Junio 10, 2010

Qué hace que un software sea seguro

Categoría: Sin categoría — dccuchile - 5:44 pm

Por Tomás Barros, profesor jornada parcial del Depto. de Ciencias de la Computación, FCFM, U. de Chile; director y gerente de NIC Labs.

"Si ellos construyeran caminos, puentes, autos, aviones y barcos como este software (Windows), la raza humana estaría condenada" Richard Sharpe.

Entre 1985 y 1987, una máquina de radioterapia, Terac-25, causó la muerte de 6 pacientes debido a una sobre dosis de radiación. En 1996, un cohete Ariana 5 de la agencia Europea explotó pocos segundos después de despegar. En ambos casos fueron errores de software. **¿Por qué los aviones de Airbus, llenos de software, no se caen regularmente como Windows? ¿Por qué los metros en París no chocan regularmente?** Una de las principales razones es que estos "software seguros críticos" están desarrollados con métodos formales; la base matemática que los provee de formas precisas para definir nociones como consistencia y completitud y, más importante aún, para especificar y verificar la 'correctitud' de la implementación de forma no ambigua, sin necesidad de ejecutar el software para determinar su comportamiento.

Robert Floyd abrió el área de métodos formales en 1967 en su artículo "Assigning Meanings to Programs". La idea básica propuesta por Floyd era agregar "etiquetas" con proposiciones lógicas en distintas partes del código, que determinarían el efecto del programa basado en una definición semántica formal del lenguaje de programación. Esta idea fue recogida por Hoare quien, en 1969, desarrolló su cálculo de pre y pos condiciones. Hoy en día Java, por ejemplo, soporta incluir estas etiquetas (conocidas como assertions) en el código y hace algunas verificaciones, aunque aún muy básicas.

Existen varias técnicas de métodos formales. Por ejemplo, en técnicas deductivas el programa es expresado como un teorema de la forma: especificación del programa => propiedad deseada; donde se busca probar ese teorema (conocido como theorem proving). Otra técnica es model-checking, donde en un modelo matemático se especifica el programa y exploran todos los posibles comportamientos de éste, junto con verificar que las propiedades deseadas se mantienen (por nombrar una, que un microondas no esté abierto y funcionando a la vez).

Para sistemas distribuidos una técnica formal es usar un álgebra de procesos. Una estructura algebraica consiste en un conjunto de elementos y sus operaciones. Por ejemplo, en algebra elemental el conjunto son los números y las operaciones la suma, resta, multiplicación y división. Además, la estructura algebraica puede acompañarse de relaciones como equivalencia (igualdad) o mayor/menor, también en el caso del álgebra elemental. En esta ocasión quiero introducir algunos elementos de un álgebra desarrollada por Robin Milner llamada CCS (Calculus of Communication Systems).

En CCS, el conjunto está definido por acciones, que son las que puede ejecutar un programa (enviar un mensaje, imprimir, leer una base de datos, etc.). Y las operaciones son:

- acción: a.P, el proceso ejecuta a y sigue operando como P
- elección: P1 + P2 , se ejecuta P1 ó P2
- paralelismo: P1 | P2 , se ejecuta P1 y P2 en paralelo
- restricción: P \ a, se ejecuta P sin la accion a

Por ejemplo, un filósofo piensa, come y luego vuelve a pensar; en CSS el filosofo estaría definido como F = piensa.come.F. El operador de restricción se utiliza para, principalmente, sincronizar acciones. Así, si queremos, por ejemplo, que 2 filósofos independientes (F1 y F2) coman exactamente al mismo tiempo, anotaríamos (F1 | F2) \ comer. Todas las operaciones tienen por supuesto una defínción formal (semántica), pero que por espacio no incluiré aquí.

Teniendo los sistemas descritos en CCS, el razonamiento de su comportamiento (y por ende la comprobación de que hace exactamente lo que debería hacer, nada más, nada menos) se realiza usando relaciones de equivalencia y orden. La relación de equivalencia más usada se llama bisimulación y está definida sobre la idea que para un observador externo es totalmente indistinguible un proceso de otro. Esto permite, por ejemplo, comprobar que un sistema se comporta de igual forma que otro conocido que está correcto, o permite reducir un sistema a uno más simple pero equivalente (algo similar a reducir una expresión en matemáticas). También hay relaciones de orden (análogas a mayor/menor) que permiten comprobar que un sistema hace al menos lo que hace otro conocido, o que a lo más hace lo mismo que otro conocido.

Comprender bien y saber usar las algebras de procesos para definir sistemas y comprobar su comportamiento requiere una formación formal teórica y bastante práctica. **El uso de métodos formales en la industria crítica, desde su concepción hasta la implementación del sistema, además de largas pruebas de testing, calidad y seguridad, hacen sin duda que los software desarrollados posean mayores garantías. Una práctica que marca la diferencia.**

Archivos

- Qué hace que un software sea seguro
- Monos al teclado II: Kolmogorov y la entropía del Universo
- La lógica de Twitter: ¿se puede razonar en poco espacio?
- Twitter: el mundo de las pildoritas de pensamiento
- Sistemas criptográficos RSA: seguros mientras no se demuestre lo contrario
- Monos al teclado, la ley del menor esfuerzo y los buscadores Web
- El futuro de la Web: ¿nuestro futuro?
- China ¿en guerra contra Internet?
- Un computador (digital) por niño
- El retraso en el cambio de hora: ¿acierto o desacierto?

Otros Blogueros

- Belisario Iturra Peralta (Noticias)
- Claudio Uson (Tecnología)
- Juan Guillermo Tejeda (Noticias)
- Tomás Flores Economista (Invertia)
- Ximena Torres Cautivo (Libros)