relazione insiemi di utenti che appartengono ad una medesima categoria.

Nel sistema il termine *utente* ha valenza del tutto generale, e comprende un qualsiasi oggetto che implementi l'interfaccia `IUser` e disponga di credenziali riconosciute dal servizio di autenticazione (`IAuthenticationService`). Poiché un oggetto può essere inserito all'interno di un container e gli utenti sono oggetti, ogni container può assumere la funzione di "gruppo" per utenti: lo stesso modello ad oggetti utilizzate per rappresentare un'organizzazione / struttura, può essere utilizzato per definire autorizzazioni e permessi.

Autorizzazioni

Tramite autorizzazioni si regola l'uso dei servizi da parte degli utenti. Le autorizzazioni sono costituire da un insieme di regole della forma:

A (non) può eseguire O (su B)

A rappresenta chi si sta autorizzando, e può essere un'utente o un container. Se A è un container, sono autorizzati tutti gli *utenti* a lui discendenti (figli, figli dei figli, etc). *O* rappresenta l'operazione che si sta autorizzando, e può essere un servizio, o un metodo di un servizio. *B* è opzionale, e deve essere specificato se l'autorizzazione è destinata ad un tipo o un oggetto specifico. Se B è un container, la regole è applicata anche ai suoi discendenti.

"Mario Rossi" può eseguire IContainer (tutti i metodi dell'interfaccia IContainer)
"Mario Rossi" può eseguire IContainer.Add (solo il metodo Add di IContainer)
"Mario Rossi" può eseguire IPrinter sull'oggetto di tipo "Stampante InkJet" (tutti i metodi dell'interfaccia IPrinter, negli oggetti del tipo specificato)
"Mario Rossi" non può eseguire IPrinter.Print sull'oggetto "HP LaserJet 600" (solo il metodo Print nelll'oggetto specificato)

Tutti gli oggetti implementano implicitamente l'interfaccia `IObject` che contiene i pseudo metodi `GetAttribute` e `SetAttribute`. Questi metodi permettono rispettivamente di leggere e modificare un attributo di un oggetto e possono essere utilizzati per controllare le generiche autorizzazioni di "lettura" e "scrittura". In caso di regole contrastanti, vengono adottati i seguenti criteri per stabilire le priorità:

- Le autorizzazioni negate hanno la priorità su quelle concesse
- Le autorizzazioni definite per un oggetto specifico hanno la priorità rispetto a quelle definite per un tipo
- Le autorizzazioni definite per un tipo hanno la priorità rispetto a quelle definite per un servizio
- Le autorizzazioni definite per un container antenato di un oggetto, hanno la priorità rispetto a quelle definite per un tipo o un servizio (se l'oggetto A è di tipo "Documento di testo" e implementa l'interfaccia "IDocument", ed è figlio di un container B, le regole definite per il container B hanno la priorità rispetto alle regole definite per il servizio "IDocument" e il tipo "Documento di testo")
- Le autorizzazioni definite per un container antenato più vicino ad un oggetto, hanno la priorità rispetto a quelle definite per un container antenato più lontano. (se B è figlio di A, e C è figlio di B (A ->B ->C), per C hanno maggiore priorità le regole definite per B, rispetto a quelle definite per A)

Le regole vengono memorizzate nel registro di sistema, e protette con firma digitale. La chiave privata per la crittografia viene generata utilizzando le credenziali dell'utente stesso, ed è calcolata di volta in volta evitando cosi di essere memorizzata. In questo modo, solo l'utente che ha definito le autorizzazioni ha il potere di modificarle o rimuoverle.

Proxy-filtro e sicurezza

Tutti i servizi sono implementanti assumendo che i controlli di sicurezza siano stati effettuati a monte di ogni richiesta. Infatti, in poche occasioni vengono utilizzati direttamente i servizi esposti da un *service object*: la maggior parte delle richieste passa attraverso *proxy-filtro*. Il primo ruolo di questi *proxy* è verificare, in base alle regole definite, che l'utente abbia le autorizzazione per utilizzare un servizio. Inoltre, quando si deve installare un modulo firmato da un autore che non gode della nostra fiducia, è possibile creare un contesto di esecuzione protetto, nel quale l'acceso ai servizi sensibili (come disco, rete, etc), è effettuato attraverso *proxy-filtro*. Questi proxy in base alle scelte dell'utente, possono:

- Ignorare tutte le richieste al servizio
- Chiedere conferma all'utente prima di inoltrare ogni richiesta al servizio.

In questo modo è sempre possibile tenere sottocontrollo l'attività di un modulo, e solo dopo aver verificato che non compromette la sicurezza e stabilità del sistema, concedergli libero accesso a tutti i servizi. Oltre agli scenari qui esposti, i *proxy-filtro* possono in genere usati in molteplici scenari.

La scrittura su disco è normalmente senza crittografia. Se per motivi di sicurezza venisse richiesto il supporto per IO crittografato, sarebbe sufficiente anteporre un proxy al servizio "IMemorySurface". Questo proxy avrebbe il compito di decrittografare le pagine di memoria dopo essere state lette, e crittografarle prima di essere scritte. Il proxy può essere utilizzato con ogni superficie di memoria, che sia essa un'unità disco, un file, o memoria RAM.

Grazie ai *proxy-filtro* è possibile aggiungere lo strato di sicurezza "a richiesta", senza compromettere le prestazioni del sistema nel caso in cui certi controlli non siano necessari.

Sessioni e autenticazione

Per poter utilizzare i servizi / oggetti forniti dal sistema, è necessario iniziare una *sessione*. Per poter iniziare una sessione, l'utente deve prima autenticarsi. In linea con i principi del sistema, anche l'autenticazione è un servizio (`IUserAuthenticationService`), che può essere implementato da più oggetti in modo diverso (ad esempio richiedendo di immettere la classica coppia utente/password, o con sistemi più moderni, quali smart card e impronte digitali). Una volta verificate le credenziali, la sessione ha inizio e l'utente, nei limiti delle sue autorizzazioni, ha accesso a tutte le risorse del sistema (servizi, oggetti). Più utenti possono aprire sessioni simultaneamente sulla stesso sistema, ma ogni sessione è isolata dalla altre, senza possibilità di intercomunicazione (se vengono adottati i due livelli di esecuzione, ognuna ha il proprio spazio di indirizzamento virtuale).

Come i *metodi*, gli *oggetti*, e i *moduli*, anche le *sessioni* hanno associato un *contesto* di esecuzione. Si ricorda che in ogni contesto, è possibile stabilire quel'è il *service object* che implementa ciascun servizio. La prima scelta viene effettuata dal programmatore del modulo, che può formalizzarla staticamente tramite dei metadati affiancati a ciascun elemento (modulo, oggetto, metodo), o dinamicamente a livello di codice. L'utente ha il potere di modificare queste scelte, definendo un insieme di regole della forma:

Il servizio S è implementato dal tipo T (nel contesto C)

S rappresenta il servizio, T il *service object* che implementa il servizio, C il contesto nel quale la regola deve essere applicata. Le regole definite dall'utente possono essere vincolanti o semplici preferenze. Nel primo caso, l'assegnazione servizio – implementazione avviene forzatamente, nel contesto specificato, e in tutti i contesti inferiori. Nel secondo l'assegnazione avviene solo se non è stata definita alcuna regola per quel servizio nel contesto specificato e in quelli superiore. In caso di regole contrastanti, vengono adottati i seguenti criteri per stabilire le priorità:

- Le regole definite per un contesto più ristretto (inferiori) hanno la priorità rispetto a quelle definite per un contesto più ampio (superiore).
- A parità di contesto, le regole definite dall'utente hanno la priorità rispetto a quelle definite dal programmatore del modulo.
- Le regole definite dall'utente come "vincolanti" hanno la priorità rispetto alle altre.

All'interno di una sessione, per ottimizzare le risorse, viene creata una sola istanza di ogni *service object*, al momento della prima richiesta. Nel caso in cui più sessioni richiedano lo stesso *service object*, la regola generale è di creare una istanza diversa per sessione, anche se non tutti i *service object* possono avere più di una istanza (ad esempio gli oggetti-driver che rappresentano periferiche hardware), I *service object* ad istanza singola, possono essere in genere condivisi tra più sessioni, ma in alcuni casi l'assegnazione deve essere esclusiva (solo una sessione può utilizzare l'oggetto)

Il servizio interfaccia grafica non può essere in genere condiviso tra più sessioni, e un solo utente/sessione ne ha il controllo (a meno non si disponga di due monitor, due testiere, etc).

Sessioni remote

Un utente del sistema locale può iniziare una sessione in un sistema remoto. In questo caso i contesto di "sistema" e "sessione" del sistema remoto, vengono fusi con quelli di "sistema" e "sessione" del sistema locale. Durante l'apertura della sessione remota, è necessario stabilire chi tra i due sistemi ha il ruolo di master. Una volta stabilita una sessione remota, il sistema locale, attraverso proxy, può utilizzare in modo trasparente i servizi del sistema remoto**, e viceversa**. Se un servizio è implementato da entrambi i sistemi, l'implementazione nel sistema master ha la priorità rispetto all'altra.

L’utente U crea una sessione S1 all’interno ella macchina M1. L’utente deve poter accedere al servizio scansione fornito dalla macchina M2, alla quale è collegata lo scanner. La macchina M1, si autentica ad M2 con il ruolo di master usando le credenziali di U, e da inizio ad una sessione S2 su M2. Il servizio di scansione M2 ha necessità di utilizzare il servizio interfaccia grafica, per poter mostrare l’anteprima della pagina scansionata. Poiché la sessione S2 è stata generate dalla sessione S1 di M1, ed M1 è master, il servizio di interfaccia grafica predefinito è quello fornito da M1. Attraverso un proxy, il servizio in esecuzione su M2 utilizza il servizio grafico fornito da M1

Sicurezza codice e moduli

Ogni modulo deve essere provvisto di firma digitale. La crittografia a chiave asimmetrica (base della firma digitale), permette di ottenere due importanti risultati: il primo, è di poter certificare l’autore di un modulo (ogni produttore possiede la propria “firma”), il secondo di garantire che un modulo non venga alterato dopo che è stata applicata la firma. Di fatto, la sicurezza dell’intero sistema si basa su un unico modulo, quello che ha il compito di verificare l’autenticità della firma e l’integrità dei modulo (compreso se stesso). Essendo purtroppo anch’esso un componente software, che risiede su unità disco, può essere soggetto ad attacchi e mutazioni. Una soluzione è isolare questo componente in una memoria in cui è possibile escludere elettricamente la scrittura, attraverso un interruttore. Per questo fine sarebbe sufficiente una qualsiasi memoria flash (es. SD card), considerando anche i bassi costi che hanno questo tipo di supporti. L’accesso in scrittura deve essere consentito solo durante l’installazione del sistema (compresi eventuali aggiornamenti successivi), e negato in tutto gli altri casi. In questo modo, anche se accidentalmente si dovesse entrare in contatto con un modulo dannoso (per scelta dell’utente, o per un buco nella sicurezza), questo non potrebbe alterare lo stato del sistema, senza essere rilevato.
In abbinamento alla firma digitale, se vengono adottati i due livelli di esecuzione, è possibile applicare un’ulteriore protezione rendendo le aree di memoria dedicate all’esecuzione del codice in sola lettura, e non concedendo il permesso di esecuzione alle altre. In questo modo, nessun attacco può causare esecuzione di codice, e nessuna esecuzione di codice può alterare il codice eseguibile di un modulo in memoria (le modifiche su disco sono bloccate dalle firma digitale)

Interfaccia utente

(testo)

Interfaccia grafica basata su pannelli

Il pannello può essere

Ambiente di programmazione testuale

Il ruolo principale della riga di comando è quello di manipolare gli oggetti di sistema, navigando nell'albero ed invocando i metodi dei servizi che espongono. Attraverso espressioni del tutto simili a xpath, è possibile interrogare l'albero degli oggetti, e

Utilizzando i servizi di formattazione (`IObjectFormatter`) è possibile avere una rappresentazione testuale di ogni oggetto, mentre con i servizi di parsing, è possibile costruire un oggetto partendo da una stringa di testo digitata da tastiera. In questo modo eventuali output potranno essere facilmente visualizzati.

Utilizzando l'ambiente a riga di comando, il contesto di esecuzione muta e in particolare tutti i servizi collegati all'interfaccia grafica a ai dispositivi di input. In questo modo, anche i moduli che fanno uso di interfaccia grafica, possono essere eseguiti nell'ambiente testuale, in quanto è presente un servizio GUI che implementa i principali widget ed elementi di interfaccia (tra cui TextBox, Button, ComboBox, CheckBox, Menu, GroupBox, ListBox, ListView, etc) attraverso i soli caratteri ascii.

Le specifiche dettagliate sulle parole chiavi e i comandi disponibili non saranno descritte in questo documento.

Esempi comandi

```
[images/imageA/levels].Add([internet/www.mysite.com/index.htm/>/body/#img[@id = "imageC"]])
```

Aggiunge un livello all'oggetto di tipo immagine "ImageA", prendendo come sorgente l'immagine con 'id "ImageC" nella pagina html di url www.mysite.com/index.htm

```
[home].Add([images/imageA/levels/#level[0]].Merge([images/imageB/levels/#level[25]]).Encode("jpeg"))
```

Fonde il livello 0 dell'immagine "ImageA" con il livello 25 dell'immagine "ImageB". L'immagine risultante, viene codificata in formato JPEG, e aggiunga al container "Home"

*[media/music].**Add**([device/soundblaster].**Aquire**(44100, 8, Stereo, 20).**Encode**("mp3"))*

Registra uno spezzone audio dalla scheda sonora "soundblaster" con frequenza di campionamento 44 khz, 8 bit per campione, stereo, e della durata di 20 secondi . L'oggetto "audio" risultante, viene codificato in formao mp3, e aggiunto al container "music"

*[media/video/videoA/frames].**Add**([device/hpscan].**Aquire**(A4, 640, 480, RGB24))*

Inserisce un fotogramma nell'oggetto "videoA", prendendo come sorgente un'immagine acquisita da uno scanner in formato A4

Implementazione

L’implementazione del sistema avverrà in due fasi, che potranno procedere in modo sequenziale o parallelo. La prima fase, si svilupperà dall’alto verso il basso, e sarà mirata ad implementare tutti i servizi ad alto livello, utilizzando un sistema operativo ospite di appoggio. La seconda fase invece si svilupperà dal basso verso l’alto, e sarà mirata ad implementare tutti i servizi a basso livello collegati alle periferiche hardware. Nella prima fase, saranno utilizzati al massimo i servizi astratti e le system call fornite dal sistema operativo ospite. Nel passare alla seconda fase, ci sarà una fase di transizione nella quale le uniche dipendenze dal sistema operativo ospite saranno quelle verso i driver delle periferiche, tutti i servizi ad un livello più alto saranno implementati manualmente. I servizi per l’accesso diretto all’hardware della seconda fase, sostituiranno le chiamate ai driver effettuate nella fase di transizione.
L’uso di un sistema operativo ospite, permette di ottenere due vantaggi: il primo, di poter disporre di un sistema stabile, con strumenti di debug avanzati per implementare i servizi non a basso livello. Il secondo la portabilità di ogni modulo del sistema, nel sistema ospite.

La portabilità tra sistemi diversi, non si realizza attraverso macchine virtuali, che interpretano dei linguaggi intermedi. I sistemi sono diversi, perché ogni sistema è pensato per assolvere al meglio un determinato compito: l’uniformare tutto attraverso macchine virtuali non permette di sfruttare questi vantaggi, rende i sistemi lenti e vanifica le spese effettuate per un hardware più potente (es. le accelerazioni delle schede video, o le estensioni di un microprocessore). La dipendenza di una applicazione, non è mai verso un particolare componente hardware (quindi una macchina), ma verso dei servizi**: l’ hardware è solo uno dei possibili mezzi attraverso il quale ottenere alcuni di quei servizio**.

Per questi ragionamenti, la portabilità di un modulo non dovrebbe essere realizzare attraverso un byte code comune, ma attraverso delle comuni interfacce e un sistema di comunicazione che metta in contatto “chi” necessita di un servizio con chi è in grado di fornirlo. Cosi ad esempio un modulo che richiede i servizi di IO, eseguito nel sistema operativo ospite, utilizzerà l’implementazione che invoca le system call del sistema ospite, mentre eseguito nel sistema “vuoto” (lo stesso modulo binario!) utilizzerà l'implementato che comanda direttamente il dispositivo hardware interessato.

Una applicazione per memorizzare le impostazioni, chiama un set di system-call proprietarie per la gestione del file, nei quali scrive una serie di stringhe di testo, formattate in un certo modo. L’applicazione, utilizzando queste system call, vuole ottenere un fine: memorizzare in modo persistente dei dati strutturati, per poi essere letti in un secondo momento. Non chiede di aprire un file, non chiede di accedere al disco! Questa esigenza dovrebbe essere mascherata da un servizio “impostazioni applicazioni”, la cui implementazione in un sistema potrebbe utilizzare un file su disco, mentre in un altro potrebbe comunicare con un servizio database remoto!

Prima fase

Il sistema operativo ospite sarà inizialmente un sistema windows. Le ragioni di questa scelta sono la maggior presenza di driver, la maggior efficienza del sottosistema grafico, e la maggior presenza di strumenti di sviluppo e debug. La scelta potrebbe ricadere su Windows 98, per l’assenza dei due livelli di esecuzioni, e la possibilità di eseguire istruzioni privilegiate a livello applicazione. Sia Windows 98 che Windows XP implementano il sottosistema Win32, di conseguenza non dovrebbero esserci difficoltà nel passare da uno all’altro.

Caricamento
Il sistema sarà rappresentato da un unico processo nel sistema operativo ospite. L’avvio di questo processo corrisponderà alla fase di boot del sistema. Questo processo si limiterà a caricare il modulo-loader, ed eseguire la sua procedura di inizializzazione. Questo modulo, in modo del tutto analogo a come avverrà nel sistema reale, leggerà le impostazioni di sistema, e caricherà gli altri moduli.

File system
Il servizio di allocazione superfici di memoria su disco, sarà implementato utilizzando il file system del sistema operativo ospite. Tutte le superfici di memoria risiederanno su una apposita cartella sottoforma di file. I file saranno chiamati con il loro ID, e non ci sarà alcuna struttura gerarchica

Gestione delle memoria
Il servizio di allocazione dinamica della memoria userà le funzioni fornite direttamente dal crt di c (malloc, free)

Moduli
I moduli saranno compilati e linkati in formato PE, sottoforma di DLL. Il servizio di caricamento moduli userà le system call del sistema operativo ospite per il caricamento dinamico di DLL.

Registro di sistema
Il registro di sistema dipende soltanto dal servizio file system, l’implementazione effettuata in questa fase potrà essere utilizzata anche nel sistema reale.

Interfaccia grafica
Il servizio di interfaccia grafica sarà implementato all’interno di una finestra del sistema ospite. La finestra potrà essere

collocata in un desktop secondario, senza barra dei titoli a tutto schermo, o nel desktop principale. Le operazioni di disegno primitive avverranno utilizzando i componenti del sistema operativo ospite (es. GDI+)

Task e scheduler
I task saranno implementati sottoforma di thread . Lo scheduler userà le system call del sistema operativo per effettuare i context switch tra thread.

Storage
Gli *storage* saranno implementati all'interno di database relazioni esistenti, sottoforma di tabelle.

Rete
I servizi di rete saranno implementati utilizzando socket e le altre system call fornite dal sistema operativo ospite

Periferiche
Tutti i servizi collegati a periferiche hardware saranno implementati richiamando le rispettive system call fornire dal sistema operativo ospite

Transizione verso la seconda fase

Interfaccia grafica
Saranno utilizzati DirectDraw / Direct3D per ottenere l'accesso diretto al frame buffer, e usufruire delle accelerazioni 2D / 3D. Tutte le operazioni di disegno primitive saranno implementate manualmente senza far uso di alcun componente del sistema ospite.

File system
Sarà utilizzata una partizione, o creato un unico file di grandi dimensione, allo scopo di simulare un'unità disco. Questo spazio sarà allocato "manualmente" con la tecnica descritta nei capitoli precedenti.

Moduli
Il caricamento dei moduli avverrà "manualmente", interpretando il formato del PE, e utilizzando i servizi di memoria per allocare i segmenti con i relativi privilegi. In caso di necessità, il modulo sarà rilocato dinamicamente.

Gestione delle memoria
Verranno invocati direttamente i servizi di allocazione virtuale forniti dal sistema operativo. Ogni macro blocco di pagine sarà utilizzato per una funzione e le richieste di allocazione dinamica saranno implementate suddividendo questi blocchi con le tecniche descritte nei capitoli precedenti. (es i pool)

Rete
Sarà implementato "manualmente" lo stack TCP/IP con i servizi fondamentali quali ARP, DNS, DHCP, ICMP. Le uniche dipendenze dal sistema operativo ospite saranno verso i driver che forniscono l'accesso ai dispositivi hardware per lo strato fisico (es. invio di un frame ethernet sulla scheda di rete)

Seconda fase

Caricamento
Verrà scritto un boot sector che utilizzando gli interrupt del BIOS (disponibili in real mode) caricherà il modulo-loader dal disco. Questo modulo, conterrà tutti i servizi necessari per un sistema minimo, e dopo aver portato il processo in protected-mode, caricherà tutti i moduli di sistema.

Interfaccia grafica
Al di fuori del sistema operativo ospite, saranno implementati i servizi video, per le schede VGA (con le estensioni VBE). Se verranno rilasciate le specifiche per una scheda video moderna, saranno utilizzati i servizi di accelerazione da essa forniti.

Gestione delle memoria
Al di fuori del sistema operativo ospite, sarà implementata il servizio di gestione della memoria virtuale.

File system
Al di fuori del sistema operativo ospite saranno implementati i servizi di accesso diretto al disco, inviando i comandi ai rispettivi controller.

Periferiche
Saranno via via implementati tutti i servizi collegati alle periferiche hardware, inviando i comandi alle rispettive porte/registri. I primi servizi ad essere implementati saranno il bus PCI, DMA, Interval Timer, Interrupt Controller, RS232, porta USB, Mouse, Tastiera, Video modalità testo, Video modalità grafica VGA, controller IDE / SATA.

Casi d’uso

Struttura ad oggetti

Lo schema a fianco illustra un ipotetico modello ad oggetti che descrivere una struttura universitaria. L’oggetto “Paolo Rossi” è di tipo “Persona”, e implementa l’interfaccia `IUser`. Si nota che“Paolo Rossi” è presente in più container, sia come elemento della rubrica di sistema, sia come studente iscritto al primo anno di ingegneria. Una “persona” assume il ruolo di studente, nel momento in cui è iscritto ad almeno un corso di laurea. (ovvero è figlio di un container “Studenti Iscritti”). “Paolo Rossi” in quanto studente di ingegneria deve poter usare la “stampante 1” del “laboratorio 1”. E’ possibile concedere l’autorizzazione direttamente all’oggetto “Paolo Rossi”, ma poiché la regola vale per tutti gli studenti di ingegneria, è più corretto assegnare tale autorizzazioni all’oggetto di tipo “Facoltà” chiamato “Ingegneria”