Simulatore di Gestione della Memoria di un Elaboratore SiGeM

http://stylosoft.altervista.org
stylosoft@gmail.com

Specifica tecnica	26 Gennaio 2008
Documento Esterno - Formale - v2.0	Specifica_tecnica.pdf

Redazione:

Alberto Zatton

Revisione:

Daniele Bonaldo

Approvazione:

Davide Compagnin

Lista di distribuzione:

Prof. Vardanega Tullio
Prof. Palazzi Claudio
Stylosoft

Registro delle modifiche:

Versione	Data	Descrizione delle modifiche
2.0	26/01/2008	Evidenziati i termini del glossario
1.0	25/01/2008	Aggiornati i diagrammi, le descrizioni e aggiunto tracciamento
0.6	24/01/2008	Scritte le prime descrizioni dei package
0.4	24/01/2008	Aggiornate le stime di fattibilità
0.2	23/01/2008	Prima Stesura

http://stylosoft.altervista.org
stylosoft@gmail.com

Sommario

Questo documento ha il fine di illustrare le scelte tecniche, tecnologiche ed architetturali per la realizzazione del sistema SiGeM.

Indice

http://stylosoft.altervista.org
stylosoft@gmail.com

1 Introduzione

1.1 Scopo del documento

Tale documento analizza lo scenario applicativo e definisce l'architettura del progetto ad un alto livello di astrazione, finalizzandone gli aspetti tecnici, operativi ed implementativi affinché si possa passare ad una progettazione di dettaglio e quindi finire alla programmazione vera e propria.

1.2 Scopo del prodotto

Lo scopo del prodotto SiGeM è quello di fornire un sistema didattico destinato allo studio dei meccanismi di gestione della memoria in un elaboratore *multiprogrammato*. Tale software consentirà all'utilizzatore di simulare il comportamento di vari algoritmi di rimpiazzo delle *pagine* e *segmenti*, sulla base di dati da lui specificati, ai fini di illustrare le problematiche inerenti alla gestione della memoria. Per una descrizione più dettagliata rimandiamo al documento [AR].

1.3 Glossario

I termini in *corsivo* sono descritti per una migliore comprensione in [G], allegato a questo documento.

1.4 Riferimenti

- [AR] Analisi_dei_requisiti.pdf
- [SF] Studio_di_fattibilita.pdf
- [G] Glossario.pdf
- [PP] Piano_di_progetto.pdf
- [NP] Norme_di_progetto.pdf

1.5 Relazione con altri progetti

Al fine di ottenere una *simulazione* più completa possibile, oltre alla gestione della memoria si è scelto di includere anche la *simulazione* della schedulazione dei processi. Per riuscire a contenere il tempo di sviluppo, si è deciso di utilizzare algoritmi già sviluppati dal gruppo BlueThoth, realizzatori del progetto “SGPEMv2”. Questa scelta è stata influenzata dalla buona realizzazione di tali algoritmi, oltre al tipo di linguaggio usato (*Java*), compatibile con il nostro progetto.

2 Definizione del prodotto

2.1 Metodo e formalismo di specifica

Per rendere più comprensibile l'architettura da noi realizzata per descrivere il progetto, utilizzeremo diagrammi delle classi in linguaggio *UML*. Di questi diagrammi si fornirà poi una descrizione in dettaglio.

Questo documento è stato redatto durante al fase di analisi (rif. [PP]), pertanto presenterà un livello di astrazione alto. Successivamente, durante la fase di progettazione, verrà analizzato ogni singolo componente, specificando con più rigore il contenuto delle classi e le relazioni tra le classi stesse.

2.2 Primo livello di decomposizione architetturale

Di seguito viene riportato il diagramma delle classi realizzato in fase di analisi, con relativa illustrazione dei vari componenti. I componenti disegnati in rosso saranno quelli sviluppati ex-novo dal nostro gruppo, mentre quelli disegnati in giallo saranno sviluppati a partire da quelli progettati dal gruppo BlueThoth (vedi 1.5).

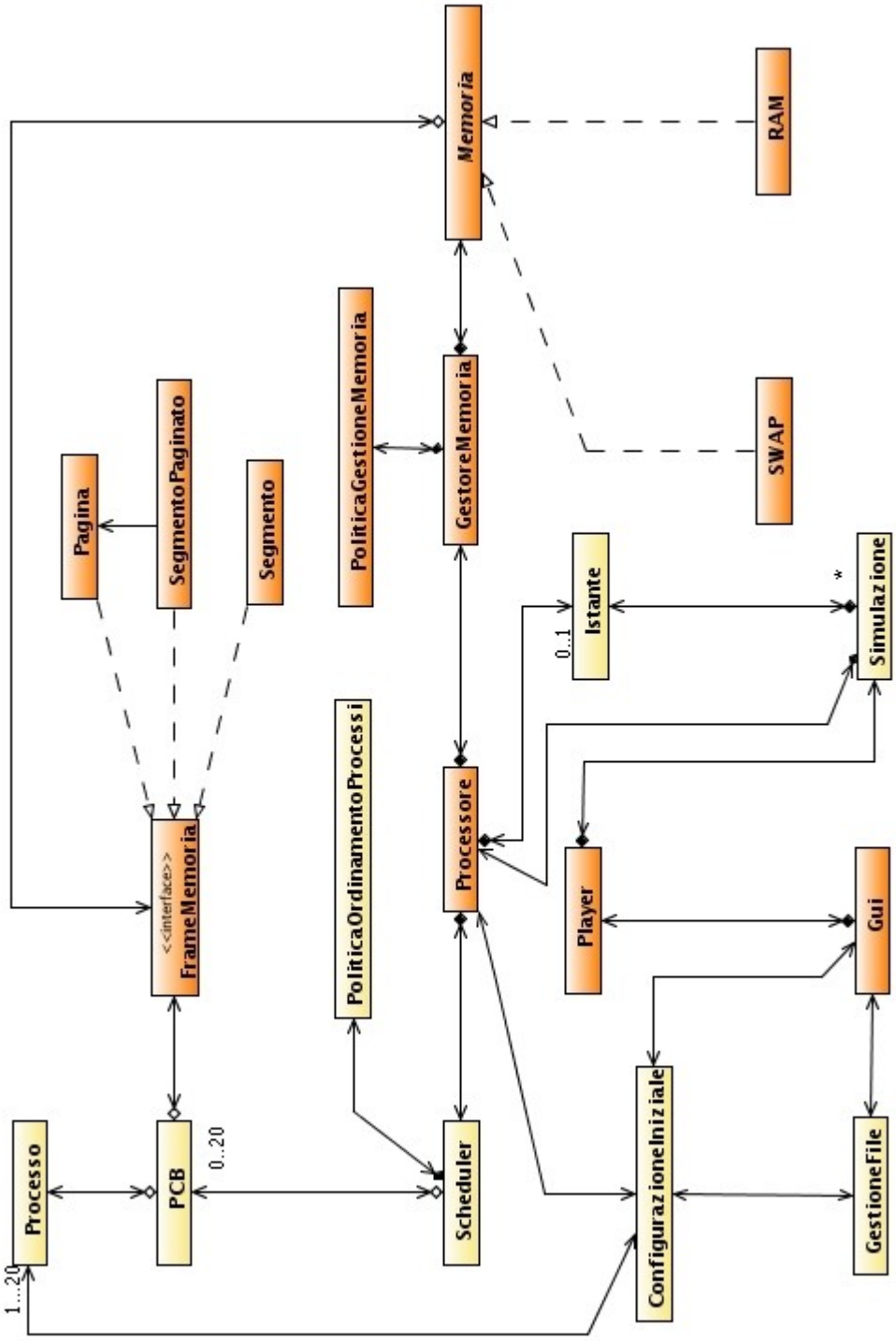
Memoria
RAM
SWAP
PoliticaGestioneMemoria
GestoreMemoria
Pagina
SegmentoPaginato
Segmento
Istante
Simulazione
0..1
*
<<interface>>
FrameMemoria
PoliticaOrdinamentoProcessi
Processore
Player
Gui
Processo
PCB
0..20
Scheduler
ConfigurazioneIniziale
GestioneFile
1...20

Figura 2.1

http://stylosoft.altervista.org
stylosoft@gmail.com

Il cuore centrale del sistema è formata dalla coppia **Scheduler-GestoreMemoria**, coordinata dalla classe **Processore**: è compito dello **Scheduler** infatti scegliere il *processo* da mandare in esecuzione, in accordo con la politica di schedulazione dei processi scelta, mentre **GestoreMemoria** si occupa della gestione delle *pagine/segmenti* di cui ha bisogno il *processo*. Compito del **Processore** è infine unire le informazioni ottenute da **Scheduler** e **GestoreMemoria** per costruire un **Istante** della *simulazione*. Per **Istante** si intende una singola unità di tempo nell'evolvere della *simulazione*: deve perciò contenere tutte le informazioni che serviranno poi per calcolare le statistiche e lo stato del sistema.

Per quanto riguarda il meccanismo di schedulazione dei processi, ci siamo ispirati al lavoro effettuato in precedenza dal gruppo BlueThoth come specificato nel punto 1.5. Le classi principali coinvolte nella schedulazione sono: **Processo**, **PCB** e **PoliticaOrdinamentoProcesso**. Un'istanza di **Processo** contiene tutte le informazioni riguardanti il *processo* al momento della sua creazione. **PCB** (*Process Control Block*) ha lo scopo di salvare, per ogni *processo*, i suoi parametri durante l'evoluzione della *simulazione*. **PoliticaOrdinamentoProcesso** sarà poi realizzata come interfaccia: ogni sua implementazione rappresenterà una delle politiche di schedulazione dei *processi* specificate in [AR], paragrafo 5, tabella A7.

La gestione della memoria rappresenta l'obiettivo centrale del nostro progetto. Le classi coinvolte sono: **RAM**, **SWAP**, **PoliticaOrdinamentoMemoria**. **RAM** rappresenta la memoria principale usata dai *processi* da simulare. **SWAP** invece vuole simulare la memoria virtuale su disco fisso. Entrambe sono concretizzazioni della classe astratta **Memoria**, che ne contiene i tratti comuni. Inoltre entrambe sono costituite da un insieme di implementazioni concrete dell'interfaccia **FrameMemoria**, che sarà illustrata in seguito. **PoliticaRimpiazzoMemoria** è un'interfaccia che sarà implementata dalle diverse politiche di rimpiazzo delle pagine, illustrate in [AR] par. 5 tabella A3, o dalle diverse politiche di rimpiazzo dei segmenti, illustrate in [AR] par. 5 tabella A4. Saranno le diverse implementazioni di questa interfaccia che si occuperanno di modificare lo stato di **RAM** e **SWAP**.

Per rappresentare le diverse modalità di organizzare la memoria, usiamo un'interfaccia **FrameMemoria**. L'implementazione concreta di questa interfaccia sarà operata dalle classi **Pagina**, **Segmento** e **SegmentoPaginato**. In tutte queste classi, come attributo indispensabile, sarà presente un identificativo univoco per distinguere le varie locazioni di memoria le une dalle altre.

Simulazione si preoccuperà di collezionare e salvare l'evoluzione della simulazione. Questo sarà realizzato tramite una collezione ordinata di istanze della classe **Istante**.

Player rappresenta la classe incaricata di scorrere la collezione ordinata di istanze della classe **Istante** contenuta in **Simulazione**. Dovrà offrire tutte le funzionalità previste in [AR], 3.3.3.

ConfigurazioneIniziale è la classe che racchiude al suo interno tutti i parametri che l'utente, all'inizio della *simulazione*, inserirà per configurare l'ambiente. I parametri sono illustrati in [AR], par. 5, tabelle A1-A2-A3-A4-A7. Questi dati potranno essere salvati e caricati da file, tramite la classe **GestioneFile**.

Gui, come suggerisce il nome, rappresenta la parte grafica del progetto. Deve permettere all'utente di interfacciarsi con la parte logica, offrendo tutte le funzionalità di quest'ultima. Dovrà essere il più possibile user-friendly, in modo da permettere una rapida configurazione della *simulazione* soprattutto pensando all'utilizzo didattico durante le lezioni. Inoltre sarà adibita a mostrare a video l'evoluzione della *simulazione* e le statistiche, in maniera esaustiva e immediata.

3 Descrizione package

Di seguito viene rappresentato il diagramma delle classi raffigurante la suddivisione in package delle classi rappresentate in Figura 2.1. Questa organizzazione permette una suddivisione logica dei vari componenti del sistema.

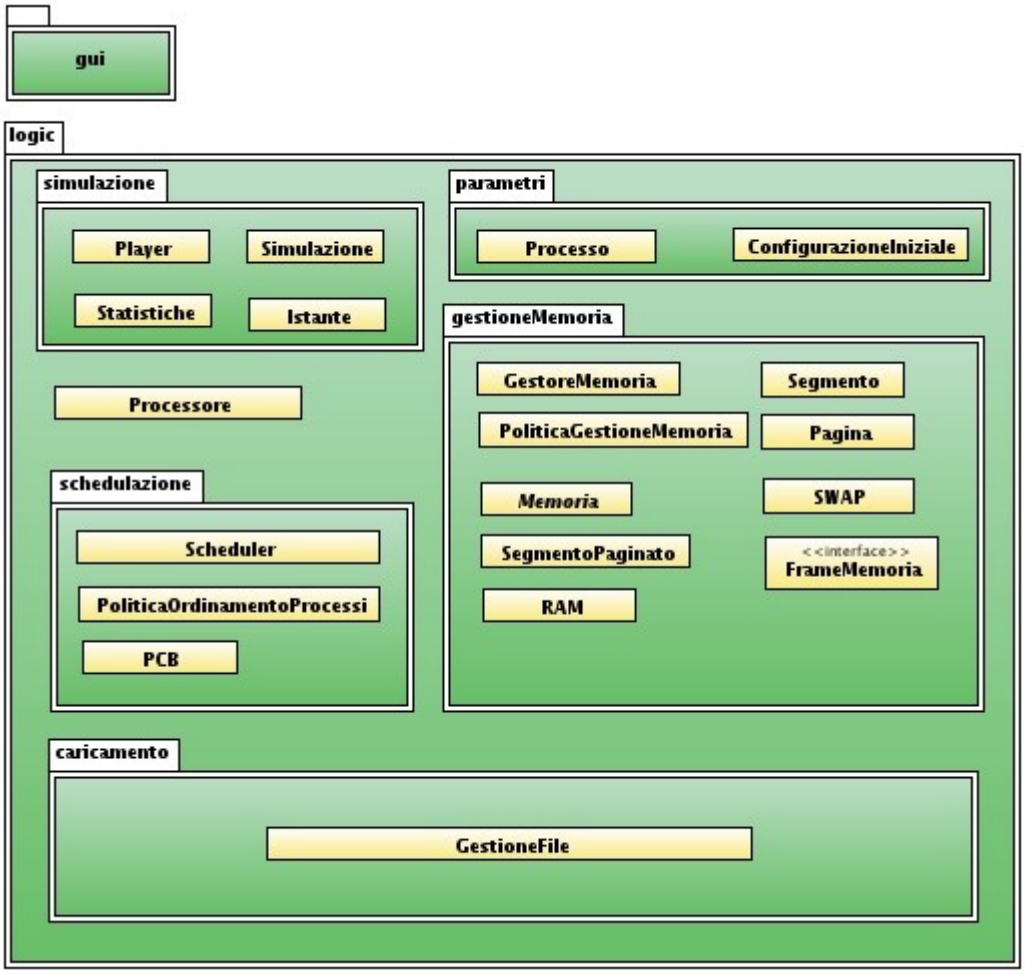

Figura 3.1

4 Stime di fattibilità e bisogno di risorse

Per l'impegno ed il tempo richiesti alla realizzazione, fare riferimento a [PP]. Per quanto riguarda le risorse hardware, umane e competenze individuali, fare riferimento a [SF].

Il progetto verrà realizzato utilizzando il linguaggio *Java* per i le seguenti motivazioni:

- tutti i componenti hanno familiarità con questo linguaggio di programmazione;

- questo linguaggio permette la portabilità del programma da una piattaforma ad un'altra, requisito implicito evidenziato in [AR] con il codice RNO08. La portabilità è stata messa in rilievo rispetto a l'ottimizzazione del codice che si può ottenere con altri linguaggi di programmazione, come ad esempio C;
- Java permette il salvataggio tramite serializzazione di oggetti, rendendo semplice la gestione dell'I/O;
- per quanto riguarda l'implementazione della parte grafica, Java mette già a disposizione librerie grafiche adatte al nostro scopo.

5 Tracciamento componenti – requisiti

Componente	Requisiti
gui	RFO11, RFO12, RFO13, RFO14, RFO15, RFO19, RFO21, RFO23, RFD02, RNO05, RNP01, RNO09, RNO10, RNO11
simulazione	RFO18, RFO19, RFO20, RFO23, RFD03
parametri	RFO03, RFO04, RFO05, RFO06, RFO07, RFO08, RFO09, RFO10, RFO17, RFO22, RFO24, RFD02, RNO06
gestioneMemoria	RFO01, RFO02, RFD01,RFP03
schedulazione	RFP01,RFP03
caricamento	RFO16, RFP02