Verbale Esterno 2016-03-30

Informazioni sul documento	
Versione	1.0.0
Redazione	Marco Boseggia
Verifica	Michael Munaro
Approvazione	Giacomo Vanin
Uso	Interno
Lista di Distribuzione	ScalateKids
	Prof. Tullio Vardanega
	Prof. Riccardo Cardin

Diario delle modifiche

Versione	Autore	Ruolo	Data	Descrizione
1.0.0	Giacomo Vanin	Responsabile	2016-03-30	Validazione documento
0.1.1	Michael Munaro	Verificatore	2016-03-30	Verifica documento
0.1.0	Marco Boseggia	Progettista	2016-03-30	Prima stesura documento
0.0.1	Marco Boseggia	Progettista	2016-03-30	Creazione scheletro del documento

Indice

1 Informazioni sulla riunione

- **Data**: 2016-03-30
- **Luogo**: Torre Archimede
- **Orario d'inizio**: 13:15
- **Durata**: 30'
- **Partecipanti interni**: *ScalateKids*
  - Andrea Giacomo Baldan
  - Alberto De Agostini
  - Marco Boseggia
  - Michael Munaro
  - Francesco Agostini
  - Giacomo Vanin
  - Davide Trevisan

2 Domande e risposte

In questo capitolo sono elencate le domande fatte dal gruppo *ScalateKids* in grassetto e le risposte del proponente *Riccardo Cardin* in corsivo.

2.1 Abbiamo trovato un componente (spring shell) che potremmo riutilizzare per la realizzazione della nostra CLI$_G$. Cosa ne pensa a riguardo?

L'idea è buona poiché permetterebbe un facile dialogo tra CLI$_G$ e driver$_G$ in quanto entrambi utilizzano JVM$_G$. Tuttavia il suo utilizzo porta ad avere molte dipendenze dovute allo spring framework$_G$ nel progetto, quindi cercate di valutare bene gli aspetti positivi e negativi di tale scelta e vedete se è il caso di intraprendere questa strada.

2.2 Abbiamo pensato ad una possibile divisione delle componenti ad alto livello secondo il pattern MVC$_G$, in particolare la CLI$_G$ sarebbe la nostra View$_G$, il Driver$_G$ sarebbe il nostro Controller$_G$ e il package sistema sarebbe il Model$_G$, secondo lei è una scelta corretta?

Non credo sia una buona idea, le componenti che dite sono componenti separate tra loro, non cercherei di forzare un collegamento tra loro dal punto di vista architetturale.

2.3 Avendo più nodi del cluster$_G$ avevamo pensato o di scegliere dei nodi designati per salvare i dati, oppure far si che ogni nodo salvi un pezzo di database e all'accensione il ripopolamento avverrebbe tramite gli stessi nodi

Credo sia opportuno ipotizzare che al riavvio del database distribuito tra diversi nodi tutti questi debbano essere presenti nuovamente altrimenti saranno necessarie operazioni da sistemista. Quindi opterei per la seconda scelta.

2.4 Se potessimo usare i Persistence Actor$_G$ potremmo far spegnere nodi inutilizzati da molto tempo con previo snapshot su disco, questo snapshot poi sarà utilizzato per "riaccendere" l'attore se richiesto

State attenti con una feature come questa. Se ad esempio decideste di mettere in sleep attori dopo 5 secondi di inattività, qualora la vostra applicazione avesse dei momenti di poco carico potrebbe andare in sleep una gran parte del database se non tutto, andando a rallentare le prossime operazioni. In più

ogni attore che va in sleep andrà a scrivere un file su disco e se non prevedete un modo intelligente di cancellarli potrebbero accumularsi in fretta.

2.5 Abbiamo deciso di avere un numero di attori main configurabile per ogni nodo; ogni volta che un main decide di creare uno storefinder per la creazione di una nuova collection$_G$ questo deve informare gli altri attori main per far si che ognuno abbia sempre una tabella aggiornata oppure ogni main ha il suo pezzetto di mappa e se la richiesta che riceve non può essere processata da lui allora manda il messaggio agli altri attori main

Le due scelte hanno i suoi pro e i suoi contro, la prima è più dispendiosa in fatto di memoria mentre la seconda è un po' più lenta. Secondo me entrambe le scelte vanno bene, cercate di pensare a cosa conviene a voi e cercate di far si che una scelta non escluda l'altra.

2.6 Come possiamo rappresentare le relazioni tra attori in UML$_G$?

Nei diagrammi delle classi e dei package dovete solo pensare alle dipendenze e non ai messaggi. I messaggi andranno rappresentati semmai nei diagrammi di sequenza.

3 Decisioni prese

- Cercare un alternativa a Spring Shell;
- Non suddividere con pattern MVC le varie componenti ;
- Dividere la persistenza dei dati tra i vari nodi del cluster;
- Valutare bene se mandare in sleep gli attori che non fanno operazioni per un certo tempo;
- Valutare se conviene tenere aggiornate le mappe di tutti gli storefinder su tutti i main oppure ogni main tiene un pezzo di mappa e instrada le richieste agli altri quando non può processare la richiesta che riceve;
- I messaggi tra attori andranno rappresentati solo nei diagrammi di sequenza