

Resumen

En este documento se presenta un caso de estudio que permitió evaluar el desempeño de la herramienta EventB2Java; dicho caso consistió en modelar formalmente un modulo de inventarios que fue traducido a código Java+JML y que posteriormente se incorporó en el software opensource Openbravo POS, la evaluación del desempeño se realizó comparando la versión original del mismo vs la modificada. En este trabajo también se mejoró el rendimiento del conjunto de clases que usa la herramienta haciendo uso de los objetos disponibles en el SDK de Java y se presentan las reglas de traducción y el análisis previo llevado a cabo para traducir de un modelo en Event-B a código Java + especificación en JML.

Abstract

In this document, a case study is presented which evaluated the performance of the EventB2Java tool. In the case study, which consisted of formally modeling a module of inventories translated to Java + JML code and later incorporating it into the open source Openbravos POS software, the performance evaluation was done by comparing the original version of the code vs the modified version. In this evaluation we found that the modified version did improve the rendition of classes using the aforementioned tool.

Keywords: Event-B, EventB2Java, Rodin, JML, Openbravo, POS

PONTIFICIA UNIVERSIDAD JAVERIANA

TESIS DE MAESTRÍA

EVALUACIÓN DEL DESEMPEÑO DEL GENERADOR DE CÓDIGO EVENTB2JAVA

Autor:

Carlos A. Hernández G.

Supervisor:

Ph.D Néstor Cataño

*Una tesis presentada en cumplimiento de los requisitos
para el grado de Magíster en Ingeniería*

en la

Pontificia Universidad Javeriana
Departamento de Electrónica y Ciencias de la Computación.

16 de julio de 2014

Agradecimientos

Esta tesis no podría haber sido posible sin la ayuda de muchas personas. Mis más profundos agradecimientos a mi supervisor Néstor Cataño. Su rigurosa supervisión de esta tesis me llevo a dar lo mejor de mí mismo. Fui realmente afortunado de trabajar con él en un tema de investigación tan apasionante como los métodos formales.

Me gustaría extender mis agradecimientos a Víctor Rivera, con quien desarollé parte de este trabajo y de quien aprendí mucho durante mi estadía en Madeira.

Finalmente, deseo expresar mi gratitud y amor a mi mama, quien me apoyo y me dio su voz de aliento durante este largo proceso. A mi padre quien desde el cielo vela por mí y cuyo apoyo en vida sembró las bases para mi presente y futuro. A mis amigos quienes me brindaron la motivación para seguir adelante.

Índice general

| | |
|---|------|
| Resumen | I |
| Abstract | II |
| Agradecimientos | I |
| Tabla de contenido | II |
| Índice de figuras | V |
| Índice de cuadros | VIII |
| Abreviaciones | IX |
| Símbolos | X |
| | |
| 1. Introducción | 1 |
| 2. Preliminares | 3 |
| 2.1. Método Event-B | 3 |
| 2.2. JML | 4 |
| 3. Problemática | 7 |
| 3.1. Planteamiento del problema | 7 |
| 3.2. Objetivo general | 8 |
| 3.3. Objetivos específicos | 8 |
| 4. Traducción de un Modelo en Evnet-B a Java+JML | 9 |
| 4.1. Versión inicial de las reglas | 10 |
| 4.2. Versión final de las reglas de traducción | 15 |
| 4.3. La herramienta de traducción EventB2Java | 19 |
| 4.4. Ejemplo de traducción | 20 |
| 5. Mejoramiento del desempeño de las clases BSet y BRalation | 24 |
| 5.1. Resultados de las pruebas | 25 |

| | |
|---|-----------|
| 6. Caso de estudio acerca de generación de código de una aplicación ERP | 30 |
| 6.1. Descripción del caso de estudio | 30 |
| 6.1.1. Objetivo general | 31 |
| 6.1.2. Objetivos específicos | 31 |
| 6.2. Openbravo ERP | 31 |
| 6.2.1. Funcionamiento de Openbravo POS | 32 |
| 6.2.1.1. Creación de clientes (<i>Customers</i>) | 34 |
| 6.2.1.2. Creación de bodegas (<i>Warehouses</i>) | 35 |
| 6.2.1.3. Creación de un impuesto (<i>Tax</i>) | 35 |
| 6.2.1.4. Creación de categorías (<i>Categories</i>) y productos (<i>Products</i>) | 37 |
| 6.2.1.5. Procesamiento de una venta (<i>Sale</i>) | 37 |
| 6.3. Documento de requerimientos (módulo de inventario) | 40 |
| 6.3.1. Estrategia de refinamiento | 43 |
| 6.3.1.1. Refinamiento 1: ref1_Enabling | 43 |
| 6.3.1.2. Refinamiento 2: ref2_Order | 43 |
| 6.3.1.3. Refinamiento 3: ref3_Categories | 44 |
| 6.3.1.4. Refinamiento 4: ref4_Taxes | 44 |
| 6.3.1.5. Refinamiento 5: ref5_Reports | 44 |
| 6.4. Modelo formal para el modulo de inventarios en Event-B | 45 |
| 6.4.1. El contexto del modelo abstracto | 45 |
| 6.4.2. Las variables del modelo abstracto | 45 |
| 6.4.3. Los invariantes y la inicialización en el modelo abstracto | 47 |
| 6.4.4. Creación de los productos | 48 |
| 6.4.5. Creación de los productos mediante listas | 50 |
| 6.4.6. Creación, edición y borrado de un proveedor | 50 |
| 6.4.7. Asociar un producto con su proveedor | 53 |
| 6.4.8. Creación y edición de un cliente | 54 |
| 6.4.9. Edición de los precios de venta y compra | 55 |
| 6.4.10. Creación y eliminación de bodegas | 56 |
| 6.4.11. Refinamiento 1 - Variables, invariantes e inicialización | 57 |
| 6.4.12. Refinamiento 1 - Eventos refinados | 58 |
| 6.4.13. Refinamiento 1 - Habilitar o deshabilitar productos | 59 |
| 6.4.14. Refinamiento 2 - El contexto | 60 |
| 6.4.15. Refinamiento 2 - Variables, invariantes e inicialización | 61 |
| 6.4.16. Refinamiento 2 - Creación de una orden de venta con su detalle . . | 62 |
| 6.4.17. Refinamiento 3 - El contexto | 63 |
| 6.4.18. Refinamiento 3 - Variables, invariantes e inicialización | 65 |
| 6.4.19. Refinamiento 3 - Eventos refinados | 66 |
| 6.4.20. Refinamiento 3 - Creación y borrado de una categoría | 66 |
| 6.4.21. Refinamiento 4 - El contexto | 67 |
| 6.4.22. Refinamiento 4 - Variables, invariantes e inicialización | 68 |
| 6.4.23. Refinamiento 4 - Creación, eliminación y actualización de un im- puesto | 69 |
| 6.4.24. Refinamiento 4 - Agregar un impuesto a un producto | 70 |
| 6.4.25. Refinamiento 4 - Agregar un impuesto a los productos de una categoría | 72 |
| 6.4.26. Refinamiento 5 - Variables, invariantes e inicialización | 73 |

| | |
|--|----------------|
| 6.4.27. Refinamiento 5 - Obtener los precios de compra y venta | 73 |
| 6.5. Resultados obtenidos de la prueba de desempeño | 75 |
| 6.5.1. Tiempos por número de operaciones tomados en el SO Windows . | 76 |
| 7. Conclusiones | 81 |
| 7.1. Trabajo futuro | 82 |
| A. Modulo de Inventario en Event-B | 83 |
| A.1. Maquina abstracta | 83 |
| A.2. Refinamiento 1: Habilitación y deshabilitación de productos, clientes y proveedores (ref1_Enabling) | 94 |
| A.3. Refinamiento 2 - Ordenes (ref2_Orders) | 102 |
| A.4. Refinamiento 3 - Categorías (ref3_Categories) | 107 |
| A.5. Refnamiento 4 - Impuestos (ref4_Taxes) | 110 |
| A.6. Refnamiento 5 - Reportes (ref5_Reports) | 114 |
| B. Resultado de las pruebas clase BSet | 117 |
| C. Resultado de las pruebas clase BRelation | 121 |
| Bibliografía | 126 |

Índice de figuras

| | |
|--|----|
| 2.1. Evento para crear una cuenta en una red social | 4 |
| 2.2. Especificación JML producida para la creación de una cuenta en una red social | 6 |
| 4.1. Menu contextual EventB2Java | 20 |
| 4.2. Evento que permite agregar una categoría | 21 |
| 5.1. Comparación del método insertar de la clase BSet Java y JML | 26 |
| 5.2. Comparación del método unión (<i>union</i>) de la clase BSet Java y JML | 26 |
| 5.3. Tiempos para el método unión (<i>union</i>) del clase BSet Java | 27 |
| 5.4. Tiempos para el método intersección (<i>intersection</i>) del clase BSet Java | 27 |
| 5.5. Comparación del método intersección (<i>intersection</i>) de la clase BSet Java y JML | 27 |
| 5.6. Comparación del método diferencia (<i>difference</i>) de la clase BSet Java y JML | 28 |
| 5.7. Tiempos para el método diferencia (<i>difference</i>) del clase BSet Java | 29 |
| 5.8. Comparación del método inserción (<i>insert</i>) de la clase BRelation Java y JML | 29 |
| 6.1. Interfaz inicial Openbravo | 32 |
| 6.2. Interfaz principal para el rol de invitado | 33 |
| 6.3. Interfaz principal para el rol empleado | 33 |
| 6.4. Interfaz principal para el rol de supervisor | 33 |
| 6.5. Interfaz principal para el rol administrador | 34 |
| 6.6. Vista para ingresar a crear un cliente | 35 |
| 6.7. Interfaz para la creación de un cliente | 35 |
| 6.8. Acceso a la vista de creación de bodegas | 36 |
| 6.9. Interfaz para la creación de una bodega | 36 |
| 6.10. Interfaz para acceder a las vistas de impuestos, productos y categorías | 36 |
| 6.11. Interfaz para la creación de un impuesto | 37 |
| 6.12. Interfaz para la creación de una categoría | 38 |
| 6.13. Interfaz para la creación de un producto | 38 |
| 6.14. Interfaz para procesar una venta | 38 |
| 6.15. Ventana para asociar un cliente a la venta | 39 |
| 6.16. Ventana para realizar el pago | 40 |
| 6.17. Factura de venta | 40 |
| 6.18. Contexto de la maquina abstracta | 46 |
| 6.19. Variables de la maquina abstracta | 46 |
| 6.20. Invariantes de la maquina abstracta | 47 |

| | |
|--|----|
| 6.21. Inicialización de variables en la maquina abstracta | 48 |
| 6.22. Evento que crea una materia prima | 49 |
| 6.23. Evento que crea un producto para reventa | 49 |
| 6.24. Evento que crea un producto manufacturado | 50 |
| 6.25. Evento que crea un conjunto de productos de tipo materia prima | 51 |
| 6.26. Evento que crea un conjunto de productos de tipo reventa | 51 |
| 6.27. Evento que crea un conjunto de productos de tipo manufacturados | 52 |
| 6.28. Evento para crear un proveedor | 52 |
| 6.29. Evento que permite borrar un proveedor | 53 |
| 6.30. Evento que permite editar un proveedor | 53 |
| 6.31. Evento que permite asociar un producto a un proveedor | 54 |
| 6.32. Evento que permite editar crear un cliente | 54 |
| 6.33. Evento que permite editar un cliente | 55 |
| 6.34. Evento que permite editar el precio de compra de un producto | 55 |
| 6.35. Evento que permite editar el precio de venta de un producto | 56 |
| 6.36. Evento para agregar una bodega | 56 |
| 6.37. Evento que permite remover una bodega | 57 |
| 6.38. Variables del refinamiento 1 | 57 |
| 6.39. Invariantes del refinamiento 1 | 58 |
| 6.40. Inicialización de las variables en el refinamiento 1 | 58 |
| 6.41. Refinamiento de evento que permite crear una materia prima | 58 |
| 6.42. Refinamiento del evento que permite crear un conjunto de productos de tipo materia prima | 59 |
| 6.43. Refinamiento del evento que permite agregar un proveedor | 59 |
| 6.44. Refinamiento del evento que permite agregar un cliente | 59 |
| 6.45. Evento que permite habilitar un producto de tipo reventa | 60 |
| 6.46. Evento que permite deshabilitar un producto de tipo reventa | 60 |
| 6.47. Contexto del refinamiento 2 | 61 |
| 6.48. Variables del refinamiento 2 | 61 |
| 6.49. Invariantes del refinamiento 2 | 62 |
| 6.50. Inicialización de las variables en el refinamiento 2 | 62 |
| 6.51. Evento que permite crear una orden de venta | 63 |
| 6.52. Evento que permite agregar el detalle del numero de productos a una orden de venta | 64 |
| 6.53. Evento que permite agregar el detalle de la fecha a una orden de venta | 64 |
| 6.54. Contexto del refinamiento 3 | 65 |
| 6.55. Variables del refinamiento 3 | 65 |
| 6.56. Invariantes del refinamiento 3 | 65 |
| 6.57. Inicialización de las variables en el refinamiento 3 | 66 |
| 6.58. Refinamiento del evento que permite crear un producto de reventa | 66 |
| 6.59. Refinamiento del evento que permite crear un conjunto de productos de reventa | 67 |
| 6.60. Evento que permite agregar una categoría | 67 |
| 6.61. Evento que permite remover una categoría | 68 |
| 6.62. Contexto del refinamiento 4 | 68 |
| 6.63. Variables del refinamiento 4 | 69 |
| 6.64. Invariantes del refinamiento 4 | 69 |

| | |
|---|-----|
| 6.65. Inicialización de las variables en el refinamiento 4 | 69 |
| 6.66. Evento que permite crear un impuesto | 70 |
| 6.67. Evento que permite eliminar un impuesto | 70 |
| 6.68. Evento que permite actualizar el porcentaje de un impuesto | 71 |
| 6.69. Evento que permite actualizar la descripción de un impuesto | 71 |
| 6.70. Evento que permite asociar un impuesto a un producto | 71 |
| 6.71. Evento que permite asociar un impuesto a una lista de productos | 72 |
| 6.72. Evento que permite asociar un impuesto a un conjunto de productos asociados por una categoría | 72 |
| 6.73. Variables del refinamiento 5 | 73 |
| 6.74. Invariantes del refinamiento 5 | 73 |
| 6.75. Inicialización de las variables en el refinamiento 5 | 73 |
| 6.76. Evento que permite obtener el precio de venta de un producto | 74 |
| 6.77. Evento que permite obtener el precio de venta con su respectivo impuesto | 74 |
| 6.78. Evento que permite calcular el valor total de un producto en una orden de venta | 75 |
| | |
| B.1. Comparación del método <i>contains</i> de la clase BSet Java y JML. | 117 |
| B.2. Comparación del método <i>is proper subset</i> de la clase BSet Java y JML. | 118 |
| B.3. Comparación del método <i>is proper superset</i> de la clase BSet Java y JML. | 118 |
| B.4. Comparación del método <i>is subset</i> de la clase BSet Java y JML. | 118 |
| B.5. Tiempos para el método <i>is subset</i> del clase BSet Java. | 118 |
| B.6. Comparación del método <i>is superset</i> de la clase BSet Java y JML. | 119 |
| B.7. Tiempos para el método <i>is superset</i> del clase BSet Java. | 120 |
| | |
| C.1. Comparación del método <i>difference</i> de la clase BRelation Java y JML. | 121 |
| C.2. Tiempos para el método <i>difference</i> del clase BRelation Java. | 121 |
| C.3. Comparación del método <i>intersection</i> de la clase BRelation Java y JML. | 122 |
| C.4. Tiempos para el método <i>intersection</i> del clase BRelation Java. | 122 |
| C.5. Comparación del método <i>inverse</i> de la clase BRelation Java y JML. | 122 |
| C.6. Tiempos para el método <i>inverse</i> del clase BRelation Java. | 122 |
| C.7. Comparación del método <i>range</i> de la clase BRelation Java y JML. | 123 |
| C.8. Tiempos para el método <i>range</i> del clase BRelation Java. | 123 |
| C.9. Comparación del método <i>union</i> de la clase BRelation Java y JML. | 123 |
| C.10. Tiempos para el método <i>union</i> del clase BRelation Java. | 123 |
| C.11. Comparación del método <i>domain</i> de la clase BRelation Java y JML. | 125 |

Índice de cuadros

| | | |
|-------|---|-----|
| 5.1. | Tiempos obtenidos para el método insertar de BSet Java y JML. | 26 |
| 5.2. | Tiempos obtenidos para el método intersección (<i>union</i>) de BSet Java y JML. | 28 |
| 5.3. | Tiempos obtenidos para el método intersección (<i>intersection</i>) de BSet Java y JML. | 28 |
| 5.4. | Tiempos obtenidos para el método diferencia (<i>difference</i>) de BSet Java y JML. | 29 |
| 5.5. | Tiempos obtenidos para el método inserción (<i>insert</i>) de BRelation Java y JML. | 29 |
| 6.1. | Tiempos para ingreso de datos en la vista <i>Categories</i> | 77 |
| 6.2. | Tiempos para ingreso de datos en la vista <i>Customers</i> | 77 |
| 6.3. | Tiempos para ingreso de datos en la vista <i>Warehouse</i> | 77 |
| 6.4. | Tiempos para ingreso de datos en la vista <i>Taxes</i> | 78 |
| 6.5. | Tiempos para ingreso de datos en la vista <i>Products</i> | 78 |
| 6.6. | Tiempos para ingreso de datos en la vista <i>Sale</i> | 78 |
| 6.7. | Tiempos para ingreso de datos en la vista <i>Categories</i> en Mac OS X | 78 |
| 6.8. | Tiempos para ingreso de datos en la vista <i>Customers</i> Mac OS X | 79 |
| 6.9. | Tiempos para ingreso de datos en la vista <i>Warehouse</i> en Mac OS X | 79 |
| 6.10. | Tiempos para ingreso de datos en la vista <i>Taxes</i> en Mac OS X | 79 |
| 6.11. | Tiempos para ingreso de datos en la vista <i>Products</i> en Mac OS X | 79 |
| 6.12. | Tiempos para ingreso de datos en la vista <i>Sales</i> en Mac OS X | 80 |
| B.1. | Tiempos obtenidos para el método diferencia (<i>difference</i>) de BSet Java y JML. | 117 |
| B.2. | Tiempos obtenidos para el método <i>is proper subset</i> de BSet Java y JML. . | 119 |
| B.3. | Tiempos obtenidos para el método <i>is proper superset</i> de BSet Java y JML. | 119 |
| B.4. | Tiempos obtenidos para el método <i>is subset</i> de BSet Java y JML. . . . | 119 |
| B.5. | Tiempos obtenidos para el método <i>is superset</i> de BSet Java y JML. . . . | 120 |
| C.1. | Tiempos obtenidos para el método <i>difference</i> de BRelation Java y JML. . | 124 |
| C.2. | Tiempos obtenidos para el método <i>intersection</i> de BRelation Java y JML. | 124 |
| C.3. | Tiempos obtenidos para el método <i>inverse</i> de BRelation Java y JML. . | 124 |
| C.4. | Tiempos obtenidos para el método <i>range</i> de BRelation Java y JML. . . . | 124 |
| C.5. | Tiempos obtenidos para el método <i>union</i> de BRelation Java y JML. . . . | 125 |
| C.6. | Tiempos obtenidos para el método <i>domain</i> de BRelation Java y JML. . | 125 |

Abreviaciones

JML Java Modeling Language (Lenguaje de modelaje para Java)

IDE Integrated Development Environment (Entorno de desarrollo integrado)

SDK Software Development Kit (Kit de desarrollo de software)

POS Point Of Sale (Punto de venta)

Símbolos

- \times Producto cartesiano
- \lhd Restricción de dominio
- \rhd Restricción de rango
- \cap Intersección de conjuntos
- \cup Unión de conjuntos
- \in Pertenece al conjunto
- \notin No pertenece al conjunto
- \subseteq Es un subconjunto
- \mathbb{N} Números naturales
- \mathbb{N}^* Números positivos
- \rightarrow Función total
- \leftrightarrow Relación
- $::=$ Asignación

Me gustaría dedicar mi tesis a mi papá quien desde el cielo vela por mí, a mi mama quien me ha apoyado y guiado en los momentos difíciles y a mis amigos que me regalaron su voz de aliento cuando lo necesitaba.

Capítulo 1

Introducción

Los métodos formales de desarrollo se caracterizan por emplear técnicas matemáticas para el análisis, construcción y verificación de aplicaciones, esto permite expresar las especificaciones de un software en lógica matemática; en la actualidad se realizan esfuerzos importantes para introducir ciertos niveles de automatismo al proceso de construcción de software, al unir estos dos aspectos se puede contribuir de una forma significativa a los procesos de desarrollo de software y a los ingenieros relacionados con esta área de la informática [3].

Uno de los métodos formales más importantes es el método B [1], el cual fue originalmente desarrollado para razonar sobre programas concurrentes. Event-B [2] es una evolución del método B que es más adecuado para el desarrollo grandes sistemas. Para realizar los diseños en Event-B se cuenta con una potente herramienta llamada Rodin [6, 13], la cual viene dotada con un conjunto de herramientas para trabajar con los modelos, como por ejemplo el editor, el generador de las obligaciones de prueba y los probadores.

El desarrollo de software en Event-B comienza con una especificación abstracta de los requerimientos de sistema y va siendo refinado a través de varios pasos para llegar a una descripción concreta del sistema, a medida que se va refinando el sistema se generar casos de prueba que deben ser verificados matemáticamente.

En este documento se presenta un caso de estudio sobre la herramienta EventB2Java, la cual busca traducir un modelo en Event-B a código Java mas su especificación en JML [11]. JML es un lenguaje de especificación de interfaz, esto significa, que permite generar la documentación técnica y describir el comportamiento de una clase Java. La herramienta da la posibilidad al usuario de traducir los modelos típicos de Event-B o los que han sido descompuestos.

Las especificaciones JML están compuestas de invariantes, pre y post condiciones, estas se escriben como comentarios encima del método u elemento que se quiere especificar, JML hace uso del diseño mediante la técnica del contrato (DBC). Este tema se aborda en mas profundidad en el capítulo 3 de este documento.

El código generado con la herramienta presenta varias ventajas para los desarrolladores, primero se cuenta con código que ha sido probado matemáticamente y que esta correcto desde fases tempranas de desarrollo, además se encuentra documentado lo cual describe en detalle el funcionamiento del mismo y finalmente este se puede verificar usando las herramientas disponibles para JML.

Finalmente, en este trabajo se presenta un caso de estudio que tiene como objetivo medir el rendimiento y desempeño del código generado, este consiste en el modelado de un sistema de inventarios en Event-B al cual se le generará el código Java con su correspondiente especificación, por ultimo, este se usara para modificar el software Openbravo, este aplicativo open source permite llevar el control de inventario de un comercio, además de brindar la posibilidad de procesar ventas y compras de productos. Una vez se obtenga una versión que es funcionalmente ideática a la original se procederá a ejecutar una serie de pruebas con el fin de determinar ventajas y falencias del código generado.

Capítulo 2

Preliminares

2.1. Método Event-B

El metodo B [1], introducido por J.R Abrial, es una estrategia de desarrollo de software en el que un modelo abstracto de un sistema es transformado en una implementación por medio de una serie de refinamientos, donde el comportamiento de cada uno de los refinamientos es demostrablemente coherente con el comportamiento de la etapa anterior. cada refinamiento agrega mas detalles a la descripción del sistema. un refinamiento genera obligaciones de prueba que deben ser formalmente verificadas para afirmar que un modelo M_{i+1} es de hecho un refinamiento de un modelo anterior M_i . Estas obligaciones de prueba son necesarias y condiciones suficientes para garantizar que, aunque a diferentes niveles de abstracción, ambos son modelos de un mismo sistema¹.

Un derivado del método B, también introducido por J.R Abrial, se llama Event-B [2]. Modelos Event-B son desarrollos completos de sistemas de transición discretas. Estos se componen de máquinas y contextos. Las máquinas contienen la partes dinámicas de un modelo (por ejemplo, variables, invariantes, eventos). Los contextos contienen la parte estática de un modelo (por ejemplo, carrier sets, constantes).

Un ejemplo de un evento que crea una cuenta en una red social se representa en la Figura 2.1. PERSON es un carrier set que representa el conjunto de todas las personas posibles en la red, y los contenidos del conjunto de todas las imágenes posibles, el texto y en el contenido en general en la red. personas es el conjunto de personas que realmente en la red, y CONTENTS es el conjunto de todas las posibles imágenes, texto y contenido en generar de la red. *persons* es el conjunto de personas actuales, *contents* es el contenido actual, *owner* es una función sobreyectiva total que mapea cada contenido con su

¹Traducido al español de [7]

propietario y *pages* es una relación total que indica que contenido es visible para las personas. El evento *create_account* agrega una nueva persona y asocia el contenido inicial a la red. A diferencia del Método B en el que las operaciones se llaman, Event-B define eventos que podrían ser ejecutados/activados cuando la guarda (la parte entre *where* y *then*) es verdadera. Por lo tanto, este evento puede ejecutar siempre que haya al menos una persona y al menos un contenido de elemento que aún no se ha añadido.

```

create_account
any p1 c1 where
  grd1 p1 ∈ PERSON \ persons
  grd2 c1 ∈ CONTENTS \ contents
then
  act1 contents := contents ∪ {c1}
  act2 persons := persons ∪ {p1}
  act3 owner := owner ∪ {c1 ↦ p1}
  act4 pages := pages ∪ {c1 ↦ p1}
  act5 viewp := viewp ∪ {c1 ↦ p1}
  act6 editp := editp ∪ {c1 ↦ p1}
end
```

FIGURA 2.1: Evento para crear una cuenta en una red social.

2.2. JML

JML [4, 5, 9, 12] es un lenguaje diseñado para la especificación de las interfaces de clases Java. Las especificaciones JML suelen ser incorporadas directamente en las implementaciones de clases Java usando marcadores de comentario especiales como `/*@ ... */` o `//@`. La especificación incluyen varias formas de invariantes y pre- y post-condiciones para los métodos. La sintaxis es intencionalmente similar a la de Java, por lo que es menos intimidante para los desarrolladores. En particular, los tipos matemáticos que son muy utilizado en otros lenguajes de especificaciones (conjuntos, secuencias, relaciones y funciones) se proporcionan en JML como clases, y las operaciones sobre esos tipos son especificados (en JML) e implementados como métodos Java².

Un especificación simple en JML para una clase Java consiste en pre- (`requires` en JML) y post-condiciones (`ensures` en JML) agregadas a los métodos, y la clase invariantes (`invariant` en JML) que restringen los posibles estados de instancias de clase. En JML las pre- y post-condiciones se incrustan como comentarios justo antes de la declaración del métodos. Los predicados en JML son aserciones lógicas de primer orden construidas a partir de expresiones booleanas en Java y distintos elementos de especificación en JML.

²Traducido al español de [7]

JML proporciona notaciones para implicaciones lógicas hacia adelante y hacia atrás, \Rightarrow y \Leftarrow , para la negación $!$, para la no equivalencia \Leftrightarrow y para la *o* (*or*) y la *y* (*and*) lógicas se usa \sqcap y \sqcup respectivamente. La notación en JML para los cuantificadores universal y existencial son $(\forall T x; E)$ y $(\exists T x; E)$, donde $T x;$ declara una variable x de tipo T , y E es la expresión que se debe mantener para cada valor de tipo T .

La Figura 2.2 presenta la traducción a JML del evento mostrado en la Figura 2.1. Un evento en Event-B puede ser activado (sus acciones pueden ser ejecutadas) cuando se satisface la guarda. Este comportamiento se modela mediante la creación de dos métodos: un método **guard** que contiene la traducción de la guarda, y un método **run** que contiene la traducción del cuerpo evento. En el método **guard**, **\result** representa el resultado devuelto por una llamada al método. En los estados del sistema en los que se satisfagan la guarda se ejecuta el método **run** el cual termina en un estado que satisface la traducción del cuerpo del evento. En los estados del sistema en los que no se satisface la guarda, el evento correspondiente no podría ser activado y así la ejecución del método **run** no debe llevarse a cabo. Esto se maneja mediante la traducción de un evento a dos casos de especificación en JML - uno donde se satisface la traducción de la guarda, y otro donde esto no ocurre. Esto se expresa en la pre-condición (cláusula **requires** en JML) de cada caso.

El primer caso de especificación para **run_create_account** (en el que **guard_create_account** retorna verdadero **true**) especifica el efecto de crear una cuenta nueva en el sistema. La cláusula **assignable** es una condición estructural que especifica qué ubicaciones pueden cambiar de el pre-estado hacia el post-estado, por ejemplo, para este caso todos los campos representan variables en Event-B. El pre-estado es el estado de entrada al método y la post-estado es el estado de salida del método. Existen dos especificaciones especiales de **assignable**, **assignable \nothing**, la cual especifica que no van a ocurrir cambios en las ubicaciones y **assignable \everything**, el cual especifica que se va a modificar todo. La post-condición (**ensures** en JML) expresa que en el estado de salida del método, la persona $p1$ se añade al sistema con la propiedad, visibilidad y todos los permisos del contenido $c1$. Las expresiones entre \old son evaluados en el pre-estado, mientras que todas las demás expresiones se evalúan en el post-estado. En el segundo caso de especificación, no hay asignaciones (**assignable**), por lo que el post-estado es igual al pre-estado cuando **guard_create_account** retorna falso en el pre-estado.

```

/*@ assignable \nothing;
ensures \result <==> (\exists Integer c1; (\exists Integer p1;
  (PERSON.difference(persons).has(p1) &&
   CONTENTS.difference(contents).has(c1))); */
public abstract boolean guard_create_account();

/*@ requires guard_create_account();
assignable contents, persons, owner, pages, viewp, editp;
ensures (\exists Integer c1; (\exists Integer p1;
  \old((PERSON.difference(persons).has(p1) &&
         CONTENTS.difference(contents).has(c1)))
  && contents.equals(\old(contents.union((new BSet<Integer>(c1)))))
  && persons.equals(\old(persons.union((new BSet<Integer>(p1)))))
  && owner.equals(\old(owner.union(
    new BRelation<Integer, Integer>().singleton(c1, p1))))
  && pages.equals(\old(pages.union(
    new BRelation<Integer, Integer>().singleton(c1, p1))))
  && viewp.equals(\old(viewp.union(
    new BRelation<Integer, Integer>().singleton(c1, p1))))
  && editp.equals(\old(editp.union(
    new BRelation<Integer, Integer>().singleton(c1, p1))))));
also
  requires !guard_create_account();
  assignable \nothing;
  ensures true; */
public abstract void run_create_account();

```

FIGURA 2.2: Especificación JML producida para la creación de una cuenta en una red social.

Capítulo 3

Problemática

3.1. Planteamiento del problema

En un mundo cambiante como el actual, donde la industria está ávida de aplicaciones que suplan las necesidades en tiempos cortos, se ha dejado a un lado el desarrollo formal de software optando por metodologías ágiles u otras herramientas, esto debido a la versatilidad y la rapidez con la que se puede construir un aplicativo. Esto supone un gran problema cuando se tratan de aplicaciones de misión critica o software que tiene que estar correcto desde fases tempranas del desarrollo.

Uno de los grandes problemas que enfrenta la industria es el distanciamiento entre los expertos en modelamiento de sistemas usando métodos formales (los cuales siguen la metodología correctness-by construction¹) y los implementadores, en [3] se aborda esta problemática mostrando cómo se han llevado a cabo esfuerzos para mezclar lo mejor de ambos métodos el ágil y formal. Como puntos de unión se plantean usar métodos formales para el diseño de casos de uso, verificación de arquitecturas de software, pruebas unitarias y desarrollo incremental. Hacia el final del artículo los autores concluyen que no hay motivos para que no se tome lo mejor de ambos mundos y se unan para obtener metodologías de desarrollo de software que permitan llevar a buen término aplicaciones cumpliendo con el tiempo y la calidad esperada.

Otro de los posibles puntos de unión que no se toco anteriormente es la documentación del código. Cuando se trabaja con métodos ágiles se puede caer en el error de no documentar bien el código o en el peor de los casos no documentar, esto puede ser resuelto con metodologías como design-by-contract, en un caso concreto Java Modeling Language (JML).

¹Esta metodología [10] requiere de una definición rigurosa de requerimientos, de un diseño sólido y verificable y un código del cual se comprende su funcionamiento.

Al agregar automatismo al proceso de ir de un método formal a una implementación de código que pueda ser usada por los desarrolladores y que además cuenta con la ventaja de estar documentado y que puede ser verificado, se estaría minimizado la brecha entre estos dos elementos importantes en el desarrollo de software.

Lo que busca este trabajo es poder generar código Java con especificación JML a partir de una modelo en Event-B, logrando con ello unir las dos metodologías de desarrollo de software correctness-by-construction y design-by-contract. Un desarrollo de este tipo permitiría obtener código que esta documentado, es decir, que explica por medio de las pre y post-condiciones y de los invariantes el funcionamiento de cada sección del mismo, por otra parte, este conservará todas las propiedades y bondades que provee un método formal.

3.2. Objetivo general

El objetivo general de esta tesis es desarrollar un caso de estudio que permita evaluar y mejorar el rendimiento del generador de código EventB2Java.

3.3. Objetivos específicos

- Mejorar la implementación de las estructuras de datos de la herramienta EventB2Java.
- Diseñar una prueba que permita medir el rendimiento de las estructuras de datos una vez modificadas, usando como punto de referencia la implementación anterior.
- Modelar un caso de estudio en Event-B y usar la herramienta EventB2Java para obtener código Java+JML a partir de el.
- Evaluar el rendimiento de la herramienta EventB2Java.

Capítulo 4

Traducción de un Modelo en Evnet-B a Java+JML

En este capitulo se presentan las reglas de traducción que permiten ir de Evnet-B a código Java+JML. La primera versión fue escritas en conjunto con Víctor Rivera y revisadas por Néstor Cataño, en la cual mi porcentaje de colaboración fue del 10 %. Este trabajo inicial fue refinado posteriormente por Víctor y Néstor hasta llegar a una versión final de las reglas [8].

Los siguientes operadores fueron definidos:

El operador **EB2Jml+Java** traduce la sintaxis de Event-B a código Java mas especificación JML.

EB2Jml va en la misma dirección que **EB2Jml+Java** pero genera solo especificación JML. Por lo tanto, se usa para traducir los invariantes.

EB2Java este operador solo traduce a código Java. Por lo tanto, es usado para traducir elementos de clase.

Operadores **TypeOf** - **PRED** - **MOD** trabajan de manera similar a los definidos previamente. **TypeOf** traduce una variable en Evnet-B a su homólogo en Java (La cual es la misma traducción para la especificación JML). **PRED** traduce un predicado de Event-B a uno en JML. Y por ultimo, **MOD** calcula el conjunto de variables modificadas por un evento específico.

4.1. Versión inicial de las reglas

Traduciendo una maquina: Una máquina se traslada a una clase Java y cada uno de los eventos es trasladado como una clase interna que extienden de la clase Thread de Java. La idea es ejecutar los métodos (la traducción de los eventos) en un entorno concurrente. Cuando la guarda de un método es evaluada y es verdadera, el hilo específico toma el control de la ejecución y lleva a cabo sus acciones en una forma atómica.

La regla M es la encargada de traducir una Maquina de Event-B, en ella se usa el resultado de traducir los conjuntos, constantes y variables, ademas de inicializar cada uno de los hilos.

$$\begin{aligned}
 EB2Java + Jmlsets s &= S \quad EB2Java + Jml(constants c) = C \\
 EB2Jml((axioms X(s, c))) &= X \quad EB2Jml((theorems T(s, c))) = T \\
 EB2Java + Jml(variables v) &= V \quad EB2Jml((invariants I(s, c, v))) = I \\
 EB2Java + Jml(events e) &= E \quad ThreadCreation = ThreadCreation \\
 EB2Jml((events initialisation begin A(s, c, v) end)) &= I1 \\
 EB2Java((events initialisation begin A(s, c, v) end)) &= I2
 \end{aligned} \tag{M}$$

EB2Java + Jml(machine M sees C

variables v

invariants I(s, c, v)

events e

end) =

E

public class MVariables{

S C V

//@ ensures I1;

MVariables(){

I2

}

}

public class M{

X T I

MVariable vars;

M(){

vars = new MVariable();

//ThreadCreation

}

}

Carrier Sets, Constantes y Variables: Los *carrier sets* de Event-B, constantes y variables son traducidos a JML con sus restricciones de especificación. Adicionalmente, se genera el código Java. Si no se posee información adicional de los *carrier sets* estos serán traducidos a conjuntos de enteros (*Integers*).

Las constantes se comportan de la misma manera que los *carrier set* pero en ellos se define un tipo de dato específico.

$$\begin{array}{c}
 \hline
 EB2Java + Jml(\text{sets } s) = \\
 / * @ \text{public model } BSet < \text{Integer} > \ s; \\
 \quad \text{public constraint } s.equals(\backslash \text{old}(s)); * / \\
 \\
 \textcolor{red}{public static final } BSet < \text{Integer} > \ s; \\
 \\
 \hline
 TypeOf(c) = \text{Type} \\
 EB2Java + Jml(\text{constants } s) = \\
 / * @ \text{public model } Type \ c; \\
 \quad \text{public constraint } c.equals(\backslash \text{old}(c)); \ * / \\
 \\
 \textcolor{red}{public static final } Type \ c;
 \end{array} \quad (\text{Set}) \quad (\text{Cons})$$

Las variables son traducidas como *model variables* en JML y poseen un tipo específico, además de ser públicas.

$$\begin{array}{c}
 \hline
 TypeOf(c) = \text{Type} \\
 EB2Java + Jml(\text{variables } v) = \\
 // @ \text{public model } Type \ v; \\
 \\
 \textcolor{red}{public } Type \ v;
 \end{array} \quad (\text{Var})$$

Axiomas, teoremas e invariantes: Los axiomas, teoremas e invariantes en Event-B son traducidos a JML como cláusulas en los invariantes como se indica abajo. El código Java requiere esta información para propósitos de verificación.

$$\begin{array}{c}
 \hline
 Pred(X(s, c)) = \text{X} \\
 EB2Jml((\text{axioms } X(s, c))) = \\
 // @ \text{public invariant } X;
 \end{array} \quad (\text{Axiom})$$

$$\frac{\text{Pred}(T(s, c)) = \text{T}}{\begin{aligned} EB2Jml(\text{axioms } T(s, c)) &= \\ // @ \text{public invariant redundantly } T; \end{aligned}} \quad (\text{Thm})$$

$$\frac{\text{Pred}(I(s, c, v)) = \text{I}}{\begin{aligned} EB2Jml(\text{invariants } I(s, c, v)) &= \\ // @ \text{public invariant } I; \end{aligned}} \quad (\text{Inv})$$

Hay dos tipos de eventos: los de inicialización , en los cuales se indican los valores iniciales de las variables y los que manejan la evolución del sistema. Un evento de inicialización es traducido a Java como el constructor de la clase que representa la maquina. Estos también son traducidos a especificación JML como pos-condición del constructor.

$$\frac{\begin{aligned} EB2Jml((A(s, c, v))) &= \text{A} \\ EB2Jml((\text{events initialisation begin } A(s, c, v) \text{ end})) &= \\ // @ \text{public initially } A; \end{aligned}}{} \quad (\text{Init1})$$

$$\frac{\begin{aligned} EB2Java((A(s, c, v))) &= \text{A} \\ EB2Java((\text{events initialisation begin } A(s, c, v) \text{ end})) &= \\ A; \end{aligned}}{} \quad (\text{Init2})$$

$$\frac{\text{Pred}(E(s, c, v)) = \text{E}}{EB2Jml(v := E) = v.equals(\text{old}(E))} \quad (\text{AsgJml})$$

$$\frac{\begin{aligned} \text{Pred}(E(s, c, v)) &= \text{E} \\ \text{TypeOf}(v) &= \text{Type} \end{aligned}}{\begin{aligned} EB2Java(v := E) &= \\ \text{Type } v \text{ tmp } &= v; \\ v &= E; \end{aligned}} \quad (\text{AsgJava})$$

$$\frac{\text{Pred}(P(s, c, v, v')) = \text{P} \text{ TypeOf}(v) = \text{Type}}{\begin{aligned} EB2Jml(v : j P) &= \\ (\backslash \text{exists } \text{Type } v0; \text{ old}(P) \&\& v.equals(v')) \end{aligned}} \quad (\text{NAsgJml})$$

Con respecto a los otros eventos, que fueron mencionados previamente, son los referentes a los que manejan la evolución del sistema que se traducen como clases independientes que se ejecutan de manera concurrente.

$TypeOf(x) = \text{Type } Pred(G(s, c, v, x)) = \mathbf{G} \ Mod(A(s, c, v, x)) = \mathbf{M}$
 $EB2Jml((A(s, c, v, x))) = \mathbf{A} \ EB2Java((A(s, c, v, x))) = \mathbf{AA}$ (Any)

$EB2Java + Jml((events evt any x where) G(s, c, v, x)$
 $then A(s, c, v, x) end) =$
 $\text{class evt extends Thread}{$
 $Machine m; \ public int eid;$
 $public evt(Machine ma; int i){$
 $eid = i;$
 $m = ma;$
 $}$
 $/* @ assignable \nnothing;$
 $ensures \result \iff (\exists Type x; G); */$
 $public boolean guard(){$
 $return G;$
 $}$
 $/* @ requires guard();$
 $assignable M;$
 $ensures (\exists Type x; \text{old}(G) \&\& A); */$
 $also$
 $requires !guard();$
 $assignable \nnothing;$
 $ensures \text{true}; */$
 $public void execute(){$
 $if (guard())\{ AA \}$
 $}$
 $public void run(){$
 $while(\text{true}){$
 $...$
 $if (guard())\{$
 $machine.util.lock(eid);$
 $execute();$
 $machine.util.unlock(eid);$
 $}$
 $...$
 $}$
 $}$

$$\begin{array}{c}
 Pred(G(s, c, v)) = \texttt{G} \ Mod(A(s, c, v)) = \texttt{M} \\
 EB2Jml((A(s, c, v))) = \texttt{A} \ EB2Java((A(s, c, v, x))) = \texttt{AA} \\
 \hline
 EB2Java + Jml((events evt when) G(s; c; v) \\
 thenA(s, c, v) end) = \\
 \textcolor{red}{class} evt \textcolor{red}{extends} Thread\{ \\
 \quad Machine m; \textcolor{red}{public} \textcolor{teal}{int} eid; \\
 \quad \textcolor{red}{public} evt(Machine ma; \textcolor{teal}{int} i)\{ \\
 \quad \quad eid = i; \\
 \quad \quad m = ma; \\
 \quad \quad \} \\
 \quad /* @ assignable \nothing; \\
 \quad \quad ensures \result <==> G; */ \\
 \quad \textcolor{red}{public} \textcolor{blue}{boolean} guard()\{ \\
 \quad \quad \textcolor{red}{return} G; \\
 \quad \quad \} \\
 \quad /* @ requires guard(); \\
 \quad \quad assignable M; \\
 \quad \quad ensures (Type x; \old(G) \&& A); */ \\
 \quad \textcolor{blue}{also} \\
 \quad \quad \textcolor{blue}{requires} !guard(); \\
 \quad \quad assignable \nothing; \\
 \quad \quad \textcolor{blue}{ensures} true; /* \\
 \quad \quad \textcolor{red}{public} \textcolor{teal}{void} execute()\{ \\
 \quad \quad \quad \textcolor{red}{if} (guard())\{ AA \} \\
 \quad \quad \quad \} \\
 \quad \quad \textcolor{red}{public} \textcolor{teal}{void} run()\{ \\
 \quad \quad \quad \textcolor{red}{while}(true)\{ \\
 \quad \quad \quad \quad ... \\
 \quad \quad \quad \quad \textcolor{red}{if} (guard())\{ \\
 \quad \quad \quad \quad machine.util.lock(eid); \\
 \quad \quad \quad \quad execute(); \\
 \quad \quad \quad \quad machine.util.unlock(eid); \\
 \quad \quad \quad \quad \} \\
 \quad \quad \quad \quad ... \\
 \quad \quad \quad \quad \} \\
 \quad \quad \quad \} \\
 \quad \quad \} \\
 \quad \} \\
 \}
 \end{array}$$

(When)

4.2. Versión final de las reglas de traducción

En la sección anterior (4.1) se presento el primer acercamiento que se dio a las reglas de traducción, estas fueron refinadas y en [8] fueron presentadas. En este articulo se presenta un nuevo operador **EB2Prog** el cual traduce una maquina en Event-B y cualquier contexto que ella vea a una clase Java con su especificación JML. **EB2Prog** depende de **EB2Java** que produce solo código Java y de **EB2Jml** que produce solo especificación JML. Por ejemplo una variable de maquina en Event-B es traducida como un atributo de clase, un invariante de maquina es traducido a un invariante de clase en JML y una constante es traducida como un atributo de clase que es limitado por un **constraint** JML.

La regla **M** traduce una maquina M que ve un contexto C . Se deben descargar todas las obligaciones de prueba antes de proceder con la traducción. Se deja la posibilidad de traducir a código Java previo a haber descargado las obligaciones puesto que esto resulta conveniente cuando se esta experimentando con modelos que están en fase de construcción.

Como ocurría en la primera versión los eventos de inicializacion son traducidos como el constructor de la clase Java obtenida de traducir la maquina. El constructor incluye la post-condicion JML para los valores iniciales de los atributos de clase. Cualquier otro evento es traducido como una clase Java que extiende de Thread y que incluye una referencia a la implementacion de la clase resultante de traducir la maquina (Ver la regla **Any** mas abajo).

Las guardas de los eventos es ejecutada concurrentemente, cuando una de las guardas es verdadera, ese evento podrá ejecutar las acciones. Solo puede haber un evento al tiempo en ejecución. La implementación de estos eventos esta compuesta por 3 métodos, *guard_evt* el cual evalúa las guardas, *run_evt* el cual modela la ejecución del mismo y el *run()* que es la sobreescritura del correspondiente a la clase Thread y en el cual se lleva a cabo la implementación de la sección critica de Lamport's Bakery. La expresión *GuardValue < Type > .next()* retorna un valor de tipo *Type* que satisface la guarda del evento.

En la regla **Any**, las variables definidas en la sección *any* del evento en Event-B es traducida como un parámetro de los métodos *run_evt* y *guard_evt*. El operador **Mod** calcula el conjunto de variables asignadas por la acción del evento. El operador **Pred** traduce los predicados de Event-B a su homologo en JML.

La especificación JML del metodo *run_evt* contiene dos casos. En el primero, la traducción de la guarda es satisfecha y el post-estado del método deberá satisfacer la traducción de las acciones. En el segundo caso, la traducción de la guarda no se satisface y el método

no está disponible a ejecutarse ni modificar ningún campo, asegurando que el post-estado es el mismo que el pre-estado. Estos dos estados están separados en JML por la palabra **also**

$$\begin{array}{l}
 EB2Prog(\text{sets } s) = \text{S} \quad EB2Prog(\text{constants } c) = \text{C} \\
 EB2Jml(\text{axioms } X(s, c)) = \text{X} \quad EB2Jml(\text{theorems } T(s, c)) = \text{T} \\
 EB2Java(\text{variables } v) = \text{V} \quad EB2Jml(\text{invariants } I(s, c, v)) = \text{I} \\
 EB2Prog(\text{events } e) = \text{E} \\
 EB2Jml(\text{event initialisation then } A(s, c, v) \text{ end}) = \text{I1} \\
 EB2Java(\text{event initialisation then } A(s, c, v) \text{ end}) = \text{I2} \\
 \hline
 EB2Prog(\text{machine } M \text{ sees } C \\
 \quad \text{variables } v \\
 \quad \text{invariants } I(s, c, v) \\
 \quad \text{event initialisation then } A(s, c, v) \text{ end} \\
 \quad \text{events } e \\
 \quad \text{end}) = \\
 \quad E \\
 \quad \textcolor{red}{public class } M \{ \\
 \quad \quad X \; T \; I \\
 \quad \quad S \; C \; V \\
 \quad \quad / * @ \textcolor{blue}{requires} \textcolor{red}{true}; \\
 \quad \quad \textcolor{blue}{assignable} \setminus \text{everything}; \\
 \quad \quad \textcolor{blue}{ensures} \; I1; \; * / \\
 \quad \quad \textcolor{red}{public} \; M() \{ \\
 \quad \quad \quad I2 \\
 \quad \quad \quad // \textcolor{blue}{Java code that spawns all events} \\
 \quad \quad \quad \} \\
 \quad \quad \} \\
 \end{array} \tag{M}$$

$EB2Jml(A(s, c, v, x)) = A \quad EB2Java(A(s, c, v, x)) = B$
 $TypeOf(x) = Type \quad Pred(G(s, c, v, x)) = G \quad Mod(A(s, c, v, x)) = D$ (Any)

```

 $EB2Prog(event evt any x where G(s, c, v, x)$   

 $then A(s, c, v, x) end) =$   

 $\textcolor{red}{public class} evt \textcolor{red}{extends} Thread \{$   

 $\textcolor{blue}{private} \textcolor{teal}{M} m; \textcolor{blue}{private} \textcolor{teal}{int} eventId;$   

 $\textcolor{blue}{/* @ requires true;}$   

 $\textcolor{blue}{assignable} \backslash everything;$   

 $\textcolor{blue}{ensures} this.m == m \&& this.event == i; */$   

 $\textcolor{blue}{public} evt(\textcolor{teal}{M} m, \textcolor{teal}{int} i) \{$   

 $\textcolor{red}{this}.m = m; \textcolor{red}{this}.event = i;$   

 $\}$   

 $\textcolor{blue}{/* @ requires true; \ assignable \nothing;}$   

 $\textcolor{blue}{ensures} \backslash result <==> G; */$   

 $\textcolor{blue}{private} \textcolor{teal}{boolean} guardEvt(\textcolor{teal}{Type} x) \{ \textcolor{red}{return} G; \}$   

 $\textcolor{blue}{/* @ requires guardEvt(x);}$   

 $\textcolor{blue}{assignable} D; \textcolor{blue}{ensures} A;$   

 $\textcolor{blue}{also}$   

 $\textcolor{blue}{requires} !guardEvt(x); \textcolor{blue}{assignable} \backslash nothing;$   

 $\textcolor{blue}{ensures} true; */$   

 $\textcolor{blue}{private} \textcolor{red}{void} runEvt(\textcolor{teal}{Type} x)\{$   

 $\textcolor{red}{if} (\text{guard\_evt}(x)) \{ B \}$   

 $\}$   

 $\textcolor{blue}{public} \textcolor{red}{void} run()\{$   

 $\textcolor{blue}{while}(\textcolor{red}{true}) \{$   

 $\dots$   

 $\textcolor{teal}{Type} x = \text{GuardValue} < \textcolor{teal}{Type} >.next();$   

 $\textcolor{red}{if} (\text{guard\_evt}(x))\{$   

 $m.\text{util}.lock(event);$   

 $\text{runEvt}(x);$   

 $m.\text{util}.unlock(event);$   

 $\}$   

 $\dots$   

 $\}$   

 $\}$   

 $\}$ 

```

$$\frac{\text{Pred}(I(s, c, v)) = I}{\begin{aligned} EB2Jml(\text{invariants } I(s, c, v)) &= \\ // @ public invariant I; \end{aligned}} \text{(Inv)}$$

$$\frac{\text{Pred}(X(s, c)) = X}{\begin{aligned} EB2Jml(\text{axioms } X(s, c)) &= \\ // @ public invariant X; \end{aligned}} \text{(Axiom)}$$

$$\frac{\text{Pred}(T(s, c)) = T}{\begin{aligned} EB2Jml(\text{theorems } T(s, c)) &= \\ // @ public invariant_redundantly T; \end{aligned}} \text{(Thm)}$$

El concepto de los *carrier set* y las constantes es igual que en la sección anterior, salvo la aplicación de los operadores, en la sección anterior se usaba *EB2Java + JML* y se realizaba la traducción a Java con su correspondiente especificación JML, ahora estas operaciones se llevan a cabo por separado y *carrier set* es inicializado en un rango determinado.

$$\frac{}{\begin{aligned} EB2Java(\text{sets } s) &= \\ \text{public static final } BSet < \text{Integer} > s &= \\ \text{new } BSet < \text{Integer} > (\text{new Range(Integer.MIN_VALUE, Integer.MAX_VALUE)}); \end{aligned}} \text{(Set)}$$

$$\frac{}{EB2Jml(\text{sets } s) = // @ public constraint s.equals(\text{\old}(s));} \text{(Set)}$$

$$\frac{}{EB2Jml(\text{constants } c) = // @ public constraint c.equals(\text{\old}(c));} \text{(Cons)}$$

$$\frac{\text{TypeOf}(c) = \text{Type } v = \text{Value} < \text{Type} > .\text{next}()}{EB2Java(\text{constants } c) = \text{public static final Type } c = v;} \text{(Cons)}$$

$$\frac{\text{TypeOf}(v) = \text{Type}}{EB2Java(\text{variables } v) = /* @ spec_{public} */ \text{private Type } v;} \text{(Var)}$$

$$\frac{\text{Pred}(E(s, c, v)) = E}{\text{EB2Jml}(v := E) = v.equals(\text{old}(E));} \text{ (Asg)}$$

$$\frac{\text{Pred}(E(s, c, v)) = E}{\begin{aligned} \text{EB2Java}(v := E) &= \\ v &= E; \end{aligned}} \text{ (Asg)}$$

$$\frac{\text{Pred}(P(s, c, v, v')) = P \text{ TypeOf}(v) = \text{Type}}{\begin{aligned} \text{EB2Jml}(v :| P) &= \\ (\backslash \text{exists } \text{Type } v'; \backslash \text{old}(P) \&& v.equals(v')) \end{aligned}} \text{ (NAsg)}$$

$$\frac{\text{Pred}(s1) = s1 \cdot \text{Pred}(s2) = s2}{\text{Pred}(s1 \subseteq s2) = s1.isSubset(s2)} \text{ (Subset)}$$

$$\frac{\text{Pred}(x) = x \text{ Pred}(s) = s}{\text{Pred}(x : s) = s.has(x)} \text{ (Has)}$$

$$\frac{\text{Pred}(r) = r \text{ Pred}(s) = s}{\text{Pred}(r[s]) = r.image(s)} \text{ (Image)}$$

4.3. La herramienta de traducción EventB2Java

La implementación de las reglas fue llevada a cabo por Víctor Rivera y probada con diferentes modelos. La herramienta se encuentra disponible en <http://poporo.uma.pt/Projects/favas/EventB2Java.html>. Para implementarla se uso el API de Rodin para recolectar todos los componentes de la maquina a ser traducidos como por ejemplo los carrier sets, axiomas, constantes, variables, invariantes y eventos, ademas de toda la información necesaria para las maquinas refinadas. Toda esta información se encuentra almacenada en la base de datos de Rodin y puede ser accedida usando el paquete org.eventb.core. Las expresiones y las declaraciones son convertidas y almacenadas como arboles de sintaxis, que pueden ser accedidos y recorridos usando la librería AST en el paquete org.eventb.core.ast. El API de Rodin también provee librerías para recorrer arboles y para agregar información a los nodos. La mayoría de la implementación de EventB2Java es realizado a través de una clase utilitaria que trae los invariantes de la maquina, el contexto con su información, los teoremas, axiomas, variables de la maquina abstracta

y las refinadas y de los eventos en general. También fueron implementados los conjuntos en Event-B y las relaciones como clases Java llamadas BSet y BRalation.

Una vez instalado el plug-in en Rodin, se puede ver un nuevo menú contextual al hacer click derecho sobre una maquina como se aprecia en la Figura 4.1, en este se presentan dos opciones: traducción a multi-hilos en la cual los eventos se ejecutan de manera concurrente y la secuencial en la que se debe hacer el llamado desde el código a cada clase que representa los eventos.

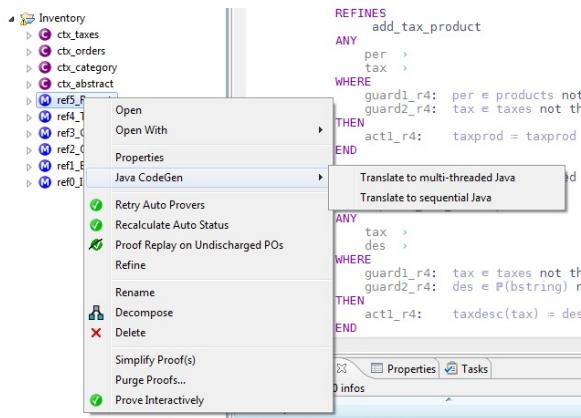


FIGURA 4.1: Menú contextual EventB2Java.

Una vez traducida la maquina deseada se creara una carpeta en el directorio del proyecto con el nombre `generated_java_NombreProyecto_sequential` en el caso de ser secuencial o terminada en `multi_threaded` en caso de ser concurrente.

En el interior de la carpeta se pueden ver los fuentes del proyecto, en el folder `src` se crean tres subdirectorios `eventb_prelude`, `Inventory_multi_threaded` y `Util`. En la primera se encuentran las clases que representan los objetos de Event-B como `BSet` y `BRelation`. En la segunda, esta la clase con el nombre de la maquina que se tradujo y que es el resultado de aplicar la regla de traducción M vista anteriormente, ademas de cada una de las clases resultantes de traducir los eventos. Finalmente, esta la clase `Utilities` en la que se encuentra la implementación del algoritmo de sección critica ademas de otras funciones necesarias para el correcto funcionamiento del proyecto.

4.4. Ejemplo de traducción

En la Figura 4.2 se muestra un evento en Event-B que permite crear una categoría de productos en un sistema de inventarios. Según la regla Any vista anteriormente las variables `cat` y `name` de la sección ANY del evento serán usadas como parámetros tanto en el método que evalúa la guarda como en el que ejecuta el cuerpo del evento, los nombres

son *guard* y *run* respectivamente. En el método *guard* se evaluara que la variable *cat* este contenida en el *carrier set CATEGORIES* substrayendo aquellos que ya fueron creados previamente en el sistema y que se encuentran en la variable de maquina *categories*, la segunda guarda indica que el parámetro *name* estará contenido en el conjunto resultante de hacer una combinatoria de la variable *bstring*. En la Figura 4.1 se presenta el código resultante de la traducción de manera secuencial. En el constructor de la clase se pasa una referencia a la maquina (Resultante de usar la regla M), con la cual se puede acceder a las variables u otros elementos en caso de ser necesario en cualquiera de los dos métodos.

```

add_category :
  ANY
    cat
    name
  WHERE
    guard1_r3: cat ∈ CATEGORIES \ categories
    guard2_r3: name ∈ P(bstring)
  THEN
    act1_r3: categories := categories ∪ cat
    act2_r3: categoryname := categoryname ∪ cat ↠ name
  END

```

FIGURA 4.2: Evento que permite agregar una categoría

```

package Inventory_sequential;

import eventb_prelude.*;
import Util.Utilities;

public class add_category{
    /*@ spec_public */ private ref5_Reports machine; // reference to the
    machine

    /*@ public normal_behavior requires true; assignable \everything;
    ensures this.machine == m; */
    public add_category(ref5_Reports m) {
        this.machine = m;
    }

    /*@ public normal_behavior requires true;
       assignable \nothing; ensures \result <==>
    (machine.CATEGORIES.difference(machine.get_categories()).has(cat) &&
     ((machine.bstring).pow()).has(name)); */
    public /*@ pure */ boolean guard_add_category( Integer cat,
        BRelation<Integer,Integer> name) {

```

```

        return
(machine.CATEGORIES.difference(machine.get_categories()).has(cat)
    && ((machine.bstring).pow()).has(name));
}

/*@ public normal_behavior
    requires guard_add_category(cat,name);
    assignable machine.categories, machine.categoryname;
    ensures guard_add_category(cat,name) &&

machine.get_categories().equals(\old((machine.get_categories().union(new
BSet<Integer>(cat)))))

&&

machine.get_categoryname().equals(\old((machine.get_categoryname().union(new
BRelation<Integer,BRelation<Integer,Integer>>
    (new Pair<Integer,BRelation<Integer,Integer>>(cat,name))))));
also
    requires !guard_add_category(cat,name);
    assignable \nothing;
    ensures true; */

public void run_add_category( Integer cat, BRelation<Integer,Integer>
name){
    if(guard_add_category(cat,name)) {
        BSet<Integer> categories_tmp =
machine.get_categories();
        BRelation<Integer,BRelation<Integer,Integer>>
categoryname_tmp = machine.get_categoryname();

        machine.set_categories((categories_tmp.union(new
BSet<Integer>(cat))));
        machine.set_categoryname((categoryname_tmp.union(new
BRelation<Integer,BRelation<Integer,Integer>>
    (new
Pair<Integer,BRelation<Integer,Integer>>(cat,name)))));

        System.out.println("add_category executed cat: " + cat
+ " name: " + name + " ");
    }
}
}

```

LISTADO 4.1: Clase resultante de traducir el evneto add_category.

Capítulo 5

Mejoramiento del desempeño de las clases BSet y BRelation

Como se menciono en el capitulo 4 al traducir un modelo en Event-B a codigo Java+JML fue necesario construir una serie de clases que permitirían trabajar con conjuntos de la misma manera que se hace en Event-B, estas clases están contenidas en la carpeta *eventb_prelude* de cada proyecto y se compone de la clase BOOL la cual permite trabajar con elementos verdaderos o falsos (true o false), NAT y NAT1 son clases que modelan el comportamiento de los números naturales y los naturales positivos respectivamente, BSet ayuda a trabajar con conjuntos y finalmente BRelation con relaciones entre conjuntos. La versión original de las clases BSet y BRelation fueron originalmente desarrolladas por Tim Wahls ¹, estas clases solo contenían código JML; posteriormente en colaboración y supervicion de Néstor fue posible escribir la version Java de las mismas.

El objetivo de probar y mejorar el desempeño de este paquete de nuevos objetos Java esta motivado por el hecho de brindar a los desarrolladores un framework adecuado para sus aplicaciones. No seria comprensible que un sistema modelado formalmente al momento de ser implementado presente un bajo rendimiento.

Es por esto que se diseño una prueba que permitiría medir el desempeño de la nueva implementación de estas clases respecto a su predecesora usada por la herramienta EventB2JML. La prueba consistía en crear objetos de tipo BSet o BRelation e insertar valores enteros entre 0 y 50000 para el primer tipo, y relaciones entre datos enteros también entre 0 y 50000 para el segundo tipo; Este proceso se llevaría a cabo tanto en la versión previa y la nueva y se compararían los tiempos con el fin de determinar cual de las dos presenta un mejor desempeño. Ademas del proceso de insercción se probaron las

¹<http://users.dickinson.edu/~wahlst/>

diferentes funciones con las que cuenta cada clase como el borrar un elemento, informar si un elemento pertenece al conjunto y realizar la unión o la intersección.

Para llevar a cabo dicho proceso fue necesario primero estudiar como reconstruir los objetos previos de tal manera que permitirá llevar a cabo los mismos procesos que sus antecesores, es decir, en el caso de un conjunto poder agregar elementos, eliminarlos, unir dos conjuntos etc y en el caso de una relación poder obtener el dominio o el rango entre otros. Estos objetos son una colección ordenada de elementos por lo que se estudio diferentes tipos de colecciones que podrían llegar a manejar dicha información como los TreeSet o los hashset.

HashSet² es implementada usando tablas hash que no garantizan tener los elementos ordenados, sus metodos de agregar, remover y de encontrar si un elemento ya ha sido creado es de orden de complejidad O(1).

La clase TreeSet³ esta basado en la estructura de un árbol rojo-negro y al contrario de HashSet permite tener los elementos ordenados, esto hace que aumente el orden de complejidad de los métodos de inserción y borrado a O(log n) pero provee de otras funciones útiles al momento de obtener por ejemplo el primer elemento o el ultimo entre otros.

Finalmente el hecho de que los elementos debían estar ordenados fue un punto critico por lo que se opto por realizar las modificaciones partiendo de este tipo de estructura.

5.1. Resultados de las pruebas

Al realizar las pruebas tanto en la clase BSet como BRelation se observa una perdida de rendimiento al momento de realizar la inserción de los datos, esto es debido a que los elementos deben estar ordenados. En la figura 5.1 se presenta mediante una gráfica de Tiempo Vs No. Operaciones el resultado de insertar cada uno de los elementos, la linea roja corresponde a la implementación de BSet Java y la azul a JML. Para una mejor comprensión y comparación de los datos en la Tabla 5.1 se muestran los tiempos para ambas implementaciones, es notable la diferencia en cuanto a esta función pero al momento de comprar el resto de los métodos con los que cuenta BSet se aprecia que la versión Java de la misma es mas eficiente.

Tres de las funciones mas importantes con las que cuenta esta clase son la unión, intersección y diferencia de conjuntos, en las Figuras 5.2, 5.4 y 5.6 se presentan los gráficos

²<http://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html>

³<http://docs.oracle.com/javase/7/docs/api/java/util/TreeSet.html>

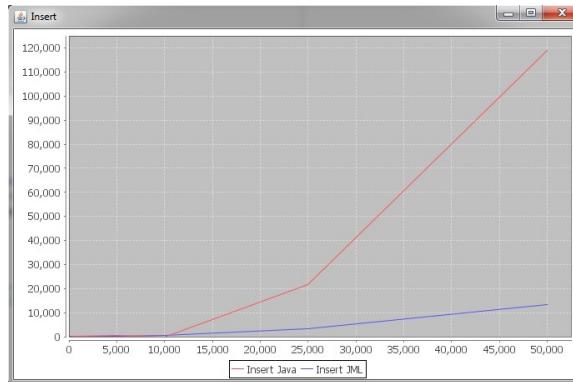
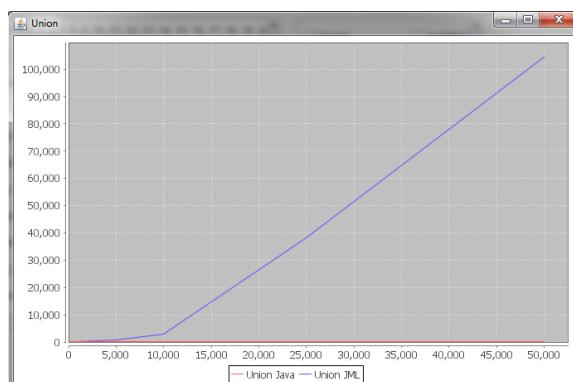


FIGURA 5.1: Comparación del método insertar de la clase BSet Java y JML.

| # Tran. | Tiempo Java | Tiempo JML |
|---------|-------------|------------|
| 100 | 4ms | 4ms |
| 500 | 22ms | 2ms |
| 1000 | 46ms | 5ms |
| 5000 | 479ms | 122ms |
| 10000 | 567ms | 491ms |
| 25000 | 21801ms | 3140ms |
| 50000 | 119135ms | 13385ms |

CUADRO 5.1: Tiempos obtenidos para el método insertar de BSet Java y JML.

correspondientes a cada una de dichas funciones, debido a que la escala no permite visualizar adecuadamente los tiempos de la implementación Java, estas se presentan de manera individual en las Figuras 5.3, 5.5 y 5.7 respectivamente, es en estas funciones donde se aprecia las mejoras considerables al rendimiento, esto es debido a que dichos procesos están basados en búsquedas sobre el conjunto de datos es mas rápido hacerlo sobre un conjunto ordenado. Los tiempos para estas tres operaciones se reúnen en las Tablas 5.2, 5.3 y 5.4.

FIGURA 5.2: Comparación del método unión (*union*) de la clase BSet Java y JML.

En el caso de la clase BRelation se observa un comportamiento similar el rendimiento baja al momento de realizar la inserción del los datos pero para las otras funciones el

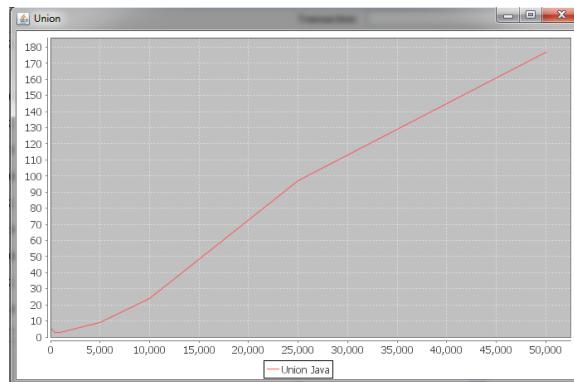


FIGURA 5.3: Tiempos para el método unión (*union*) del clase BSet Java.

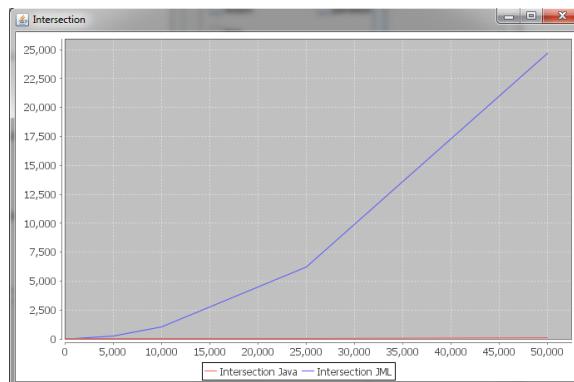


FIGURA 5.4: Tiempos para el método intersección (*intersection*) del clase BSet Java.

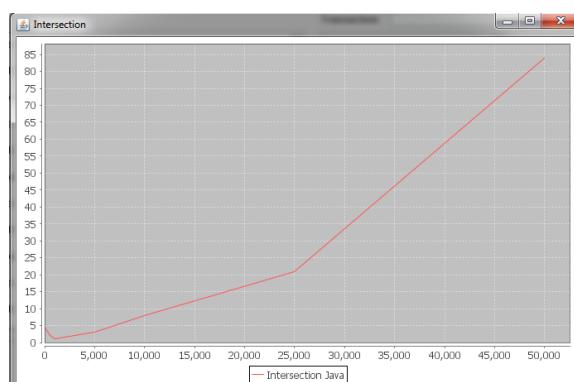


FIGURA 5.5: Comparación del método intersección (*intersection*) de la clase BSet Java y JML.

| # Tran. | Tiempo Java | Tiempo JML |
|---------|-------------|------------|
| 100 | 5ms | 105ms |
| 500 | 3ms | 137ms |
| 1000 | 3ms | 155ms |
| 5000 | 9ms | 781ms |
| 10000 | 24ms | 2870ms |
| 25000 | 97ms | 38291ms |
| 50000 | 177ms | 104558ms |

CUADRO 5.2: Tiempos obtenidos para el método intersección (*union*) de BSet Java y JML.

| # Tran. | Tiempo Java | Tiempo JML |
|---------|-------------|------------|
| 100 | 4ms | 37ms |
| 500 | 2ms | 69ms |
| 1000 | 1ms | 45ms |
| 5000 | 3ms | 278ms |
| 10000 | 8ms | 1010ms |
| 25000 | 21ms | 6207ms |
| 50000 | 84ms | 24704ms |

CUADRO 5.3: Tiempos obtenidos para el método intersección (*intersection*) de BSet Java y JML.

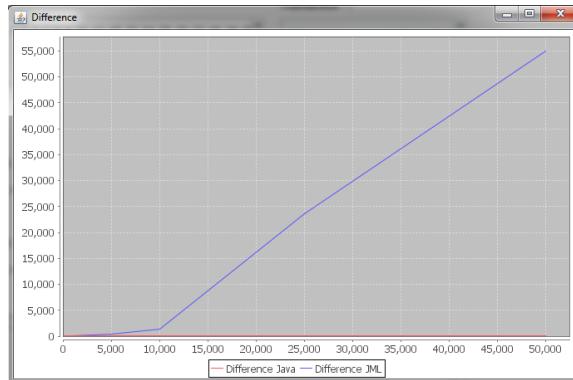
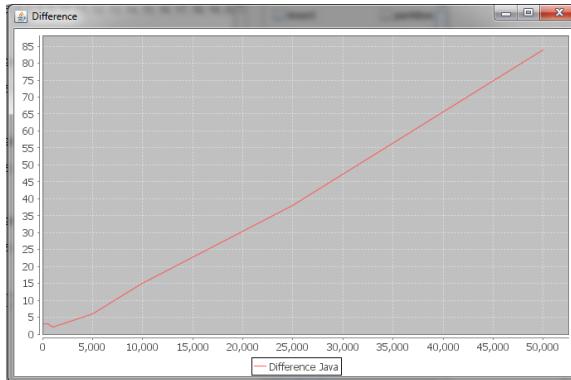
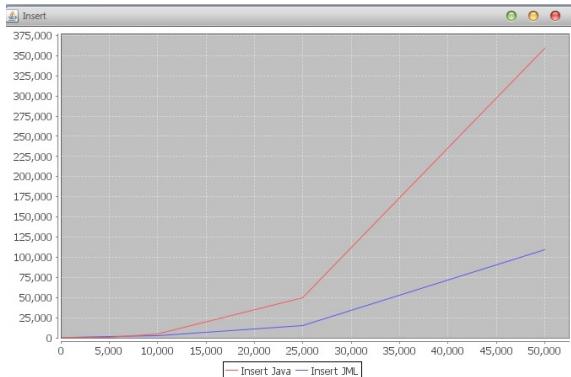


FIGURA 5.6: Comparación del método diferencia (*difference*) de la clase BSet Java y JML.

rendimiento mejor por mucho la implementación anterior, esto se aprecia en la Figura 5.8 y ??, en la Tabla 5.5 se presentan los tiempos para dicho proceso. Los gráficos de las funciones principales de esta clase junto con sus correspondientes tiempos se presenta en el Apéndice C y los de la clase BSet en el Apéndice B.

FIGURA 5.7: Tiempos para el método diferencia (*difference*) del clase BSet Java.

| # Tran. | Tiempo Java | Tiempo JML |
|---------|-------------|------------|
| 100 | 3ms | 76ms |
| 500 | 3ms | 99ms |
| 1000 | 2ms | 112ms |
| 5000 | 6ms | 415ms |
| 10000 | 15ms | 1369ms |
| 25000 | 38ms | 23542ms |
| 50000 | 84ms | 55012ms |

CUADRO 5.4: Tiempos obtenidos para el método diferencia (*difference*) de BSet Java y JML.FIGURA 5.8: Comparación del método inserción (*insert*) de la clase BRelation Java y JML.

| # Tran. | Tiempo Java | Tiempo JML |
|---------|-------------|------------|
| 100 | 15ms | 43ms |
| 500 | 30ms | 21ms |
| 1000 | 45ms | 69ms |
| 5000 | 1017ms | 1489ms |
| 10000 | 4313ms | 2468ms |
| 25000 | 49242ms | 14862ms |
| 50000 | 359508ms | 108898ms |

CUADRO 5.5: Tiempos obtenidos para el método inserción (*insert*) de BRelation Java y JML.

Capítulo 6

Caso de estudio acerca de generación de código de una aplicación ERP

6.1. Descripción del caso de estudio

Como caso de estudio se modelara en Event-B los componentes mas importantes de una aplicación POS (Punto de Venta por sus siglas en inglés¹). En el modelo se contempla la posibilidad de crear impuestos (*Taxes*), bodegas (*Warehouse*), categorías de productos (*Categories*), productos (*Products*), clientes (*Clients*) y el procesamiento de una ventas (*Sales*), además de otras características. Posteriormente usando la herramienta EventB2Java, la cual recibe como insumo el modelo formal escrito en Event-B, se obtendrá código Java que finalmente será usado para modificar la versión original del programa de código abierto Openbravo POS, este aplicativo será descrito mas adelante.

Lo que se busca inicialmente es tener una versión que realiza las mismas funciones que el programa POS original, es decir, que sea capaz de procesar una venta una vez ingresado los datos necesarios para ello.

Una vez se tenga una versión que realiza las mismas tareas, se procederá con la realización de pruebas de rendimiento sobre las dos aplicaciones, la modificada y la original, en dos sistemas operativos distintos (Windows y Mac) esto con el fin de analizar que

¹Corresponde a los equipos y software que ayudan a la gestión de establecimientos de ventas, estos pueden contar con interfaces amigable a los vendedores.

tanto difieren los tiempos cuando se mantiene la información en memoria (código generado) respecto a guardarla directamente en una base de datos y si podrían presentarse problemas de memoria.

La prueba consiste en crear por medio de la interfaz de usuario un numero determinado de elementos (impuestos, bodegas y demás) empezando en 100 e ir incrementando hasta llegar a 50000.

6.1.1. Objetivo general

El objetivo general del trabajo descrito en este capítulo es realizar pruebas de rendimiento sobre un programa POS open source (Openbravo), que ha sido modificado con código Java generado con la herramienta EventB2Java a partir de un modelo en Event-B, con el fin de determinar que desempeño tiene y que inconvenientes se presentan con el uso de la memoria, tomando como punto de comparación la versión original del mismo.

6.1.2. Objetivos específicos

- Modelar un sistema de inventarios formalmente y traducirlo a código Java.
- Modificar un software POS libre usando el código generado y verificar que realiza correctamente los procesos.
- Determinar si la aplicación tiene un buen desempeño por medio de pruebas de rendimiento, ejecutadas en dos sistemas operativos distintos (Windows y Mac) y usando como punto de comparación la versión original del mismo.
- Identificar problemas de uso memoria de la versión modificada con código formal.

6.2. Openbravo ERP

Openbravo ERP² es una aplicación enfocada a pequeñas y medianas empresas, que permite mantener información gerencial que manejan muchos negocios asociados a la producción y a aspectos de distribución de las compañías que producen bienes o servicios además de los aspectos contables de la misma. Actualmente hay dos versiones, una *Community* que es de libre distribución pero con muchas restricciones y una profesional.

Openbravo permite automatizar y registrar los procesos de negocio más comunes como ventas, compras, fabricación, finanzas entre otros.

²<http://www.openbravo.com/>

Para el caso de estudio nos centraremos en el modulo de compras y ventas que se gestiona por medio de Opebravo POS³, el cual es una aplicación de escritorio para empresas comerciales y de hostelería, esta se encuentra completamente integrada a la aplicación ERP, por lo que es posible actualizar los datos de existencias de productos, diarios financieros y datos de los clientes.

El punto de venta es donde se completa una transacción de venta. El cliente en este caso realiza un pago a cambio de bienes o servicios, normalmente se presentan varias opciones de pago como efectivo, tarjeta débito o crédito e incluso cheques. Una vez hecha la transacción normalmente se imprimirá un recibo, de igual forma, se cuenta con cierto número de reportes que permite al usuario analizar la información referente al negocio.

6.2.1. Funcionamiento de Openbravo POS

En la Figura 6.1 se presenta la interfaz inicial de Openbravo, en la parte inferior se muestran los diferentes roles que se pueden tomar al momento de manejar el programa, el rol de invitado (*Guest*) solo tendrá disponible la opción de poder realizar una venta tal como se presenta en la Figura 6.2. Para mayor comodidad solo se muestra la sección correspondiente al menú de los siguientes roles. En el rol de empleado (*Employee* Figura 6.3) se cuenta con la opción de cambiar password, hacer entradas o salidas de efectivo (Opción *Payments*) y editar una venta (*Edit sales*) además de la opción del rol anterior.



FIGURA 6.1: Interfaz inicial Openbravo.

El rol de supervisor o jefe (*Manager* Figura 6.4) permite realizar cierre de caja (Verificar que el efectivo de la caja concuerda con el movimiento diario) y modificar información de clientes pero además se le otorgan permisos para llevara a cabo algunas tareas de configuración del sistema como la creación de clientes, productos, categorías e impuestos; estos privilegios también los tiene el administrador (*Administrator* Figura 6.5) además

³<http://www.openbravo.com/retail>

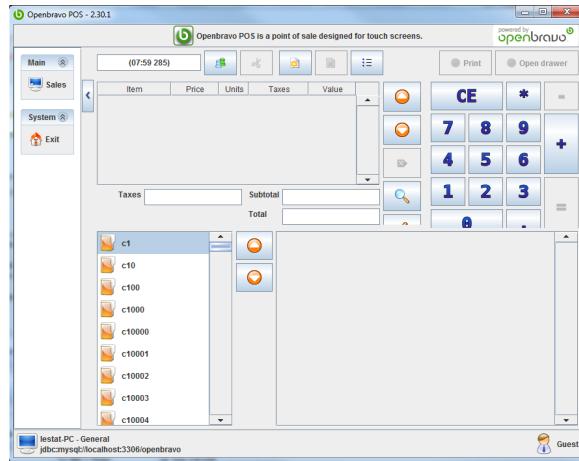


FIGURA 6.2: Interfaz principal para el rol de invitado.

de la posibilidad de crear bodegas, imprimir recibos por pantalla, realizar tareas de mantenimiento y cambiar la configuración del sistema.

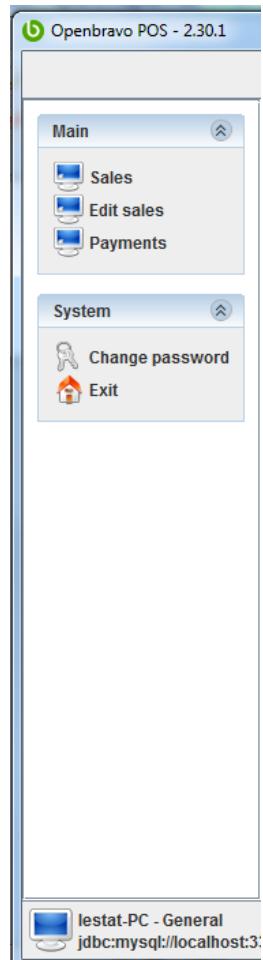


FIGURA 6.3: Interfaz principal para el rol empleado.

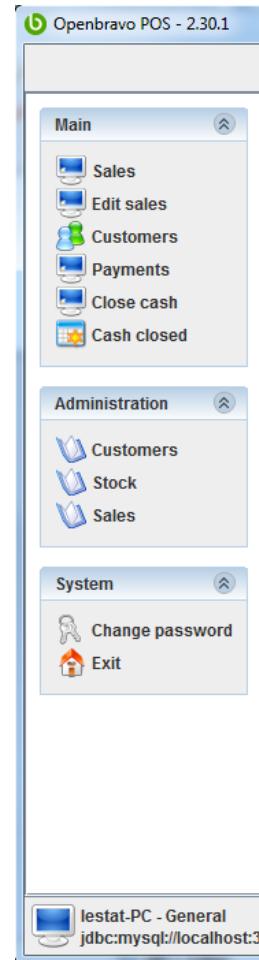


FIGURA 6.4: Interfaz principal para el rol de supervisor.

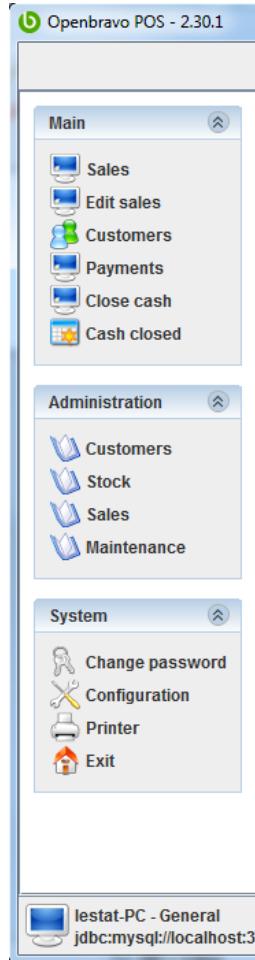


FIGURA 6.5: Interfaz principal para el rol administrador.

6.2.1.1. Creación de clientes (*Customers*)

Para crear un cliente se ha de ingresar por la opción *Customers* en la sección de *Administration* ubicada a la izquierda en el menú, cabe recordar que esta opción solo está disponible para administradores y supervisores; una vez se despliega la vista se presiona el botón *Customers* (Ver Figura 6.6).

En el formulario (Ver Figura 6.7) es necesario ingresar información en tres campos, el primero es el id (auto generado), seguido de la llave de búsqueda (*Search key*) y finalmente el nombre (*Name*), los demás campos son opcionales y no son usados en el modelo, en el caso de la llave sera usada al momento de realizar la venta para buscar al cliente al cual se le asocia la factura.

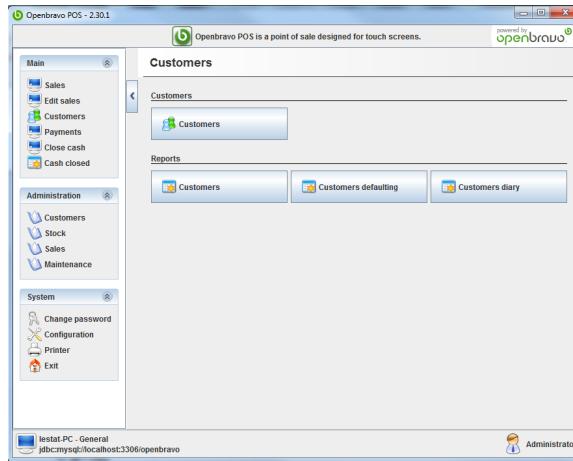


FIGURA 6.6: Vista para ingresar a crear un cliente.

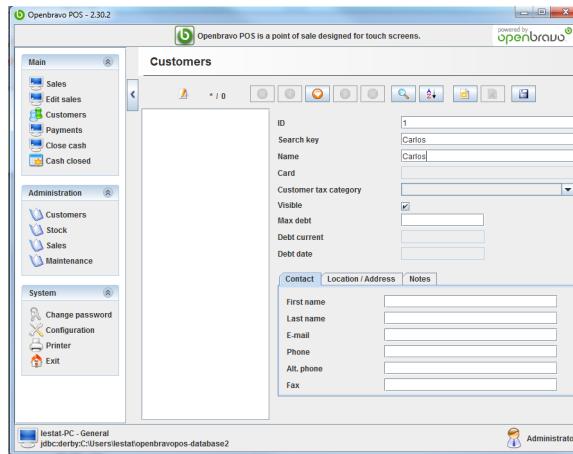


FIGURA 6.7: Interfaz para la creación de un cliente.

6.2.1.2. Creación de bodegas (*Warehouses*)

A la vista de bodegas se accede por *Maintenance* y posteriormente se hace click en *Warehouses* (Ver Figura 6.8). Para la creación son requeridos dos campos el id (auto generado) y el nombre como se aprecia en la Figura 6.9.

Al igual que ocurre en clientes el campo de dirección (*Address*) no es usado por el modelo por lo que se deja en blanco.

6.2.1.3. Creación de un impuesto (*Tax*)

En el menú *Stock* ubicado en la sección de *Administration* (Figura 6.10) encontramos el botón para acceder a la vista de impuestos (*Taxes*), además de otras vistas importantes para el proceso como categorías (*Categories*) y productos (*Products*) las cuales serán explicadas juntas mas adelante.

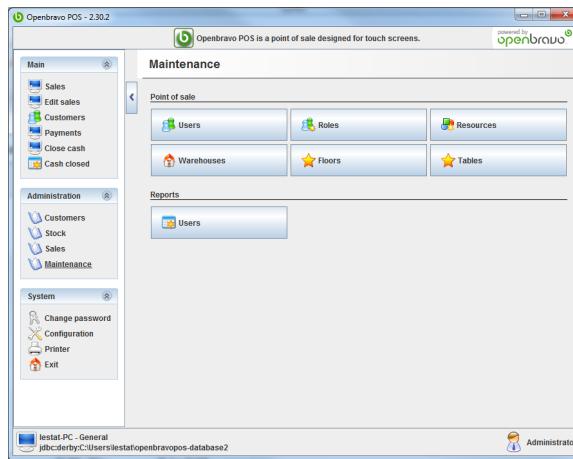


FIGURA 6.8: Acceso a la vista de creación de bodegas.

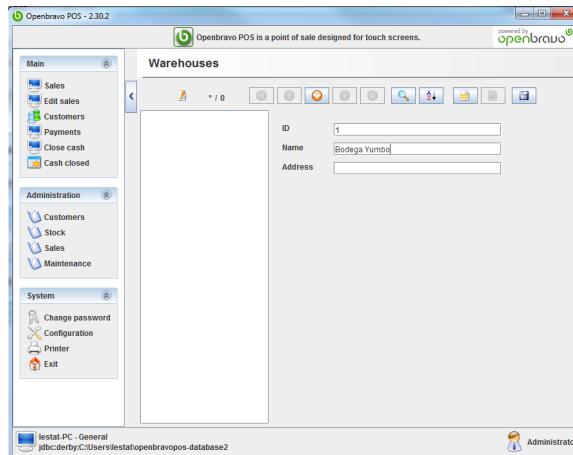


FIGURA 6.9: Interfaz para la creación de una bodega.

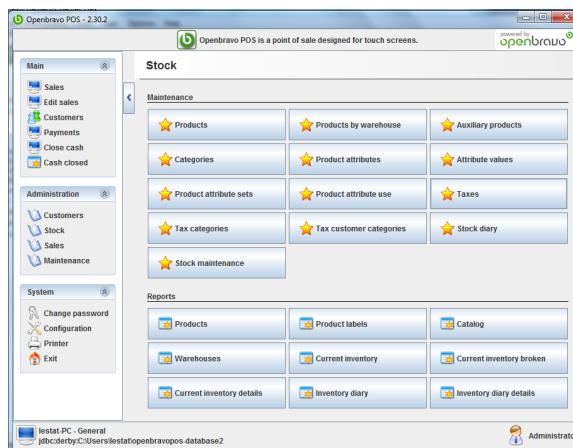


FIGURA 6.10: Interfaz para acceder a las vistas de impuestos, productos y categorías.

Para crear un impuesto deberemos ingresar el nombre, el porcentaje que sera aplicado al valor del producto al momento de la venta y un id, en la Figura 6.11 se puede observar que se esta creando un impuesto llamado *IVA* con un porcentaje del 16 %.

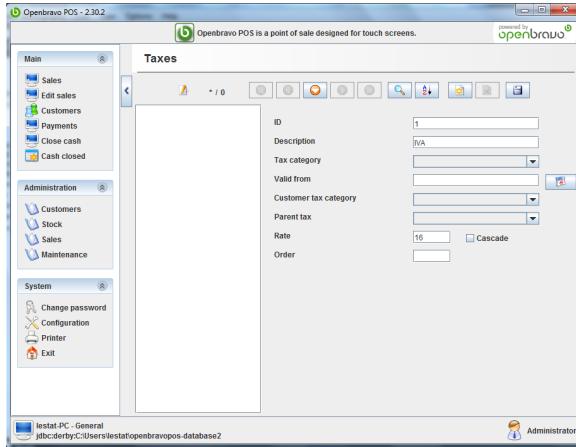


FIGURA 6.11: Interfaz para la creación de un impuesto.

6.2.1.4. Creación de categorías (*Categories*) y productos (*Products*)

La creación de los productos y las categorías van de la mano puesto que estas son la clasificación de los productos por ciertas características, como se aprecia en la Figura 6.12 se esta creando un grupo llamado *Herramientas* (el id es auto generado), este grupo es a su vez usado para crear un producto como se puede observar en la Figura 6.13, además de la categoría también es necesario usar información previamente ingresada como la bodega en la cual se almacena el producto y el impuesto que se le aplicara al momento de realizar la venta; otros datos son requeridos para crear el producto como es el caso del precio de compra y de venta, en caso de ser una materia prima solo el primer dato sera requerido, si es un producto manufacturado solo el segundo lo sera y en caso de ser un articulo de reventa los dos campos serán requeridos; también es necesario ingresar un nombre y un id (auto generado), el porcentaje de utilidad que aparece al lado del precio de venta es calculado automáticamente y aunque se muestra no es usado por el modelo.

6.2.1.5. Procesamiento de una venta (*Sale*)

Procesar una venta es la función mas usada por los cajeros en los establecimientos comerciales y es en esta vista donde se usa toda la información ingresada previamente, en la Figura 6.14 se muestra la interfaz para procesar estas operaciones en Openbravo, en ella se aprecian varios elementos importantes, estos sera explicados siguiendo la numeración que se encuentra en la imagen.

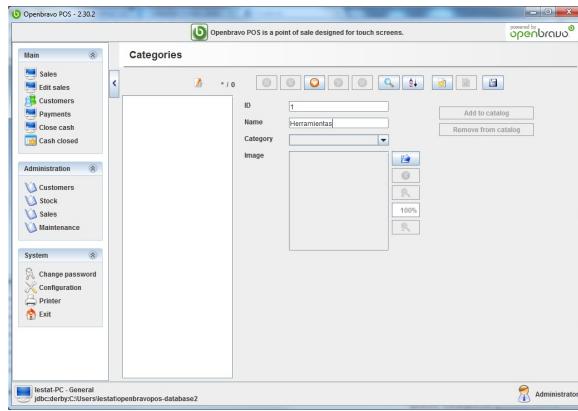


FIGURA 6.12: Interfaz para la creación de una categoría.

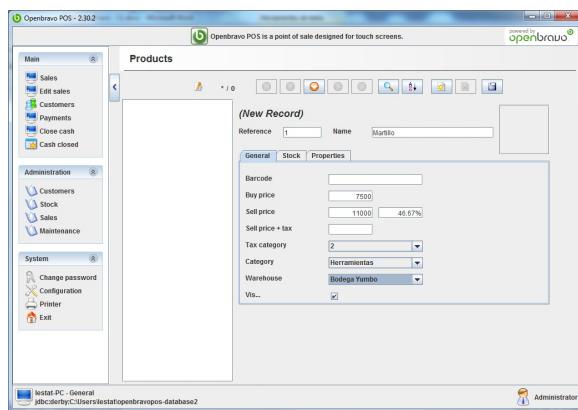


FIGURA 6.13: Interfaz para la creación de un producto.

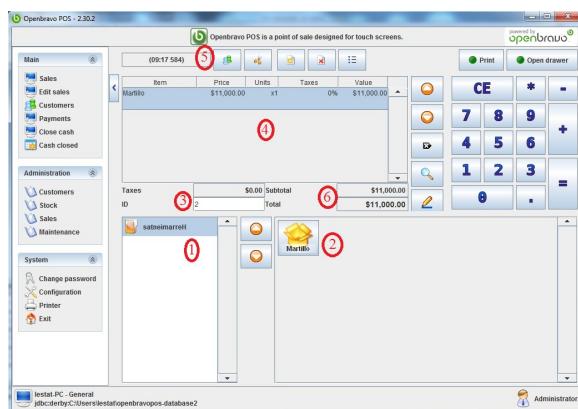


FIGURA 6.14: Interfaz para procesar una venta.

1. En esta región se listan todas las categorías que han sido creadas en el sistema.
2. Cuando se selecciona alguna de las categorías los productos que están asociadas a esta se muestran en la zona marcada con el numero 2, en caso de que el producto tenga asociado una imagen se mostrara en vez del icono de la caja abierta.
3. El id es auto generado y es usado por el modelo para el almacenamiento en memoria de la venta.
4. Al momento de seleccionar un producto este se agrega a la lista.
5. Todas las ventas tienen asociado un cliente, por medio de este botón se accede a la vista que permite buscar y asociar el cliente a la venta.
6. El total de la venta se actualiza y se muestra en el campo *Total* siendo la suma de el *Subtotal* y los impuestos.

En la Figura 6.15 se presenta la ventana para buscar el cliente, en la llave de búsqueda o en el nombre se ingresa la información por la cual se desea encontrar al cliente, la lista se mostrara en el recuadro de abajo, para seleccionarlo se debe hacer doble click sobre el nombre del cliente deseado.

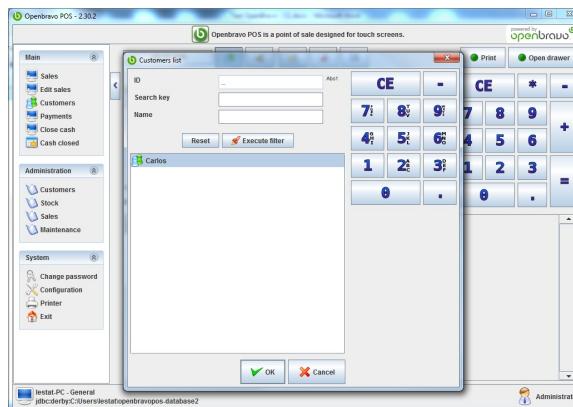


FIGURA 6.15: Ventana para asociar un cliente a la venta.

Una vez se ha ingresado toda la información necesaria solo resta llevar a cabo el pago, para ello presionamos en el igual (=), en la vista principal de la venta y se desplegará una ventana como se muestra en la Figura 6.16, en ella se puede apreciar los diferentes medios de pago con los que se cuenta en Openbravo.

Finalmente una vez la venta ha sido procesada solo resta imprimir el recibo como se observa en la Figura 6.17.



FIGURA 6.16: Ventana para realizar el pago.

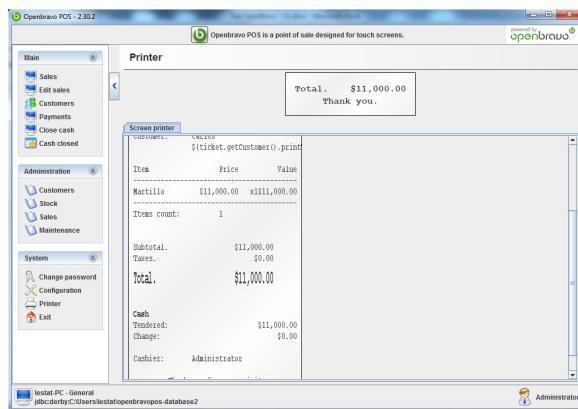


FIGURA 6.17: Factura de venta.

6.3. Documento de requerimientos (módulo de inventario)

El objetivo de nuestro sistema es mantener los inventarios de un sistema ERP (Enterprise Resource Planning). Nuestro sistema permite a la empresa a mantener un registro de la entrada y salida de productos. Los requisitos funcionales del sistema a continuación se han dividido en dos categorías:

- Los Requisitos sobre la funcionalidad del sistema se etiquetaran como FUN.
- Requisitos para la validación de la funcionalidad del sistema se etiquetaran como INV.

Es importante que las empresas lleven un registro de los flujos de mercancías en su tienda, los distintos tipos de productos, y la materia prima almacenada para ser utilizada en la fabricación de nuevos productos o para ser vendidos a otras empresas.

Supongamos que la empresa *Puertas Cruz* compra láminas de acero y tablas de madera, que se utilizan para la fabricación de puertas. El usuario debe ser capaz de crear un nuevo

producto en el sistema, en este caso nos estamos refiriendo a una materia prima, de igual forma se puede modificar el stock cuando son ingresadas a la bodega y cuando son usadas para fabricar una puerta.

- El sistema permitirá al usuario añadir un producto de tipo materia prima FUN-1
- El sistema permitirá al usuario actualizar la existencia de un producto de tipo materia prima FUN-2

En el caso de la fabricación de la puerta esta entra en la categoría de productos manufacturados, el sistema deberá permitir crear los productos y actualizar su existencia una vez se termine su realización. Para dicho proceso se hace uso de materia prima por lo que esta deberá ser descontada de stock disponible.

- El sistema permitirá al usuario crear un producto de tipo manufacturado FUN-3
- El sistema permitirá al usuario actualizar la existencia de un producto manufacturado FUN-4
- El sistema debe permitir descontar materias primas del stock FUN-5

No solo existen empresas dedicadas a manufacturar productos, también hay las que se dedican a revender productos, el sistema debe permitir crear dichos artículos.

- El sistema debe permitir crear un producto para reventa FUN-6
- El sistema permitirá al usuario actualizar la existencia de un producto para reventa FUN-7

Los proveedores pueden ser empresas o particulares que abastecen a la empresa con los bienes necesarios para el correcto funcionamiento. El sistema debe ser capaz de crear o editar los proveedores. El sistema debe validar que el usuario no puede introducir dos proveedores idénticos.

- El sistema permitirá al usuario añadir un nuevo proveedor FUN-8
- El sistema permitirá al usuario editar un proveedor existente FUN-9
- Los proveedores almacenados en el sistema son todos diferentes INV-1

Los clientes son las personas a las que se les vende los productos. El sistema debe permitir al usuario crear un nuevo cliente o editar uno existente. Los usuarios existentes almacenados en el sistema son todos diferentes.

- El sistema permitirá al usuario añadir un nuevo cliente FUN-10
- El sistema permitirá al usuario editar un cliente existente FUN-11
- Los clientes almacenados en el sistema son todos diferentes INV-2

Dependiendo del tipo de producto, este debe tener ya sea un precio de compra asociada o una de venta. El sistema debe permitir para validar esto.

- Productos en la categoría materia prima tienen un precio de compra correspondiente. Los productos de esta categoría no tienen un precio de venta asociada FUN-12
- Productos en la categoría reventa tienen un precio de compra y venta asociado FUN-13
- Productos en la categoría manufacturados tienen un precio de venta asociada FUN-14
- Los precios de compra y venta de un producto debe ser mayor o igual a 0 INV-3

Debido a la fluctuación del mercado, los precios de un producto pueden cambiar. El sistema debe permitir la modificación de los precios de compra y venta de productos.

- El sistema permitirá a los usuarios cambiar la compra o venta precio de un producto FUN-15

Las empresas cuentan con bodegas para almacenar los productos, el sistema debe permitir crearlas y modificar su nombre.

- El sistema permitirá a los usuarios crear bodegas FUN-16
- El sistema debe permitir al usuario cambiar el nombre de una bodega FUN-17

6.3.1. Estrategia de refinamiento

6.3.1.1. Refinamiento 1: ref1_Enabling

En las empresas por lo general no se permite eliminar la información sobre los clientes, productos y proveedores, debido a que deben mantener un registro histórico de la información asociada a ellos. Cuando una empresa decide borrar un cliente, proveedor o producto registrado, entonces la empresa puede desactivarlos del el sistema. Además, con el tiempo pueden habilitarlas de nuevo.

- El sistema permitirá al usuario deshabilitar o habilitar un producto REF_1_FUN-18
- El sistema permitirá al usuario deshabilitar o habilitar un proveedor REF_1_FUN-19
- El sistema permitirá al usuario deshabilitar o habilitar un cliente REF_1_FUN-20

6.3.1.2. Refinamiento 2: ref2_Order

El sistema debe ser capaz de crear una orden de compra / venta de productos. Un pedido puede incluir productos ofrecidos por una sola empresa. Si queremos hacer puertas y tenemos que comprar productos de diversas empresas, entonces tiene que colocar varias órdenes de compra, uno por cada empresa. Cada orden de compra debe indicar el número de artículos que se compran de cada producto en particular.

- El sistema debe permitir colocar una nueva orden de compra. Esta orden debe estar asociado a un proveedor REF_2_FUN-21
- El sistema debe ser capaz de ejecutar una orden de compra. Cuando los productos se compran, su información se almacena, y la tienda se actualiza para reflejar el número de artículos comprados de esos productos REF_2_FUN-22

Para vender productos a los clientes de la empresa utiliza una orden de venta, los cuales decrementan un número de unidades a las existencias de los productos.

- El sistema debe ser capaz de crear una orden de venta de los productos en el almacén. La orden de venta debe estar asociada sólo a un cliente registrado y el producto debe tener un precio asociado con el fin de ser vendidos. Una orden de venta no se puede asociar a varios clientes diferentes. REF_2_FUN-23

- El sistema debe ser capaz de vender un producto que se asocia a una orden de venta. Una vez ejecutada la orden de venta el dinero es atribuido a la empresa, y el producto se elimina de la lista de productos disponibles en la tienda. Varios productos se pueden asociar una orden de venta, pero una orden de venta se asocia a un solo cliente registrado. REF_2_FUN-24

6.3.1.3. Refinamiento 3: ref3_Categories

Para una mejor organización las compañías clasifican sus productos por diferentes conceptos dependiendo de la actividad económica que esta lleve a cabo, por ejemplo, una empresa que vende materiales para construcción puede clasificar martillos y palas en una categoría herramientas.

- El sistema debe permitir crear categorías REF_3_FUN-25
- El sistema debe permitir al usuario asociar un producto a una categoría REF_3_FUN-26
- El sistema permitirá editar el nombre de una categoría REF_3_FUN-27

6.3.1.4. Refinamiento 4: ref4_Taxes

Las empresas deben pagar impuestos por la comercialización de sus productos, el sistema debe permitir crear un impuesto y asociarlo a un producto.

- El sistema debe permitir al usuario adicionar un tipo impuesto REF_4_FUN-28
- El sistema permitirá editar el impuesto REF_4_FUN-29
- El sistema debe permitir al usuario asociar un impuesto a un producto o un conjunto de productos REF_4_FUN-30

6.3.1.5. Refinamiento 5: ref5_Reports

Los directivos en una empresa normalmente toman decisiones y supervisan el correcto funcionamiento usando informes o reportes de los diferentes procesos , el sistema debe permitir listar los productos con sus precios de compra o venta y de igual forma con la aplicación del impuesto.

- El sistema debe permitir listar los productos con su respectivo precio de compra o venta REF_5_FUN-31
- El sistema debe permitir listar los productos con el precio calculado con el impuesto REF_5_FUN-32

6.4. Modelo formal para el modulo de inventarios en Event-B

En la sección anterior se presentaron los requerimientos de un modulo de inventarios para un sistema POS, en la presente sección se muestran los elementos importantes del modelo construido a partir de dichos requerimientos en Event-B. En el Apéndice A se puede observar el modelo completo.

6.4.1. El contexto del modelo abstracto

El contexto abstracto consta de 5 conjuntos y 5 constantes como se aprecia en la Figura 6.18. *SUPPLIERS* contiene todos los proveedores que no están creados en la compañía, de igual forma ocurre con *CUSTOMERS* (clientes), *PRODUCTS* (productos) y *WAREHOUSES* (bodegas). Finalmente en los conjuntos nos resta por ver *NAMES* la cual contendría todos los nombres posibles para los diferentes elementos en el sistema como los nombres para proveedores, productos y clientes entre otros.

En las constantes tenemos elementos que nos van a ayudar con diferentes aspectos del manejo de un sistema de inventarios como es el caso de las fechas (*bday*, *bmonth*, *byear* y *bdate*) y la constante *bstring* que sera una función parcial entre dos enteros, esta nos ayudara a representar los nombres en Event-B como una secuencia ordenada de letras (las letras serán tratadas como números decimales según la tabla ASCII).

6.4.2. Las variables del modelo abstracto

En la Figura 6.19 se muestran las diferentes variables usadas en el modelo abstracto. Los clientes de la compañía se almacenaran en *customers* y los proveedores en *suppliers*, además de sus respectivos nombres *customername* y *suppliername*. El conjunto de todos los productos de la empresa estarían en *products* y mas específicamente las materias primas en *rawmaterial*, los productos para reventa en *resale* y finalmente los manufacturados en *manufacturing*. La materia prima que se uso para fabricar un producto se guardara en *material*. Dependiendo del tipo de producto este tendrá o un precio de

```

CONTEXT
  ctx_abstract
SETS
  SUPPLIERS
  CUSTOMERS
  PRODUCTS
  NAMES
  WAREHOUSES
CONSTANTS
  bstring
  bday
  bmonth
  byear
  bdate
AXIOMS
  ax1: bstring ∈ N ↗ N
  ax2: finite(bstring)
  ax3: dom(bstring) = 1..card(bstring)
  ax4: bday = 1.,31
  ax5: bmonth = 1.,12
  ax6: byear = 1.,3000
  ax7: bdate ∈ (bday × bmonth) ↗ byear
END

```

FIGURA 6.18: Contexto de la maquina abstracta

compra o uno de venta o ambos esto se guardara en *pprice* (compra) y *sprice* (venta). Las bodegas donde la empresa almacena los productos se guardan en *warehouses* con su respectivo nombre *warehousename*. Las materias primas y los productos para reventa son adquiridos mediante un proveedor esta relación se almacena en *prodsupplier*. Finalmente el numero total de productos en una bodega y la cantidad de unidades por cada uno de ellos se guarda en *stock* y *stocknumber* respectivamente.

```

MACHINE
  ref0_Inventory
SEES
  ctx_orders
VARIABLES
  customers suppliers pprice sprice
  rawmaterial manufacturing resale material
  products customername suppliernames warehouses
  warehousename stock stocknumber prodsupplier

```

FIGURA 6.19: Variables de la maquina abstracta

6.4.3. Los invariantes y la inicialización en el modelo abstracto

Como invariantes tendremos que los clientes (*customers*) serán un subconjunto de conjunto global de clientes (*CUSTOMERS*) visto en el contexto, igual ocurrirá para los proveedores, los productos y las bodegas (*inv2*, *inv3* y *inv13*). Como se vio anteriormente la compañía puede tener diferentes tipos de producto, la unión de todos ellos deberá ser igual al total de productos en la empresa (*inv4*). Otra de las restricciones vistas anteriormente es que hay productos que pueden tener un precio de venta o uno de compra asociado y en el caso de los productos de reventa los dos, esto se aprecia en los invariantes *inv6* y *inv7*, en el primero se aprecia que los únicos productos con precio de compra son los de reventa (*resale*) y las materias primas (*rawmaterial*); por otro lado los manufacturados (*manufacturing*) tendrán precio de venta igual que los de reventa.

En la Figura 6.20 también se aprecian otros aspectos importantes del modelo como el uso de la constante *bstring* vista en el contexto, para poder definir los nombres (*customername*, *warehousename* y *suppliernamde*) como una relación entre un powerset⁴ de la constante y un elemento bien sea un cliente, proveedor o bodega (*inv11*, *inv12* y *inv14*).

INVARIANTS

- inv1:** $customers \subseteq CUSTOMERS$
- inv2:** $suppliers \subseteq SUPPLIERS$
- inv3:** $products \subseteq PRODUCTS$
- inv4:** $resale \cup rawmaterial \cup manufacturing = products$
- inv5:** $resale \cap rawmaterial = \emptyset$
- inv6:** $pprice \in (resale \cup rawmaterial) \rightarrow \mathbb{N}$
- inv7:** $sprice \in (resale \cup manufacturing) \rightarrow \mathbb{N}$
- inv8:** $resale \cap manufacturing = \emptyset$
- inv9:** $rawmaterial \cap manufacturing = \emptyset$
- inv10:** $material \in (manufacturing \leftrightarrow \mathbb{P}(rawmaterial))$
- inv11:** $customername \in customers \rightarrow \mathbb{P}(bstring)$
- inv12:** $suppliernname \in suppliers \rightarrow \mathbb{P}(bstring)$
- inv13:** $warehouses \subseteq WAREHOUSES$
- inv14:** $warehousename \in warehouses \rightarrow \mathbb{P}(bstring)$
- inv15:** $stock \in warehouses \leftrightarrow products$
- inv16:** $stocknumber \in stock \rightarrow \mathbb{N}$
- inv17:** $prodsupplier \in suppliers \leftrightarrow products$

FIGURA 6.20: Invariantes de la maquina abstracta

Las variables del modelo se inicializan todas iguales al conjunto vacío como se aprecia en la Figura Figura 6.21.

⁴Es un conjunto de todos los subconjuntos posibles de S, este incluye el conjunto vacío y S.

```

EVENTS
INITIALISATION:notextendedordinary
THEN
  act1: rawmaterial :=  $\emptyset$ 
  act2: resale :=  $\emptyset$ 
  act3: suppliers :=  $\emptyset$ 
  act4: customers :=  $\emptyset$ 
  act5: pprice :=  $\emptyset$ 
  act6: sprice :=  $\emptyset$ 
  act7: products :=  $\emptyset$ 
  act8: manufacturing :=  $\emptyset$ 
  act9: material :=  $\emptyset$ 
  act10: customername :=  $\emptyset$ 
  act11: suppliername :=  $\emptyset$ 
  act12: warehouses :=  $\emptyset$ 
  act13: warehousename :=  $\emptyset$ 
  act14: stock :=  $\emptyset$ 
  act15: stocknumber :=  $\emptyset$ 
  act16: prodsupplier :=  $\emptyset$ 
END

```

FIGURA 6.21: Inicialización de variables en la maquina abstracta

6.4.4. Creación de los productos

La creación de los diferentes tipos de producto se lleva a cabo de una manera mas o menos similar en todos los casos, si observamos el evento para crear una materia prima (*create_raw_material* Figura 6.22) se ve que en principio (*gur1*) se verifica que ya no haya sido creado dentro del sistema, luego se verifica que el precio de compra sea un natural mayor a 0 (*grd2*), en (*grd3*) se valida que la bodega este creada y finalmente que el producto no se encuentre en el stock.

Este mismo proceso se lleva a cabo con la creación de productos para reventa (Figura 6.23) y manufacturados (Figura 6.24), con la respectiva validación de los precios de compra y venta.

Una vez revisadas las guardas de los eventos, nos centramos en el cuerpo de los mismos, donde se realizan las asignaciones pertinentes, en el caso de un producto para reventa se actualizará la variable *resale* (Figura 6.23 en la acción *act1*), seguida de los precios de compra y venta, *pprice* y *sprice* respectivamente, finalmente se asigna el producto a la bodega indicada en la variable *war*.

El mismo proceso se lleva a cabo para los otros dos tipos de productos.

```

create_raw_material :
ANY
  pro
  purchaseprice
  war
WHERE
  grd1: pro ∈ PRODUCTS \ products
  grd2: purchaseprice ∈ N1
  grd3: war ∈ warehouses
  grd4: war ↦ pro ∉ stock
THEN
  act1: rawmaterial := rawmaterial ∪ pro
  act2: pprice := pprice ∪ pro ↦ purchaseprice
  act3: products := products ∪ pro
  act4: stocknumber := stocknumber ∪ (war ↦ pro) ↦ 0
  act5: stock := stock ∪ war ↦ pro
END

```

FIGURA 6.22: Evento que crea una materia prima

```

create_resale_product :
ANY
  pro
  purchaseprice
  sellprice
  war
WHERE
  grd1: pro ∈ PRODUCTS \ products
  grd2: purchaseprice ∈ N1
  grd3: sellprice ∈ N1
  grd4: war ∈ warehouses
  grd5: war ↦ pro ∉ stock
THEN
  act1: resale := resale ∪ pro
  act2: pprice := pprice ∪ pro ↦ purchaseprice
  act3: sprice := sprice ∪ pro ↦ sellprice
  act4: products := products ∪ pro
  act5: stocknumber := stocknumber ∪ (war ↦ pro) ↦ 0
  act6: stock := stock ∪ war ↦ pro
END

```

FIGURA 6.23: Evento que crea un producto para reventa

```

create_manufacturing_product :
  ANY
    pro
    sellprice
    rawmaterialpro
    war
  WHERE
    grd1: pro ∈ PRODUCTS \ products
    grd2: sellprice ∈  $\mathbb{N}_1$ 
    grd3: rawmaterialpro ⊆ rawmaterial
    grd4: pro ↪ rawmaterialpro ∉ material
    grd5: war ↪ pro ∈ stock
  THEN
    act1: manufacturing := manufacturing ∪ pro
    act2: material := material ∪ pro ↪ rawmaterialpro
    act3: sprice := sprice ∪ pro ↪ sellprice
    act5: products := products ∪ pro
    act6: stocknumber := stocknumber ∪ (war ↪ pro) ↪ 0
    act7: stock := stock ∪ war ↪ pro
  END

```

FIGURA 6.24: Evento que crea un producto manufacturado

6.4.5. Creación de los productos mediante listas

El modelo también contempla la posibilidad de crear productos mediante listas, esto permite pasar un conjunto de productos de un tipo (reventa, manufacturado o materia prima) y crearlos al mismo tiempo, en las Figuras 6.25, 6.26 y 6.26 se muestra los eventos para crear mediante una lista los productos. En *create_raw_material_list* se puede observar que se le esta pasando la variable *lis* y en la *grd1* se verifica que sea un subconjunto de productos *PRODUCTS* que no han sido creados en el sistema, esta validación se lleva a cabo por los otros dos eventos.

Los precios de compra, venta y las bodegas también son tratados como listas y se les realizan las mismas validaciones vistas en la sección anterior.

El proceso de actualizar las variables en el cuerpo del evento es similar a cuando se crea un solo producto salvo que en el caso de *stocknumber* y *stock* se hace un producto cartesiano entre los productos de la lista y la bodega.

6.4.6. Creación, edición y borrado de un proveedor

Los proveedores son los encargados de surtir a la empresa con los elementos necesarios para llevar a cabo todas sus operaciones, dichos elementos pueden ser desde artículos de

```

create_raw_material_list :
  ANY
    lis
    pprice
    war
  WHERE
    grd1: lis ⊆ PRODUCTS \ products
    grd2: pprice ∈ lis → N
    grd3: war ∈ warehouses
    grd4: (war × lis) ∩ stock = ∅
  THEN
    act1: rawmaterial := rawmaterial ∪ lis
    act2: pprice := pprice ∪ pprice
    act3: products := products ∪ lis
    act4: stocknumber := stocknumber ∪ ((war × lis) × 0)
    act5: stock := stock ∪ (war × lis)
  END

```

FIGURA 6.25: Evento que crea un conjunto de productos de tipo materia prima

```

create_resale_product_list :
  ANY
    lis
    pprice
    sprice
    war
  WHERE
    grd1: lis ⊆ PRODUCTS \ products
    grd2: pprice ∈ lis → N
    grd3: sprice ∈ lis → N
    grd4: war ∈ warehouses
    grd5: (war × lis) ∩ stock = ∅
  THEN
    act1: resale := resale ∪ lis
    act2: pprice := pprice ∪ pprice
    act3: sprice := sprice ∪ sprice
    act4: products := products ∪ lis
    act5: stocknumber := stocknumber ∪ ((war × lis) × 0)
    act6: stock := stock ∪ (war × lis)
  END

```

FIGURA 6.26: Evento que crea un conjunto de productos de tipo reventa

```

create_manufacturing_product_list :
  ANY
    lis
    spricel
    rawmaterialpro
    war
  WHERE
    grd1: lis ⊆ PRODUCTS \ products
    grd2: spricel ∈ lis → N
    grd3: rawmaterialpro ∈ lis ↔ P(rawmaterial)
    grd4: war ∈ warehouses
    grd5: (war × lis) ∩ stock = ∅
  THEN
    act1: manufacturing := manufacturing ∪ lis
    act2: material := material ∪ rawmaterialpro
    act3: sprice := sprice ∪ spricel
    act4: products := products ∪ lis
    act5: pprice := pprice ∪ (l × 0)   act6: stocknumber := stocknumber ∪ ((war × lis) × 0)
    act7: stock := stock ∪ (war × lis)
  END

```

FIGURA 6.27: Evento que crea un conjunto de productos de tipo manufacturados

oficina hasta materias primas o productos para la reventa, en la Figura 6.28 se presenta el evento para la creación de un proveedor en el modelo. Se aprecian dos elementos importantes el nombre y el identificador del proveedor, antes de proceder con la ejecución del cuerpo del evento se debe verificar que el proveedor (parámetro *sup*) no haya sido creada en el sistema y además que el nombre sea un *powerset* de la variable *bstring*. Una vez validadas las guardas se puede modificar la variable *suppliers* para que contenga al nuevo proveedor y se actualiza *suppliername* con la relación entre el identificador y el nombre del proveedor.

```

add_supplier :
  ANY
    sup
    name
  WHERE
    grd1: sup ∈ SUPPLIERS \ suppliers
    grd2: name ∈ P(bstring)
  THEN
    act1: suppliers := suppliers ∪ sup
    act2: suppliername := suppliername ∪ sup ↦ name
  END

```

FIGURA 6.28: Evento para crear un proveedor

Los proveedores también pueden ser eliminados del sistema para ello es necesario validar que el identificador esta en los creados previamente (*grd1*), una vez hecha la validación se elimina de la variable (*suppliers*) y se borra cualquier registro que se encuentre en los nombres (*suppliername*) y en los que han provisto de productos a la empresa (*prodsupplier*), esto se hace mediante una restricción de dominio.

```
delete_supplier :
  ANY
    sup
  WHERE
    grd1: sup ∈ suppliers
  THEN
    act1: suppliers := suppliers \ sup
    act2: suppliername := sup ⋸ suppliername
    act3: prodsupplier := sup ⋸ prodsupplier
  END
```

FIGURA 6.29: Evento que permite borrar un proveedor

El sistema también dota al usuario con la posibilidad de editar el nombre de un proveedor en caso de haberlo errado, para ello se debe pasar el nuevo nombre en la variable (*name*) que debe ser estar contenida en el *powerset* de *bstring* (igual que ocurría con el nombre original) y el identificador que ya este creado en el sistema según la guarda *grd2*, una vez validado esto se procede a actualizar la variable *suppliername* con el nuevo nombre.

```
edit_supplier_name :
  ANY
    sup
    name
  WHERE
    grd1: sup ∈ suppliers
    grd2: name ∈  $\mathbb{P}(\text{bstring})$ 
  THEN
    act1: suppliername(sup) := name
  END
```

FIGURA 6.30: Evento que permite editar un proveedor

6.4.7. Asociar un producto con su proveedor

En las empresas es importante saber que proveedor surtió que producto, en el modelo esto se hace mediante el evento *add_product_supplier* (Figura 6.31). Este evento necesita de dos parámetros, el identificador del proveedor y del producto. Las guardas *grd1* y

grd2 verifican que corresponden a productos y proveedores creados previamente en el sistema, finalmente se actualiza la variable *prodsupplier* para reflejar la relación que existe entre estos dos elementos.

```

add_product_supplier :
  ANY
    prod
    sup
  WHERE
    grd1: prod ∈ products
    grd2: sup ∈ suppliers
  THEN
    act1: prodsupplier := prodsupplier ∪ sup ↳ prod
  END

```

FIGURA 6.31: Evento que permite asociar un producto a un proveedor

6.4.8. Creación y edición de un cliente

El sistema permite crear un cliente y editararlo al igual que ocurría con los proveedores, los eventos correspondientes son presentados en las Figuras 6.32 y 6.33.

En el caso de la creación es necesario pasar un identificador (que no debe haber sido creado previamente como indica la guarda *grd1*) y un nombre que al igual que los anteriores debe pertenecer al powerset de *bstring* (*grd2*), una vez validadas las guardas se actualiza la variable *customer* con el correspondiente identificador y la variable *customername* con la relación entre el nombre y el cliente.

```

add_customer :
  ANY
    cust
    name
  WHERE
    grd1: cust ∈ CUSTOMERS \ customers
    grd2: name ∈ ℙ(bstring)
  THEN
    act1: customers := customers ∪ cust
    act2: customername := customername ∪ cust ↳ name
  END

```

FIGURA 6.32: Evento que permite editar crear un cliente

La edición permite cambiar el nombre del cliente, este proceso es similar al de proveedores.

```

edit_customer :
ANY
  cust
  newcust
  newname
WHERE
  grd1: cust ∈ dom(customername)
  grd2: newcust ∈ CUSTOMERS \ customers
  grd3: newname ∈ ℙ(bstring)
THEN
  act1: customers := (customers \ cust) ∪ newcust
  act2: customername := (customername \ cust ↦ customername(cust)) ∪ newcust ↦ newname
END

```

FIGURA 6.33: Evento que permite editar un cliente

6.4.9. Edición de los precios de venta y compra

Los precios de los productos varían muchas veces por factores externos a la compañía, como por ejemplo la entrada de un nuevo competidor al mercado o reducción de la demanda de un producto, para ello el sistema permite modificar el precio de compra o venta de los productos, recordando las restricciones que existen para cada producto. Editar un precio de venta (evento *edit_purchase_price* Figura 6.34) solo esta disponible para productos que son materias primas o de reventa esto se verifica mediante la *grd1*, el precio debe ser un valor natural mayor a 1 (*grd4*).

```

edit_purchase_price :
ANY
  pro
  pri
WHERE
  grd1: pro ∈ products \ manufacturing
  grd4: pri ∈ ℑ
  grd5: pro ∈ dom(pprice)
THEN
  act1: pprice := (pprice \ pro ↦ pprice(pro)) ∪ pro ↦ pri
END

```

FIGURA 6.34: Evento que permite editar el precio de compra de un producto

Para la edición del precio de venta se debe validar que el producto a modificar es manufaturado o de reventa (*grd1*), una vez hechas las validaciones correspondientes se puede proceder con la modificación de la variable *sprice* para que contenga el nuevo precio de venta para el producto.

```

edit_sell_price :
ANY
    pro
    pri
WHERE
    grd1: pro ∈ products \ rawmaterial
    grd4: pri ∈ N1
    grd5: pro ∈ dom(sprice)
THEN
    act1: sprice := (sprice \ pro ↦ sprice(pro)) ∪ pro ↦ pri
END

```

FIGURA 6.35: Evento que permite editar el precio de venta de un producto

6.4.10. Creación y eliminación de bodegas

La creación de las bodegas (evento *add_warehouse* Figura 6.36) es similar a la de clientes y proveedores, esta recibe un identificador de la bodega *war* se valida que no exista mediante la *grd1* y un nombre *name*. Una vez validado los datos se procede a almacenarlos en las variables correspondientes *warehouses* y *warehousename* respectivamente.

```

add_warehouse :
ANY
    war
    name
WHERE
    grd1: war ∈ WAREHOUSES \ warehouses
    grd2: name ∈ P(bstring)
THEN
    act1: warehouses := warehouses ∪ war
    act2: warehousename := warehousename ∪ war ↦ name
END

```

FIGURA 6.36: Evento para agregar una bodega

Para eliminar una bodega requerimos del identificador de la misma *war*, el eventos se muestra en la Figura 6.37, una vez validado que el identificador corresponde a una de las bodegas creadas previamente en el sistema, se procede con borrarla modificando la variable *warehouses* para que excluya dicho valor y del nombre *warehousename*, esto se hace mediante una restricción de dominio, esta misma restricción se usa para eliminar los registros que se hayan guardado en el *stock* de la bodega y el el numero de unidades del producto.

```

remove_warehouse :
ANY
    war
WHERE
    grd1: war ∈ warehouses
THEN
    act1: warehouses := warehouses \ war
    act2: warehousename := war ⋙ warehousename
    act3: stock := (war ⋙ stock)
    act4: stocknumber := ((war × products) ⋙ stocknumber)
END

```

FIGURA 6.37: Evento que permite remover una bodega

Este es el modelo inicial de nuestro modulo de inventarios, que falta refinar considerando otros aspectos importantes que manejan las empresas.

6.4.11. Refinamiento 1 - Variables, invariantes e inicialización

En el refinamiento se tiene en cuenta un elemento importante en las empresas y es que normalmente no se borra la información, esto es debido a que se deben guardar los históricos de las operaciones. Lo que se hace es deshabilitar un producto, cliente o proveedor una vez se descontinua por alguna razón.

En la Figura 6.38 se muestran las variables para este refinamiento. En *enabled_supplier*, *enabled_customer* y *enabled_product* vamos a guardar que proveedores, clientes y productos están habilitados en el sistema respectivamente.

```

VARIABLES
    enabled_supplier
    enabled_customer
    enabled_product

```

FIGURA 6.38: Variables del refinamiento 1

Como se aprecia en la sección correspondiente a los invariantes (Figura 6.39) las variables van a ser subconjuntos de su respectivo elemento objetivo, por ejemplo *enabled_customer* sera un subconjunto de *customers*.

En la Figura 6.40 se presenta la sección del modelo que corresponde a la inicialización de las nuevas variables, a estas se les asigna el conjunto vacío.

INVARIANTS

inv1_r1: $enabled_product \subseteq products$
inv2_r1: $enabled_supplier \subseteq suppliers$
inv3_r1: $enabled_customer \subseteq customers$

FIGURA 6.39: Invariantes del refinamiento 1

EVENTS

INITIALISATION:

THEN

act1_r1: $enabled_product := \emptyset$
act2_r1: $enabled_supplier := \emptyset$
act3_r1: $enabled_customer := \emptyset$

END

FIGURA 6.40: Inicialización de las variables en el refinamiento 1

6.4.12. Refinamiento 1 - Eventos refinados

Debido a la incorporación de las nuevas variables se hace necesario refinar los eventos en los cuales se crean productos, clientes y proveedores, esto con el fin de agregar dichos elementos a la variable correspondiente y dejarlos habilitados por defecto al crearlos. En la Figura 6.41 y 6.42 se presentan los refinamientos para los eventos que permiten crear una materia prima individual y por lista respectivamente. Como se puede apreciar en el *act1_r1* de ambos eventos ahora una vez creado el producto este es adicionado al conjunto de productos habilitados *enabled_product*. El proceso es igual para productos manufacturados y de reventa.

```
create_raw_material :  

  REFINES  

    create_raw_material  

  ANY  

  WHERE  

  THEN  

    act1_r1:  $enabled\_product := enabled\_product \cup pro$   

  END
```

FIGURA 6.41: Refinamiento de evento que permite crear una materia prima

Como se dijo anteriormente los proveedores y los clientes también pueden ser habilitados o deshabilitados en el sistema, por lo cual es necesario refinar los eventos correspondientes a su creación. En la Figura 6.43 se muestra el evento para crear un proveedor, en el *act1_r1* una vez creado el proveedor se agrega al conjunto *enabled_supplier*, Igual ocurre

```

create_raw_material_list :
  REFINES
    create_raw_material_list
  ANY
  WHERE
  THEN
    act1_r1: enabled_product := enabled_product ∪ lis
  END

```

FIGURA 6.42: Refinamiento del evento que permite crear un conjunto de productos de tipo materia prima

para el evento de creación de clientes (Figura 6.44) en el cual se actualiza la variable *enabled_customer*.

```

add_supplier :
  REFINES
    add_supplier
  ANY
  WHERE
  THEN
    act1_r1: enabled_supplier := enabled_supplier ∪ sup
  END

```

FIGURA 6.43: Refinamiento del evento que permite agregar un proveedor

```

add_customer :
  REFINES
    add_customer
  ANY
  WHERE
  THEN
    act1_r1: enabled_customer := enabled_customer ∪ cust
  END

```

FIGURA 6.44: Refinamiento del evento que permite agregar un cliente

6.4.13. Refinamiento 1 - Habilitar o deshabilitar productos

Es necesario contar con la posibilidad de habilitar o deshabilitar los productos, clientes y proveedores, para ello el modelo cuenta con estos dos eventos para cada elemento del sistema, en la Figura 6.45 se aprecia el evento que permite habilitar un producto para reventa (*enable_resale_product*), en la guarda se verifica que el producto que se

va a habilitar se de este tipo y además (en la guarda $grd2$) que no se encuentre en el conjunto de los productos habilitados, una vez verificado esto se procede a agregarlo a la variable $enabled_product$ en el cuerpo del evento.

```

enable_resale_product :
ANY
pro
WHERE
  grd1: pro ∈ resale
  grd2: pro ∉ enabled_product
THEN
  act1: enabled_product := enabled_product ∪ pro
END

```

FIGURA 6.45: Evento que permite habilitar un producto de tipo reventa

El evento para deshabilitar un producto se presenta en la Figura 6.46, se realiza la validación de que el producto sea de tipo reventa y que el producto este contenido en el conjunto de productos habilitados ($grd2$), finalmente se substraerá el elemento del conjunto como se aprecia en el $act1$.

```

disable_resale_product :
ANY
pro
WHERE
  grd1: pro ∈ resale
  grd2: pro ∈ enabled_product
THEN
  act1: enabled_product := (enabled_product \ pro)
END

```

FIGURA 6.46: Evento que permite deshabilitar un producto de tipo reventa

Los demás eventos para habilitar y deshabilitar clientes, proveedores y los otros tipos de producto, funcionan de manera similar a la expuesta anteriormente y se pueden apreciar en el Apéndice A.

6.4.14. Refinamiento 2 - El contexto

En el contexto del refinamiento 2 (Figura 6.47) agregamos un nuevo conjunto *ORDERS* el cual sera el conjunto global de ordenes de compra y venta en la compañía, en la sección de *EXTENDS* se aprecia que es un refinamiento al contexto abstracto explicado previamente.

```

CONTEXT
  ctx_orders
EXTENDS
  ctx_abstract
SETS
  ORDERS
END

```

FIGURA 6.47: Contexto del refinamiento 2

6.4.15. Refinamiento 2 - Variables, invariantes e inicialización

En el refinamiento 2 se introducen las ordenes de compra y venta, en el primer caso se ejecuta una orden de compra al momento del ingreso de productos a la empresa y en caso de salida de los mismos se ejecuta una orden de venta.

En la Figura 6.48 se muestra la sección de variables para este refinamiento, en la variable *orders* se almacenara el conjunto de todas las ordenes en el sistema. Las ordenes de compra se guardaran en *porders* y las de venta en *sorders*. El detalle de una factura son los productos que se adquieren o venden con cada una de ellas, este detalle se almacena en *prodorders* y además en el caso de ser de venta se guarda el cliente al cual se le esta entregando el articulo, esto se guarda en la variable *custorders*. El numero de productos adquiridos o entregados se guardara en las variables *sorder_detail_numitems* y *porder_detail_numitems*. Al igual que el numero de productos también es necesario guardar la fecha de la factura esta sera guardada en *sorder_detail_date* y *porder_detail_date*.

```

VARIABLES
  orders
  porders
  sorders
  prodorders
  custorders
  sorder_detail_numitems
  porder_detail_numitems
  sorder_detail_date
  porder_detail_date

```

FIGURA 6.48: Variables del refinamiento 2

En los invariantes (Figura 6.49) lo primero que definiremos es que la variable *orders* sera un subconjunto del todas las ordenes de compra y venta *ORDERS*. En el invariante 2 y 3 (*inv2_r2* y *inv3_r2*) se expresa que las ordenes están compuestas por la unión de ordenes de compra y de venta y además que entre ellas dos no pueden haber elementos en

común, es decir, que una orden no puede ser de compra y de venta al mismo tiempo. El detalle de la factura es decir la lista de productos asociados a ella guardado en la variable *prodorders* se expresa como una relación entre un producto y una orden. La variable *custorders* se expresa como una relación entre un cliente y una orden. Finalmente en los invariantes del 6 al 9 se definen las reglas para el numero de productos y la fecha en la cual se creo la orden.

INVARIANTS

```

inv1_r2: orders ⊆ ORDERS
inv2_r2: porders ∪ sorders = orders
inv3_r2: porders ∩ sorders = ∅
inv4_r2: prodorders ∈ orders ↔ products
inv5_r2: custorders ∈ orders ↔ customers
inv6_r2: sorder_detail_numitems ∈ (orders × products) ↦ ℕ1
inv7_r2: porder_detail_numitems ∈ (orders × products) ↦ ℕ1
inv8_r2: sorder_detail_date ∈ (orders × products) ↦ bdate
inv9_r2: porder_detail_date ∈ (orders × products) ↦ bdate

```

FIGURA 6.49: Invariantes del refinamiento 2

Las variables se inicializan todas igual al conjunto vacío como se aprecia en la Figura 6.50.

EVENTS

INITIALISATION:

THEN

```

init1_r2: orders := ∅
init2_r2: porders := ∅
init3_r2: sorders := ∅
init4_r2: prodorders := ∅
init5_r2: custorders := ∅
init6_r2: sorder_detail_numitems := ∅
inti7_r2: porder_detail_numitems := ∅
init8_r2: sorder_detail_date := ∅
init9_r2: porder_detail_date := ∅

```

END

FIGURA 6.50: Inicialización de las variables en el refinamiento 2

6.4.16. Refinamiento 2 - Creación de una orden de venta con su detalle

Para crear una orden de venta necesitaremos un identificador de la misma y un cliente, como se ve en la Figura 6.51 en la sección del ANY, al primer parámetro se le valida que ya no este creado en el sistema mientras que con el segundo se valida que sea un cliente

previamente creado. Una vez hecha las validaciones se procede a agregar el identificador a la variable de ordenes de venta *sorders* y a la de ordenes globales *orders*, además como detalle de la factura se relacionan el cliente y el identificador *custorders*.

```
create_sell_order :
  ANY
    ord
    cust
  WHERE
    guard1_r2: ord ∈ ORDERS \ orders
    guard2_r2: cust ∈ customers
  THEN
    act1_r2: sorders := sorders ∪ ord
    act2_r2: orders := orders ∪ ord
    act3_r2: custorders := custorders ∪ ord ↳ cust
  END
```

FIGURA 6.51: Evento que permite crear una orden de venta

Para agregar los productos y la cantidad de los mismos que son vendidos usaremos el evento *add_sell_order_detail_numitems* (Figura 6.52) el cual recibe como parámetro el identificador de la orden y el cliente, los cuales ya deben estar creados como nos indica *guard1_r2* y *guard2_r2*, es necesario también el producto que se quiere asociar a la orden y el numero de unidades, en el primer caso este ya debe pertenecer a los productos de la compañía y el numero debe ser un valor entero mayor a 1.

Una vez hecha las validaciones se procede a actualizar la variable *prodorders* indicándole el producto a que orden estaría asociado y finalmente se modifica la variable *sorder_detail_numitems* con la cantidad de productos vendidos.

Finalmente solo nos resta indicar la fecha en la que fue expedida la orden, esta sigue un proceso similar al de los productos, en primer lugar como se ve en la Figura 6.53 se verifica que el indicativo de la orden este creado en el sistema, que el producto ya haya sido creado y que la fecha sea de tipo *bdate* (vista en el contexto abstracto). Una vez validadas las guardas se procede a actualizar la variable *prodorders* con el indicativo y el producto y se modifica el valor en *sorder_detail_date* con la fecha.

El mismo proceso se sigue para la creación de las ordenes de compra.

6.4.17. Refinamiento 3 - El contexto

El contexto para este nuevo refinamiento extiende del de ordenes y agrega un nuevo conjunto como se ve en la Figura 6.55. El conjunto *CATEGORIES* contiene todas las

```

add_sell_order_detail_numitems :
  ANY
    ord
    cust
    pro
    n
  WHERE
    guard1_r2: ord  $\mapsto$  cust  $\in$  custorders
    guard2_r2: ord  $\in$  sorders
    guard3_r2: pro  $\in$  products
    guard4_r2: n  $\in \mathbb{N}_1$ 
  THEN
    act1_r2: prodorders := prodorders  $\cup$  ord  $\mapsto$  pro
    act2_r2: sorder_detail_numitems(ord  $\mapsto$  pro) := n
  END

```

FIGURA 6.52: Evento que permite agregar el detalle del numero de productos a una orden de venta

```

add_sell_order_detail_date :
  ANY
    ord
    cust
    pro
    des
  WHERE
    guard1_r2: ord  $\mapsto$  cust  $\in$  custorders
    guard2_r2: ord  $\in$  sorders
    guard3_r2: pro  $\in$  products
    guard4_r2: des  $\in$  bdate
  THEN
    act1_r2: prodorders := prodorders  $\cup$  ord  $\mapsto$  pro
    act2_r2: sorder_detail_date(ord  $\mapsto$  pro) := des
  END

```

FIGURA 6.53: Evento que permite agregar el detalle de la fecha a una orden de venta

posibles categorías a las cuales se puede asociar un producto.

```

CONTEXT
  ctx_category
EXTENDS
  ctx_orders
SETS
  CATEGORIES
END

```

FIGURA 6.54: Contexto del refinamiento 3

6.4.18. Refinamiento 3 - Variables, invariantes e inicialización

En el refinamiento 3 incluimos el concepto de las categorías, estas son asociaciones de productos por características similares. En la Figura 6.55 se muestran las variables introducidas al modelo, en *categories* guardaremos las categorías, en *prodcat* tendremos los productos a que categoría pertenecen y finalmente en *categoryname* guardaremos los nombres de las categorías.

```

VARIABLES
  categories
  prodcat
  categoryname

```

FIGURA 6.55: Variables del refinamiento 3

Las categorías serán un subconjunto de todas las categorías posibles para los productos, como se aprecia en la sección de los los invariantes (Figura 6.56). La variable *prodcat* permite asociar un producto con una categoría mediante una relación y finalmente *categoryname* sera una relación entre una categoría y *powerset* de la constante *bstring*.

```

INVARIANTS
  inv1_r3: categories ⊆ CATEGORIES
  inv2_r3: prodcat ∈ products → categories
  inv3_r3: categoryname ∈ categories → ℙ(bstring)

```

FIGURA 6.56: Invariantes del refinamiento 3

Por ultimo todas las variables se igualan al conjunto vacío como se muestra en la Figura 6.57.

```

EVENTS
INITIALISATION:
THEN
  act1_r3: categories := ∅
  act2_r3: prodcat := ∅
  act3_r3: categoryname := ∅
END

```

FIGURA 6.57: Inicialización de las variables en el refinamiento 3

6.4.19. Refinamiento 3 - Eventos refinados

Al introducir el concepto de las categorías se hace necesario refinar los eventos correspondientes a la creación de los productos de los diferentes tipos, en la Figura 6.23 se presenta el refinamiento para el evento *create_resale_product*, en el se aprecia que se esta pasando un nuevo parámetro llamado *cat* que sera un identificador a una categoría previamente creada como nos indica la guarda *guard1_r3*, finalmente se actualiza la variable *prodcat* para reflejar la relación que existe entre la categoría y el producto.

```

create_resale_product :
REFINES
  create_resale_product
ANY
  cat
WHERE
  guard1_r3: cat ∈ categories
THEN
  act1_r3: prodcat := prodcat ∪ pro ↦ cat
END

```

FIGURA 6.58: Refinamiento del evento que permite crear un producto de reventa

Al igual que el anterior el evento que permite crear una lista de productos de tipo reventa también debe ser refinado como se aprecia en la Figura 6.26, la diferencia radica en que al momento de actualizar la variable *prodcat* realizamos un producto cartesiano entre la lista de productos y la categoría.

6.4.20. Refinamiento 3 - Creación y borrado de una categoría

En la Figura 6.60 se presenta el evento que permite agregar una nueva categoría al sistema, para ellos recibe como parámetros un identificador y un nombre, como se ve en la guarda *guard1_r3* el identificador no debe haber sido creado previamente en el

```

create_resale_product_list :
  REFINES
    create_resale_product_list
  ANY
    cat
  WHERE
    guard1_r3: cat ∈ categories
  THEN
    act1_r3: prodcat := prodcat ∪ (lis × cat)
  END

```

FIGURA 6.59: Refinamiento del evento que permite crear un conjunto de productos de reventa

sistema y el nombre debe ser un *powerset* de tipo *bstring*. Una vez hecha las validación se procede a actualizar la variable *categories* con el nuevo identificador y *categoryname* con la relación entre el nombre y la categoría.

```

add_category :
  ANY
    cat
    name
  WHERE
    guard1_r3: cat ∈ CATEGORIES \ categories
    guard2_r3: name ∈ ℙ(bstring)
  THEN
    act1_r3: categories := categories ∪ cat
    act2_r3: categoryname := categoryname ∪ cat ↦ name
  END

```

FIGURA 6.60: Evento que permite agregar una categoría

Para remover una categoría es necesario contar con el identificador y validar que haya sido creado previamente en el sistema como se ve en la guarda *guard1_r3* (Figura 6.62), luego de esto se procede a eliminar el identificador de la variable *categories* y de *categoryname*.

6.4.21. Refinamiento 4 - El contexto

En el refinamiento 4 se introduce el concepto de los impuestos (*taxes*), estos son cobrados por las compañías al momento de vender productos y finalmente reportados ante el estado para su recaudo, en la Figura 6.62 se muestra el nuevo contexto el cual refina al de categorías, este agrega un nuevo conjunto de todos los impuestos posibles *TAXES*

```

remove_category :
ANY
cat
WHERE
  guard1_r3: cat ∈ categories
THEN
  act1_r3: categories := categories \ cat
  act2_r3: categoryname := cat ◁ categoryname
END

```

FIGURA 6.61: Evento que permite remover una categoría

y además define un constante con su respectivo axioma *ax1_taxes* que restringe el valor posible de esta a un rango entre 0 y 100, esto es por que el impuesto cobrado es un porcentaje sobre el valor del producto.

```

CONTEXT
  ctx_taxes
EXTENDS
  ctx_category
SETS
  TAXES
CONSTANTS
  percentage
AXIOMS
  ax1_taxes : percentage = 0.,100
END

```

FIGURA 6.62: Contexto del refinamiento 4

6.4.22. Refinamiento 4 - Variables, invariantes e inicialización

Se introducen cuatro nuevas variables al modelo (Figura 6.63), en *taxes* estaría almacenado todo los impuestos creados en el sistema, en *taxprod* estaría que impuesto tiene cada producto, la descripción estaría guardada en *taxdesc* y finalmente el porcentaje del impuesto se almacenara en *taxper*.

En los invariantes (Figura 6.64) se puede observar que la variable *taxes* es un subconjunto de todos los posibles impuestos (*inv1_r4*), *taxprod* contendrá la relación entre productos e impuestos, cada impuesto debe tener un porcentaje como se ve en *taxper* y por ultimo la descripción del impuesto esta dada por la relación entre el identificador y un *powerset* de tipo *bstring*.

VARIABLES

taxes
taxprod
taxper
taxdesc

FIGURA 6.63: Variables del refinamiento 4

INVARIANTS

inv1_r4: $\text{taxes} \subseteq \text{TAXES}$
inv2_r4: $\text{taxprod} \in \text{products} \leftrightarrow \text{taxes}$
inv3_r4: $\text{taxper} \in \text{taxes} \rightarrow \text{percentage}$
inv4_r4: $\text{taxdesc} \in \text{taxes} \rightarrow \mathbb{P}(\text{bstring})$

FIGURA 6.64: Invariantes del refinamiento 4

Finalmente todas las variables se igualan al conjunto vacío como se muestra en la Figura 6.65.

EVENTS

INITIALISATION:

THEN

init1_r4: $\text{taxes} := \emptyset$
init2_r4: $\text{taxprod} := \emptyset$
init3_r4: $\text{taxper} := \emptyset$
init4_r4: $\text{taxdesc} := \emptyset$

END

FIGURA 6.65: Inicialización de las variables en el refinamiento 4

6.4.23. Refinamiento 4 - Creación, eliminación y actualización de un impuesto

Para crear un impuesto (Figura 6.66) se necesita 3 parámetros, el identificador, el porcentaje y la descripción. Como guardas se valida que el identificador no este previamente creado en el sistema, que el porcentaje este en el rango de 0..100 y la descripción debe ser un powerset de la constante *bstring*.

El impuesto tambien puede ser removido del sistema como se muestra en el evento en la Figura 6.67, para ello solo requeriremos del identificador del mismo, validaremos que este pertenezca a los creados previamente y procederemos a removerlo de las diferentes variables *taxes*, *taxprod*, *taxper* y *taxdesc*.

```

create_tax :
  ANY
    tax
    per
    des
  WHERE
    guard1_r4: tax ∈ TAXES \ taxes
    guard2_r4: per ∈ percentage
    guard3_r4: des ∈ ℙ(bstring)
  THEN
    act1_r4: taxes := taxes ∪ tax
    act2_r4: taxper := taxper ∪ tax ↦ per
    act3_r4: taxdesc := taxdesc ∪ tax ↦ des
  END

```

FIGURA 6.66: Evento que permite crear un impuesto

```

remove_tax :
  ANY
    tax
  WHERE
    guard1_r4: tax ∈ taxes
  THEN
    act1_r4: taxes := taxes \ tax
    act2_r4: taxprod := (taxprod ▷ tax)
    act3_r4: taxper := taxper \ tax ↦ taxper(tax)
    act4_r4: taxdesc := taxdesc \ tax ↦ taxdesc(tax)
  END

```

FIGURA 6.67: Evento que permite eliminar un impuesto

Es posible actualizar el porcentaje y la descripción del evento, el primer caso se presenta en la Figura 6.68 y el segundo en la Figura 6.69, para llevar a cabo esta tarea es necesario contar con el identificador del impuesto al cual se le quiere actualizar alguno de los dos elementos, también, es requerido bien sea un nuevo porcentaje y la nueva descripción. Finalmente se actualiza la variable correspondiente en el primer caso sera *taxper* y en el segundo *taxdesc*.

6.4.24. Refinamiento 4 - Agregar un impuesto a un producto

Por ultimo el impuesto debe ser aplicado a un producto o un conjunto de ellos, para el primer caso usaremos el evento *add_tax_product* (Figura 6.70), para que este funcione es necesario pasarle el identificador del producto y el del impuesto a aplicar, en la

```

update_tax :
ANY
tax
per
WHERE
guard1_r4: tax ∈ taxes
guard2_r4: per ∈ percentage
THEN
act1_r4: taxper := (taxper \ tax ↦ taxper(tax)) ∪ tax ↦ per
END

```

FIGURA 6.68: Evento que permite actualizar el porcentaje de un impuesto

```

update_tax_description :
ANY
tax
des
WHERE
guard1_r4: tax ∈ taxes
guard2_r4: des ∈ ℙ(bstring)
THEN
act1_r4: taxdesc(tax) := des
END

```

FIGURA 6.69: Evento que permite actualizar la descripción de un impuesto

guarda *guard1_r4* verificamos que el parámetro es efectivamente un producto creado previamente y en *guard2_r4* se verifica lo mismo para el impuesto. Una vez realizadas las validaciones pertinentes se procede a actualizar la variable *taxprod* para que contenga esta nueva asociación.

```

add_tax_product :
ANY
per
tax
WHERE
guard1_r4: per ∈ products
guard2_r4: tax ∈ taxes
THEN
act1_r4: taxprod := taxprod ∪ per ↦ tax
END

```

FIGURA 6.70: Evento que permite asociar un impuesto a un producto

También se puede aplicar el impuesto a un conjunto de productos (evento *add_tax plist*

Figura 6.71), en *guard1_r4* validamos que efectivamente todos los elementos de la lista estén contenidos en los productos previamente creados, y mediante un producto cartesiano entre el impuesto y la lista se actualiza la variable *taxprod*.

```
add_tax plist :
ANY
  lis
  tax
WHERE
  guard1_r4: lis ⊆ products
  guard2_r4: tax ∈ taxes
THEN
  act1_r4: taxprod := taxprod ∪ (lis × tax)
END
```

FIGURA 6.71: Evento que permite asociar un impuesto a una lista de productos

6.4.25. Refinamiento 4 - Agregar un impuesto a los productos de una categoría

Un opción interesante es la de agregar un impuesto a un conjunto de productos que pertenecen a una categoría, esta característica nos la brinda el evento *add_tax_pccategory* (Figura 6.72). Al evento debemos pasarle como parámetros el identificador del impuesto y el de la categoría, se validara que pertenecen a los creados en el sistema y finalmente se actualizara la variable *taxprod*, para llevar a cabo este proceso haremos un producto cartesiano entre el impuesto y todos los productos resultantes de buscar en la variable *prodcat* usando la categoría.

```
add_tax_pccategory :
ANY
  cat
  tax
WHERE
  guard1_r4: cat ∈ categories
  guard2_r4: tax ∈ taxes
THEN
  act1_r4: taxprod := taxprod ∪ (prodcat ∼ [cat] × tax)
END
```

FIGURA 6.72: Evento que permite asociar un impuesto a un conjunto de productos asociados por una categoría

6.4.26. Refinamiento 5 - Variables, invariantes e inicialización

En el refinamiento 5 incluiremos el concepto de reportes, en el listaremos por ejemplo los valores de los productos con su impuesto o buscaremos los precios de compra o venta de los productos.

En la Figura 6.73 se presentan las variables para este nuevo refinamiento, en *p-sprice* guardaremos los precios de venta y en *p-pprice* los de compra.

```
VARIABLES
  p-sprice
  p-pprice
```

FIGURA 6.73: Variables del refinamiento 5

En la sección de invariantes (Figura 6.74) vemos que el precio de venta y de compra pertenece a los naturales.

```
INVARIANTS
  inv1_r5: p-sprice ∈ N
  inv2_r5: p-pprice ∈ N
```

FIGURA 6.74: Invariantes del refinamiento 5

Finalmente las dos variables se igualan a 0 como se aprecia en la sección de inicialización en la Figura 6.75.

```
EVENTS
INITIALISATION:
THEN
  init1_r5: p-sprice := 0
  init2_r5: p-pprice := 0
END
```

FIGURA 6.75: Inicialización de las variables en el refinamiento 5

6.4.27. Refinamiento 5 - Obtener los precios de compra y venta

En un establecimiento comercial es necesario consultar constantemente los precios de compra y venta de los productos, el evento *get_product_sell_price* (Figura 6.76) permite buscar este dato sin agregar el impuesto para ello es necesario pasar como parámetro el identificador del producto, se valida que pertenezca al dominio de la variable *sprice*,

recordemos que esta variable guarda una relación entre un producto y su respectivo precio de venta. Por ultimo se extrae el valor del producto y se guarda en la variable *p-sprice*.

```
get_product_sell_price :
ANY
  p
WHERE
  guard1_r5:  $p \in \text{dom}(\text{sprice})$ 
THEN
  act1_r5:  $p\_sprice := \text{sprice}(p)$ 
END
```

FIGURA 6.76: Evento que permite obtener el precio de venta de un producto

Una opción muy útil es ver el precio de venta mas el impuesto, esto no lo permite hacer el evento *get_product_taxed_sell_price* (Figura 6.77), para ello necesitaremos el identificador del producto y el del impuesto que se le desea aplicar, al igual que el evento anterior este también requiere validar que el producto pertenezca al dominio de *sprice* y además que el impuesto y el producto estén asociados en la variable *taxprod*, una vez validadas las guardas se procede a multiplicar el valor de venta del producto por el del impuesto y guardarlo en *p-sprice*.

```
get_product_taxed_sell_price :
ANY
  pro
  tax
WHERE
  guard1_r5:  $pro \in \text{dom}(\text{sprice})$ 
  guard2_r5:  $pro \mapsto tax \in \text{taxprod}$ 
THEN
  act1_r5:  $p\_sprice := \text{sprice}(pro) * (1 + \text{taxper}(tax) \div 100)$ 
END
```

FIGURA 6.77: Evento que permite obtener el precio de venta con su respectivo impuesto

Por ultimo el evento *get_product_taxed_sell_price_in_order* (Figura 6.78) nos permite además ver el total por producto mas su respectivo impuesto en una orden, en este caso se considera también el numero de artículos vendidos.

En el caso de los precios de compra los eventos se comportan de forma similar a la antes explicada.

```

get_product_taxed_sell_price_in_order :
ANY
  pro
  tax
  ord
WHERE
  guard1_r5: pro ∈ dom(sprice)
  guard2_r5: pro ↦ tax ∈ taxprod
  guard3_r5: ord ↦ pro ∈ dom(sorder_detail_numitems)
THEN
  act1_r5: p_sprice := sprice(pro) * (1 + taxper(tax) ÷ 100)*
    sorder_detail_numitems(ord ↦ pro)
END

```

FIGURA 6.78: Evento que permite calcular el valor total de un producto en una orden de venta

6.5. Resultados obtenidos de la prueba de desempeño

A continuación se presentaran y analizaran los resultados obtenidos de realizar la prueba de rendimiento que permitía ver si el código generado a partir del modelo e incorporado a Openbravo presentaba un buen desempeño, respecto a la aplicación original.

Para determinar si el sistema operativo podría llegar a influir en la prueba esta se realizo en dos pc diferentes un Mac y otro Windows.

Características del PC con sistema operativo Windows:

Modelo: Acer aspire V3-571G-6602

Procesador: Intel Core i5-2450M 2.5GHz

RAM: 8 GB

Sistema operativo: Windows 7 64 bits

Cache: 3MB L3

Chipset: Intel HM77

Características del PC con sistema operativo Mac:

Modelo: Mac Pro

Procesador: Quad-Core Intel Xeon 3.2 GHz

RAM: 8 GB

Sistema operativo: Mac OS X 10.6.8

Cache: 8MB L3

Chipset:

6.5.1. Tiempos por número de operaciones tomados en el SO Windows

En la sección [6.2.1](#) se explico el funcionamiento de Openbravo y se vio que campos eran requeridos para crear cada uno de los elementos en el sistema, ya sea un cliente, proveedor, producto, bodega o una venta. Cada elemento es considerado como una transacción en las tablas presentadas a continuación, por lo que ver el numero 100 en la primera columna de la tabla [6.1](#) correspondiente a las categorías (*Categories*), es equivalente a que se midio el tiempo de crear esos 100 clientes.

Los tiempos presentados en las tablas a continuación están en dos formatos; a las izquierdas están en milisegundos y a la derecha en horas, minutos y segundos (12508214 ms = 3 hr. 28 min. 28 sec. 214 ms) esto con el fin de que sea más fácil de entender el tiempo que tarda en completarse un numero de determinado de transacciones.

En la version Openbravo modifica con código generado formalmente se tomo el tiempo en que se daba en el botón guardar de la vista hasta que se terminaba de actualizar las variables en memoria. Con la version original se tomo el tiempo que tarda la transacción desde que se da en el botón guardar hasta que se actualiza la base de datos, para revisar esto se creo un trigger por tabla que informaba de los cambios.

Si se observa las tablas de categorías (Tabla [6.1](#)), clientes (Tabla [6.2](#)) y bodegas (Tabla [6.3](#)), los tiempos entre la version modificada y la original no difieren en mucho, esto supone que el rendimiento de la aplicación en estos procesos se lleva a cabo de una manera eficiente.

El mismo comportamiento se observa en los tiempos obtenidos en el sistema operativo Mac, si se observan las Tablas [6.7](#), [6.8](#) y [6.9](#), por lo que aunque cambiemos de sistema operativo el patrón se conserva, es decir, es eficiente para estos 3 procesos.

Al observar la tabla correspondiente a los impuestos (Tabla [6.4](#)) notamos que el patrón cambia y la aplicación que fue modificada con código formal se torna ineficiente respecto a la original, al final el proceso de crear 50000 elementos en memoria tardo el doble. Esto mismo se repite al realizar la prueba en Mac (Tabla [6.10](#)) tarda el doble en llevar a cabo todo el proceso.

Finalmente en los procesos mas importantes de un sistema de inventarios correspondientes a crear los productos y a procesar una venta el código generado presenta problemas de memoria, en el caso de los productos Tabla [6.5](#) no se logra llegar a crear los 1000

| # Tran. | Tiempo Openbravo Modificado | Tiempo Openbravo |
|---------|------------------------------------|-----------------------------------|
| 100 | 49849ms ≈ 49sec 849ms | 42834ms ≈ 42sec 834ms |
| 500 | 250165ms ≈ 4min 10sec 165ms | 238247ms ≈ 4min |
| 1000 | 500531ms ≈ 8min 20sec 531ms | 498925ms ≈ 8min 18sec 925ms |
| 5000 | 2503507ms ≈ 41min 43sec 507ms | 2399211ms ≈ 40min |
| 10000 | 5006065ms ≈ 1hr 23min 26sec 65ms | 5001035ms ≈ 1hr 23min 21 sec 35ms |
| 25000 | 12508214ms ≈ 3hr 28min 28sec 214ms | 12488004ms ≈ 3hr 28min 8 sec 4ms |
| 50000 | 2501844 ms ≈ 7hr | 25004113ms ≈ 7hr |

CUADRO 6.1: Tiempos para ingreso de datos en la vista *Categories*

| # Tran. | Tiempo Openbravo Modificado | Tiempo Openbravo |
|---------|------------------------------------|------------------------------------|
| 100 | 49800ms ≈ 49sec 800ms | 44359ms ≈ 44 sec. 359 ms |
| 500 | 249915ms ≈ 4min 9sec 915ms | 245378ms ≈ 4min 5sec 378ms |
| 1000 | 500090ms ≈ 8min 20sec 90ms | 499835ms ≈ 8min 19sec 835ms |
| 5000 | 2501495ms ≈ 41min 41sec 495ms | 2402305ms ≈ 40min 2sec 305ms |
| 10000 | 5002852ms ≈ 1hr 23min 22sec 852ms | 5002124ms ≈ 1hr 23min 22sec 124ms |
| 25000 | 12505173ms ≈ 3hr 28min 25sec 173ms | 12499578ms ≈ 3hr 28min 19sec 578ms |
| 50000 | 25013000ms ≈ 7hr | 25010558ms ≈ 7 hr |

CUADRO 6.2: Tiempos para ingreso de datos en la vista *Customers*

| # Tran. | Tiempo Openbravo Modificado | Tiempo Openbravo |
|---------|-----------------------------------|------------------------------------|
| 100 | 49832ms ≈ 49sec 832ms | 49005ms ≈ 49sec 5ms |
| 500 | 249872ms ≈ 4min 9sec. 872ms | 247706ms ≈ 4min 7sec 706ms |
| 1000 | 499900ms ≈ 8min 19sec 900ms | 501087ms ≈ 8min 21sec 87ms |
| 5000 | 2500288ms ≈ 41min 40sec 288ms | 2499875ms ≈ 41min 39sec 875ms |
| 10000 | 5000879ms ≈ 1hr 23min 20sec 879ms | 4998799ms ≈ 1hr 23min 18sec 799ms |
| 25000 | 12501987ms ≈ 3hr 28min 22sec | 12500456ms ≈ 3hr 28min 20sec 456ms |
| 50000 | 25003659ms ≈ 7hr | 25000145ms ≈ 7hr |

CUADRO 6.3: Tiempos para ingreso de datos en la vista *Warehouse*

productos, en el caso de Mac (Tabla 6.11) ocurre exactamente lo mismo. En el caso de la venta Tabla 6.6, se pasa de las mil transacciones pero no mas halla de eso, igual ocurre en Mac Tabla 6.12.

Al intentar deducir el motivo por el cual en impuestos (*Taxes*) teníamos una baja de rendimiento y finalmente en ventas y productos no era posible completar las pruebas, notamos que el problema se presenta al incrementar el numero de variables afectadas en estos eventos. En el caso de la creación de un impuesto se veían afectadas 3, mientras que con las categorías eran 2 y en el caso de los productos eran 9.

| # Tran. | Tiempo Openbravo Modificado | Tiempo Openbravo |
|---------|----------------------------------|------------------------------------|
| 100 | 96329ms ≈ 1min 36sec 329ms | 49299ms ≈ 49sec 299ms |
| 500 | 492173ms ≈ 8min 12sec 173ms | 272089ms ≈ 4min 32sec 89ms |
| 1000 | 989181ms ≈ 16min 29sec 181ms | 504005ms ≈ 8min 24sec 5ms |
| 5000 | 4983618ms ≈ 1hr 23min 3sec 618ms | 2500231ms ≈ 41min 40sec 231ms |
| 10000 | 9972074ms ≈ 2hr 46min 12sec 74ms | 5001078ms ≈ 1hr 23min 21sec 78ms |
| 25000 | 25028657ms ≈ 7hr | 12497419ms ≈ 3hr 28min 17sec 419ms |
| 50000 | 50437495ms ≈ 14hr 37sec 495ms | 25013997ms ≈ 7hr |

CUADRO 6.4: Tiempos para ingreso de datos en la vista *Taxes*

| # Tran. | Tiempo Openbravo Modificado | Tiempo Openbravo |
|---------|-------------------------------|--------------------------------------|
| 100 | 155116ms ≈ 2min 35sec 116ms | 209887ms ≈ 3min 29sec 887ms |
| 500 | 1226294ms ≈ 20min 26sec 294ms | 1069887ms ≈ 17min 49sec 887ms |
| 1000 | | 2146649ms ≈ 35min 46sec 649ms |
| 5000 | | 10760505ms ≈ 3hr |
| 10000 | | 21529096ms ≈ 6hr |
| 25000 | | 53831922ms ≈ 15hr |
| 50000 | | 107679533ms ≈ 29hr 54min 39sec 533ms |

CUADRO 6.5: Tiempos para ingreso de datos en la vista *Products*

| # Tran. | Tiempo Openbravo Modificado | Tiempo Openbravo |
|---------|-------------------------------|--------------------------------------|
| 100 | 199804ms ≈ 3min 19sec 804ms | 209268ms ≈ 3min 29sec 268ms |
| 500 | 1216285ms ≈ 20min 16sec 285ms | 1068165ms ≈ 17min 48sec 165ms |
| 1000 | 2217610ms ≈ 37min | 214500ms ≈ 35min 45sec 7ms |
| 5000 | | 10758005ms ≈ 3hr |
| 10000 | | 21527029ms ≈ 6hr |
| 25000 | | 53830489ms ≈ 15hr |
| 50000 | | 107663467ms ≈ 29hr 54min 23sec 467ms |

CUADRO 6.6: Tiempos para ingreso de datos en la vista *Sale*

| # Tran. | Tiempo Openbravo Modificado | Tiempo Openbravo |
|---------|------------------------------------|----------------------------------|
| 100 | 69068ms ≈ 1min 9sec 68ms | 58054ms ≈ 1min |
| 500 | 349115ms ≈ 5min 49sec 115ms | 298045ms ≈ 5min |
| 1000 | 699200ms ≈ 11min 39sec 200ms | 576997ms ≈ 9min 37sec |
| 5000 | 3499762ms ≈ 1hr | 3129745ms ≈ 52min 9sec 745ms |
| 10000 | 7000478ms ≈ 2hr | 6785987ms ≈ 1hr 53min 6sec |
| 25000 | 17502499ms ≈ 4hr 51min 42sec 499ms | 16487965ms ≈ 4hr 34min 48sec |
| 50000 | 35005764ms ≈ 9hr 43min 25sec 764ms | 34087145ms ≈ hr 28min 7sec 145ms |

CUADRO 6.7: Tiempos para ingreso de datos en la vista *Categories* en Mac OS X

| # Tran. | Tiempo Openbravo Modificado | Tiempo Openbravo |
|---------|--|---------------------------------------|
| 100 | 48755ms ≈ 48 sec 755 ms | 49015ms ≈ 49 sec 15 ms |
| 500 | 248789ms ≈ 4 min 8 sec 789 ms | 245897ms ≈ 4 min 5 sec 897 ms |
| 1000 | 498834ms ≈ 8 min 18 sec 834 ms | 489103ms ≈ 8 min 9 sec 103 ms |
| 5000 | 2499265ms ≈ 41 min 39 sec 265 ms | 2489995ms ≈ 41 min 30 sec |
| 10000 | 4999766ms ≈ 1 hr 23 min 19 sec 766 ms | 5000127ms ≈ 1 hr 23 min 20 sec 127 ms |
| 25000 | 12501439ms ≈ 3 hr 28 min 21 sec 439 ms | 12406987ms ≈ 3 hr 26 min 47 sec |
| 50000 | 25004778ms ≈ 7 hr | 25001149ms ≈ 7 hr |

CUADRO 6.8: Tiempos para ingreso de datos en la vista *Customers* Mac OS X

| # Tran. | Tiempo Openbravo Modificado | Tiempo Openbravo |
|---------|---------------------------------------|---------------------------------------|
| 100 | 48778ms ≈ 48 sec 778 ms | 47564ms ≈ 47 sec 564 ms |
| 500 | 248837ms ≈ 4 min 8 sec 837 ms | 247125ms ≈ 4 min 7 sec 125 ms |
| 1000 | 498933ms ≈ 8 min 18 sec 933 ms | 499015ms ≈ 8 min 19 sec 15 ms |
| 5000 | 2499513ms ≈ 41 min 39 sec 513 ms | 2500021ms ≈ 41 min 40 sec 21 ms |
| 10000 | 5000184ms ≈ 1 hr 23 min 20 sec 184 ms | 5001005ms ≈ 1 hr 23 min 21 sec 5 ms |
| 25000 | 12502097ms ≈ 3 hr 28 min 22 sec 97 ms | 12501087ms ≈ 3 hr 28 min 21 sec 87 ms |
| 50000 | 25005492ms ≈ 7 hr | 25078002ms ≈ 7 hr |

CUADRO 6.9: Tiempos para ingreso de datos en la vista *Warehouse* en Mac OS X

| # Tran. | Tiempo Openbravo Modificado | Tiempo Openbravo |
|---------|--|--|
| 100 | 49022ms ≈ 49 sec 22 ms | 49078ms ≈ 49 sec 78 ms |
| 500 | 249077ms ≈ 4 min 9 sec 77 ms | 244956ms ≈ 4 min 5 sec |
| 1000 | 499136ms ≈ 8 min 19 sec 136 ms | 498124ms ≈ 8 min 18 sec 124 ms |
| 5000 | 2499695ms ≈ 41 min 39 sec 695 ms | 2455973ms ≈ 40 min 56 sec |
| 10000 | 5000407ms ≈ 1 hr 23 min 20 sec 407 ms | 5001008ms ≈ 1 hr 23 min 21 sec 8 ms |
| 25000 | 12502364ms ≈ 3 hr 28 min 22 sec 364 ms | 12500254ms ≈ 3 hr 28 min 20 sec 254 ms |
| 50000 | 25005635ms ≈ 7 hr | 24998467ms ≈ 7 hr |

CUADRO 6.10: Tiempos para ingreso de datos en la vista *Taxes* en Mac OS X

| # Tran. | Tiempo Openbravo Modificado | Tiempo Openbravo |
|---------|---------------------------------|--|
| 100 | 124898ms ≈ 2 min 4 sec 898 ms | 209989ms ≈ 3 min 30 sec |
| 500 | 627522ms ≈ 10 min 27 sec 522 ms | 1070389ms ≈ 17 min 50 sec 389 ms |
| 1000 | | 2146876ms ≈ 35 min 46 sec 876 ms |
| 5000 | | 10761743ms ≈ 3 hr |
| 10000 | | 21529979ms ≈ 6 hr |
| 25000 | | 53833952ms ≈ 15 hr |
| 50000 | | 107689271ms ≈ 29 hr 54 min 49 sec 271 ms |

CUADRO 6.11: Tiempos para ingreso de datos en la vista *Products* en Mac OS X

| # Tran. | Tiempo Openbravo Modificado | Tiempo Openbravo |
|---------|----------------------------------|--|
| 100 | 199940ms ≈ 3 min 19 sec 940 ms | 209971ms ≈ 3 min 30 sec |
| 500 | 1001602ms ≈ 16 min 41 sec 602 ms | 1068886ms ≈ 17 min 48 sec 886 ms |
| 1000 | 2003616ms ≈ 33 min 23 sec 616 ms | 2145767ms ≈ 35 min 45 sec 767 ms |
| 5000 | | 10758975ms ≈ 3 hr |
| 10000 | | 21527799ms ≈ 6 hr |
| 25000 | | 53831251ms ≈ 15 hr |
| 50000 | | 107663695ms ≈ 29 hr 54 min 23 sec 695 ms |

CUADRO 6.12: Tiempos para ingreso de datos en la vista *Sales* en Mac OS X

Capítulo 7

Conclusiones

1. A excepción del método insertar de las clases BSet y BRelation el rendimiento mejoró considerablemente, esto debido a que la inserción se hace de manera ordenada esto reduce el tiempo al momento de ingresar los datos, pero es sumamente eficiente con operaciones de búsqueda. La estructura de datos seleccionada fue TreeSet de Java, con la cual se reescribió desde 0 este conjunto de clases.
2. El código generado a partir de un modelo en Event-B no presenta un buen desempeño para grandes volúmenes de datos. Los tiempos entre la versión de OpenBravo + EventB2Java y la original son muy cercanos cuando las operaciones no requieren la modificación de más de dos variables en el código generado por la herramienta o cuando el numero de operaciones de almacenamiento de datos en memoria no excede los 500, pero cuando el numero de variables aumenta la aplicación baja considerablemente su rendimiento hasta el punto de presentar problemas de memoria.
3. Cuando las operaciones en el código generado requiere de modificar 3 o más variables el tiempo crece de manera exponencial o agota la memoria de la maquina virtual de java (para la creación de productos y realizar ventas se genero un problema de java heap memory en el caso de Windows alrededor de las 700 a 800 transacciones y en el caso de Mac alrededor de las 1000).
4. La mayor dificultad en la prueba de desempeño de la aplicación, radicaba en el hecho de que el código generado por EventB2Java hace uso de la memoria en cambio la versión original de Openbravo hace uso de una base de datos, por lo que tomar los tiempos de los procesos no se hacia de forma idéntica, en el caso del Openbravo original se realizo mediante un trigger que se disparaba cuando se modificaba la tabla sobre la cual se quería analizar los tiempos, una vez se

hacia un cambio se guardaba en un archivo el tiempo. En el caso de la versión modificada esto se hacia directamente en el programa y se media el tiempo que tomaba actualizando las variables en el código formal.

5. Las pruebas fueron susceptibles a fallos en el PC por lo que en algunas ocasiones era necesario volver a empezar con la creación de los distintos elementos (Impuestos, Categorías, Bodegas, etc) entre 2 y hasta 3 veces, para obtener los datos adecuadamente, esto supuso grandes perdidas de tiempo, puesto que como se vio en el caso de los impuesto el correr las 50000 transacciones tardo al rededor de 14 horas.
6. El modelo formal incluyó elementos que son tópicos de un aplicativo de inventarios pero que no fueron usados en Openbravo, como por ejemplo los eventos de los reportes o los eventos de creación y borrado de proveedores. En el modelo tampoco se tuvo en cuenta el nivel de detalle que presentaban los formularios para ingresar productos, clientes, bodegas etc, por ejemplo en las bodegas no se incluyó la dirección de la misma, si se hubieran incluido todos los campos, ningún proceso de la versión modificada con código formal podría haberse comparado respecto a la original puesto que el numero de variables a modificar excedería el límite de 3 que es el umbral eficiente.

7.1. Trabajo futuro

1. El mayor de los inconvenientes en la prueba es el hecho de que EventB2Java genere código que deja la información en memoria, para aplicaciones comerciales lo más normal es hacer uso de base de datos para guardar la información, se plante entonces repetir las pruebas incorporando el trabajo realizado por Tim¹ (Generación de código + base de datos).
2. La industria de la informática se está moviendo hacia la nube, se propone evaluar la generación de código que sea ejecutable en estos ambientes.

¹ <http://users.dickinson.edu/~wahlst/eventb2sql/eventb2sql.html>

Apéndice A

Modulo de inventario en Event-B

A.1. Maquina abstracta

CONTEXT

ctx_abstract

SETS

SUPPLIERS

CUSTOMERS

PRODUCTS

NAMES

WAREHOUSES

CONSTANTS

bstring

bday

bmonth

byear

bdate

AXIOMS

ax1: bstring $\in \mathbb{N} \rightarrow \mathbb{N}$

ax2: finite(bstring)

ax3: dom(bstring) = 1..card(bstring)

ax4: bday = 1..31

ax5: bmonth = 1..12

ax6: byear = 1..3000

ax7: bdate $\in (bday \times bmonth) \leftrightarrow byear$

END

MACHINE

ref0_Inventory

SEES

ctx_orders

VARIABLES

customers

suppliers

pprice

sprice

rawmaterial

manufacturing

resale

material

products

customername

suppliername

warehouses

warehousename

stock

stocknumber

prodsupplier

INVARIANTS**inv1:** customers \subseteq CUSTOMERS**inv2:** suppliers \subseteq SUPPLIERS**inv3:** products \subseteq PRODUCTS**inv4:** resale \cup rawmaterial \cup manufacturing=products**inv5:** resale \cap rawmaterial= \emptyset **inv6:** pprice \in (resale \cup rawmaterial) $\rightarrow \mathbb{N}$ **inv7:** sprice \in (resale \cup manufacturing) $\rightarrow \mathbb{N}$ **inv8:** resale \cap manufacturing= \emptyset **inv9:** rawmaterial \cap manufacturing = \emptyset **inv10:** material \in (manufacturing $\leftrightarrow \mathbb{P}(\text{rawmaterial})$)**inv11:** customername \in customers $\rightarrow \mathbb{P}(\text{bstring})$ **inv12:** suppliername \in suppliers $\rightarrow \mathbb{P}(\text{bstring})$ **inv13:** warehouses \subseteq WAREHOUSES**inv14:** warehousename \in warehouses $\rightarrow \mathbb{P}(\text{bstring})$ **inv15:** stock \in warehouses \leftrightarrow products

inv16: stocknumber \in stock $\rightarrow \mathbb{N}$

inv17: prodsupplier \in suppliers \leftrightarrow products

EVENTS

INITIALISATION:

THEN

act1: rawmaterial := \emptyset

act2: resale := \emptyset

act3: suppliers := \emptyset

act4: customers := \emptyset

act5: pprice := \emptyset

act6: sprice := \emptyset

act7: products := \emptyset

act8: manufacturing := \emptyset

act9: material := \emptyset

act10: customername := \emptyset

act11: suppliername := \emptyset

act12: warehouses := \emptyset

act13: warehousename := \emptyset

act14: stock := \emptyset

act15: stocknumber := \emptyset

act16: prodsupplier := \emptyset

END

create_raw_material:

ANY

pro

purchaseprice

war

WHERE

grd1: pro \in PRODUCTS \ products

grd2: purchaseprice $\in \mathbb{N}1$

grd3: war \in warehouses

grd4: war \mapsto pro \notin stock

THEN

act1: rawmaterial := rawmaterial \cup pro

act2: pprice := pprice \cup pro \mapsto purchaseprice

act3: products := products \cup pro

act4: stocknumber := stocknumber \cup (war \mapsto pro) \mapsto 0
act5: stock := stock \cup war \mapsto pro
END

create_raw_material_list:
ANY
lis
ppricel
war
WHERE
grd1: lis \subseteq PRODUCTS \ products
grd2: ppricel \in lis \rightarrow \mathbb{N}
grd3: war \in warehouses
grd4: (war \times lis) \cap stock = \emptyset
THEN
act1: rawmaterial:=rawmaterial \cup lis
act2: pprice:=pprice \cup ppricel
act4: products:=products \cup lis
act5: stocknumber:=stocknumber \cup ((war \times lis) \times 0)
act6: stock:=stock \cup (war \times lis)
END

stock_raw_material:
ANY
pro
n
war
WHERE
grd1: pro \in rawmaterial
grd2: n \in \mathbb{N}
grd3: war \mapsto pro \in stock
THEN
act1: stocknumber(war \mapsto pro) := stocknumber(war \mapsto pro) + n
END

unstock_raw_material:

ANY

pro

n

war

WHERE

grd1: pro \in rawmaterial

grd2: n $\in \mathbb{N}$

grd3: war \mapsto pro \in stock

grd4: stocknumber(war \mapsto pro)-n ≥ 0

THEN

act1: stocknumber(war \mapsto pro) := stocknumber(war \mapsto pro)-n

END

create_resale_product:

ANY

pro

purchaseprice

sellprice

war

WHERE

grd1: pro \in PRODUCTS \ products

grd2: purchaseprice $\in \mathbb{N}$

grd3: sellprice $\in \mathbb{N}$

grd4: war \in warehouses

grd5: war \mapsto pro \notin stock

THEN

act1: resale:=resale \cup pro

act2: pprice:=pprice \cup pro \mapsto purchaseprice

act3: sprice:=sprice \cup pro \mapsto sellprice

act4: products:=products \cup pro

act5: stocknumber:=stocknumber \cup (war \mapsto pro) \mapsto 0

act6: stock:=stock \cup war \mapsto pro

END

create_resale_product_list:

ANY

lis

pprice
 sprice
 war
WHERE
grd1: lis \subseteq PRODUCTS \ products
grd2: pprice \in lis $\rightarrow \mathbb{N}$
grd3: sprice \in lis $\rightarrow \mathbb{N}$
grd4: war \in warehouses
grd5: (war \times lis) \cap stock = \emptyset
THEN
act1: resale:=resale \cup lis
act2: pprice:=pprice \cup pprice
act3: sprice:=sprice \cup sprice
act4: products:=products \cup lis
act5: stocknumber := stocknumber \cup ((war \times lis) \times 0)
act6: stock:=stock \cup (war \times lis)
END

stock_resale_product:
ANY
 pro
 n
 war
WHERE
grd1: pro \in resale
grd2: n $\in \mathbb{N}$
grd3: war \mapsto pro \in stock
THEN
act1: stocknumber(war \mapsto pro) := stocknumber(war \mapsto pro)+n
END

unstock_resale_product:
ANY
 pro
 n
 war
WHERE
grd1: pro \in resale

grd2: $n \in \mathbb{N}1$
grd3: $\text{war} \mapsto \text{pro} \in \text{stock}$
grd4: $\text{stocknumber}(\text{war} \mapsto \text{pro}) - n \geq 0$
THEN
act1: $\text{stocknumber}(\text{war} \mapsto \text{pro}) := \text{stocknumber}(\text{war} \mapsto \text{pro}) - n$
END

create_manufacturing_product:
ANY
 cpro
 csellprice
 crawmaterialpro
 cwar
WHERE
grd1: $\text{cpro} \in \text{PRODUCTS} \setminus \text{products}$
grd2: $\text{csellprice} \in \mathbb{N}1$
grd3: $\text{crawmaterialpro} \subseteq \text{rawmaterial}$
grd4: $\text{cpro} \mapsto \text{crawmaterialpro} \notin \text{material}$
grd5: $\text{cwar} \mapsto \text{cpro} \in \text{stock}$
THEN
act1: $\text{manufacturing} := \text{manufacturing} \cup \text{cpro}$
act2: $\text{material} := \text{material} \cup \text{cpro} \mapsto \text{crawmaterialpro}$
act3: $\text{sprice} := \text{sprice} \cup \text{cpro} \mapsto \text{csellprice}$
act5: $\text{products} := \text{products} \cup \text{cpro}$
act6: $\text{stocknumber} := \text{stocknumber} \cup (\text{cwar} \mapsto \text{cpro}) \mapsto 0$
act7: $\text{stock} := \text{stock} \cup \text{cwar} \mapsto \text{cpro}$
END

create_manufacturing_product_list:
ANY
 lis
 spricel
 rawmaterialpro
 war
WHERE
grd1: $\text{lis} \subseteq \text{PRODUCTS} \setminus \text{products}$
grd2: $\text{spricel} \in \text{lis} \rightarrow \mathbb{N}$
grd3: $\text{rawmaterialpro} \in \text{lis} \leftrightarrow \mathbb{P}(\text{rawmaterial})$

grd4: war \in warehouses
grd5: (war \times lis) \cap stock = \emptyset
THEN
act1: manufacturing:=manufacturing \cup lis
act2: material:=material \cup rawmaterialpro
act3: sprice:=sprice \cup spricel
act4: products:=products \cup lis
act6: stocknumber := stocknumber \cup ((war \times lis) \times 0)
act7: stock:=stock \cup (war \times lis)
END

stock_manufacturing_product:

ANY
 pro
 n
 war
WHERE
grd1: pro \in manufacturing
grd2: n $\in \mathbb{N}1$
grd3: war \mapsto pro \in stock
THEN
act1: stocknumber(war \mapsto pro) := stocknumber(war \mapsto pro)+n
END

unstock_manufacturing_product:

ANY
 pro
 n
 war
WHERE
grd1: pro \in manufacturing
grd2: n $\in \mathbb{N}1$
grd3: war \mapsto pro \in stock
grd4: stocknumber(war \mapsto pro) - n=0
THEN
act1: stocknumber(war \mapsto pro) := stocknumber(war \mapsto pro) - n
END

add_supplier:
ANY
sup
name
WHERE
grd1: sup \in SUPPLIERS \ suppliers
grd2: name \in $\mathbb{P}(\text{bstring})$
THEN
act1: suppliers:=suppliers \cup sup
act2: suppliername:=suppliername \cup sup \mapsto name
END

delete_supplier:
ANY
sup
WHERE
grd1: sup \in suppliers
THEN
act1: suppliers := suppliers \ sup
act2: suppliername := sup \triangleleft suppliername
act3: prodsupplier := sup \triangleleft prodsupplier
END

edit_supplier_name:
ANY
sup
name
WHERE
grd1: sup \in suppliers
grd2: name \in $\mathbb{P}(\text{bstring})$
THEN
act1: suppliername(sup) := name
END

add_product_supplier:
ANY

prod
sup
WHERE
grd1: prod ∈ products
grd2: sup ∈ suppliers
THEN
act1: prodsupplier := prodsupplier ∪ sup ↦ prod
END

delete_product_supplier:
ANY
pro
sup
WHERE
grd1: pro ∈ products
grd2: sup ∈ suppliers
THEN
act1: prodsupplier := prodsupplier ∪ sup ↦ pro
END

add_customer:
ANY
cust
name
WHERE
grd1: cust ∈ CUSTOMERS \ customers
grd2: name ∈ $\mathbb{P}(\text{bstring})$
THEN
act1: customers:=customers ∪ cust
act2: customername:=customername ∪ cust ↦ name
END

edit_customer:
ANY
cust
newcust

newname
WHERE
grd1: cust $\in \text{dom}(\text{customername})$
grd2: newcust $\in \text{CUSTOMERS} \setminus \text{customers}$
grd3: newname $\in \mathbb{P}(\text{bstring})$
THEN
act1: customers:=(customers \setminus cust) \cup newcust
act2: customername := (customername \setminus cust \mapsto customername(cust)) \cup newcust \mapsto newname
END

edit_purchase_price:
ANY
pro
pri
WHERE
grd1: pro $\in \text{products} \setminus \text{manufacturing}$
grd4: pri $\in \mathbb{N}^1$
grd5: pro $\in \text{dom}(\text{pprice})$
THEN
act1: pprice:=(pprice \setminus pro \mapsto pprice(pro)) \cup pro \mapsto pri
END

edit_sell_price:
ANY
pro
pri
WHERE
grd1: pro $\in \text{products} \setminus \text{rawmaterial}$
grd4: pri $\in \mathbb{N}^1$
grd5: pro $\in \text{dom}(\text{sprice})$
THEN
act1: sprice:=(sprice \setminus pro \mapsto sprice(pro)) \cup pro \mapsto pri
END

add_warehouse:
ANY
war

name
WHERE
grd1: war ∈ WAREHOUSES \ warehouses
grd2: name ∈ ℙ(bstring)
THEN
act1: warehouses := warehouses ∪ war
act2: warehousename := warehousename ∪ war ↦ name
END

remove_warehouse:
ANY
war
WHERE
grd1: war ∈ warehouses
THEN
act1: warehouses := warehouses \ war
act2: warehousename := war ⊲ warehousename
act3: stock := (war ⊲ stock)
act4: stocknumber := ((war × products) ⊲ stocknumber)
END

A.2. Refinamiento 1: Habilitación y deshabilitación de productos, clientes y proveedores (ref1_Enabling)

MACHINE
ref1_Enabling REFINES
ref0_Inventory SEES
ctx_orders VARIABLES
enabled_supplier
enabled_customer
enabled_product
INVARIANTS
inv1_r1: enabled_product ⊆ products
inv2_r1: enabled_supplier ⊆ suppliers
inv3_r1: enabled_customer ⊆ customers
EVENTS

INITIALISATION:

THEN

act1_r1: enabled_product := \emptyset

act2_r1: enabled_supplier := \emptyset

act3_r1: enabled_customer := \emptyset

END

create_raw_material:

REFINES

create_raw_material

ANY

WHERE

THEN

act1_r1: enabled_product:=enabled_product \cup pro

END

create_raw_material_list:

REFINES

create_raw_material_list

ANY

WHERE

THEN

act1_r1: enabled_product:=enabled_product \cup lis

END

stock_raw_material:

REFINES

stock_raw_material

ANY

WHERE

guard1_r1: pro \in enabled_product

THEN

END

```
unstock_raw_material:  
REFINES  
unstock_raw_material  
ANY  
WHERE  
guard1_r1: pro ∈ enabled_product  
THEN  
END
```

```
create_manufacturing_product:  
REFINES  
create_manufacturing_product  
ANY  
WHERE  
THEN  
act1_r1: enabled_product := enabled_product ∪ pro  
END
```

```
create_manufacturing_product_list:  
REFINES  
create_manufacturing_product_list  
ANY  
WHERE  
THEN  
act1_r1: enabled_product := enabled_product ∪ lis  
END
```

```
stock_manufacturing_product:  
REFINES  
stock_manufacturing_product  
ANY  
WHERE
```

guard1_r1: pro ∈ enabled_product
THEN
END

unstock_manufacturing_product:
REFINES
unstock_manufacturing_product
ANY
WHERE
guard1_r1: pro ∈ enabled_product
THEN
END

create_resale_product:
REFINES
create_resale_product
ANY
WHERE
THEN
act1_r1: enabled_product := enabled_product ∪ pro
END

create_resale_product_list:
REFINES
create_resale_product_list
ANY
WHERE
THEN
act1_r1: enabled_product := enabled_product ∪ lis
END

stock_resale_product:
REFINES

```
stock_resale_product
ANY
WHERE
guard1_r1: pro ∈ enabled_product
THEN
END
```

```
unstock_resale_product:
REFINES
unstock_resale_product
ANY
WHERE
guard1_r1: pro ∈ enabled_product
THEN
END
```

```
add_supplier:
REFINES
add_supplier
ANY
WHERE
act1_r1: enabled_supplier := enabled_supplier ∪ sup
END
```

```
add_customer:
REFINES
add_customer
ANY
WHERE
act1_r1: enabled_customer := enabled_customer ∪ cust
END
```

```
edit_customer:  
REFINES  
edit_customer  
ANY  
WHERE  
THEN  
act1_r1: enabled_customer := (enabled_customer \ cust) ∪ newcust  
END
```

```
delete_supplier:  
REFINES  
delete_supplier  
ANY  
WHERE  
THEN  
act1_r1: enabled_supplier := enabled_supplier \ sup  
END
```

```
enable_resale_product:  
ANY  
pro  
WHERE  
grd1: pro ∈ resale  
grd2: pro ∉ enabled_product  
THEN  
act1: enabled_product := enabled_product ∪ pro  
END
```

```
disable_resale_product:  
ANY  
pro  
WHERE
```

grd1: pro \in resale
grd2: pro \in enabled_product
THEN
act1: enabled_product := (enabled_product \ pro)
END

enable_supplier:
ANY
sup
WHERE
grd1: sup \in suppliers
grd2: sup \notin enabled_supplier
THEN
act1: enabled_supplier := enabled_supplier \cup sup
END

disable_supplier:
ANY
sup
WHERE
grd1: sup \in suppliers
grd2: sup \in enabled_supplier
THEN
act1: enabled_supplier := (enabled_supplier \ sup)
END

enable_customer:
ANY
cust
WHERE
grd1: cust \in customers
grd2: cust \notin enabled_customer
THEN
act1: enabled_customer := enabled_customer \cup cust

END

disable_customer:
ANY
cust
WHERE
grd1: cust \in customers
grd2: cust \in enabled_customer
THEN
act1: enabled_customer:=(enabled_customer \ cust)
END

enable_rawmaterial_product:
ANY
pro
WHERE
grd1: pro \in rawmaterial
grd2: pro \notin enabled_product
THEN
act1: enabled_product:=enabled_product \cup pro
END

disable_rawmaterial_product:
ANY
pro
WHERE
grd1: pro \in rawmaterial
grd2: pro \in enabled_product
THEN
act1: enabled_product := (enabled_product \ pro)
END

```

enable_manufacturing:
ANY
pro
WHERE
grd1: pro ∈ manufacturing
grd2: pro ∉ enabled_product
THEN
act1: enabled_product:=enabled_product ∪ pro
END

```

```

disable_manufacturing:
ANY
pro
WHERE
grd1: pro ∈ manufacturing
grd2: pro ∈ enabled_product
THEN
act1: enabled_product := (enabled_product \ pro)
END

```

A.3. Refinamiento 2 - Ordenes (ref2_Orders)

MACHINE
 ref2_Orders
 REFINES
 ref1_Eabling
 SEES
 ctx_orders

VARIABLES
 orders
 porders
 sorders
 prodorders
 custorders

sorder_detail_numitems
 porder_detail_numitems
 sorder_detail_date
 porder_detail_date

INVARIANTS

inv1_r2: orders \subseteq ORDERS
inv2_r2: porders \cup sorders = orders
inv3_r2: porders \cap sorders = \emptyset
inv4_r2: prodorders \in orders \leftrightarrow products
inv5_r2: custorders \in orders \leftrightarrow customers
inv6_r2: sorder_detail_numitems \in (orders \times products) \rightarrowtail N1
inv7_r2: porder_detail_numitems \in (orders products) \rightarrowtail N1
inv8_r2: sorder_detail_date \in (orders \times products) \rightarrowtail bdate
inv9_r2: porder_detail_date \in (orders \times products) \rightarrowtail bdate

EVENTS

INITIALISATION:

THEN

init1_r2: orders := \emptyset
init2_r2: porders := \emptyset
init3_r2: sorders := \emptyset
init4_r2: prodorders := \emptyset
init5_r2: custorders := \emptyset
init6_r2: sorder_detail_numitems := \emptyset
init7_r2: porder_detail_numitems := \emptyset
init8_r2: sorder_detail_date := \emptyset
init9_r2: porder_detail_date := \emptyset

END

edit_customer:

REFINES

edit_customer ANY

WHERE

THEN

act1_rf2: custorders := custorders \triangleright cust
END

create_sell_order:

ANY

ord

cust

WHERE

guard1_r2: ord \in ORDERS \ orders

guard2_r2: cust \in customers

THEN

act1_r2: sorders := sorders \cup ord

act2_r2: orders := orders \cup ord

act3_r2: custorders := custorders \cup ord \mapsto cust

END

add_sell_order_detail_numitems:

ANY

ord

cust

pro

n

WHERE

guard1_r2: ord \mapsto cust \in custorders

guard2_r2: ord \in sorders

guard3_r2: pro \in products

guard4_r2: n \in N1

THEN

act1_r2: prodorders := prodorders \cup ord \mapsto pro

act2_r2: sorder_detail_numitems(ord \mapsto pro) := n

END

add_sell_order_detail_date:

ANY

ord

cust

pro

des

WHERE

guard1_r2: $\text{ord} \mapsto \text{cust} \in \text{custorders}$

guard2_r2: $\text{ord} \in \text{sorders}$

guard3_r2: $\text{pro} \in \text{products}$

guard4_r2: $\text{des} \in \text{bdate}$

THEN

act1_r2: $\text{prodorders} := \text{prodorders} \cup \text{ord} \mapsto \text{pro}$

act2_r2: $\text{sorder_detail_date}(\text{ord} \mapsto \text{pro}) := \text{des}$

END

delete_sell_order:

ANY

ord

WHERE

guard1_r2: $\text{ord} \in \text{sorders}$

THEN

act1_r2: $\text{sorders} := \text{sorders} \cup \text{ord}$

act2_r2: $\text{orders} := \text{orders} \cup \text{ord}$

act3_r2: $\text{prodorders} := \text{ord} \triangleleft \text{prodorders}$

act4_r2: $\text{custorders} := \text{ord} \triangleleft \text{custorders}$

act5_r2: $\text{sorder_detail_numitems} := (\text{ord} \times \text{products}) \triangleleft \text{sorder_detail_numitems}$

act6_r2: $\text{sorder_detail_date} := (\text{ord} \times \text{products}) \triangleleft \text{sorder_detail_date}$

END

create_purchase_order:

ANY

ord

cust

WHERE

guard1_r2: $\text{ord} \in \text{ORDERS} \setminus \text{orders}$

guard2_r2: cust \in customers
THEN
act1_r2: porders := porders \cup ord
act2_r2: orders := orders \cup ord
act3_r2: custorders := custorders \cup ord \mapsto cust
END

add_purchase_order_detail_numitems:
ANY
ord
cust
pro
n
WHERE
guard1_r2: aord \mapsto cust \in custorders
guard2_r2: ord \in porders
guard3_r2: pro \in products
guard4_r2: n $\in \mathbb{N}_1$
THEN
act1_r2: prodorders := prodorders \cup ord \mapsto pro
act2_r2: porder_detail_numitems(ord \mapsto pro) := n
END

add_purchase_order_detail_date:
ANY
ord
cust
pro
des
WHERE
guard1_r2: ord \mapsto date_cust \in custorders
guard2_r2: ord \in porders
guard3_r2: pro \in products
guard4_r2: des \in bdate
THEN
act1_r2: prodorders := prodorders \cup ord \mapsto pro

act2_r2: porder_detail_date(ord \mapsto pro) := des
END

delete_purchase_order:
ANY
ord
WHERE
guard1_r2: ord \in porders
THEN
act1_r2: porders := porders \cup ord
act2_r2: orders := orders \cup ord
act3_r2: prodorders := ord \triangleleft prodorders
act4_r2: custorders := ord \triangleleft custorders
act5_r2: porder_detail_numitems := (ord \times products) \triangleleft porder_detail_numitems
act6_r2: porder_detail_date := (ord \times products) \triangleleft porder_detail_date
END

A.4. Refinamiento 3 - Categorias (ref3_Categories)

MACHINE
ref3_Categories
REFINES
ref2_Orders
SEES
ctx_category

VARIABLES
categories
prodcat
categoryname

INVARIANTS
inv1_r3: categories \subseteq CATEGORIES
inv2_r3: prodcat \in products \rightarrow categories

inv3_r3: categoryname \in categories $\rightarrow \mathbb{P}(\text{bstring})$

EVENTS

INITIALISATION:

THEN

act1_r3: categories := \emptyset

act2_r3: prodcat := \emptyset

act3_r3: categoryname:= \emptyset

END

create_resale_product:

REFINES

create_resale_product ANY

cat

WHERE

guard1_r3: cat \in categories

THEN

act1_r3: prodcat := prodcat \cup pro \mapsto cat

END

create_resale_product_list:

REFINES

create_resale_product_list ANY

cat

WHERE

guard1_r3: cat \in categories

THEN

act1_r3: prodcat := prodcat \cup (lis \times cat)

END

create_manufacturing_product:

REFINES

create_manufacturing_product ANY

cat
WHERE
guard1_r3: cat ∈ categories
THEN
act1_r3: prodcat := prodcat ∪ pro ↦ cat
END

create_manufacturing_product_list:
REFINES
create_manufacturing_product_list ANY
cat
WHERE
guard1_r3: cat ∈ categories
THEN
act1_r3: prodcat := prodcat ∪ (lis × cat)
END

create_raw_material:
REFINES
create_raw_material ANY
cat
WHERE
guard1_r3: cat ∈ categories
THEN
act1_r3: prodcat := prodcat ∪ pro ↦ cat
END

create_raw_material_list:
REFINES
create_raw_material_list ANY
cat
WHERE
guard1_r3: cat ∈ categories
THEN

act1_r3: prodcat := prodcat \cup (lis \times cat)
END

add_category:
ANY
cat
name
WHERE
guard1_r3: cat \in CATEGORIES \ categories
guard2_r3: name \in $\mathbb{P}(\text{bstring})$
THEN
act1_r3: categories := categories \cup cat
act2_r3: categoryname := categoryname \cup cat \mapsto name
END

remove_category:
ANY
cat
WHERE
guard1_r3: cat \in CATEGORIES \ categories
THEN
act1_r3: categories := categories \ cat
act2_r3: categoryname := cat \triangleleft categoryname
END

A.5. Refinamiento 4 - Impuestos (ref4_Taxes)

MACHINE
ref4_Taxes
REFINES
ref3_Categories
SEES
ctx_taxes

VARIABLES

taxes
 taxprod
 taxper
 taxdesc
 categoryname

INVARIANTS

inv1_r4: taxes \subseteq TAXES
inv2_r4: taxprod \in products \leftrightarrow taxes
inv3_r4: taxper \in taxes \rightarrow percentage
inv4_r4: taxdesc \in taxes $\rightarrow \mathbb{P}(\text{bstring})$

EVENTS

INITIALISATION:

THEN

init1_r4: taxes := \emptyset
init2_r4: taxprod := \emptyset
init3_r4: taxper := \emptyset
init4_r4: taxdesc := \emptyset

END

create_tax:

ANY

tax

per

des

WHERE

guard1_r4: tax \in TAXES \ taxes
guard2_r4: per \in percentage
guard3_r4: des $\in \mathbb{P}(\text{bstring})$

THEN

act1_r4: taxes := taxes \cup tax
act2_r4: taxper := taxper \cup tax \mapsto per

act3_r4: taxdesc := taxdesc \cup tax \mapsto des
END

remove_tax:
ANY
remove_tax_tax
WHERE
guard1_r4: remove_tax_tax \in taxes
THEN
act1_r4: taxes := taxes \ tax
act2_r4: taxprod := (taxprod \triangleright tax)
act3_r4: taxper := taxper \ tax \mapsto taxper(tax)
act4_r4: taxdesc := taxdesc \ tax \mapsto taxdesc(tax)
END

update_tax:
ANY
tax
per
WHERE
guard1_r4: tax \in taxes
guard2_r4: per \in percentage
THEN
act1_r4: taxper := (taxper \ tax \mapsto taxper(tax)) \cup tax \mapsto per
END

update_tax_description:
ANY
tax
des
WHERE
guard1_r4: tax \in taxes
guard2_r4: des \in $\mathbb{P}(\text{bstring})$
THEN

act1_r4: taxdesc(tax) := des
END

add_tax_product:
ANY
per
tax
WHERE
guard1_r4: per ∈ products
guard2_r4: tax ∈ taxes
THEN
act1_r4: taxprod := taxprod ∪ per ↦ tax
END

add_tax plist:
ANY
lis
tax
WHERE
guard1_r4: lis ⊆ products
guard2_r4: tax ∈ taxes
THEN
act1_r4: taxprod := taxprod ∪ (lis × tax)
END

add_tax_pccategory:
ANY
cat
tax
WHERE
guard1_r4: cat ∈ categories
guard2_r4: tax ∈ taxes
THEN
act1_r4: taxprod := taxprod ∪ (prodcat[cat] × tax)

END

A.6. Refinamiento 5 - Reportes (ref5_Reports)

MACHINE

ref5_Reports

REFINES

ref4_Taxes

SEES

ctx_taxes

VARIABLES

p_sprice

p_pprice

INVARIANTS

inv1_r5: p_sprice $\in \mathbb{N}$

inv2_r5: p_pprice $\in \mathbb{N}$

EVENTS

INITIALISATION:

THEN

init1_r5: p_sprice := 0

init2_r5: p_pprice := 0

END

get_product_sell_price:

ANY

p

WHERE

guard1_r5: p $\in \text{dom}(\text{sprice})$

THEN

act1_r5: p_sprice := sprice(p)
END

get_product_taxed_sell_price:
ANY
pro
tax
WHERE
guard1_r5: pro ∈ dom(sprice)
guard2_r5: pro ↦ tax ∈ taxprod
THEN
act1_r5: p_sprice := sprice(pro) * (1 + taxper(tax) ÷ 100)
END

get_product_taxed_sell_price_in_order:
ANY
pro
tax
ord
WHERE
guard1_r5: pro ∈ dom(sprice)
guard2_r5: pro ↦ tax ∈ taxprod
guard3_r5: ord ↦ pro ∈ dom(sorder_detail_numitems)
THEN
act1_r5: p_sprice:= sprice(pro) * (1 + taxper(tax) ÷ 100) * sorder_detail_numitems(ord
↦ pro)
END

get_product_purchase_price:
ANY
pro
WHERE
guard1_r5: pro ∈ dom(pprice)
THEN

act1_r5: p_pprice := pprice(pro)
END

get_product_taxed_purchase_price:
ANY
pro
tax
WHERE
guard1_r5: pro ∈ dom(pprice)
guard2_r5: pro ↦ tax ∈ taxprod
THEN
act1_r5: p_pprice := pprice(pro) * (1 + taxper(tax) ÷ 100)
END

get_product_taxed_purchase_price_in_order:
ANY
pro
tax
ord
WHERE
guard1_r5: pro ∈ dom(pprice)
guard2_r5: pro ↦ tax ∈ taxprod
guard3_r5: ord ↦ pro ∈ dom(porder_detail_numitems)
THEN
act1_r5: p_pprice:= pprice(pro) * (1 + taxper(tax) ÷ 100) * porder_detail_numitems(ord
↦ pro)
END

Apéndice B

Resultado de las pruebas clase BSet

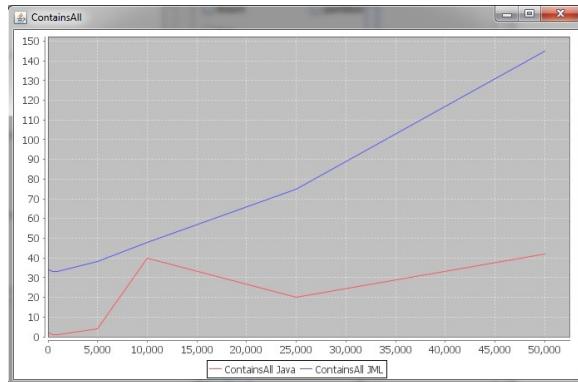


FIGURA B.1: Comparación del método *contains* de la clase BSet Java y JML.

| # Tran. | Tiempo Java | Tiempo JML |
|---------|-------------|------------|
| 100 | 2 | 34 |
| 500 | 1 | 33 |
| 1000 | 1 | 33 |
| 5000 | 4 | 38 |
| 10000 | 40 | 48 |
| 25000 | 20 | 75 |
| 50000 | 42 | 145 |

CUADRO B.1: Tiempos obtenidos para el método diferencia (*difference*) de BSet Java y JML.

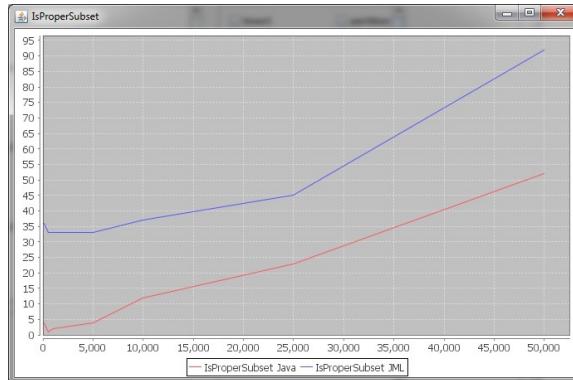


FIGURA B.2: Comparación del método *is proper subset* de la clase BSet Java y JML.

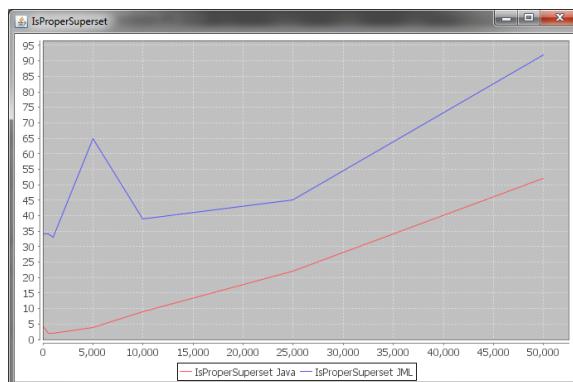


FIGURA B.3: Comparación del método *is proper superset* de la clase BSet Java y JML.

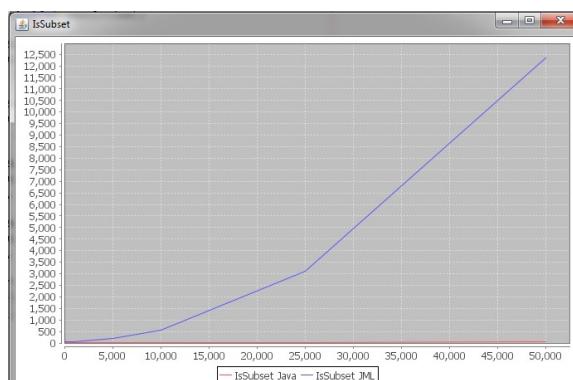
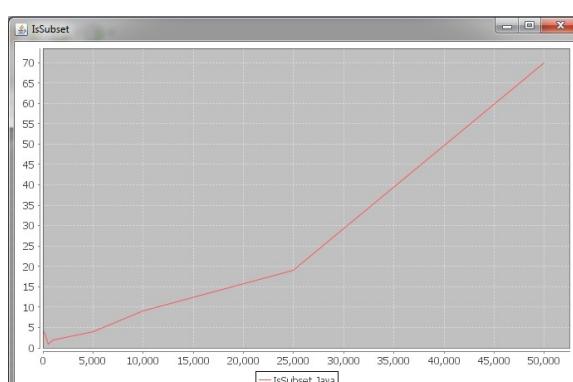


FIGURA B.4: Comparación del método *is subset* de la clase BSet Java y JML.



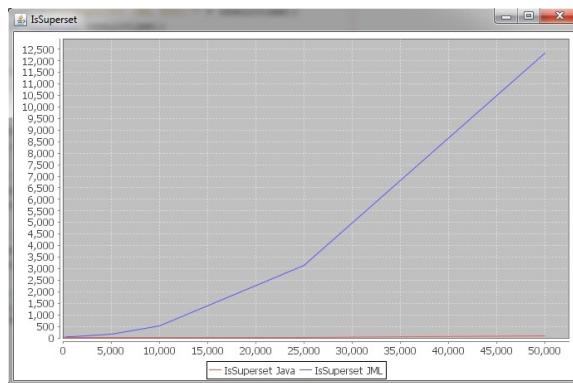
| # Tran. | Tiempo Java | Tiempo JML |
|---------|-------------|------------|
| 100 | 4 | 36 |
| 500 | 1 | 33 |
| 1000 | 2 | 33 |
| 5000 | 4 | 33 |
| 10000 | 12 | 37 |
| 25000 | 23 | 45 |
| 50000 | 52 | 92 |

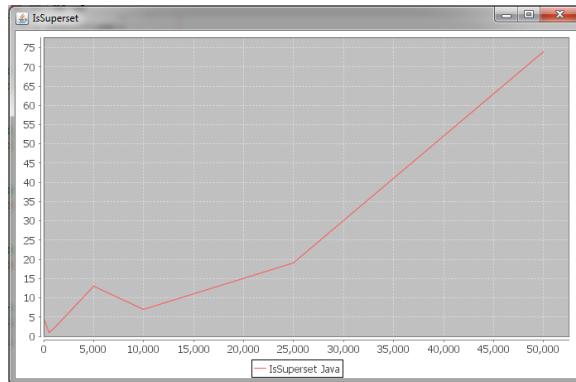
CUADRO B.2: Tiempos obtenidos para el método *is proper subset* de BSet Java y JML.

| # Tran. | Tiempo Java | Tiempo JML |
|---------|-------------|------------|
| 100 | 4 | 34 |
| 500 | 2 | 34 |
| 1000 | 2 | 33 |
| 5000 | 4 | 65 |
| 10000 | 9 | 39 |
| 25000 | 22 | 45 |
| 50000 | 52 | 92 |

CUADRO B.3: Tiempos obtenidos para el método *is proper superset* de BSet Java y JML.

| # Tran. | Tiempo Java | Tiempo JML |
|---------|-------------|------------|
| 100 | 4 | 36 |
| 500 | 1 | 66 |
| 1000 | 2 | 37 |
| 5000 | 4 | 181 |
| 10000 | 9 | 550 |
| 25000 | 19 | 3092 |
| 50000 | 70 | 12355 |

CUADRO B.4: Tiempos obtenidos para el método *is subset* de BSet Java y JML.FIGURA B.6: Comparación del método *is superset* de la clase BSet Java y JML.

FIGURA B.7: Tiempos para el método *is superset* del clase BSet Java.

| # Tran. | Tiempo Java | Tiempo JML |
|---------|-------------|------------|
| 100 | 4 | 34 |
| 500 | 1 | 60 |
| 1000 | 2 | 38 |
| 5000 | 13 | 156 |
| 10000 | 7 | 522 |
| 25000 | 19 | 3134 |
| 50000 | 74 | 12343 |

CUADRO B.5: Tiempos obtenidos para el método *is superset* de BSet Java y JML.

Apéndice C

Resultado de las pruebas clase BRelation

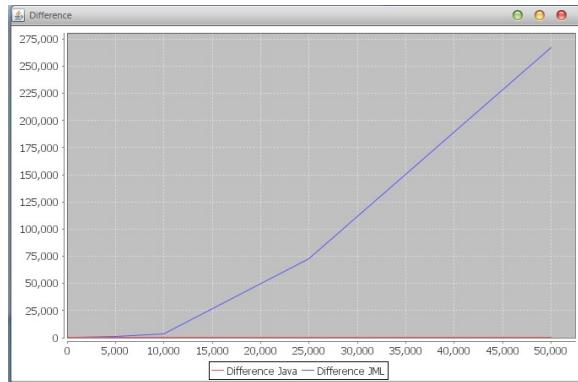


FIGURA C.1: Comparación del método *difference* de la clase BRelation Java y JML.

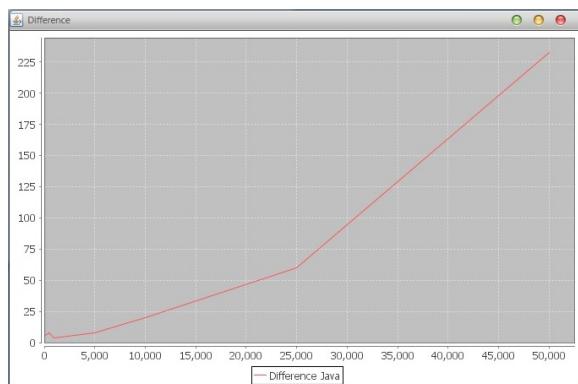
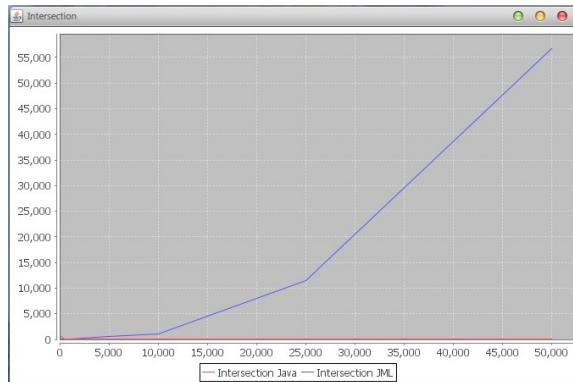
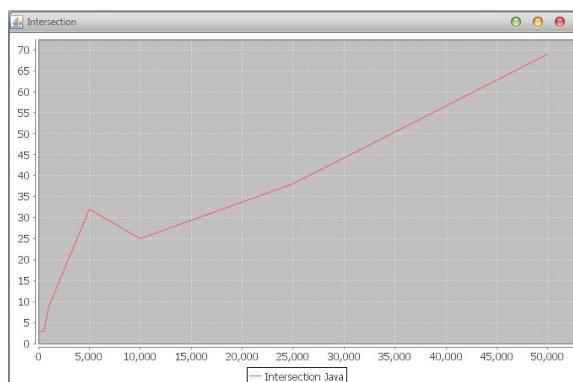
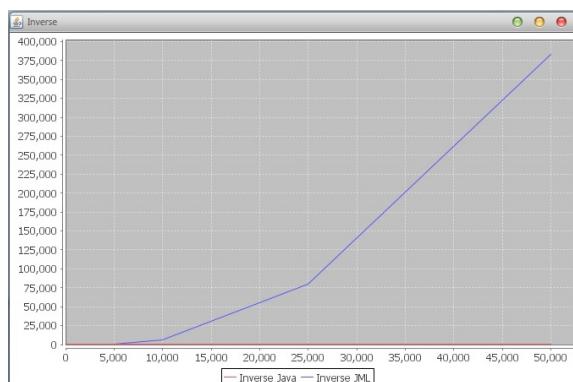
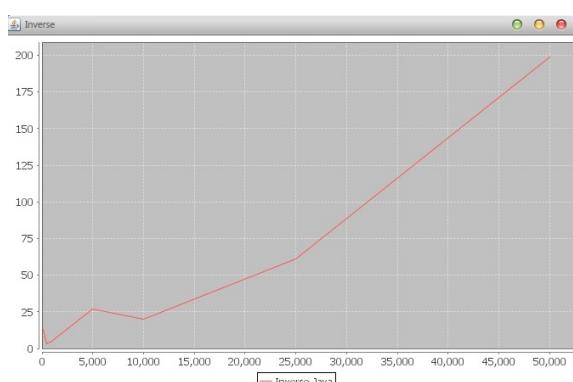
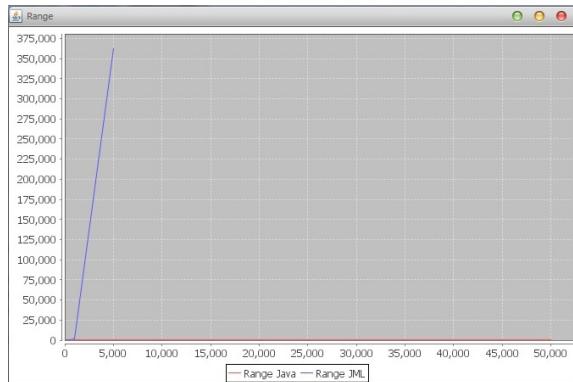
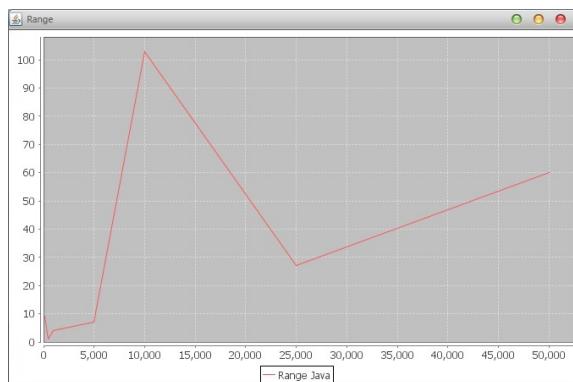
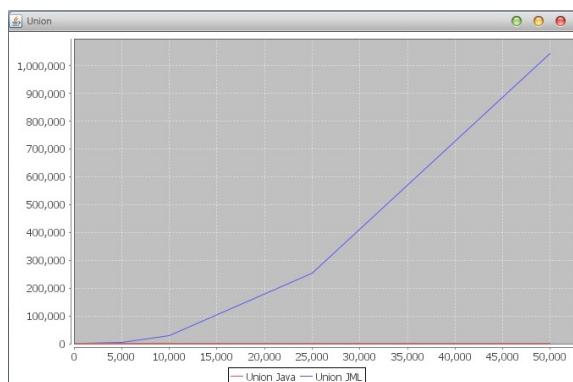
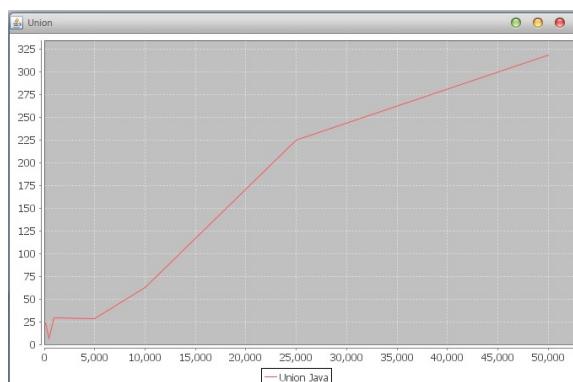


FIGURA C.2: Tiempos para el método *difference* del clase BRelation Java.

FIGURA C.3: Comparación del método *intersection* de la clase BRelation Java y JML.FIGURA C.4: Tiempos para el método *intersection* del clase BRelation Java.FIGURA C.5: Comparación del método *inverse* de la clase BRelation Java y JML.FIGURA C.6: Tiempos para el método *inverse* del clase BRelation Java.

FIGURA C.7: Comparación del método *range* de la clase BRelation Java y JML.FIGURA C.8: Tiempos para el método *range* del clase BRelation Java.FIGURA C.9: Comparación del método *union* de la clase BRelation Java y JML.FIGURA C.10: Tiempos para el método *union* del clase BRelation Java.

| # Tran. | Tiempo Java | Tiempo JML |
|---------|-------------|------------|
| 100 | 6ms | 76ms |
| 500 | 8ms | 83ms |
| 1000 | 4ms | 100ms |
| 5000 | 8ms | 807ms |
| 10000 | 20ms | 3657ms |
| 25000 | 60ms | 72553ms |
| 50000 | 233ms | 267398ms |

CUADRO C.1: Tiempos obtenidos para el método *difference* de BRelation Java y JML.

| # Tran. | Tiempo Java | Tiempo JML |
|---------|-------------|------------|
| 100 | 3ms | 48ms |
| 500 | 3ms | 21ms |
| 1000 | 9ms | 49ms |
| 5000 | 32ms | 547ms |
| 10000 | 25ms | 1081ms |
| 25000 | 38ms | 11437ms |
| 50000 | 69ms | 56771ms |

CUADRO C.2: Tiempos obtenidos para el método *intersection* de BRelation Java y JML.

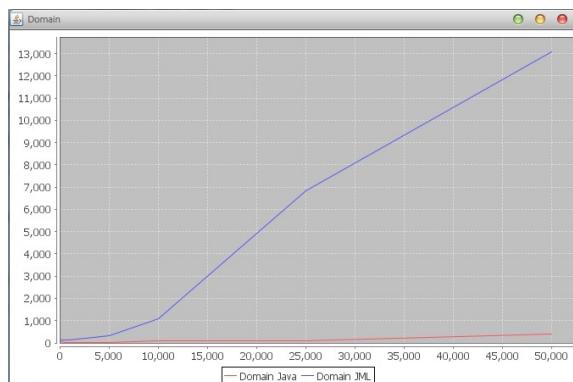
| # Tran. | Tiempo Java | Tiempo JML |
|---------|-------------|------------|
| 100 | 13ms | 134ms |
| 500 | 3ms | 82ms |
| 1000 | 5ms | 169ms |
| 5000 | 27ms | 1033ms |
| 10000 | 20ms | 6091ms |
| 25000 | 61ms | 79330ms |
| 50000 | 199ms | 383610ms |

CUADRO C.3: Tiempos obtenidos para el método *inverse* de BRelation Java y JML.

| # Tran. | Tiempo Java | Tiempo JML |
|---------|-------------|------------|
| 100 | 9ms | 37ms |
| 500 | 1ms | 245ms |
| 1000 | 4ms | 1685ms |
| 5000 | 7ms | 362555ms |
| 10000 | 103ms | |
| 25000 | 27ms | |
| 50000 | 60ms | |

CUADRO C.4: Tiempos obtenidos para el método *range* de BRelation Java y JML.

| # Tran. | Tiempo Java | Tiempo JML |
|---------|-------------|------------|
| 100 | 24ms | 149ms |
| 500 | 6ms | 272ms |
| 1000 | 29ms | 357ms |
| 5000 | 28ms | 4222ms |
| 10000 | 63ms | 28560ms |
| 25000 | 225ms | 252696ms |
| 50000 | 319ms | 1044858ms |

CUADRO C.5: Tiempos obtenidos para el método *union* de BRelation Java y JML.FIGURA C.11: Comparación del método *domain* de la clase BRelation Java y JML.

| # Tran. | Tiempo Java | Tiempo JML |
|---------|-------------|------------|
| 100 | 64ms | 115ms |
| 500 | 6ms | 117ms |
| 1000 | 7ms | 118ms |
| 5000 | 18ms | 337ms |
| 10000 | 86ms | 1094ms |
| 25000 | 102ms | 6822ms |
| 50000 | 383ms | 13097ms |

CUADRO C.6: Tiempos obtenidos para el método *domain* de BRelation Java y JML.

Bibliografía

- [1] J. R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, New York, NY, USA, 1996.
- [2] J. R. Abrial. *Modeling in Event-B: System and Software Design*. Cambridge University Press, New York, NY, USA, 2010.
- [3] S. Black, P. P. Boca, J. P. Bowen, J. Gorman, and M. Hinchey. Formal versus agile: Survival of the fittest. *IEEE*, 42(9):37 – 45, 2009.
- [4] Cees-Bart Breunesse, Néstor Cataño, Marieke Huisman, and Bart Jacobs. Formal methods for smart cards: An experience report. *Science of Computer Programming*, 55(1-3):53–80, March 2005.
- [5] L. Burdy, Y. Cheon, D. Cok, M. Ernst, J. Kiniry, G. T. Leavens, K. R. M. Leino, and E. Poll. An overview of JML tools and applications. *International Journal on STTT*, 7(3):212–232, 2005.
- [6] M. J. Butler, C. B. Jones, A. Romanovsky, and E. Troubitsyna, editors. *Rigorous Development of Complex Fault-Tolerant Systems [FP6 IST-511599 RODIN project]*, LNCS. Springer, 2006.
- [7] N. Cataño, T. Wahls, C. Rueda, Víctor Rivera, and Danni Yu. Translating B machines to JML specifications. In *27th ACM Symposium on Applied Computing, Software Verification and Testing track (SAC-SVT)*, Trento, Italy, March 26-30 2012 (to appear).
- [8] Néstor Cataño, Tim Wahls, and Víctor Rivera. Translating event-b to jml-specified java programs. In *Proceedings of 29th ACM Symposium on Applied Computing, Software Verification and Testing track*, Gyeongju, Korea, 2014. ACM.
- [9] Néstor Cataño, Tim Wahls, Víctor Rivera, and Camilo Rueda. Translating Event-B machines to JML specifications. Submitted to TOSEM, 2012.
- [10] A. Hall and R. Chapman. Correctness by construction: Developing a commercial secure system. *IEEE*, 19(1):18–25, Jan/Feb 2002.

- [11] G.T. Leavens, A.L. Baker, and C. Ruby. Preliminary design of JML: A behavioral interface specification language for Java. *ACM SIGSOFT*, 31(3):1–38, 2006.
- [12] G.T. Leavens, E. Poll, C. Clifton, Y. Cheon, C. Ruby, D. Cok, P. Müller, J. Kiniry, and P. Chalin. JML reference manual. http://www.eecs.ucf.edu/~leavens/-JML/jmlrefman/jmlrefman_toc.html, 2012.
- [13] Rigorous Development of Complex Fault-Tolerant Systems. Accessed September 2012. <http://sourceforge.net/projects/rodin-b-sharp/>, (2011).