HAPPY CODE INC

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

Definizione del Prodotto

Versione 0.9 - 16 febbraio 2008

Capitolato: "BR-jsys"

Data creazione:	18/11/2007
Versione:	0.9
Stato del documento:	Formale, esterno
Redazione:	Carraro Filippo, Luca Appon
Revisione:	Michele Bortolato
Approvazione:	Luca Appon

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
0.9	15/02/2008	Correzione del documento
0.8	14/02/2008	Modifica alla descrizione delle componenti
0.7	13/02/2008	Aggiunta delle componenti e descrizione nome del file nel modello di documento
0.6	08/02/2008	Inserimento tabella di tracciamento componenti - requisiti
0.5	07/02/2008	Modifica al capitolo standard di progetto
0.4	06/02/2008	Aggiunti riferimenti
0.3	05/02/2008	Aggiunta del nome del file nel modello di documento
0.2	26/01/2008	Documento sottoposto a revisionamento automatico
0.1	25/01/2008	Stesura preliminare del documento

Indice

Capitolo 1

Introduzione

1.1 Scopo del documento

Il seguente documento ha lo scopo di descrivere l'architettura logica prodotta al termine della fase di progettazione architetturale. Descriverà in dettaglio i campi dati e i metodi appartenenti alle singole componenti del sistema "BR-jsys".

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";
- Verbale dell'incontro con il proponente "Incontro2007-11-22.pdf";
- Verbale dell'incontro con il proponente "Incontro2008-02-05.pdf";
- Analisi dei requisiti " AnalisiDeiRequisiti2.6.pdf ";
- Piano di Qualifica " PianoDiQualifica.1.4.pdf ";
- Norme di Progetto " NormeDiProgetto.2.0.pdf ";
- Specifica Tecnica " SpecificaTecnica.1.0.pdf ";

- “Ingegneria del Software” 8a edizione - Ian Sommerville;
- “The Definitive ANTLR Reference”;
- Manuale di eXist reperibile all’URL: *exist.sourceforge.net*.

Capitolo 2

Standard di progetto

2.1 Standard di progettazione architetturale

Per quanto riguarda le rappresentazioni grafiche dell'architettura del sistema "BR-jsys", ci siamo basati sulle regole definite da UML 2.1. In particolare, ci siamo appoggiati all'uso di "Poseidon for UML - professional edition 6.0.2". Per i diagrammi use-cases ci siamo affidati invece a Dia 0.96.1.

2.2 Standard di documentazione del codice

Per le regole di documentazione del codice si faccia riferimento al documento "Norme di Progetto" allegato.

2.3 Standard di denominazione di entità e relazioni

Per le regole di denominazione delle entità, delle relazioni e degli oggetti, si faccia riferimento al documento "Norme di Progetto" allegato.

2.4 Standard di programmazione

Anche in questo caso, si faccia riferimento al documento "Norme di Progetto" allegato.

2.5 Strumenti di lavoro

L'azienda ha fatto uso dei seguenti strumenti di lavoro:

- Eclipse 3.3.1.1;
- eXist 13.1.1.2;

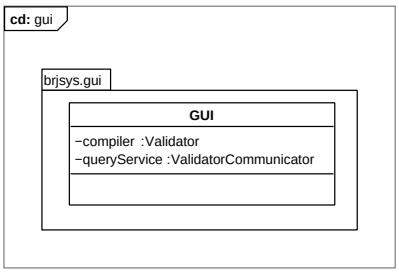
- Latex 3.14;
- ANTLRWorks 1.1.5.

Capitolo 3

Specifica delle componenti

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 tratteremo successivamente.

Interfacce con e relazioni di uso da altre componenti

Nessuna.

Attività svolte e dati trattati

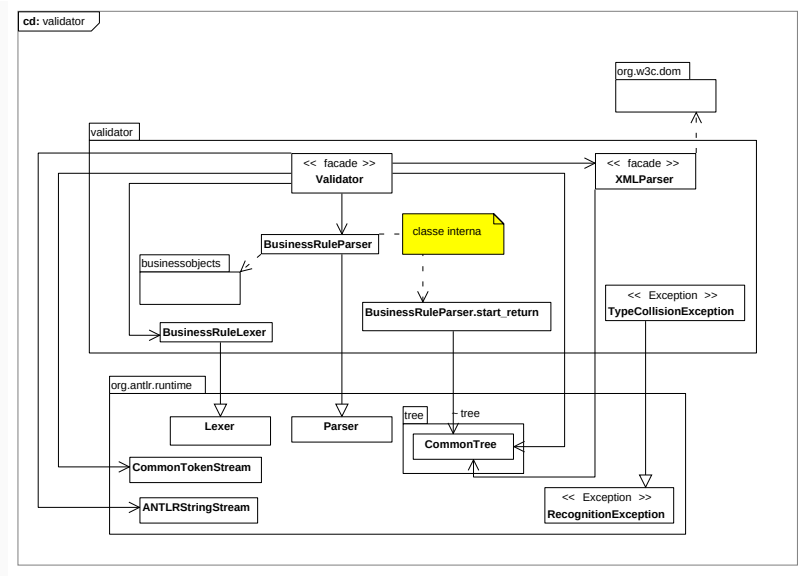
Gui possiede i campi dato necessari per comunicare col validatore e col DBMS.

Attributo	Descrizione
-compiler:Validator	Rappresenta il validatore per le business rule
-repository:GUICommunicator	Collega la componente al DBMS consentendogli di effettuare le query.

Metodo	Descrizione
+main(args:String[]):Validator	Metodo standard in java per l'avvio dell'applicazione. Si incaricherà di creare un oggetto Gui
+Gui()	Costruttore per Gui.

3.2 Validatore

3.2.1 Diagramma delle classi



3.2.2 Validator

Tipo, obiettivo e funzione del componente

La componente Validator 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 ed inserimento che avremmo altrimenti dovuto riportare direttamente laddove la componente GUI l'avesse richiesto.

Relazioni d'uso di altre componenti

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

Interfacce con e relazioni di uso da altre componenti

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

Attività svolte e dati trattati

Attributo	Descrizione
-repository:ValidatorCommunicator	Collega la componente al DBMS per inserire la business rule validata nel repository.

Metodo	Descrizione
+Validator()	Costruttore di Validator.
+validate(rule:BusinessRule):boolean {throws RecognitionException }	Unico metodo della componente che ricevuto in ingresso una business rule ne effettua la validazione. Qualora si verificassero errori in fase di compilazione si restituirà un eccezione che verrà gestita dal chiamante. Se la business rule viene inserita, il metodo ritornerà *true*. Ritornerò invece *false* se la regola ha un nome che è già presente nel repository.

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 Business-RuleLexer. Quest'ultima verrà approfondita successivamente.

Attività svolte e dati trattati

La componente, generata tramite un generatore di parser, dispone di numerosi campi dato per identificare i tokens della grammatica, nonchè vari metodi per la loro gestione. In definitiva si dovrà usare solo il suo costruttore.

Metodo	Descrizione
+BusinessRuleLexer(input: AntlrStringStream)	Costruttore del componente.

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 sia sintattico che semantico della regola, facendo un ulteriore 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 infine in grado di dare informazioni accurate riguardo eventuali errori in fase di validazione.

Relazioni d'uso di altre componenti

Questa componente 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 invece 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

Questa componente, generata tramite un generatore automatico di parser, fornisce i campi dato per il riconoscimento dei token in appoggio a quelli già forniti dalla componente BusinessRuleParser. Contiene vari metodi privati per il controllo sintattico e semantico, per il parsing e contiene infine classi innestate incaricate di generare l'AST.

Attributo	Descrizione
+endParsing: BusinessRuleParser.start_return	Campo dati che contiene l'AST da convertire in XML.

Metodo	Descrizione
+BusinessRuleParser(input: CommonTokenStream, associated: String)	Costruttore della componente. Il primo parametro identifica lo stream generato nelle fasi precedenti, il secondo dice qual'è il business object associato alla regola.
+start(): BusinessRuleParser.start_return {throws RecognitionException }	Risultato del parsing corretto di una business rule. Contiene al suo interno l'AST. Nel caso di errore nella validazione il chiamante dovrà gestire opportunamente l'eccezione.

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.

Attributo	Descrizione
-first:Class	Identifica il tipo del primo elemento che ha causato un'operazione incompatibile.
-second:Class	Identifica il tipo del secondo elemento che ha causato un'operazione incompatibile

Metodo	Descrizione
+TypeCollisionException(typeA: Class, typeB:Class)	Costruttore a due parametri che rappresentano i due tipi che sono entrati in conflitto.
+printStackTrace()	Metodo ridefinito che consente di visualizzare correttamente la struttura dell'errore avvenuto.

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 BusinessRuleParser in 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 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.

Attributo	Descrizione
-simbols:HashTable<Integer,String>	Nell'AST il tipo di token viene rappresentato tramite un numero univoco. Per poter associare in maniera rapida i tokens ai numeri che li rappresentano è dunque necessario generare una tabella Hash. Le informazioni riguardo l'associazione sono riportate in un file di nome *BusinessRule.tokens* generato automaticamente dal generatore di parser.

Metodo	Descrizione
-generateSimbols(): HashTable<Integer, String >	Questo metodo legge il file *BusinessRule.tokens* dal quale ricava tutte le associazioni tra il numero associato al token e la stringa che gli si associa in fase di costruzione dell'attributo *simbols*.
+parse(AST:Tree, rule:BusinessRule): String	Metodo pubblico che si incarica di effettuare la conversione da AST a stringa che rappresenta l'elemento XML. Si dovrà seguire la procedura adeguata per la creazione di un elemento strutturato XML secondo le specifiche delle librerie *org.w3c.dom*
-scanAST(AST: Tree, root: Element, doc: Document) Element	Metodo privato che scorre ricorsivamente l'AST per convertirla in un elemento XML. La regola verrà successivamente inserita nel repository.

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

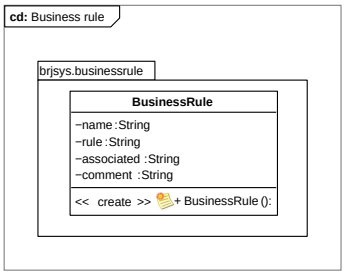
BusinessRuleParser necessita di questa componente per effettuare il controllo dei tipi qualora fosse necessario.

Attività svolte e dati trattati

Il componente businessobjects 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

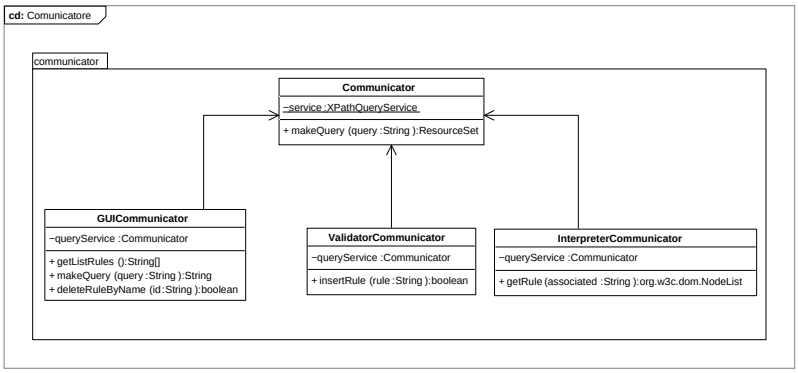
La componente contiene i campi dato Stringa per rappresentare una business rule ossia name, associatedObject, rule e comment, quest'ultima può non essere istanziata. Vengono inoltre messi a disposizione il costruttore e il metodo ridefinito toString().

Attributo	Descrizione
+name:String	Denota il nome della business rule.
+associated:String	Denota il business object associato.
+rule:String	Denota la business rule scritta dall'utente e da validare.
+comment:String	Denota un commento opzionale che l'utente può associare a quella business rule.

Metodo	Descrizione
BusinessRule(nameR: String , associatedR: String, ruleR: String, commentR: String)	Costruttore di BusinessRule.

3.4 Comunicatore

3.4.1 Diagramma delle classi



Durante la fase di progettazione, la nostra azienda ha fatto diversi studi sui DBMS in grado di trattare basi di dati XML. Si è infine presa la decisione di adottare eXist, in quanto, oltre ad essere stato consigliato dal proponente, si è rivelato il più adatto alle nostre esigenze. eXist oltre a garantire buone prestazioni, mette a disposizione più funzionalità rispetto ai suoi concorrenti e consente di sviluppare applicazioni java che si interfacciano al DBMS stesso.

3.4.2 Communicator

Tipo, obiettivo e funzione del 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 le 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. Quest'ultima consentirà successivamente di interrogare il repositorydirettamente, passandogli la stringa che rappresenta la query composta secondo le specifiche XQuery.

Attributo	Descrizione
-service:XPathQueryService	Consente, se istanziato correttamente, di comunicare con risorse presenti su eXist. In particolare faremo in modo di configurarlo affinchè faccia query solamente nel repository(posizionato in una posizione specifica nel DBMS).
-correctUsername:String	Memorizza il corretto username utilizzato per l'accesso.
-correctPassword:String	memorizza la corretta password utilizzata per l'accesso.

Metodo	Descrizione
+Communicator(username: String, password: String)	Costruttore di Communicator. Nel caso si acceda alla classe per la prima volta durante l'esecuzione del programma si istanzierà l'attributo service. Se la connessione andrà a buon fine gli attributi "correctUsername" e "correctPassword" verranno settati ai valori corrispondenti. Se l'attributo service è già stato istanziato non si farà altro che assicurarsi che i parametri in ingresso al costruttore corrispondano a quelli presenti negli attributi.
+makeQuery(query: String) :ResourceSet	Metodo pubblico che ricevuto in ingresso una query scritta secondo le specifiche XQuery la esegue e ne restituisce il risultato. Si dovrà gestire il caso di query mal formata.

3.4.3 GUICommunicator

Tipo, obiettivo e funzione del componente

La componente 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

GUICommunicator sarà strutturata nel seguente modo:

Attributo	Descrizione
-queryService:Communicator	Permette alla componente di colloquiare con eXist.

Metodo	Descrizione
+GUICommunicator(username: String, password:String)	Costruttore del componente, username e password saranno utilizzati per l'istanziazione di queryService.
+deleteRuleByName(id:String): boolean	Metodo per Cancellare una business rule dato il suo identificativo. Se la business rule è stata trovata nel repository e cancellata ritornerà *true*, altrimenti *false*.
+makeQuery(query:String): String	Metodo per eseguire una query in XQuery. Si dovrà fare attenzione che la struttura della query non contenga istruzioni XQuery Update che permetterebbero altrimenti all'utente di modificare a piacimento il repository rendendolo inconsistente.
+getListRules():String[]	Metodo che ritorna informazioni riguardo le business rules scritte nel repository. In particolare, l'array di ritorno avrà un elemento per ogni business rule nel repository. Ogni elemento conterrà nome, regola, business object associato ed eventuale commento.

3.4.4 InterpreterCommunicator

Tipo, obiettivo e funzione del 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

La componente 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

InterpreterCommunicator deve soltanto fornire al chiamante le business rules associate al business object.

Attributo	Descrizione
-queryService:Communicator	Permette alla componente di colloquiare con eXist.

Metodo	Descrizione
+InterpreterCommunicator(username:String, password:String)	Costruttore con funzionamento analogo a GUICommunicator.
+getRule(associated:String): NodeList	Dato in input un business object ritorna tutte le business rules ad esso associate.

3.4.5 ValidatorCommunicator

Tipo, obiettivo e funzione del 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

ValidatorCommunicator deve soltanto occuparsi di inserire la business rule nel repository. L'inserimento avverrà soltanto se la business rule ha un nome che non è ancora presente nel repository.

Attributo	Descrizione
-queryService:Communicator	Permette alla componente di colloquiare con eXist.

Metodo	Descrizione
+ValidatorCommunicator(username:String, password:String)	Costruttore con comportamento analogo a GUICommunicator.
+insertRule(rule:String, idRule:String): boolean	Inserisce la business rule nel repository soltanto se il suo nome non è già presente. Se l'inserimento è andato a buon fine ritornerà *true*.

Capitolo 4

Appendice

4.1 Tracciamento relazione componenti - requisiti

Componente	Metodo Associato	Requisiti
Gui	+Gui()	F4, F5, F7, F8
	+main(args:String[])	–
Validator	+Validator()	F2, F5, F8
	+validate()	F2, F3, F4, F5, F8, Npr1, NQ1
BusinessRuleLexer	+BusinessRuleLexer()	F2, F4, NQ1
BusinessRuleParser	BusinessRuleParser()	F2, F4, F10
	+start()	F2, F4, F10, NQ1
TypeCollisionException	+TypeCollisionException()	F2, F4, F10, NQ1
	+printStackTrace()	F2, F4, F10
XMLParser	-generateSimbols()	NPo1
	+parse()	NPo1
	-scanAST()	NPo1
businessobject	–	F2, F10, F11
BusinessRule	+BusinessRule()	F2, F10, F11, NU1, NU2, NU3, NQ1
Communicator	+Communicator()	F3, F5, F6, NPr1
	+makeQuery()	F3, F5, F6, NPr1
GUICommunicator	+GUICommunicator()	F5, F7, F8
	+deleteRuleByName()	F5, F7, F8
	+makeQuery()	F5, F8
	+getListRules()	F5, F8
InterpreterCommunicator	+InterpreterCommunicator()	F5, F6
	+getRule()	F5, F6
ValidatorCommunicator	+ValidatorCommunicator()	F3
	+insertRule()	F3