La técnica de prueba «Mutación Evolutiva»

Inmaculada Medina Bulo

Universidad de Cádiz

Grupo UCASE de Ingeniería del Software
Departamento de Ingeniería Informática
Universidad de Cádiz

Universidad Complutense de Madrid, Junio 2013

Índice

1 Introducción y motivación

2 Prueba de mutaciones

3 WS-BPEL

4 Algoritmos genéticos

5 Mutación evolutiva

6 Experimentos

7 Conclusiones y trabajo futuro

Sección 1 | Introducción y motivación

Introducción

Contexto

Línea de investigación

Reducción del coste computacional de la aplicación de la prueba de mutaciones a composiciones de servicios web WS-BPEL

Técnicas

- Prueba de mutaciones
- Algoritmos genéticos

Contribuciones principales

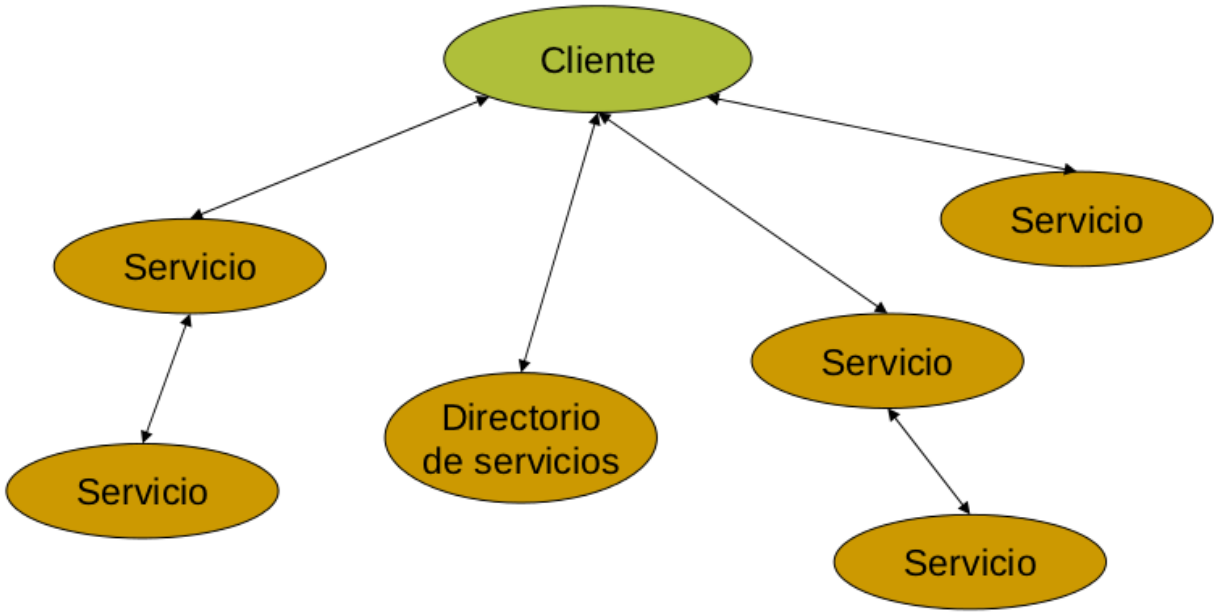
- «Mutación evolutiva»: una técnica de reducción del coste computacional de la prueba de mutaciones
- GAmera: un sistema para la aplicación de la prueba de mutaciones a composiciones de SW que implementa dicha técnica

Motivación

Composiciones de servicios web WS-BPEL

¿Por qué emplear composiciones de servicios WS-BPEL?

- Los servicios web facilitan un desarrollo rápido de aplicaciones
  - Menor coste de desarrollo
  - Mayor flexibilidad para crear aplicaciones distribuidas
- Estas aplicaciones pueden organizarse como composiciones de servicios web, donde WS-BPEL es el principal lenguaje disponible



Motivación

Principales razones para usar la prueba de mutaciones

- Se ha aplicado con éxito a programas escritos en diferentes lenguajes: Fortran, Java, C, SQL, etc.
- Se han publicado pocos artículos en este campo, la mayoría relacionados con la generación de casos de prueba
- Se carecía de un sistema de generación de mutantes para WS-BPEL

Es importante avanzar en el estudio de técnicas de prueba apropiadas para composiciones de servicios web WS-BPEL

Sección 2 | Prueba de mutaciones

Prueba de mutaciones

Definición

Prueba de mutaciones

- Es una técnica de prueba de software de caja blanca
- Consiste en inyectar fallos por medio de mutaciones en el código
- Las mutaciones representan cambios sintácticos en el programa original y son generadas mediante operadores de mutación
- El programa resultante recibe el nombre de mutante

Ejemplo

Original

```
if (a > 5000) ...
```

Mutante

```
if (a < 5000) ...
```

Prueba de mutaciones

Funcionamiento

Elementos necesarios

1. Definición de los operadores de mutación (dependen del lenguaje)
2. Código del programa original y conjunto de casos de prueba

Pasos

1. Análisis de los operadores de mutación a aplicar
2. Generación de los mutantes
3. Ejecución de los mutantes:
   Cada mutante se ejecuta sobre cada caso de prueba
4. Comparación de resultados:
   Se comparan las salidas del mutante y del programa original
   - Si las salidas son diferentes, el mutante está muerto
   - Si son iguales para todos los casos de prueba, el mutante sigue vivo

Prueba de mutaciones

Clasificación de los mutantes

Tipos de mutantes

- **Muerto** Algún caso de prueba permite distinguir al programa original del mutante
  - **Débil** Lo matan todos los casos de prueba
  - **Resistente** Lo mata un único caso de prueba
  - **Resistente difícil de matar** Lo mata un único caso de prueba que sólo lo mata él
- **Vivo** Ningún caso de prueba pemite distinguir al mutante del programa original
  - **Equivalente** El mutante siempre produce idénticos resultados a los del programa original
  - **Potencialmente equivalente** El conjunto de casos de prueba no es suficiente para detectar diferencias, aunque no se descarta su existencia
- **Erróneo** El mutante no se puede ejecutar

Prueba de mutaciones

Objetivos

Utilidad

- Detectar posibles defectos de un software mediante una prueba exhaustiva de sus componentes
- Medir la calidad de un conjunto de casos de prueba mediante la puntuación de mutación (*mutation score*), que es el porcentaje de mutantes muertos:

$$\text{Puntuación de mutación} = \frac{\text{Mut. muertos}}{\text{Mut. totales} - \text{Mut. equivalentes}} \cdot 100\,\%$$

- Servir de guía para generar nuevos casos de prueba que maten a los potencialmente equivalentes e incrementar así la puntuación

Cuando la puntuación es del 100 %, el conjunto de casos de prueba es capaz de detectar todos los fallos introducidos por los mutantes

Prueba de mutaciones

Matriz de ejecución

Definición

Es una matriz donde las filas representan mutantes, las columnas casos de prueba, y cada elemento representa el resultado de comparar la salida del mutante frente al programa original para el caso de prueba dado

- Si ambas salidas coinciden, su valor es 0
- Si difieren, su valor es 1
- Si la ejecución falla (mutante erróneo), su valor es 2

	Matriz de ejecución				
	T1	T2	T3	T4	
M1	0	1	1	1	Muerto
M2	0	0	0	1	Muerto resistente
M3	2	2	2	2	Erróneo
M4	0	0	0	0	Vivo
M5	1	1	1	1	Muerto débil

Ejemplo

Operadores de mutación

- Cambiar operador aritmético por otro ($+$, $-$, $*$, /)
- Cambiar una variable por otra

Programa original

```
int suma (int a, int b) { return a + b; }
```

Mutantes generados

M1: { return a - b; }
M2: { return a * b; }
M3: { return a / b; }
M4: { return b + b; }
M5: { return a + a; }

	Datos de prueba			
	(0,0)	(-1,0)	(1,0)	Tipo mutante
P	0	-1	1	–
M1	0	-1	1	Pot. equiv.
M2	0	0	0	Muerto
M3	error	error	error	Erróneo
M4	0	0	0	Muerto
M5	0	-2	2	Muerto

Ejemplo

Operadores de mutación

- Cambiar operador aritmético por otro ($+$, $-$, $*$, /)
- Cambiar una variable por otra

Programa original

```
int suma (int a, int b) { return a + b; }
```

Mutantes generados

M1: { return a - b; }
M2: { return a * b; }
M3: { return a / b; }
M4: { return b + b; }
M5: { return a + a; }

	Datos de prueba			
	(0,0)	(-1,0)	(1,0)	Tipo mutante
P	0	-1	1	–
M1	0	-1	1	Pot. equiv.
M2	0	0	0	Muerto
M3	error	error	error	Erróneo
M4	0	0	0	Muerto
M5	0	-2	2	Muerto

Ejemplo

Operadores de mutación

- Cambiar operador aritmético por otro ($+$, $-$, $*$, $/$)
- Cambiar una variable por otra

Programa original

```
int suma (int a, int b) { return a + b; }
```

Mutantes generados

M1: { return a - b; }
M2: { return a * b; }
M3: { return a / b; }
M4: { return b + b; }
M5: { return a + a; }

	Datos de prueba			
	(0,0)	(-1,0)	(1,0)	Tipo mutante
P	0	-1	1	–
M1	0	-1	1	Pot. equiv.
M2	0	0	0	Muerto
M3	error	error	error	Erróneo
M4	0	0	0	Muerto
M5	0	-2	2	Muerto

Ejemplo

Operadores de mutación

- Cambiar operador aritmético por otro ($+$, $-$, $*$, /)
- Cambiar una variable por otra

Programa original

```
int suma (int a, int b) { return a + b; }
```

Mutantes generados

```
M1: { return a - b; }
M2: { return a * b; }
M3: { return a / b; }
M4: { return b + b; }
M5: { return a + a; }
```

	Matriz de ejecución		
	(0,0)	(-1,0)	(1,0)
M1	0	0	0
M2	0	1	1
M3	2	2	2
M4	0	1	1
M5	0	1	1

Prueba de mutaciones

Operadores de mutación

Definición de los operadores de mutación

- Dependen del lenguaje de programación: C, Java, Fortran, SQL, WS-BPEL

Tipo de operadores de mutación

- Existen operadores que modelan errores típicos cometidos por los programadores
- Otros operadores intentan cumplir algunos criterios de cobertura de pruebas
  - Ejecutar todas las sentencias
  - Ejecutar todas las condiciones

Prueba de mutaciones

Órdenes de las mutaciones

Mutación de primer orden

- Cuando se aplica un único operador de mutación al programa original se habla de mutación de primer orden

Mutación de orden superior

- Es posible aplicar operadores de mutación en cadena, de manera que al primer mutante se puede aplicar un segundo operador de mutación, al mutante resultante otro operador, y así sucesivamente. En tal caso se habla de mutaciones de orden superior
- El orden de la mutación indica el número de operadores aplicados
- Esta técnica de mutantes de orden superior se ha encontrado efectiva como técnica de reducción de mutantes debido a que un mutante de orden superior engloba los fallos de los mutantes de primer orden que producen sus operadores de mutación

Dificultades

- Detección de mutantes equivalentes. ¿Es el conjunto de casos de prueba suficiente para diferenciarlos?
- Elevado número de mutantes, que implica un elevado coste computacional
  - Técnicas de reducción del tiempo de ejecución
  - Técnicas de reducción del número de mutantes

Prueba de mutaciones

Reducción del coste de ejecución

Mutación débil (*Weak Mutation*)

Compara el estado interno del mutante y el programa original inmediatamente después de la porción mutada

Mutación firme (*Firm Mutation*)

Compara el estado intermedio después de la ejecución así como la salida final

Técnicas de optimización del tiempo de ejecución

Ejecutan al mutante lo más rápidamente posible, como por ejemplo empleando compiladores específicos, los metamutantes y los generadores a partir de código objeto

Soporte de plataformas avanzadas

Distribución del coste entre varios equipos, supercomputadores, etc.

Prueba de mutaciones

Reducción del número de mutantes

Mutación aleatoria (*Mutant Sampling*)

Sólo se ejecuta un subconjunto de mutantes seleccionados aleatoriamente

Mutación selectiva (*Selective Mutation*)

Sólo se aplica un subconjunto de los operadores de mutación definidos para el lenguaje

Mutación de agrupamiento (*Mutant Clustering*)

Agrupa a los mutantes en función del conjunto de casos de prueba que lo mata, y de cada grupo selecciona un pequeño número de mutantes

Mutación evolutiva (*Evolutionary Mutation Testing*)

Generar de forma seleccionada un subconjunto de mutantes mediante un algoritmo evolutivo

Sección 3 | WS-BPEL

El lenguaje WS-BPEL

Orquestación de servicios

Orquestación (composición de servicios)

- La evolución del software hacia las arquitecturas orientadas a servicios ha provocado la definición de un lenguaje que facilite la orquestación (composición) de servicios web $\Longrightarrow$ El estándar OASIS WS-BPEL 2.0
- La orquestación de servicios web se basa en un modelo centralizado en el cuál las interacciones no se realizan directamente entre los servicios web, sino que existe una entidad encargada de definir la lógica de interacción
- El motor de ejecución BPEL es el encargado de dirigir la orquestación, y por tanto, el que controla la lógica de interacción entre los distintos servicios web

El lenguaje WS-BPEL

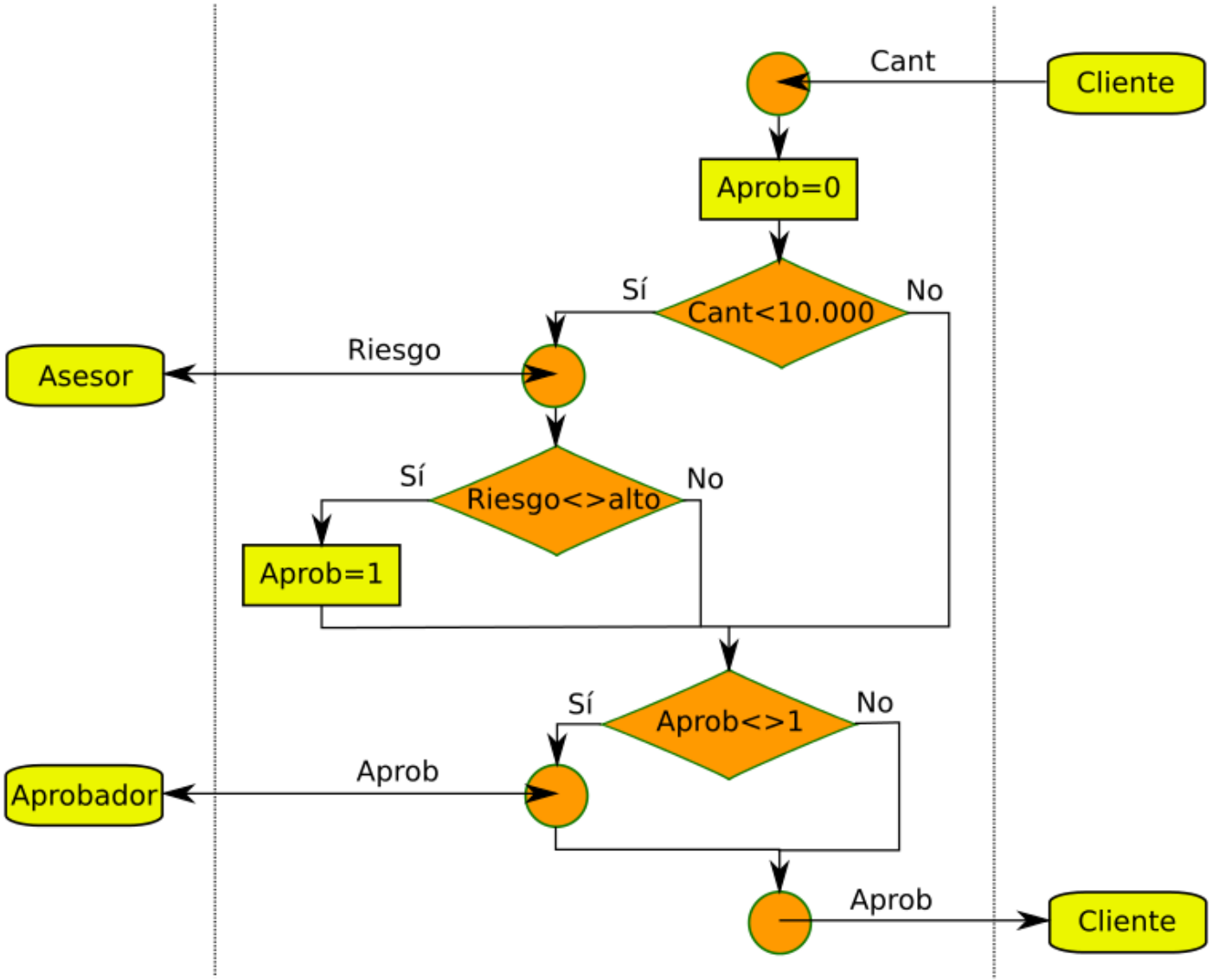
Estructura fichero WS-BPEL

¿Cómo es un fichero WS-BPEL?

- WS-BPEL es un lenguaje de programación tipo XML.
- Comienza importando los ficheros XSD donde se definen los tipos de datos y los ficheros WSDL con la descripción de los servicios con los que opera
- A continuación define los denominados partnerlinks que son las distintas llamadas que se realizarán a los servicios definidos anteriormente
- Finalmente se definen las variables a emplear y se indica la lógica de la composición

El lenguaje WS-BPEL

Composición de servicios Préstamo



El lenguaje WS-BPEL

Estructura del Préstamo

```
<process
   name="LoanApprovalProcess"
   targetNamespace="http://enterprise.netbeans.org/bpel/N6_ServicioPrestamo/LoanApprovalProcess"
   xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:tns="http://enterprise.netbeans.org/bpel/N6_ServicioPrestamo/LoanApprovalProcess"
   xmlns:ns1="http://j2ee.netbeans.org/wsdl/ApprovalService"
   xmlns:ns2="http://j2ee.netbeans.org/wsdl/AssessorService"
   xmlns:ns0="http://xml.netbeans.org/schema/Loans"
   xmlns:ns3="http://j2ee.netbeans.org/wsdl/LoanService">
  <import namespace="http://j2ee.netbeans.org/wsdl/ApprovalService"
          location="ApprovalService.wsdl"
          importType="http://schemas.xmlsoap.org/wsdl/"/>
  <import namespace="http://j2ee.netbeans.org/wsdl/AssessorService"
          location="AssessorService.wsdl"
          importType="http://schemas.xmlsoap.org/wsdl/"/>
  <import namespace="http://j2ee.netbeans.org/wsdl/LoanService"
          location="LoanService.wsdl"
          importType="http://schemas.xmlsoap.org/wsdl/"/>
```

El lenguaje WS-BPEL

Estructura del Préstamo

```
<partnerLinks>
  <partnerLink name="assessor"
               partnerLinkType="ns2:AssessorService1"
               partnerRole="AssessorServicePortTypeRole"/>
  <partnerLink name="approver"
               partnerLinkType="ns1:ApprovalService1"
               partnerRole="ApprovalServicePortTypeRole"/>
  <partnerLink name="client" partnerLinkType="ns3:LoanService1"
               myRole="LoanServicePortTypeRole"/>
</partnerLinks>
<variables>
  <variable name="processOutput" messageType="ns3:LoanServiceOperationReply"/>
  <variable name="processInput" messageType="ns3:LoanServiceOperationRequest"/>
  <variable name="assessorOutput" messageType="ns2:AssessorServiceOperationReply"/>
  <variable name="assessorInput" messageType="ns2:AssessorServiceOperationRequest" />
  <variable name="approverOutput" messageType="ns1:ApprovalServiceOperationReply"/>
  <variable name="approverInput" messageType="ns1:ApprovalServiceOperationRequest"/>
</variables>
```

El lenguaje WS-BPEL

Estructura del Préstamo

```
<sequence name="Main">
  <receive name="Receive1" createInstance="yes" partnerLink="client"
           operation="grantLoan" portType="ns3:LoanServicePortType"
           variable="processInput"/>
  <assign name="DefaultValues">
    <copy>
      <from>
        <literal><ns0:AssessorRequest><ns0:amount>0.0</ns0:amount></ns0:AssessorRequest></literal>
      </from>
      <to part="input" variable="assessorInput"/>
    </copy>
    <copy>
      <from>
        <literal><ns0:ApprovalResponse><ns0:accept>false</ns0:accept></ns0:ApprovalResponse></literal>
      </from>
      <to part="output" variable="processOutput"/>
    </copy>
    <copy>
      <from>
        <literal><ns0:ApprovalRequest><ns0:amount>0.0</ns0:amount></ns0:ApprovalRequest></literal>
      </from>
      <to part="input" variable="approverInput"/>
    </copy>
  </assign>
```

El lenguaje WS-BPEL

Estructura del Préstamo

```
<if name="If1">
  <condition> ( number(string($processInput.input/ns0:amount)) &lt;= 10000 ) </condition>
  <sequence name="SmallAmount">
    <assign name="copyLoanInfoToAssessorInput">
      <copy>
        <from>$processInput.input/ns0:amount</from>
        <to>$assessorInput.input/ns0:amount</to>
      </copy>
    </assign>
    <invoke name="queryAssessor" partnerLink="assessor"
            operation="assessLoan" portType="ns2:AssessorServicePortType"
            inputVariable="assessorInput"
            outputVariable="assessorOutput"/>
    <if name="If2">
      <condition> ( string($assessorOutput.output/ns0:risk) = 'high' ) </condition>
      <sequence name="SmallAmountHighRisk">
        <assign name="copyLoanInfoToApproverInput">
          <copy>
            <from>$processInput.input/ns0:amount</from>
            <to>$approverInput.input/ns0:amount</to>
          </copy>
        </assign>
        <invoke name="queryApprover" partnerLink="approver"
                operation="approveLoan"
                portType="ns1:ApprovalServicePortType"
                inputVariable="approverInput"
                outputVariable="approverOutput"/>
        <assign name="copyApproval">
          <copy>
            <from>$approverOutput.output/ns0:accept</from>
            <to>$processOutput.output/ns0:accept</to>
          </copy>
```

El lenguaje WS-BPEL

Estructura del Préstamo

```
      <else>
        <sequence name="SmallAmountLowRisk">
          <assign name="approveLoan">
            <copy>
              <from>true()</from>
              <to>$processOutput.output/ns0:accept</to>
            </copy>
          </assign>
        </sequence>
      </else>
    </if>
  </sequence>
  <else>
    <sequence name="LargeAmount">
      <assign name="copyLoanInfoToApproverInput2">
        <copy>
          <from>$processInput.input/ns0:amount</from>
          <to>$approverInput.input/ns0:amount</to>
        </copy>
      </assign>
```

El lenguaje WS-BPEL

Estructura del Préstamo

```
    </if>
    <reply name="Reply1" partnerLink="client" operation="grantLoan"
           portType="ns3:LoanServicePortType" variable="processOutput"/>
  </sequence>
</process>
```

El lenguaje WS-BPEL

Operadores de mutación

- Hemos definido un conjunto de 26 operadores que modelan errores típicos que pueden cometer los programadores de WS-BPEL
- Estos operadores se clasifican en cuatro categorías:
  1. Operadores de sustitución de identificadores (I)
  2. Operadores de expresión (E)
  3. Operadores de actividad (A)
  4. Operadores de excepciones y eventos (X)

Estructura del nombre

X	Y Z

X: la categoría

YZ: el operador

El lenguaje WS-BPEL

Operadores de mutación

Operadores de sustitución de identificadores

OPER.	DESCRIPCIÓN
ISV	Sustituye un identificador de variable por otro del mismo tipo

Operadores de expresión

OPER.	DESCRIPCIÓN
EAA	Sustituye un operador aritmético por otro del mismo tipo
EEU	Elimina el operador menos unario de una expresión
ERR	Sustituye un operador relacional por otro del mismo tipo
ELL	Sustituye un operador lógico por otro del mismo tipo
☆ ECC	Sustitye un operador de camino por otro del mismo tipo
ECN	Modifica una constante numérica
☆ EMD	Modifica una duración de fecha
☆ EMF	Modifica una expresión de tiempo

El lenguaje WS-BPEL

Operadores de mutación

Operadores de actividad relacionados con la concurrencia

OPER.	DESCRIPCIÓN
☆ ACI	Cambia el atributo `createInstance` de una actividad de mensaje entrante a *no*
☆ AFP	Sustituye una actividad secuencial `forEach` por una paralela
☆ ASF	Sustituye una actividad `sequence` por una actividad `flow`
☆ AIS	Cambia el atributo `isolated` de un `scope` a *no*

Operadores de actividad no concurrentes

OPER.	DESCRIPCIÓN
AIE	Elimina un elemento `elseif` o `else` de una actividad `if`
AWR	Sustituye una actividad `while` por un `repeatUntil` y viceversa
☆ AJC	Elimina el atributo `joinCondition` de una actividad
☆ ASI	Intercambia el orden de dos actividades `sequence` hijas
☆ APM	Elimina un elemento `onMessage` de una actividad `pick`
☆ APA	Elimina el elemento `onAlarm` de una actividad `pick` o de un manejador de eventos

El lenguaje WS-BPEL

Operadores de mutación

Operadores de excepción y eventos

OPER.	DESCRIPCIÓN
☆ XMF	Elimina un elemento `catch` o el elemento `catchAll` de un manejador de fallos
☆ XRF	Elimina el atributo `faultName` de una actividad `reply`
☆ XMC	Elimina la definición de un manejador de compensación
☆ XMT	Elimina la definición de un manejador de terminación
XTF	Sustituye la activación de un manejador de fallos por una actividad `throw`
☆ XER	Elimina una actividad `rethrow`
☆ XEE	Elimina un elemento `onEvent` de un manejador de eventos

Operadores de mutación

¿Quieres saber más?

- Mutation operators for WS-BPEL 2.0 (ICSSEA 2008), Estero, A. y otros
- Operadores de mutación para WS-BPEL 2.0 (PRIS 2008), Estero, A. y otros
- Software Unit Test Coverage and Adequacy, Zhu y otros

Sección 4 | Algoritmos genéticos

Algoritmos Genéticos

Características

- Son técnicas de búsquedas probabilísticas basadas en conceptos de la teoría de la evolución y la genética natural
- Se genera un conjunto de posibles soluciones población al problema de forma aleatoria. Cada solución individual recibe el nombre de individuo
- Cada individuo tiene asociado un valor de su calidad (aptitud)
- En cada iteración (generación) se realizan procesos de selección y de reproducción (cruce y mutación), que producen una mejora de la aptitud de la población
- Se puede renovar la población completa (generacional) o un pequeño porcentaje de individuos (estado permanente)
- Si aplicamos elitismo, el mejor individuo siempre pasa de una generación a otra

Algoritmos Genéticos

Individuo

Individuo

- Cada individuo codifica una solución del problema
- El individuo es configurable al problema que se quiera resolver
- Podemos tener algoritmos genético con individuos de enteros, de reales, de cadenas, de secuencias, binarios, etc.
- La elección del individuo decidirá los operadores genéticos a emplear

Función de aptitud

- A cada individuo se le debe asignar un valor que mida cómo de buena es la solución que representa respecto al problema
- Normalmente se suele representar mediante una fórmula matemática que representa el problema a optimizar

Algoritmo genético

Esquema básico

Algoritmo-Genético : $T_P \times p_c \times p_m \times G \longrightarrow ind$

$t \longleftarrow 0$

$P_t \longleftarrow$ Crear_Población_Inicial(T_P)

Evaluar_Aptitud()

Mientras $t < G$

  $t \longleftarrow t + 1$

  Mientras $|P_t| < T_P$

    Seleccionar_Operador(p_c, p_m)

    Seleccionar_Individuos_Reproducción(P_{t-1})

    Aplicar_Operador()

    Reemplazar_Individuos(P_t)

  Evaluar_Aptitud()

$ind \longleftarrow$ Mejor_Individuo()

Devolver ind

Algoritmos Genéticos

Operadores genéticos

Selección

- Selecciona individuos de una población para la reproducción
- Esta selección puede ser proporcional o no a la aptitud del individuo
- Normalmente, se diseña generalmente para dar mayores oportunidades de reproducción a los individuos más aptos dentro de la población
- Algunos individuos serán seleccionados más de una vez: los mejores individuos obtendrán más copias, y los peores morirán

Tipos de selección

Ruleta Cada individuo tiene una probabilidad de salir de acuerdo a su aptitud

Torneo Se seleccionan k individuos y se escoge el mejor

Algoritmos Genéticos

Operadores genéticos

Cruce

- Este operador genera dos individuos nuevos, denominados hijos, a partir de dos individuos seleccionados previamente, denominados padres
- Los hijos heredan parte de la información almacenada en cada uno de los dos padres
- El operador dependerá de la codificación de los individuos

Mutación

- Altera la información almacenada en un individuo

Algoritmos Genéticos

Reemplazo

- Introduce los hijos recién creados en la población

Tipos

Reemplazo Generacional reemplaza todos los individuos de la población actual con los hijos producidos durante la reproducción

Estado estable se van insertando los hijos a medida que se van creando, bien de forma automática o bien cuando el valor de aptitud del hijo es mejor que el individuo a ser reemplazado

Elitismo

El mejor individuo de la población siempre sobrevive a la próxima generación, a menos que un individuo mejor haya sido creado

Criterio de parada

- Alcanzar un número máximo de generaciones
- Alcanzar un determinado valor en la función de aptitud
- Sobrepasar un tiempo de ejecución máximo
- Exceder un número máximo de evaluaciones entre actualizaciones del mejor individuo de la población

Algoritmos genéticos

¿Cómo resolver un problema con AG?

Se necesita

- Definir al individuo
- Establecer la función de aptitud
- Decidir cómo se generará la población inicial
- Establecer el método de selección/reemplazo de individuos
- Definir los operadores de cruce y mutación
- Decidir cómo se realiza la generación
- Establecer el criterio de parada

Sección 5 | Mutación evolutiva

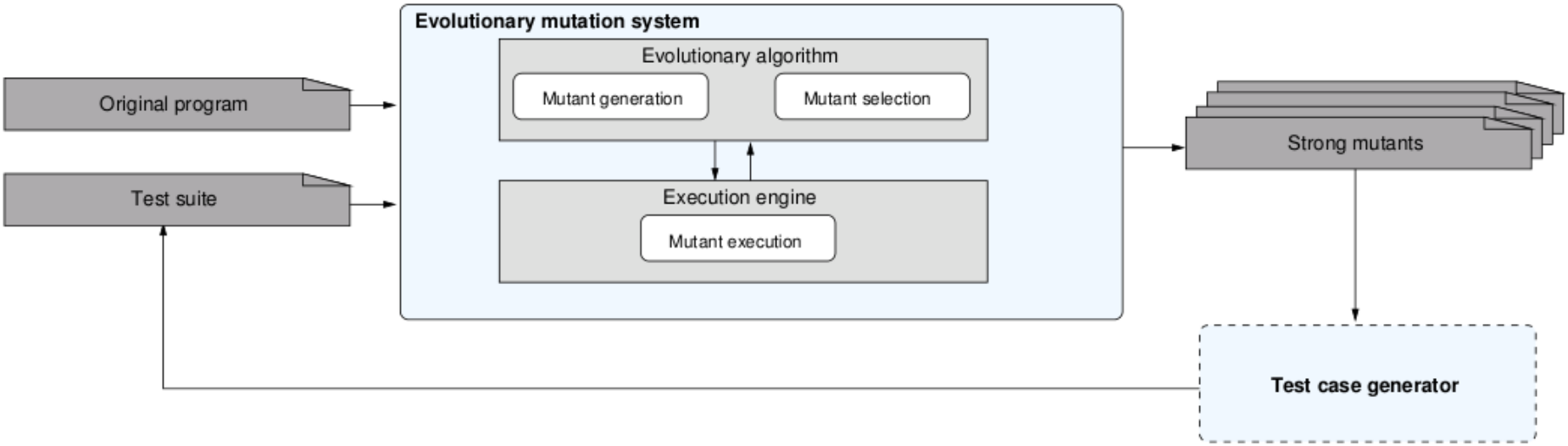
Mutación evolutiva

Características

- Técnica de reducción de mutantes
- Emplea algoritmos evolutivos
- No genera todos los mutantes, sólo un subconjunto de ellos
- Su función de aptitud favorece a los denominados mutantes fuertes, es decir, los mutantes potencialmente equivalentes y los resistentes difíciles de matar
- Es independiente del lenguaje y se puede implementar con cualquier algoritmo evolutivo
- Se dispone de la herramienta GAmera que aplica la prueba de mutaciones al lenguaje WS-BPEL mediante el empleo de un algoritmo genético

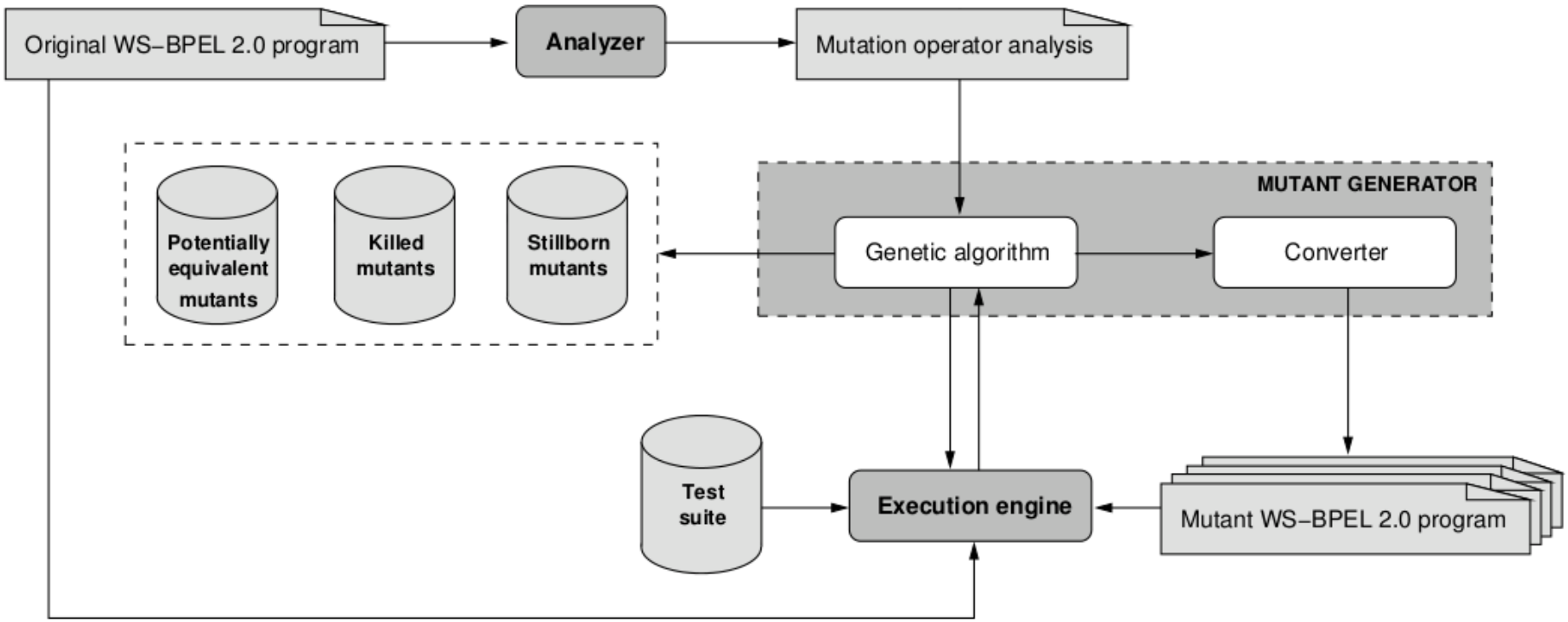
Mutación evolutiva

Esquema de un sistema de mutación evolutiva



Mutación evolutiva

GAmera: Sistema de generación de mutantes para WS-BPEL



Mutación evolutiva

GAmera

Características

- Primer generador de mutantes basado en algoritmos genéticos
- Primer generador de mutantes para WS-BPEL
- Se puede adaptar a otros lenguajes:
  - Analizador + Conversor + Motor de Ejecución
  - MuBPEL $\longrightarrow$ WS-BPEL
- Genera subconjunto de mutantes, potenciando los denominados mutantes fuertes, que son los de más calidad

GAmera

Analizador WS-BPEL

Características

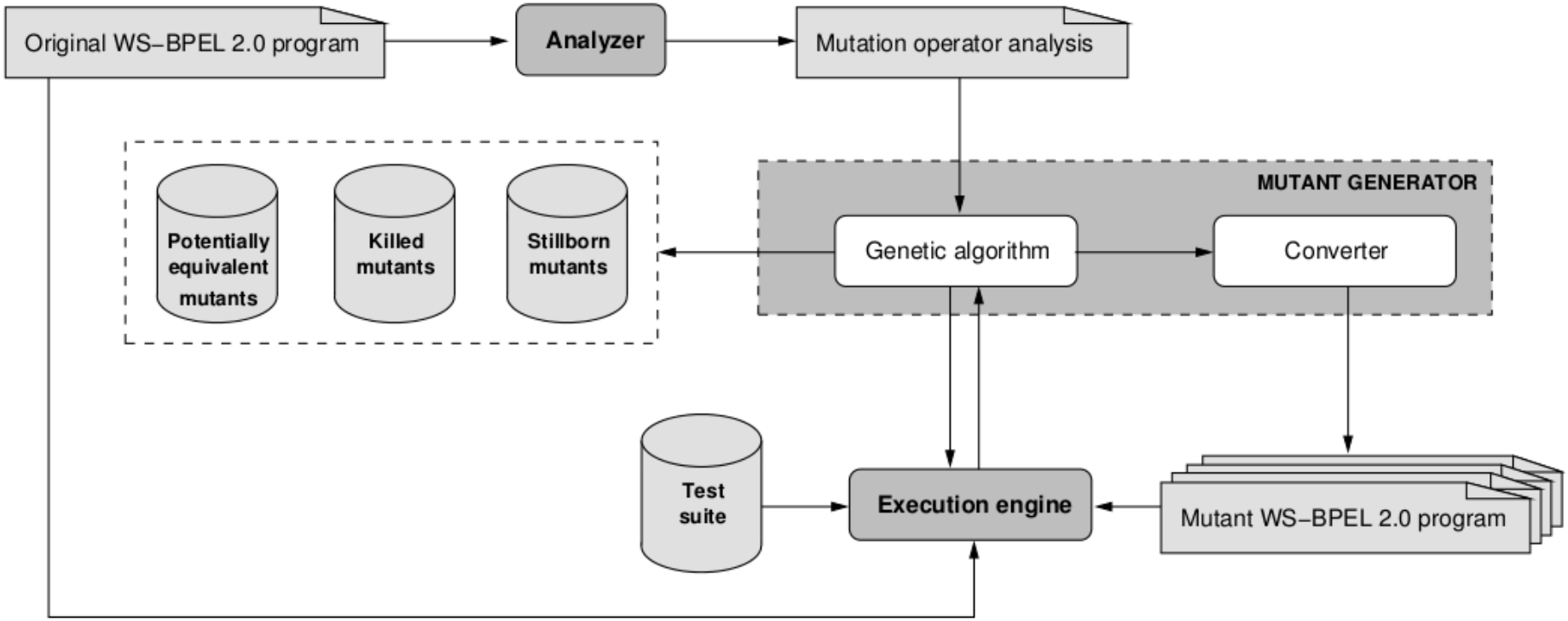
- Identifica los diferentes elementos del programa original que pueden ser mutados
- Lista para cada operador de mutación, su nombre y el número de instancias donde puede aplicarse

Ejemplo de salida de analizador

```
<if name="A">                                    ERR 2
   <condition>
          ( $X &lt;= 10000 )                     ECN 1
   </condition>
    ...                                          AIE 2
<elsif name="B">
   <condition>
          ( $Y = 'low' )
   </condition>
...
```

Mutación evolutiva

GAmera: Sistema de generación de mutantes para WS-BPEL



GAmera

Representación de los individuos

Operator	Location	Attribute

Descripción

Un individuo codifica a un mutante mediante tres campos:

- **Operador** Operador de mutación que se aplica
- **Localidad** Instancia donde puede aplicarse el operador
- **Atributo** Valor necesario para aplicar el operador

La codificación del individuo propuesta es independiente del lenguaje donde se emplee

GAmera

Ejemplo de codificación de un mutante

Individuo (ERR, 2, 3)

Programa Original

```
<if name="discount">
  <condition>
    $buy &gt; 5000
  </condition>
  <invoke name="10off" ... />
  <elseif>
    <condition>
      $buy &gt; 2500
    </condition>
    <invoke name="5off" ... />
  </elseif>
  <else>
    <reply name="nooff" ... />
  </else>
</if>
```

Mutante

```
<if name="discount">
  <condition>
    $buy &gt; 5000
  </condition>
  <invoke name="10off" ... />
  <elseif>
    <condition>
      $buy = 2500
    </condition>
    <invoke name="5off" ... />
  </elseif>
  <else>
    <reply name="nooff" ... />
  </else>
</if>
```

GAmera

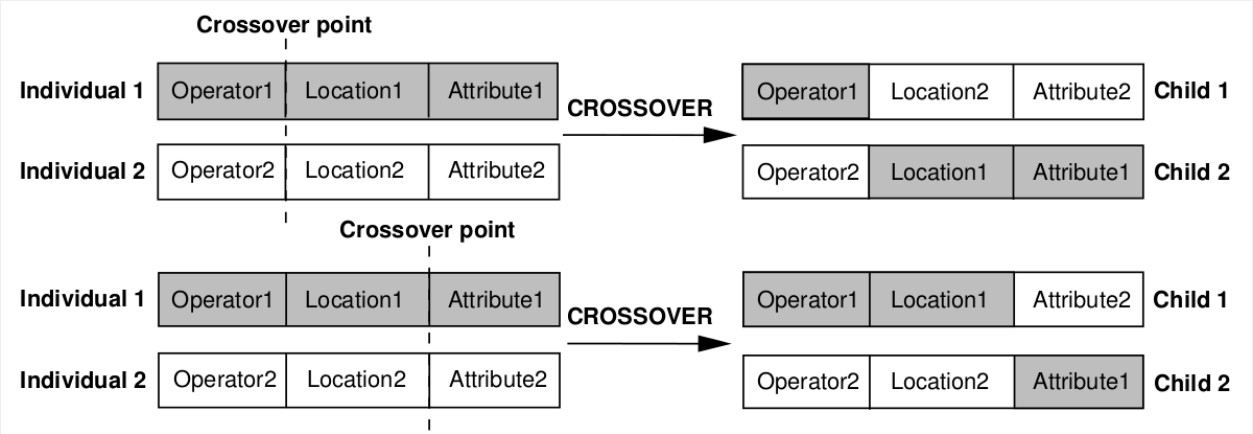
Operadores de mutación y sus atributos

Operador	Valor	Máx. Valor del atributo
ISV	1	N
EAA	2	5	+, -, *, div, mod
EEU	3	1	
ERR	4	6	<, >, >=, <=, =, !=
ELL	5	2	and, or
ECC	6	2	/, //
ECN	7	4	+1, -1, añadir, eliminar
EMD	8	2	0, mitad
EMF	9	2	0, mitad
ACI	10	1	
AFP	11	1	
ASF	12	1	
AIS	13	1	
...	...	...	...

GAmera

Operadores genéticos

Cruce

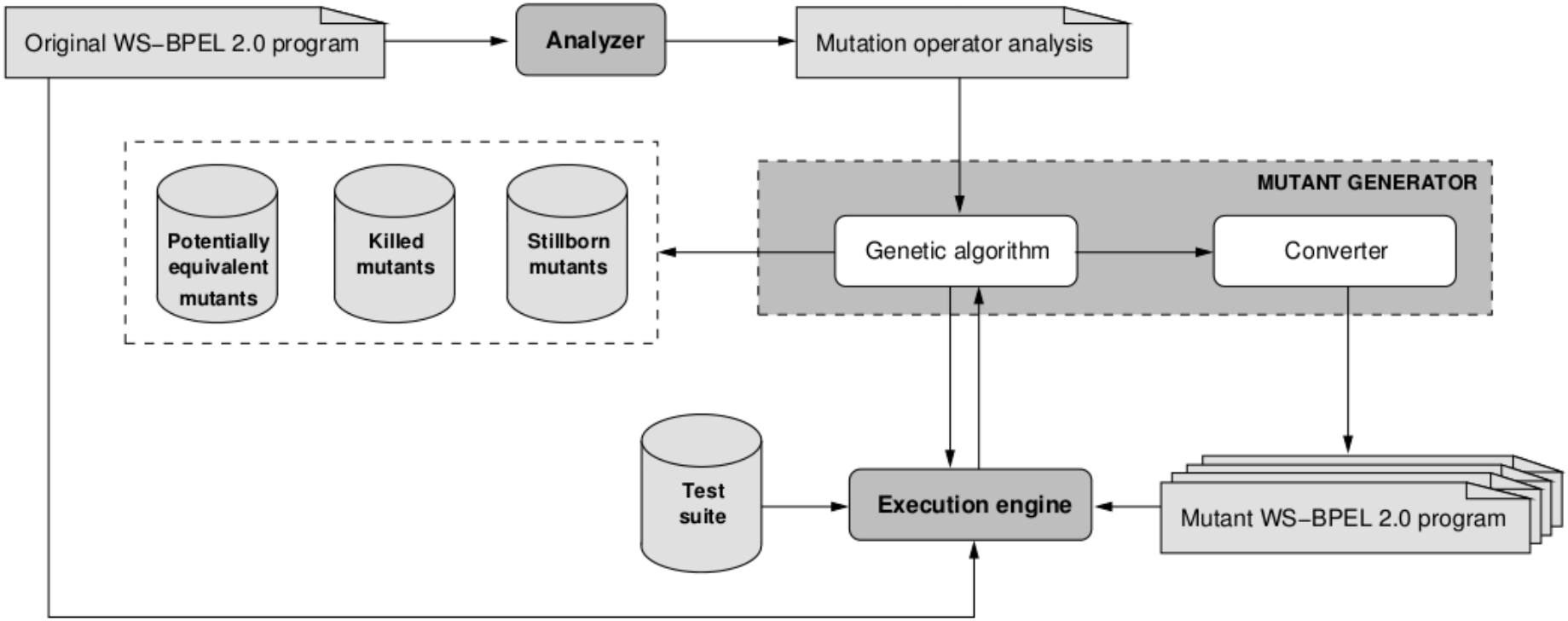


Mutación

- Consiste en realizar una perturbación al valor de un campo del individuo seleccionado

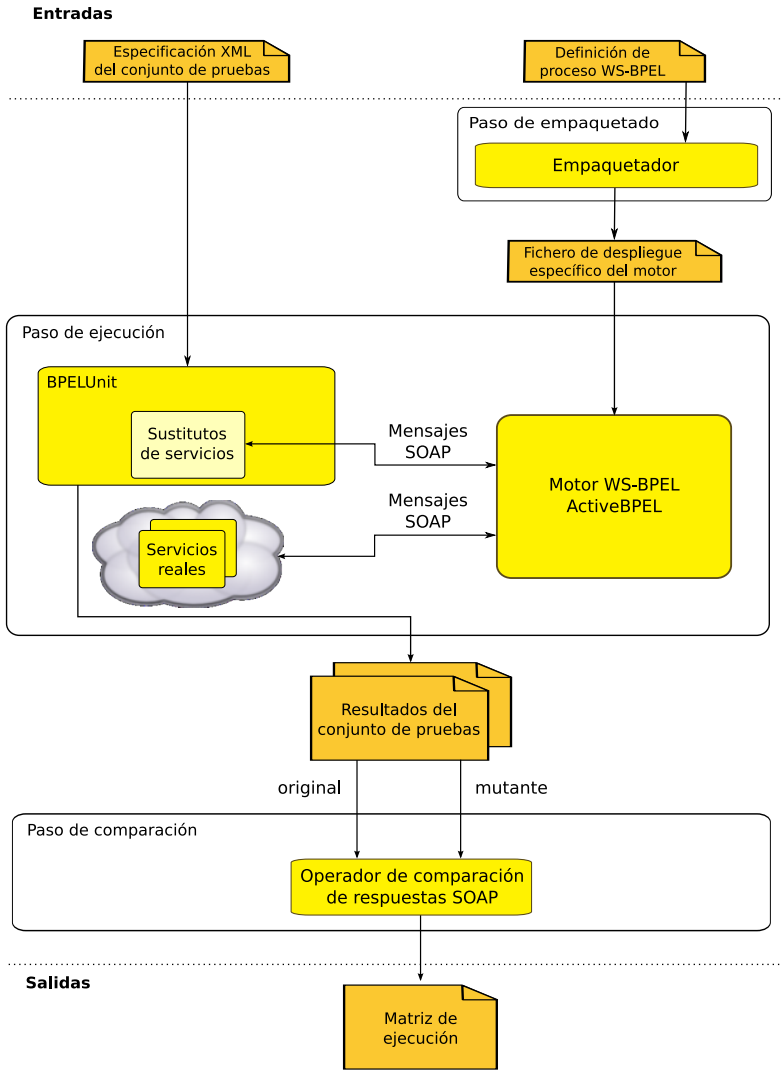
Mutación evolutiva

GAmera: Sistema de generación de mutantes para WS-BPEL



GAmera

Sistema de ejecución



Matriz de ejecución

$$(m_{ij})_{M\times T} = \begin{pmatrix} 0 & 0 & 1 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \\ & & \dots\dots\dots & & \\ 0 & 0 & 0 & \dots & 0 \\ 2 & 2 & 2 & \dots & 2 \end{pmatrix}$$

- Un mutante i está muerto, $\sum_{j=1}^{T} m_{ij} = 1$
- Un mutante i está vivo, $\sum_{j=1}^{T} m_{ij} = 0$
- Un mutante i es erróneo, $\sum_{j=1}^{T} m_{ij} = 2T$

Aptitud

$$\text{Aptitud}(I) = M \cdot T - \sum_{j=1}^{T} \left(m_{Ij} \cdot \sum_{i=1}^{M} m_{ij} \right)$$

- Se penaliza a los mutantes débiles, es decir, los grupos de mutantes que son matados con el mismo caso de prueba
- Se favorece a los mutantes fuertes, es decir:
  - Mutantes resistentes difíciles de matar: aquellos que son matados por un caso de prueba que sólo lo mata a él
  - Los que sigan vivos: que se catalogan como posibles mutantes equivalentes. Estos pueden darnos información sobre el diseño de nuevos casos de prueba $\Longrightarrow$ mejorar la calidad del conjunto de casos de prueba

Características del algoritmo genético

- El algoritmo genético dispone de dos poblaciones
- La primera y principal es la que va evolucionando a lo largo de las distintas generaciones
- Tiene un tamaño fijo especificado como parámetro de entrada
- En cada generación un porcentaje es generado aleatoriamente y el resto mediante operaciones de cruce y mutación
- La selección de los individuos se realiza mediante la ruleta
- La segunda población actúa como salón de la fama y almacena todos los mutantes que va generando en todo el proceso
- Esta población sirve como memoria y se utiliza para calcular la aptitud de los individuos de la población principal

GAmera

Ejemplo de coevolución

	T1	T2	T3	T4	T5	T6	Fitness (only Mi)	Fitness (all)
M1	1	0	1	0	0	0	18 - 2 = 16	30 - 5 = 25
M2	0	0	0	1	0	1	18 - 3 = 15	30 - 3 = 27
M3	0	1	0	1	1	0	18 - 4 = 14	30 - 6 = 24
S1	0	0	1	0	1	0		
S2	1	1	1	0	0	0		

Mutación evolutiva

¿Quieres saber más?

Mutación evolutiva

- Evolutionary Mutation Testing (IST 2011)
  Domínguez, Juan José y otros

Sección 6 | Experimentos

Experimentos

Composiciones utilizadas

- Servicio de reserva de viajes
- Metabúsqueda
- Préstamo extendido

Estimación de los valores óptimos de configuración del AG

- Tamaño de la población
- Probabilidad de cruce
- Número de nuevos mutantes generados entre generaciones
- Porcentaje de mutantes a generar

Experimentos

Datos de las composiciones

	LOC	|NM|	|NS|	|T|	GT (h)	RT (h)
TRSE	363	210	68	10	0.04	2.51
MS	597	529	101	9	0.14	3.21
LAE	1520	3661	1362	21	1.74	94.85

- LOC: líneas de código
- |NM|: número de mutantes
- |NS|: número de mutantes fuertes
- |T|: número de casos de prueba
- GT: tiempo de generación
- RT: tiempo de ejecución

Sección 7 | Conclusiones y trabajo futuro

Conclusiones y trabajo futuro

Conclusiones

- Se ha diseñado, la «mutación evolutiva», una nueva técnica de prueba de mutaciones que permite reducir el coste computacional
- Se ha desarrollado la herramienta, «GAmera», que implementa dicha técnica para composiciones de servicios WS-BPEL
- Los experimentos realizados son positivos y muestran que el sistema genera subconjuntos de mutantes de alta calidad

Trabajo futuro

- Aplicar la mutación evolutiva a otros lenguajes de programación
- Desarrollar un generador de casos de prueba para WS-BPEL
- Desarrollar un sistema de generación de mutantes de orden superior para WS-BPEL
- Realizar un análisis de la calidad de los mutantes

¡Gracias por su atención!

Grupo UCASE de Ingeniería del Software
Departamento de Ingeniería Informática
Universidad de Cádiz

inmaculada.medina@uca.es

Trabajo financiado por el proyecto MoDSOA (TIN 2011-27242) del Programa Nacional de I+D+i del Ministerio de Ciencia e Innovación