0.1 Evaluación de la herramienta para la definición del DSL.

Para la construcción de TextRAM, se analizarón dos herramientas: Xtext y Rascal. Las principales características de Xtext fueron vistas en [11!!!].

0.1.1 Rascal

Rascal (Storm (2011)] es un lenguaje de meta-programación extensible que proporciona mecanismos de análisis de código fuente y transformación. El código fuente es la objeto primario de manipulación de Rascal.

Rascal proporciona una interfaz simple de programación para extender el IDE de eclipse para ajustarlo a nuevos lenguajes. Actualmente Rascal es utilizado como vehículo de investigación para analizar código existente e implementar DSL.

EASY (Extracción, análisis y sintesis) es un paradigma con las siguientes características:

- *Extracción*: los meta-programas inicia por la extracción de información ("facts") desde un programa de entrada.
- *Análisis*: los "facts" derivados son computados y la información es enriquecida.
- *Síntesis*: Finalmente el meta-programa producira un tipo de resultado. Ejemplos: transformación de un código fuente (eliminación de código muerto), reportes (estadística del número públicas sin utilizar), visualización.

La figura [] muestra el ciclo de vida del paradigma EASY.

Rascal fue diseñado para seguir el paradigma de EASY. Un DSL puede ser implementado con RASCAL aplicando el paradigma EASY:

- Extracción: La información de entrada es el código fuente del DSL. El AST puede ser derivado a partir la sintaxis concreta.
- Análisis: el producto de la extracción es el AST. En la etapa de análisis se pueden realizar las validaciones, restricciones, verificacione de tipo, restricciones de alcance.
- Sintesis: cubre las tareas de visualización, generación de código y optimización.

Las principales cualidades de Rascal son:

- **Sintaxis familiar y flujo de control**: la sintaxis de Rascal es parecida a C, Java, Javascript o C#. Las estructuras de control obedecen las mismas reglas sintácticas de un GPL basado en llaves.
- **Datos imutables**: El cambio de un valor, siempre tendra como resultado otro valor.
- **Tipos de datos integrados y "pattern matching"**: Rascal integra una colección de tipos integrados (integer, boolean, string, real, tuple, list, set, relation,map, parse trees, source locations, date-time). Adicionalmente soporta "Abstract Data Types" (ADTs). Todos los tipos de datos tiene una representación literal, por tanto se puede utilizar "pattern matching".
- **Construcciones específicas de dominio**: un ejemplo es la sentencia `visit` que emula el funcionamiento del patrón de diseño `Visitor`. `visit` puede ser utilizado el análisis y transformación de código fuente.
- **Gramática arbitraria libre de contexto**: gracias a esto, Rascal genera en forma automática "parsers" a partir de la gramática.
- **Plantillas de cadenas**: utilizadas para la generación de código.

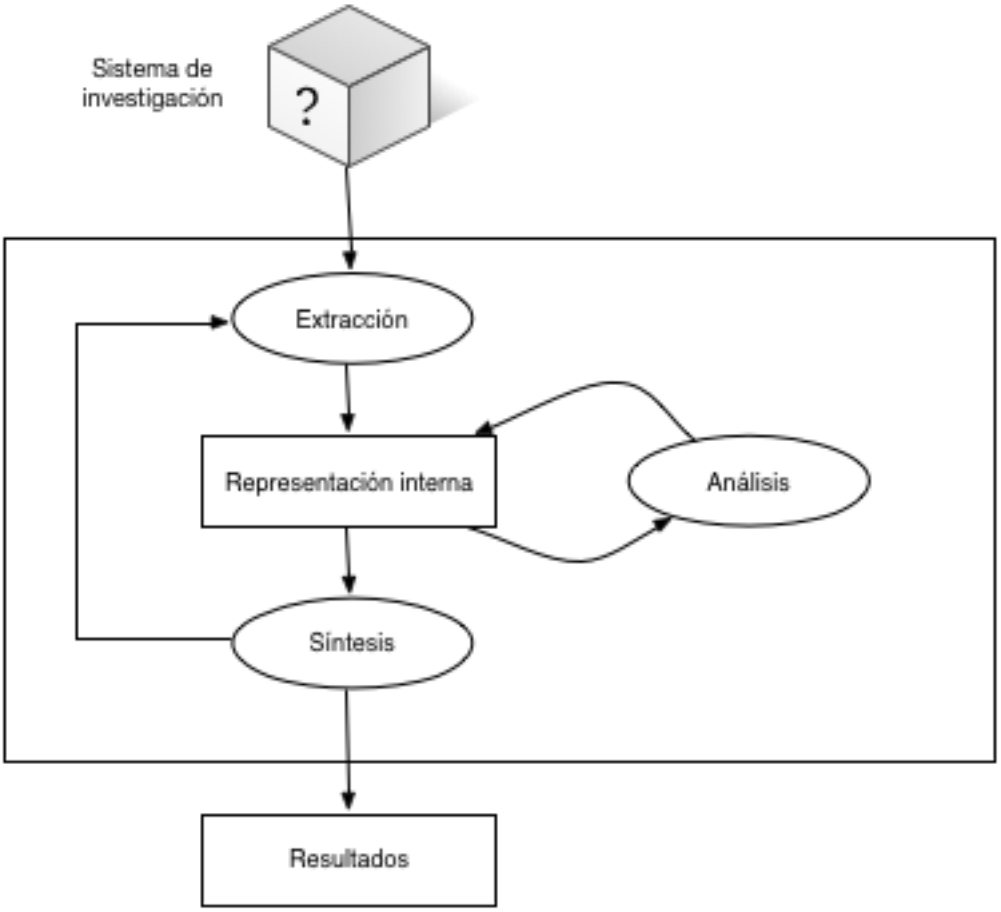

Figure 1: Paradigma EASY

- **Integración con Java**: algunas tareas requieren de cualidades que Rascal no proporciona. Para resolver este problema es posible utiliza código Java por medio de anotaciones en los encabezados de las funciones de Rascal.
- **Integración del IDE con Eclipse**: Rascal puede ser instalado en un IDE basado en Eclipse, dando como resultado el soporte a "syntax highlighting, outling, interactive visualization" e incluso un REPL ("Read-Eval-Print-Loop").

0.1.2 Diferencias entre Xtext y Rascal

Xtext y Rascal son clasificados como "Languages workbenches". Un "language workbench" es una herramienta que provee mecanismos de alto nivel para la implementación de lenguajes (específicos a dominio) (Erdweg et al., 2013).

Diversos "languages workbenches" son estudiados y comparados en una competición anual llamada "Language Workbenche Challenge" (Xtext, n.d.). La Figura !!!! muestra el modelo de los "language workbench", dicho modelo es utilizado como base de comparación.

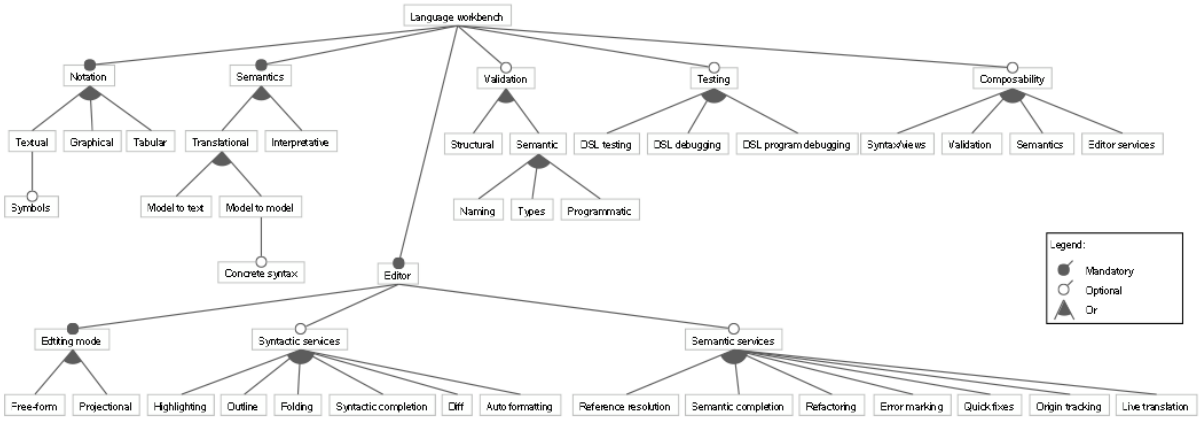

Figure 2: Modelo un "Language Workbench" copiado de (Xtext, n.d.)

Rascal y Xtext comparten las mismas cualidades (descritas en la Figura !!) para la construcción de TextRam se ha elegido Xtext debido a que dicha herramienta está intimamente ligada a Ecore que es el lenguaje de meta-modelado utilizado por RAM.

Erdweg, S., Storm, T. van der, Völter, M., Boersma, M., Bosman, R., Cook, W. R., ... others. (2013). The state of the art in language workbenches. In *Software language engineering* (pp. 197–217). Springer.

Storm, T. van der. (2011). *The rascal language workbench*. CWI Technical Report SEN-1111, CWI.

Xtext. (n.d.). Language workbench challenge - comparing tools of the trade. http://www.languageworkbenches.net.