IronWorks

Utility per la Costruzione di Software Robusto

Verbale esterno 2018/03/12

Versione	-
Redattori	Mirko Gibin
Verificatori	Stefano Nordio
	Antonio Moz
Responsabili	Francesco Sacchetto
Uso	Esterno
Distribuzione	Gruppo Swear on Code
	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Gregorio Piccoli, Zucchetti S.p.A.

Descrizione

Verbale di incontro per il progetto **IronWorks** tra i componenti del gruppo Swear on Code ed il proponente Gregorio Piccoli, Zucchetti S.p.A.

1 Informazioni Generali

1.1 Informazioni Incontro

- **Data:** 2018/03/12;
- **Luogo:** Padova, via Cittadella 7;
- **Ora d'inizio:** 14:30;
- **Ora di fine:** 16:30;
- **Partecipanti del gruppo:**
  - Antonio Moz;
  - Francesco Sacchetto;
  - Mirko Gibin.
- **Partecipanti esterni:**
  - Gregorio Piccoli, Zucchetti S.p.A.

1.2 Ambiguità

Al fine di dipanare qualsiasi dubbio o ambiguità relativa al linguaggio impiegato nel documento viene fornito il *Glossario v1.0.0*, documento contenente la definizione di tutti i termini scritti in corsivo e marcati con una 'G' pedice.

1.3 Riferimenti

1.3.1 Informativi

- *Glossario v1.0.0*.

2 Ordine del Giorno

Chiarimenti inerenti il progetto **IronWorks** e la sua implementazione. Il *proponente*$_G$ fornisce idee e risposte circa le seguenti domande:

- È possibile utilizzare *Slack*$_G$ per una comunicazione più immediata?
- Che cosa si intende con "Definire le regole per sviluppare manualmente il codice degli oggetti *boundary*$_G$ e *control*$_G$ che possano utilizzare le classi generate in automatico"?
- Quali tecnologie consiglia di utilizzare?
- In cosa consiste il codice di manutenzione delle tabelle nel *database*$_G$?
- Come gestiamo requisiti obbligatori e requisiti opzionali?
- Il glossario è una cosa interna o necessaria anche a voi?
- Cos'è in sostanza il formalismo di *Warnier-Orr*$_G$?
- Che cosa si intende con "Specificare delle regole di controllo e calcolo delle classi persistenti generando il codice di verifica"?
- Perché includere i programmi di *data entry*$_G$?

2.1 È possibile utilizzare *Slack*$_G$ per una comunicazione più immediata?

Il canale di comunicazione principale è l'indirizzo email, inoltre ci si incontra fisicamente almeno un paio di volte. *Slack*$_G$ non è necessario in quanto non ci sono scadenze dell'ultimo minuto.

2.2 Che cosa si intende con "Definire le regole per sviluppare manualmente il codice degli oggetti *boundary*$_G$ e *control*$_G$ che possano utilizzare le classi generate in automatico"?

Ci sono 3 macro classi: *boundary*$_G$, *control*$_G$, *entità*$_G$.
Ci sono 2 tipi di entità: supportate da un *file*$_G$ *JSon*$_G$ o *XML*$_G$ come entità *singleton*$_G$ (alcune operazioni potrebbero non essere disponibili), ed entità archivi su cui voglio fare operazioni *CRUD*$_G$. A queste attività si aggiunge una funzionalità di ricerca dalla quale potrei ottenere una classe o nessuna se questa non esiste. Se esiste potrei fare altre operazioni come una mappatura dell'oggetto, quindi l'*utente*$_G$ dovrebbe già avere la possibilità di effettuare delle operazioni. Ci sono 2 tipi di interfacce: grafiche o *machine-to-machine*$_G$. Le problematicità sono relative alle credenziali, sicurezza, *stateless*$_G$, *connectionless*$_G$; ci sono già delle informazioni utili che possono essere create in modo automatico.
Il *controller*$_G$ è il programma più generico possibile, ma gli vengono passati dei dati che devono essere controllati, devo dare delle informazioni al *controller*$_G$ sul dato che passo. Il lato *client*$_G$ è un punto di massima pericolosità lato sicurezza, i dati potrebbero essere manipolati, non posso affidarmi unicamente al controllo dell'interfaccia.

L'obiettivo sarebbe quello di estrarre i controlli lato interfaccia e metterli in uno stadio intermedio tra interfaccia e *controller*$_G$, in modo da evitare manipolazioni lato *client*$_G$ ed effettuando controlli lato *server*$_G$. Sono metodi obbligatori da implementare, che sono utili proprio per questa validazione.
Da quello che vedo nel grafico faccio saltare fuori delle classi con dei metodi virtuali. L'*utente*$_G$ deve per forza implementare determinati metodi (ad esempio un *getter*$_G$ o un metodo per istanziarlo) se vuole ottenere delle informazioni. Quindi è una sorta di regola da seguire, obbligo l'*utente*$_G$ alla creazione. Aggiungo dettagli tra *casi d'uso*$_G$ e *diagramma delle classi*$_G$, aggiungo continuità tra analisi e progettazione dove prima c'era un salto. Aggiungendo anche la parte della valutazione, aumentandone il collegamento.

2.3 Quali tecnologie consiglia di utilizzare?

JointJS$_G$ con *Backbone.js*$_G$ come *framework*$_G$ per scrivere, soprattutto per quanto concerne il disegno del diagramma. Ce ne sono altre ma sono consigliate *Backbone.js*$_G$, *Angular*$_G$ e *React*$_G$. Cercare, trovare, valutare le tecnologie, senza ovviamente perdere troppo tempo in *librerie*$_G$ esoteriche. Lo scopo è anche esplorare dove si può arrivare.

2.4 In cosa consiste il codice di manutenzione delle tabelle nel *database*$_G$?

Identificare la chiave primaria, il tracciato *record*$_G$ (nel caso di una procedura di una tabella), ed in caso di *JSon*$_G$ bisognerà capire come fare. È importante capire la struttura, non serve andare nel dettaglio, tanto le procedure sono quelle e si possono costruire automaticamente. Ad esempio con *Hibernate*$_G$ se creo *XML*$_G$ si arrangia lui senza che io debba creare tutta la gestione. Si potrebbe sostituire quel punto obbligatorio con il punto opzionale di *Hibernate*$_G$.

2.5 Come gestiamo requisiti obbligatori e requisiti opzionali?

In definitiva il requisito obbligatorio è quello di avere il programma che crea il disegno che poi dovrà essere salvato. I requisiti opzionali ve li gestite per conto vostro tenendo presenti 4 temi principali:

- *Hibernate*$_G$;
- Gestione eccezioni (percorso positivo e percorso di quando ci sono errori);
- Controllo sul canale (tracciato con una serie di check);
- Divisione dei *controller*$_G$ (*splitter*$_G$, elementi *array*$_G$, unione dei flussi).

L'idea è quella di esplorare, in quanto essendo territori nuovi ancora non si sa il risultato. Provare ma non riuscirci (ovviamente giustificando l'infattibilità) è comunque una cosa accettata, in quanto è una strada che non funziona o da riesplorare.
Ricordate il termine "seemliness", mantenere il tutto aggiornato, fare in modo che ci sia continuità. Se non si riesce ad aggiornarlo tanto vale buttarlo via, se non c'è correlazione tra schema e codice non c'è una mappa da seguire, o addirittura questa è sbagliata. Si vuole avere un prototipo iniziale sul quale aggiungere dettagli con continuità volta per volta, senza ad un certo punto uscire di strada e buttare via tutto.

2.6 Il glossario è una cosa interna o necessaria anche a voi?

Analizzando i progetti mediamente un glossario salta fuori, quindi è ritenuto necessario.

2.7 Cos'è in sostanza il formalismo di *Warnier-Orr*$_G$?

Warnier-Orr$_G$ è in sostanza *JSon*$_G$ degli anni 50. In *JSon*$_G$ la molteplicità è in *array*$_G$, in *Warnier-Orr*$_G$ si utilizza '*'. È solo una strutturazione a livelli, potrebbe essere un buon modo per descrivere le cose che si passano. Il programmatore *utente*$_G$ dovrebbe usarlo per indicare cosa richiede, in forma di struttura, cosa ha messo dentro. *JSon*$_G$ è da programmazione, *Warnier-Orr*$_G$ è più dichiarativo. È un buon modo per descrivere il documento che viene passato tra le due parti. Con *Warnier-Orr*$_G$ faccio il tracciato, potrebbe diventare *JSon*$_G$, *XML Schema*$_G$ o qualsiasi altra cosa che descrive.

2.8 Che cosa si intende con "Specificare delle regole di controllo e calcolo delle classi persistenti generando il codice di verifica"?

Quando arrivo all'entità, questa riceve dei dati. Potrebbe essere interessante descrivere il tracciato *record*$_G$. Il tracciato *record*$_G$ sono dei dati, il salvataggio dei dati e basta. Potrebbe risultare utile fare dei controlli prima di salvarli. Le regole di calcolo sono un punto di passaggio tra ricevere i dati e passarli.
Vi sono 3 casi principali:

- Ricevo, controllo, rigetto in quanto non completo;
- Ricevo, controllo, aggiungo dei dati che mancano (calculator);
- Ricevo, controllo e lo proietto al prossimo passo.

2.9 Perché includere i programmi di *data entry*$_G$?

Possono essere un aiuto per fare i test, programmi che abbiano dei campi dati da inserire per verificare, un prototipo di *data entry*$_G$. Con un *data entry*$_G$ prototipale il check lo fa il pezzo di passaggio, il *controller*$_G$ lo passa all'entità che lavora direttamente con *Hibernate*$_G$.

3 Tracciamento decisioni

ID	Decisione
VE_20180312.1	Utilizzo di swearoncode@gmail.com con Gregorio Piccoli, Zucchetti S.p.A.
VE_20180312.2	Utilizzo di *JointJS*$_G$ con *Backbone.js*$_G$ o altro *framework*$_G$
VE_20180312.3	Il glossario è un documento necessariamente esterno

Tabella 1: Decisioni riunione esterna del 2018/03/12