HAPPY CODE INC

BR-jsys *business rules* per sistemi gestionali in architettura J2EE

Specifica Tecnica

Versione 1.0 - 16 febbraio 2008

Capitolato: "BR-jsys"

Data creazione:	18/11/2007
Versione:	1.0
Stato del documento:	formale, esterno
Redazione:	Michele Bortolato
Revisione:	Alessia Trivellato, Elena Trivellato
Approvazione:	Mattia Meroi

Lista di distribuzione

HappyCode inc	Gruppo di lavoro
Tullio Vardanega, Renato Conte	Committente
Zucchetti S.r.l	Azienda proponente

Diario delle modifiche

Versione	Data rilascio	Descrizione
1.0	16/02/2008	Inserimento diagramma di collaborazione e diagramma di sequenza.
0.9	15/02/2008	Correzione grammaticale
0.8	12/02/2008	Modifiche ai diagrammi di attività e alle loro descrizioni
0.7	08/02/2008	Aggiunta dei diagrammi di attività e rispettive descrizioni
0.6	06/02/2008	Aggiunta dei diagrammi delle componenti e loro descrizione
0.5	05/02/2008	Aggiunta del nome del file nel modello di documento
0.4	22/01/2008	Modifica al layout dei documenti
0.3	22/12/2007	Aggiornamento requisiti
0.2	21/12/2007	Documento sottoposto a revisionamento automatico
0.1	18/12/2007	Stesura preliminare del documento

Indice

Capitolo 1

Introduzione

1.1 Scopo del documento

Il presente documento descriverà il sistema software "BR-jsys" dal punto di vista architetturale e da noi implementato secondo le esigenze identificate nel documento "Analisi dei Requisiti". Attraverso l'uso di diagrammi UML, nella fattispecie con alcuni diagrammi delle classi, identificheremo alcune componenti principali nel quale scomporre il sistema. Procederemo inoltre con un approccio di tipo "Bottom-up" nelle procedure di verifica delle componenti. Questo documento fornirà quindi una visione più dettagliata delle componenti da realizzare; verrà definito il contesto d'uso del sistema e fornita la decomposizione di questo in componenti principali.

1.2 Scopo del prodotto

Per lo scopo del prodotto si faccia riferimento al documento "Analisi dei Requisiti".

1.3 Glossario

Viene fornito come documento esterno chiamato Glossario.1.8.pdf .

1.4 Riferimenti

- Capitolato d'appalto concorso per sistema "BR-jsys";
- "Analisi dei Requisiti";
- "Norme di Progetto";
- "Piano di Qualifica";

- “Piano di Progetto”;
- “Ingegneria del software” 8a edizione - Ian Sommerville;
- “The Definitive ANTLR Reference”;
- Incontro con il proponente “Incontro2008-01-17.pdf”;
- Incontro con il proponente “Incontro2008-02-05.pdf”.

Capitolo 2

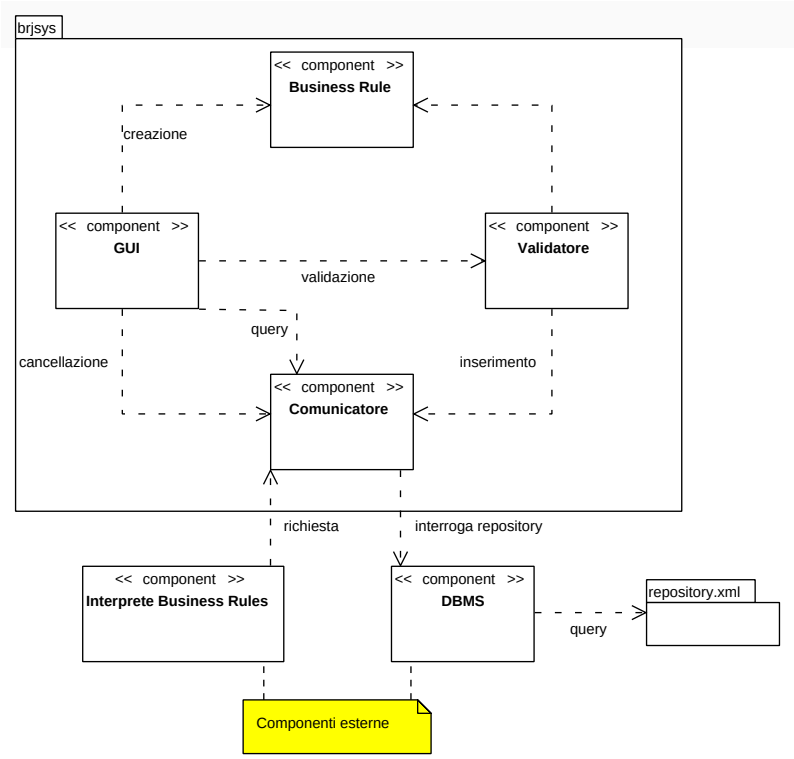
Definizione del prodotto

2.1 Metodo e formalismo di specifica

La progettazione del prodotto è stata attuata con l'uso di diagrammi UML 2.1, in quanto linguaggio internazionalmente riconosciuto e standardizzato. Il software di cui l'azienda ha fatto uso per la loro creazione è *Poseidon for UML Professional edition 6.0.2 (Evaluation Copy)*. L'utilizzo di tale software deriva dalla completezza di strumenti di cui è fornito, i quali soddisfano in pieno le nostre necessità.

2.2 Presentazione dell'architettura generale del sistema e identificazione dei componenti architetturali di alto livello

Lo schema generale del prodotto è specificato nell'immagine sottostante.



Il prodotto è stato suddiviso in quattro macrocomponenti:

1. **GUI**: Fornisce all'utente un interfaccia grafica minimale, consentendogli di effettuare operazioni di cancellazione e interrogazione del repository in maniera user-friendly.

2. **Business Rule**: Rappresenta la business rule che l'utente dichiara e che deve essere passata al validatore per la compilazione.

3. **Validatore**: Accetta in input una business rule, la valida e la inserisce, se scritta correttamente, nel repository.

4. **Comunicatore**: Si connette al DBMS esterno e comunica con esso tramite i formalismi del linguaggio XQuery. Sarà appunto tramite Query che il DBMS effettuerà operazioni di inserimento, cancellazione o ricerca nel repository, qualora un componente lo richieda.

Questa suddivisione, per quanto minimale, consente di individuare le componenti che verranno poi ampliamente descritte nel successivo capitolo.

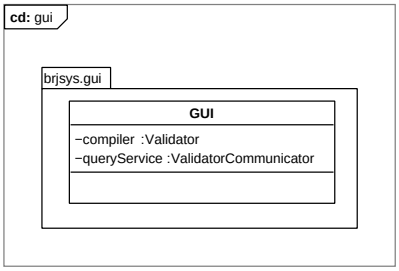
Capitolo 3

Descrizione dei singoli componenti

Analizzeremo qui le quattro macrocomponenti elencate in precedenza e ne daremo una descrizione più dettagliata.

3.1 GUI

3.1.1 Diagramma delle classi



3.1.2 Gui

Tipo, obiettivo e funzione del componente

Questa componente, realizzata tramite una singola classe java, fornisce all'utente un'interfaccia minimale che gli consente di effettuare operazioni di cancellazione e querying sul repository. Nel caso di esecuzione di una query definita dall'utente, verranno fornite anche informazioni relative ai tempi d'esecuzione.

Relazioni d'uso di altre componenti

Questa componente utilizza:

- BusinessRule per dichiarare una nuova business rule da spedire al validatore;
- Validator per effettuare la validazione di una business rule;
- GUICommunicator che verrà trattata successivamente.

Interfacce con e relazioni di uso da altre componenti

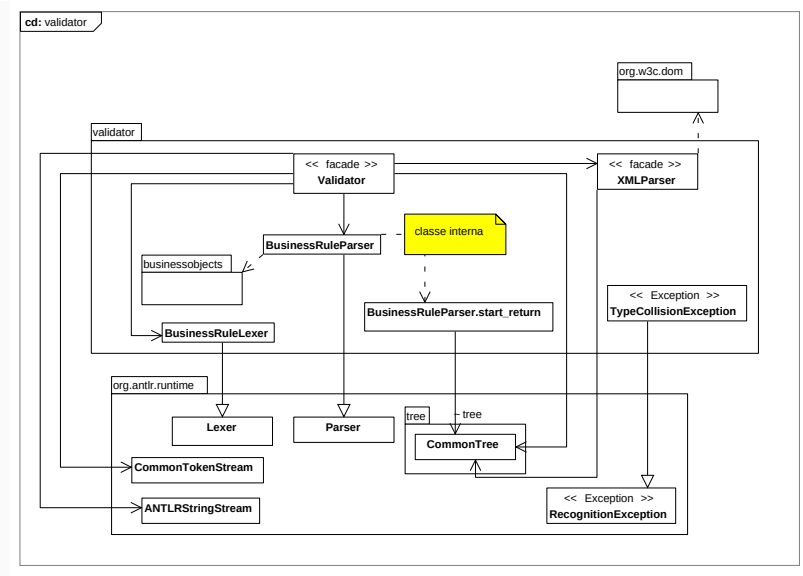
Nessuna.

Attività svolte e dati trattati

La componente Gui possiede i campi dato "GUICommunicator" e "Validator" che verranno utilizzati per fornire funzionalità di interfacciamento. Tali funzionalità verranno trattate in seguito.

3.2 Validatore

3.2.1 Diagramma delle classi



3.2.2 Validator

Tipo, obiettivo e funzione del componente

Questa componente effettua la validazione di una business rule, dando la possibilità all'utilizzatore di evitare le singole operazioni di compilazione. Ha il ruolo di *façade*: fornisce un'interfaccia unificata che riesce a gestire in

maniera semplice ed immediata le operazioni di compilazione e inserimento che altrimenti dovremmo riportare direttamente dove la componente GUI lo richiedesse.

Relazioni d'uso di altre componenti

Validator usa le componenti BusinessRuleParser, BusinessRuleLexer, XMLParser e ValidatorCommunicator. Quest'ultime verranno ampliamente descritte successivamente.

Interfacce con e relazioni di uso da altre componenti

La componente Validator è in relazione con la componente GUI, al fine di rendere possibile la validazione.

Attività svolte e dati trattati

Validator contiene solo il metodo validate(). Tale metodo effettua la validazione della business rule.

3.2.3 BusinessRuleLexer

Tipo, obiettivo e funzione del componente

Questa componente, derivata da org.antlr.runtime.Lexer, non è altro che una classe wrapper per la stringa che rappresenta la business rule. Essa aggiunge funzionalità alla stringa, necessarie per il successivo parsing.

Relazioni d'uso di altre componenti

Nessuna.

Interfacce con e relazioni di uso da altre componenti

La componente BusinessRuleParser necessita della componente BusinessRuleLexer. Quest'ultima verrà approfondita in seguito.

Attività svolte e dati trattati

BusinessRuleLexer mette a disposizione vari metodi per la lettura del testo della business rule, nonchè per la gestione di eventuali eccezioni avvenute in fase di validazione.
Nota:*Questa classe è stata creata utilizzando uno strumento automatico per la generazione di parser data in input la specifica di una grammatica.*

3.2.4 BusinessRuleParser

Tipo, obiettivo e funzione del componente

Questa componente, derivata da org.antlr.runtime.Parser, effettua il parsing della stringa che rappresenta la business rule. Effettua quindi il controllo sintattico della regola e il controllo semantico facendo un controllo sui tipi dei dati (siano essi costanti oppure campi dati di business objects). Mentre effettua la validazione, BusinessRuleParser genera l'albero di parsing secondo le specifiche presenti nel controllo semantico. È in grado infine di dare informazioni accurate riguardo eventuali errori in fase di validazione.

Relazioni d'uso di altre componenti

La componente BusinessRuleParser necessita di un TokenStream, fornitogli indirettamente da BusinessRuleLexer. Deve riferirsi poi alla componente BusinessObjects per effettuare il controllo sui tipi per il business object associato. Per trattare gli errori in fase di validazione, ha bisogno infine della componente TypeCollisionException.

Interfacce con e relazioni di uso da altre componenti

BusinessRuleParser viene messa in relazione con XMLParser che verrà trattata successivamente. È utilizzata inoltre dalla componente Validator.

Attività svolte e dati trattati

BusinessRuleParser mette a disposizione vari metodi per effettuare il parsing e i test semantici. Dispone inoltre di numerosi campi dato per la ricognizione dei token.
Nota:*Questa classe è stata creata utilizzando uno strumento automatico per la generazione di parser data in input la specifica di una grammatica.*

3.2.5 TypeCollisionException

Tipo, obiettivo e funzione del componente

Questa componente, derivata dalla classe org.antlr.runtime.RecognitionException, permette di ricavare informazioni sugli eventuali errori avvenuti in fase di parsing della business rule. In definitiva aggiunge alla sua superclasse la possibilità di riportare informazioni su errori di tipo.

Relazioni d'uso di altre componenti

Nessuna.

Interfacce con e relazioni di uso da altre componenti

La componente TypeCollisionException è utilizzata da BusinessRuleParser per sollevare eccezioni derivanti da errori di tipo.

Attività svolte e dati trattati

Viene ridefinito il metodo printStackTrace() e messo a disposizione un costruttore per avere informazioni sui tipi che si sono rivelati incompatibili.

3.2.6 XMLParser

Tipo, obiettivo e funzione del componente

La componente XMLParser si occupa di effettuare la conversione dell'albero sintattico prodotto dalla componente BusinessRuleParserin un elemento XML rappresentante la business rule. L'elemento XML risultante conterrà:

- il nome della business rule, che dovrà essere univoco;
- il nome del business object associato alla regola;
- la struttura dell'AST della business rule, scritta secondo la metodologia elemento-attributo tipica di XML;
- la struttura dell'AST della business rule, scritta linearmente secondo la notazione prefissa rappresentata da un singolo attributo XML;
- la business rule effettivamente digitata dall'utente;
- eventuali commenti associati dall'utente alla business rule.

XMLParser ha in questo contesto il ruolo del design pattern *façade*. La creazione di un elemento XML tramite il package java *org.w3c.dom* necessita di operazioni sequenziali che coinvolgono numerose classi e numerosi metodi. Implementare tutte queste procedure direttamente nella classe Validator renderebbe il codice meno leggibile e meno manipolabile.

Relazioni d'uso di altre componenti

XMLParser ha bisogno dell'AST prodotto da BusinessRuleParser, nonchè necessita della componente business rule per ricavare le informazioni da inserire nell'elemento XML.

Interfacce con e relazioni di uso da altre componenti

La componente ValidatorCommunicator ha bisogno dell'elemento XML prodotto da XMLParser per avviare la procedura di inserimento nel repository.

Attività svolte e dati trattati

XMLParser contiene le operazioni per scorrere l'AST ed effettuare la traduzione in XML. Dispone inoltre di una tabella Hash statica che serve per associare i valori numerici che il parser usa per identificare i token ai nomi dei tokens definiti dall'utente.

3.2.7 businessobjects

Tipo, obiettivo e funzione del componente

La componente businessobjects si occupa di offrire un namespace comune a tutti i business objects che durante la validazione di una business rule vengono interpellati richiedendo accesso ai suoi campi o sottocampi.

Relazioni d'uso di altre componenti

Nessuna.

Interfacce con e relazioni di uso da altre componenti

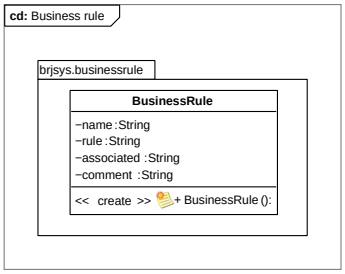
BusinessRuleParsernecessita di questa componente per effettuare il controllo dei tipi qualora fosse necessario.

Attività svolte e dati trattati

Il componente businessbjects offre le classi java che rappresentano i business objects.

3.3 Business Rule

3.3.1 Diagramma delle classi



3.3.2 BusinessRule

Tipo, obiettivo e funzione del componente

La componente BusinessRule rappresenta la business rule che l'utente inserisce e vuole validare.

Relazioni d'uso di altre componenti

Nessuna.

Interfacce con e relazioni di uso da altre componenti

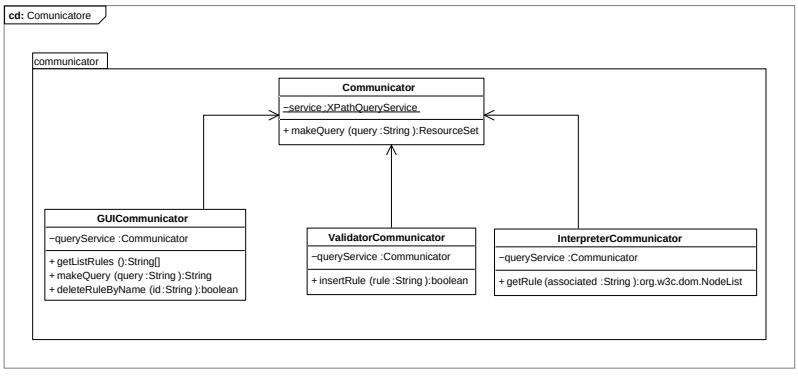
La componente GUI utilizza BusinessRule nel caso in cui l'utente voglia inserire una nuova business rule. La componente Validator effettua la validazione di un'istanza di BusinessRule, tramite la chiamata del suo metodo validate().

Attività svolte e dati trattati

Business Rule contiene i campi dato Stringa per rappesentare una business rule (name, associatedObject, rule e comment, dove quest'ultimo può anche non essere istanziato). Vengono messi inoltre a disposizione il costruttore e il metodo ridefinito toString().

3.4 Comunicatore

3.4.1 Diagramma delle classi



3.4.2 Communicator

Tipo, obiettivo e funzione del componente

La componente Communicator fornisce alle componenti che la utilizzano la possibilità di effettuare query di qualsiasi tipo al repository presente nel DBMS.

Relazioni d'uso di altre componenti

Communicator necessita di interagire con la componente esterna DBMS per interrogare il repository.

Interfacce con e relazioni di uso da altre componenti

Le componenti GUICommunicator, InterpreterCommunicator e Validator-Communicator necessitano di Communicator per potergli passare query specifiche. Le loro descrizioni verranno trattate successivamente.

Attività svolte e dati trattati

Quando viene istanziata per la prima volta, Communicator inizializza la connessione al DBMS tramite la definizione di una variabile statica. Consente successivamente di interrogare il repository direttamente, passandogli la stringa che rappresenta la query composta secondo le specifiche XQuery.

3.4.3 GUICommunicator

Tipo, obiettivo e funzione del componente

La componente GUICommunicator contiene metodi necessari per modellare query di cancellazione (tramite operazioni XQuery Update) e query di semplice interrogazione.

Relazioni d'uso di altre componenti

GUICommunicator utilizza la componente Communicator per accedere al repository ed interrogarlo.

Interfacce con e relazioni di uso da altre componenti

GUICommunicator viene messa in relazione con la componente GUI, dalla quale riceve richieste di cancellazione e di querying.

Attività svolte e dati trattati

La componente fornisce le seguenti operazioni:

- deleteRuleByName(): dato il nome di una regola, provvede ad eliminarla dal repository nel caso questa regola sia presente;
- makeQuery(): permette l'esecuzione di una semplice query purchè non implichi modifiche strutturali al repository;
- getListRules(): ritorna per ogni business rule, nome, business object associato e struttura.

3.4.4 InterpreterCommunicator

Tipo, obiettivo e funzione del componente

La componente InterpreterCommunicator contiene metodi necessari per rispondere a richieste di business rule da parte di un interprete esterno.

Relazioni d'uso di altre componenti

InterpreterCommunicator utilizza la componente Communicator per accedere al repository ed interrogarlo.

Interfacce con e relazioni di uso da altre componenti

InterpreterCommunicator viene messa in relazione con l'interprete esterno, dal quale riceve richieste di business rules associate ad un determinato business object.

Attività svolte e dati trattati

La componente InterpreterCommunicator offre il metodo getRule() per richiedere le business rule associate al business object.

3.4.5 ValidatorCommunicator

Tipo, obiettivo e funzione del componente

La componente ValidatorCommunicator contiene metodi necessari per effettuare l'inserimento di una business rule validata già tradotta in XML.

Relazioni d'uso di altre componenti

ValidatorCommunicator utilizza la componente Communicator per accedere al repository ed interrogarlo.

Interfacce con e relazioni di uso da altre componenti

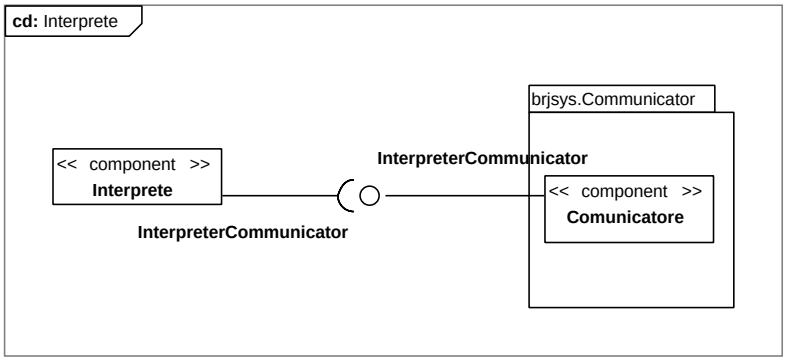
ValidatorCommunicator viene messa in relazione con la componente Validator per effettuare l'inserimento.

Attività svolte e dati trattati

La componente offre il metodo insertRule() per inserire la business rule nel repository. L'inserimento avverrà soltanto se la business rule ha un nome che non è ancora presente nel repository.

Capitolo 4

Componenti esterne



4.1 Interprete

Tipo, obiettivo e funzione del componente

Questo componente si incarica di eseguire le business rules associate ad un determinato business object. Per ottenere le business rules, deve poter interrogare il DBMS attraverso un opportuna interfaccia col sistema BR-jsys che risolverà per lui la richiesta.

Relazioni d'uso di altre componenti

La componente Interprete utilizza la componente InterpreterCommunicator per comunicare col DBMS.

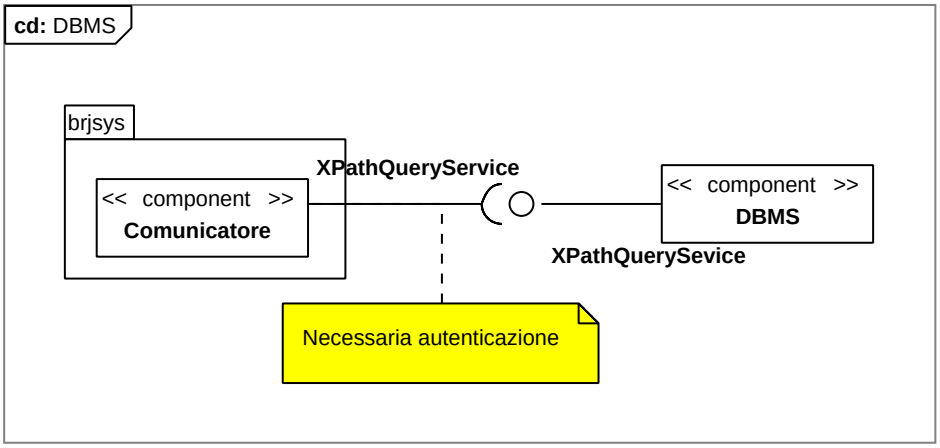
Interfacce con e relazioni di uso da altre componenti

Nessuna.

Attività svolte e dati trattati

Non siamo tenuti a considerare i dettagli riguardanti le operazioni specifiche della componente. L'unico vincolo posto è che comunichi con il DBMS tramite l'interfaccia InterpreterCommunicator.

4.2 DBMS



Tipo, obiettivo e funzione del componente

Questa componente serve a comunicare con il repository in modo efficiente. La comunicazione avverrà tramite linguaggio XQuery.

Relazioni d'uso di altre componenti

Nessuna.

Interfacce con e relazioni di uso da altre componenti

Previa una connessione corretta, il DBMS mette a disposizione un'interfaccia XPathQueryService con la quale interagire col repository presente all'interno del DBMS stesso.

Attività svolte e dati trattati

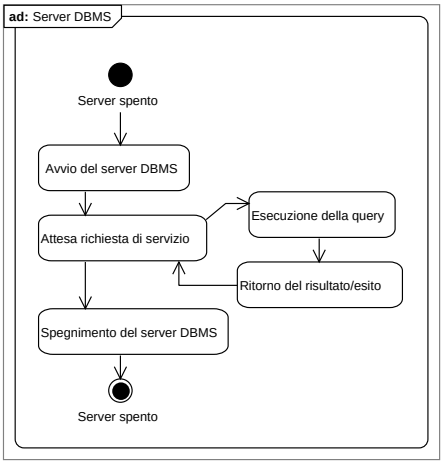
Una corretta autenticazione farà in modo che l'interfaccia di tipo XPathQueryService possa interagire col repository tramite il metodo query(). Quest'ultimo accetta come parametro una query impostata secondo le specifiche XQuery.

Capitolo 5

Diagrammi

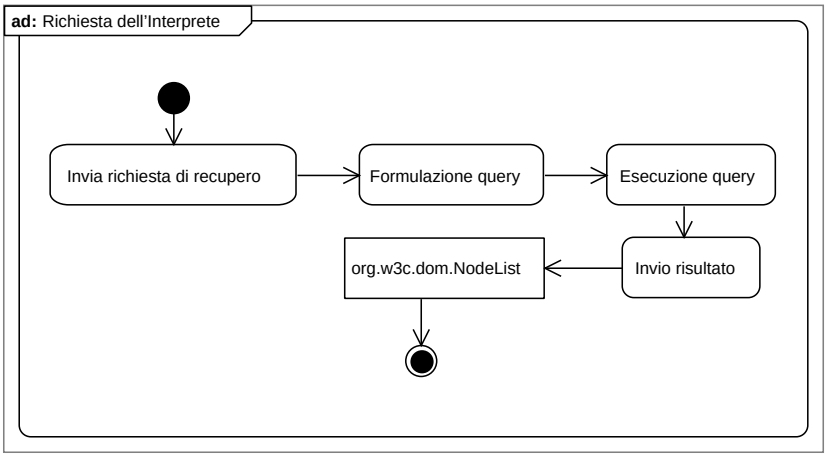
5.1 Diagrammi di attività

5.1.1 Server DBMS



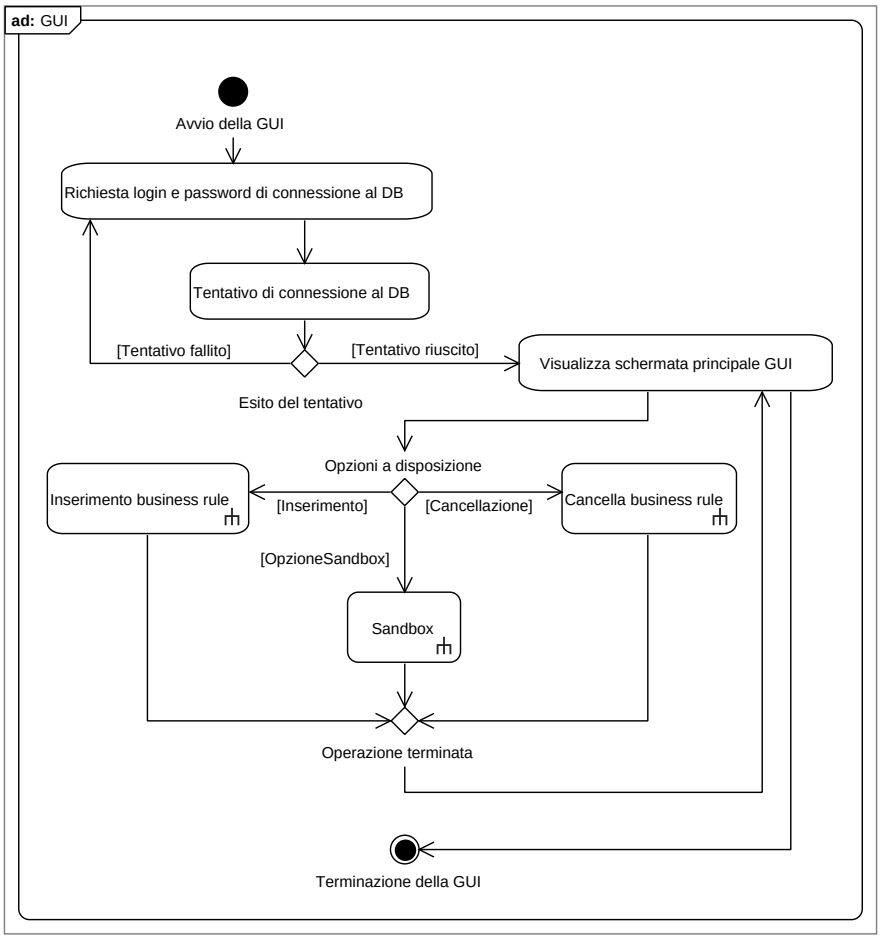
Il server DBMS, con il quale interagisce il prodotto "BR-jsys", viene avviato. Una volta completata la procedura di avvio, entra in uno stato di attesa di richieste da parte della GUI e dell'interprete. Ricevuta una richiesta, la relativa query viene inviata al DBMS e il risultato/esito della query viene ritornato all'utente. Il server torna quindi in uno stato di attesa finchè eventualmente viene spento.

5.1.2 Richiesta dell'interprete



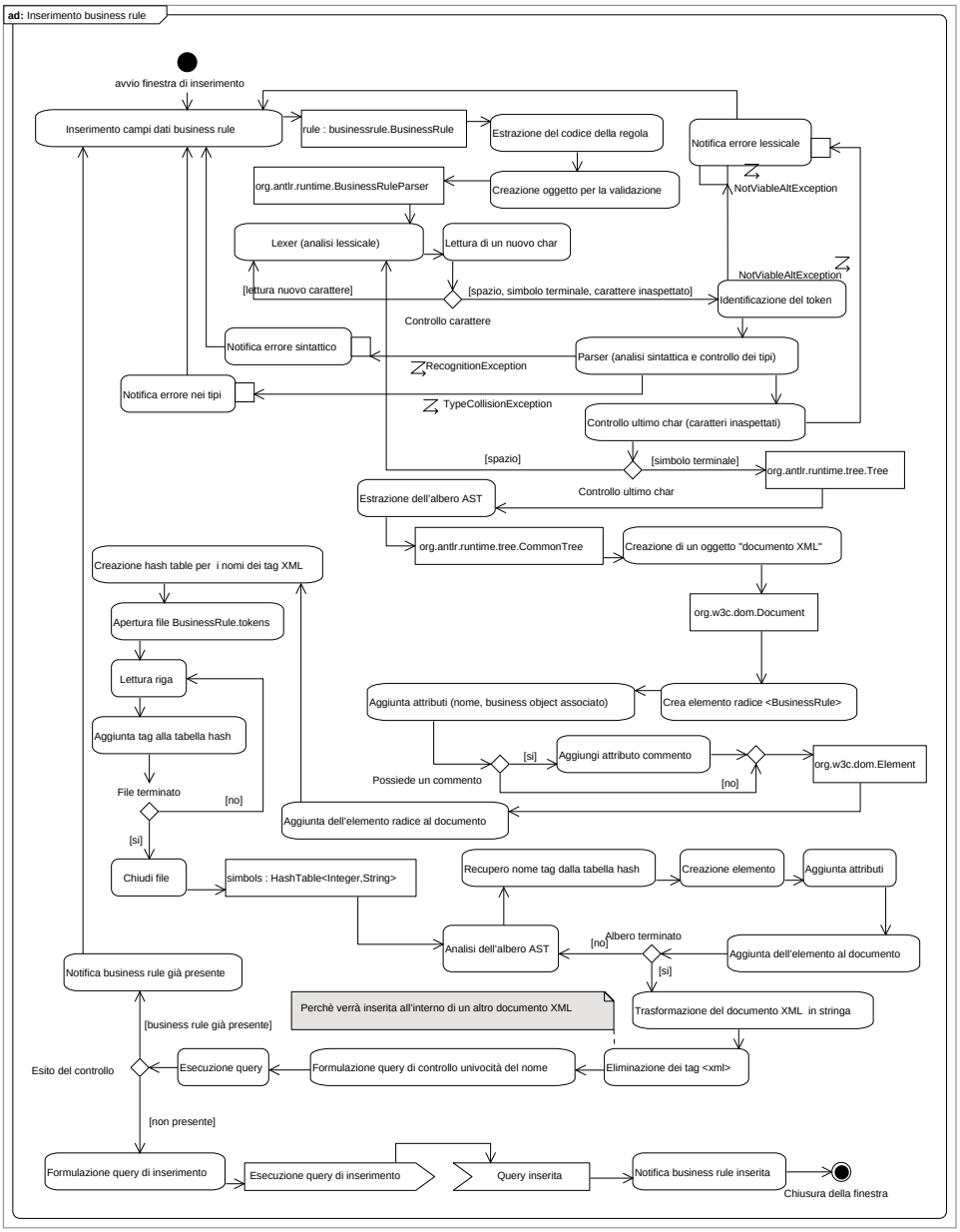
L'interprete invia la richiesta al sistema, il quale formula la query da inviare al DBMS. La query viene quindi eseguita e il risultato viene tornato all'interprete sottoforma di un oggetto *org.w3c.dom.NodeList*.

5.1.3 GUI



All'avvio della GUI viene visualizzato un messaggio in cui vengono richiesti *login* e *password*. Si cerca quindi di stabilire una connessione qualificata al DB. Si controlla quindi l'esito del tentativo di connessione; se il tentativo ha dato esito negativo si richiede la *login* e la *password* all'utente. Se, al contrario, l'esito e positivo, ossia si ha disposizione una connessione, viene visualizzata la schermata della GUI dalla quale possono essere lanciati : *l'inserimento di una business rule*, *l'avvio di una Sandbox*, *la cancellazione di una business rule*. Nel diagramma queste tre attività sono rappresentate come delle *sub-activity* che vengono illustrate di seguito. Al termine di ogni operazione l'utente viene riportato alla finestra principale dove può effettuare una nuova operazione oppure terminare l'esecuzione della GUI stessa.

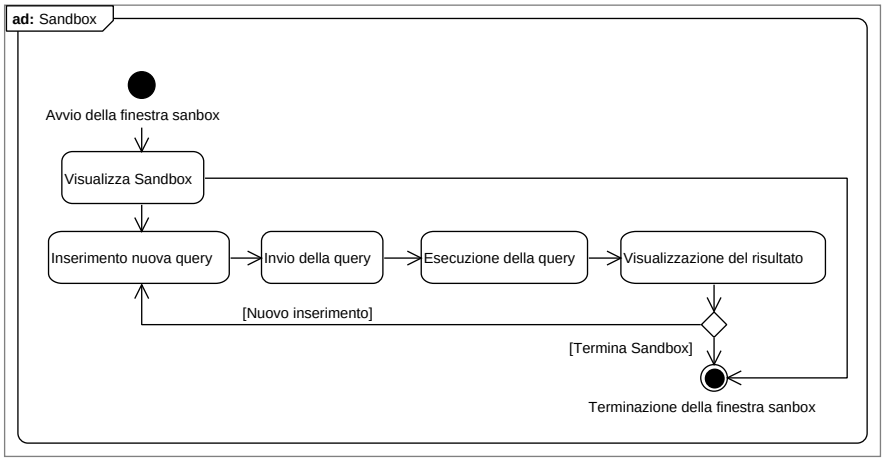
5.1.4 Inserisci business rule



All'avvio della finestra di inserimento business rule questa richiede l'immissione dei relativi campi dati. Dall'oggetto *rule* si estrae il codice della regola e lo si trasforma in un array di char per creare il relativo oggetto per la validazione. Viene quindi effettuata l'analisi lessicale esaminando un carattere per volta fino alla terminazione di un token. Una volta in possesso di un

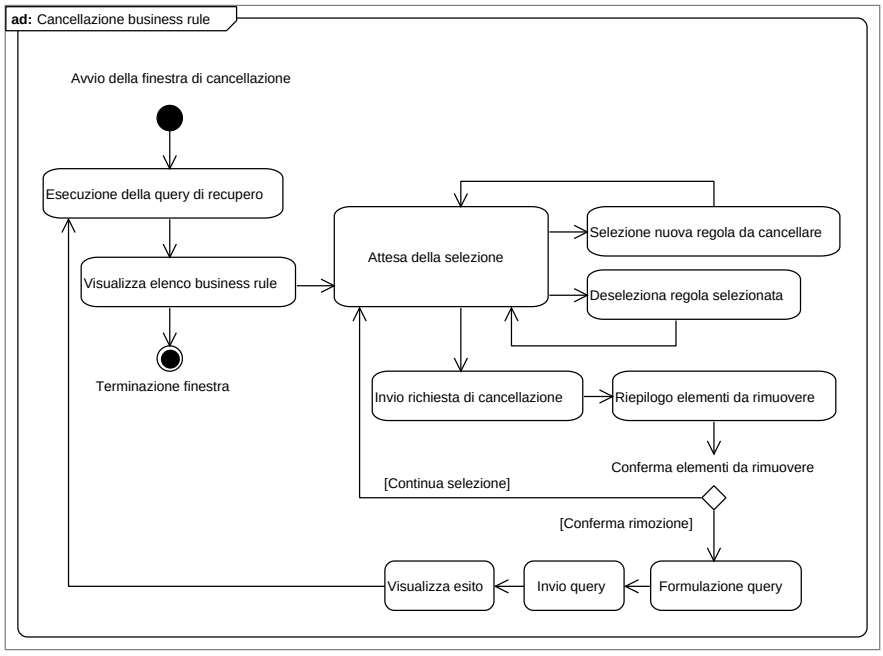
token si cerca di identificarlo confrontandolo con gli operatori e le keyword del linguaggio. Se il token non viene identificato viene lanciata un'eccezione *NotViableAltException* che servirà per notificare all'utente la presenza di un errore lessicale. Una volta identificato il token si passa all'analisi sintattica ed al controllo dei tipi da parte del parser; tale operazione può: sollevare una RecognitionException in caso di errore sintattico, sollevare una TypeCollisionException in caso di errore nei tipi. Si procede poi all'analisi dell' ultimo carattere letto. Se questo è uno spazio si continua l'analisi della regola. Se eventualmente è un carattere inaspettato viene lanciata l'eccezione *NotViableAltException*. Se l'ultimo carattere letto è il simbolo terminale della regola (l'ultima posizione dell'array di char), si termina la validazione ritornando un oggetto *org.antlr.runtime.tree.Tree*. Dall'oggetto viene estratto l'AST e istanziato org.antlr.runtime.tree.CommonTree. Viene poi creato un documento XML. Viene creato l'elemento radice *BusinessRule*. Vengono aggiunti gli attributi *nome*, *business object associato* ed eventualmente il *commento* se presente. Una volta terminata la creazione l'elemento viene aggiunto al documento. Viene creata una tabella hash partendo dalla lettura riga per riga del file *BusinessRule.tokens*. La tabella hash chiamata *simbols* conterrà i nomi dei tag da usare per i vari elementi del documento XML. Si procede all'analisi dell'albero AST. viene invocata una funzione ricorsiva che recupera il codice dell'elemento presente nell'AST, lo associa al nome del tag presente nella tabella hash, crea l'elemento e i suoi attributi, aggiunge infine l'elemento al documento. Una volta terminata l'analisi dell'albero AST il documento viene quindi trasformato in stringa e i tag XML eliminati affinchè tale stringa possa essere inserita in un altro documento XML. Giunti a questo punto si verifica tramite una query l'univocità del nome della business rule. Se l'esecuzione di questa query notifica che una business rule con lo stesso nome è già presente nel repository si torna alla finestra iniziale di inserimento dei campi dati, notificandolo all'utente. Nel caso in cui invece, il nome assegnato alla business rule sia disponibile viene formulata ed eseguita una query di inserimento della regola e quindi notificato l'avvenuto inserimento. A questo punto la finestra di inserimento può essere chiusa.

5.1.5 Sandbox



La *Sandbox* è una finestra il cui scopo è permettere ad un utente di eseguire dei test sul repository. Viene quindi fornito uno spazio in cui digitare una query. La query viene inviata al DBMS ed eseguita. Vengono infine presentati a video i risultati della query evidenziando il tempo impiegato per ottenerli. L'utente può poi terminare la finestra *Sandbox* o inserire una nuova query.

5.1.6 Cancellazione business rule

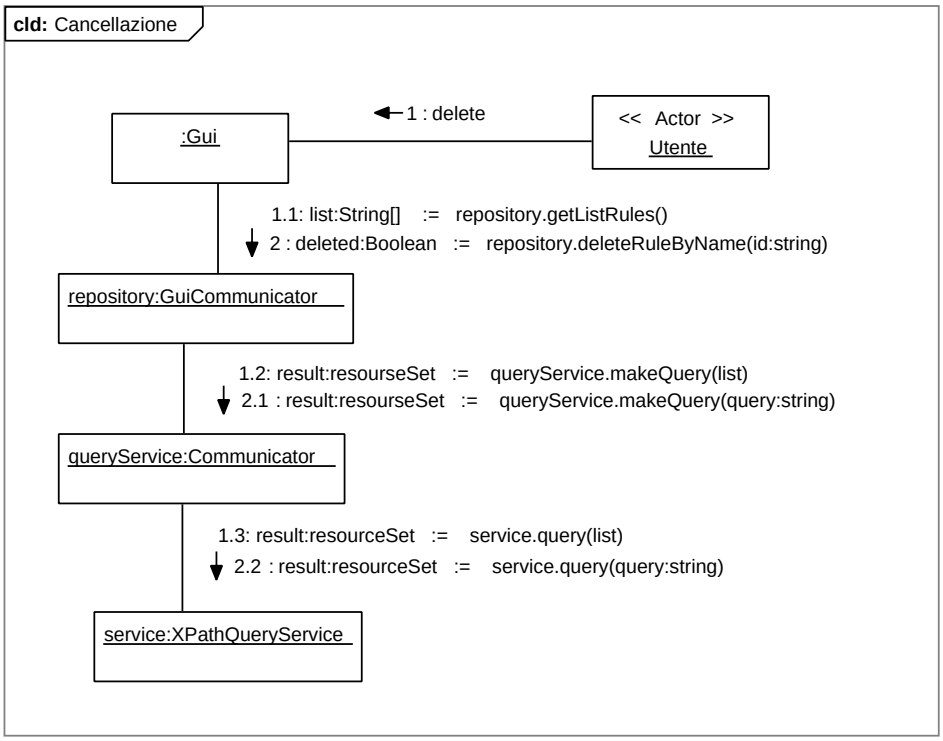


Viene inizialmente effettuata una query sul DBMS atta a recuperare un elenco completo delle business rules. Una volta recuperato l'elenco delle business rules presenti nel repository queste vengono visualizzate sinteticamente in forma tabellare, accompagnate da una *checkbox*. Si attende quindi la selezione/deselezione dell'utente. L'utente notifica al sistema (mediante un bottone) l'intenzione di cancellare le business rules selezionate. Viene poi mostrato un riepilogo delle business rules che verranno eliminate; se l'utente non conferma, il sistema torna ad attendere la selezione. Se, al contrario, la rimozione viene confermata si formula allora la query di rimozione, la quale viene infine eseguita. Al termine della rimozione viene effettuata nuovamente una query di recupero per visualizzare le business rules rimaste nel repository.

5.2 Diagrammi di collaborazione

Viene presentato di seguito un diagramma di collaborazione riguardante le relazioni tra le classi utilizzate nella cancellazione di una business rule dal repository.

5.2.1 Cancellazione business rule

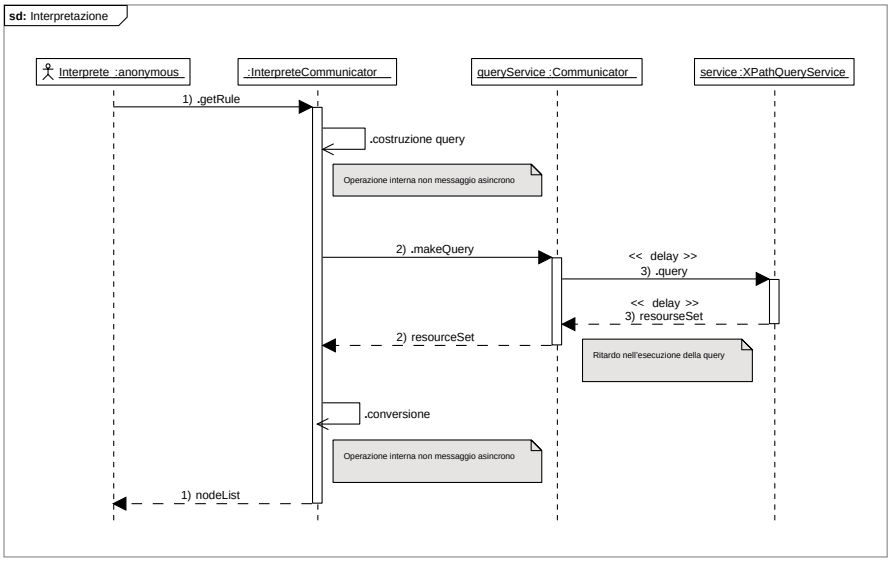


L'utente chiede di cancellare una business rule alla GUI. Questa richiesta provoca innanzitutto l'invocazione di *getListRules()* (1.1) sul campo dati *repository* della GUI. L'oggetto repository che appartiene alla classe *GuiCommunicator* invoca *makeQuery(list)* (1.2) per recuperare la lista di business rules dal repository. E di conseguenza verrà invocata *query()* (1.3) sull'oggetto *service* della classe *XPathQuery* che si occuperà di eseguire la query. Il risultato verrà riportato alla GUI mediante l'oggetto *result* della classe *ResourseSet* che conterrà appunto il risultato della query eseguita. Una volta poi effettuta la selezione delle business rules da rimuovere viene invocato il metodo *deleteRuleByName(id)* (2). Vengono quindi invocati gli stessi metodi (2.1,2.2) utilizzati precedentemente, ma questa volta la query eseguita ritornerà un *result* vuoto perchè è una query di cancellazione e quindi *deleteRuleByName(id)* si limiterà a tornare un *Boolean* che indichi l'esito positivo o negativo della cancellazione.

5.3 Diagrammi di sequenza

Viene di seguito illustrato un diagramma di sequenza che chiarisca la sequenza di chiamate e invocazioni fatte dall'interprete nella richiesta di business rules al sistema *BR-jsys*. Non riuscendo a definire operazioni interne diverse dalla ricorsione in *Poseidon*, le operazioni interne *costruzione query* e *conversione* sono state realizzate con delle frecce vuote indicanti un messaggio asincrono, piuttosto che delle frecce piene. Tali messaggi sono da intendersi tuttavia come messaggi sincroni.

5.3.1 Interpretazione



L'interprete chiama *getRule()* su un suo campo dati interno appartenente alla classe *InterpreteCommunicator*. Il metodo getRule() viene invocato con un parametro di tipo *String* contenente il nome del business object associato. Viene poi costruita la query che viene lanciata mediante il metodo *makeQuery* dell'oggetto *queryService*. La query viene poi eseguita dal metodo *query* dell'oggetto *service*. Viene poi ritornato un *resourceSet* che verrà convertito in un *nodeList* e ritornato all'interprete. Il messaggio 3) è un messaggio con ritardo, ma non siamo riusciti a rappresentarlo con *Poseidon*.

Capitolo 6

Stime di fattibilità e di bisogno di risorse

Dopo aver analizzato il problema attraverso schemi progettuali sono state individuate le risorse necessarie per la realizzazione del prodotto. Con l'utilizzo di software open source siamo riusciti a contenere i costi e contemporaneamente a rendere disponibili tutte le risorse. Tutte le risorse necessarie ai nostri componenti per affrontare le varie problematiche di comunicazione, lo sviluppo del codice, la gestione degli archivi, la verifica dei documenti e del sistema, sono state descritte nel documento PianoDiQualifica.1.4.pdf . Nella fase di specifica tecnica e successivamente di progettazione verranno utilizzati diagrammi UML realizzati con *Poseidon for UML Professional edition 6.0.2* (nella versione trial), in quanto più completo e usabile rispetto ad altri software simili, come ad esempio ArgoUML. Solo i diagrammi use case sono stati realizzati con Dia in quanto progettati prima di affidarsi a Poseidon. Il progetto sarà realizzato mediante il linguaggio java come da requisito implicito del capitolato. Useremo la versione 6 essendo quella di più recente sviluppo. Per la produzione della Gui e delle classi java abbiamo utilizzato Eclipse versione 3.3.1.1, un IDE in grado di fornire numerose funzionalità per lo sviluppo del software. Per la parte riguardante la grammatica, dopo varie ricerche e confronti, ci siamo affidati invece ad Antlr 3.0.1 come generatore di parser e AntlrWorks 1.1.5 come ambiente grafico di sviluppo in quanto offrivano la maggior chiarezza e risposta alle nostre aspettative. Tutta la documentazione sarà redatta utilizzando Kile 2.0 un user-friendly Tex/Latex editor completo e chiaro nelle sue funzionalità. Lo sviluppo avrà luogo singolarmente o in piccoli gruppi a seconda della natura della problematica da affrontare. Il documento/codice avrà ad ogni modo un unico proprietario incaricato di renderlo pubblico tramite server SVN. Utilizzando queste risorse e visti i tempi di consegna del prodotto, la ditta HappyCode ritiene quindi soddisfacibili le richieste del proponente. Per tempistiche più dettagliate si rimanda al PianoDiProgetto.1.7.pdf .

Capitolo 7

Tracciamento della relazione componenti-requisiti

Componente	**Componente Specifico**	**Requisiti Associati**
GUI		F4, F5, F7, F8
Validatore	Validator	F2, F3, F4, F5, F8, NPR1, NQ1
	BusinessRuleLexer	F2, F4, NQ1
	BusinessRuleParser	F2, F4, F10, NQ1
	TypeCollisionException	F2, F4, F10, NQ1
	XMLParser	NPo1
	businessobjects	F2, F10, F11
BusinessRule		F2, F10, F11, NU1, NU2, NU3, NQ1
Comunicatore	Communicator	F3, F5, F6, NPr1
	GUICommunicator	F5, F7, F8
	InterpreterCommunicator	F5, F6
	ValidatorCommunicator	F3