SPI&FM
Software Process Improvement and Formal Methods group

UCA
Universidad de Cádiz

Generación dinámica de invariantes en composiciones de servicios web con WS-BPEL

Manuel Palomo Duarte

Índice

- Introducción
  - SOA y WS-BPEL
  - Prueba de software con invariantes
- Trabajo realizado
  - Generación dinámica de invariantes en WS-BPEL
  - Experimentos
- Trabajo actual
- Líneas futuras
- Referencias

SOA

- Arquitecturas Orientadas a Servicio (SOA)
  - Conjunto de tecnologías con las características:
    - Funcionamiento distribuidas
    - Base Internet: HTTP + SOAP + XML
    - Independientes de lenguaje de programación, sistema operativo y plataforma hardware
    - Extensibles
    - Gobernables
  - Están cambiando la informática “de gran escala”
    - Mucho interés empresarial y grandes clientes

SOA

- Algunas de las tecnologías en SOA:
  - WS-Policy: negociación de condiciones
  - WS-Security: seguridad
  - UDDI: descubrimiento dinámico de servicios
  - WS-Transaction: transacciones
- Importancia de los estándares abiertos:
  - W3C, OASIS
  - Gobernabilidad

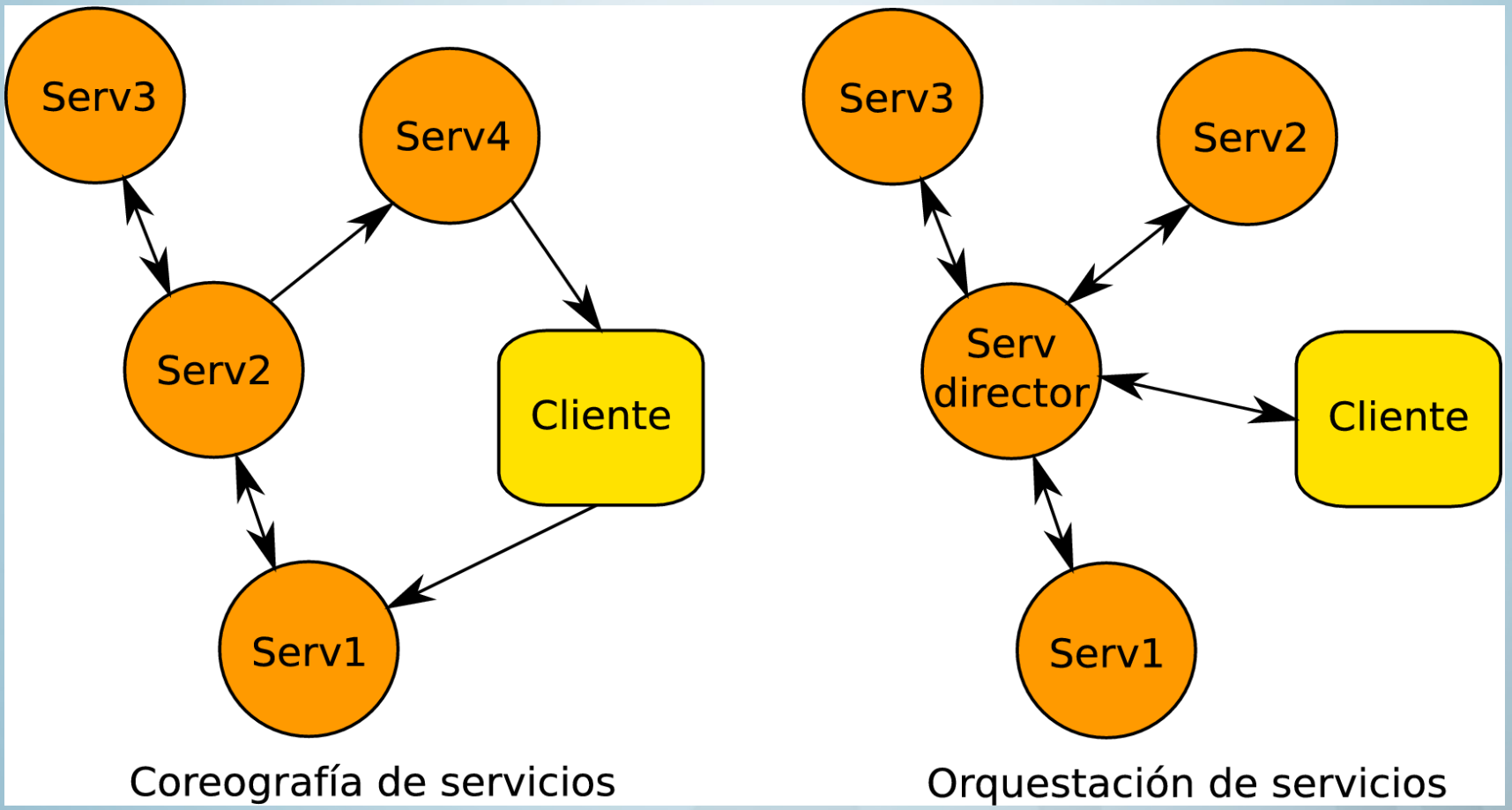
SOA

- La base de SOA son los Servicios Web (WS):
  - Software diseñado para:
    - Ejecutarse en un servidor HTTP
    - Servir peticiones recibidas en formato SOAP
    - Descrito en WSDL
    - Facilita la interacción automática
    - Identificado por URI
    - Fácilmente sustituible para el proveedor
    - Fácilmente sustituible para el consumidor

WS-BPEL

- Los WS ofrecen funcionalidades concretas
- Si quiero hacer un WS basado en otro(s) WS tengo dos opciones (sig. página):
  - Coreografía: no existe control centralizado
    - Poco éxito: complicado, débil ante fallos
  - Orquestación: existe director
    - Bastante éxito empresarial
    - Algunas empresas propusieron BPEL4WS (1.1)
    - OASIS estandarizó WS-BPEL 2.0

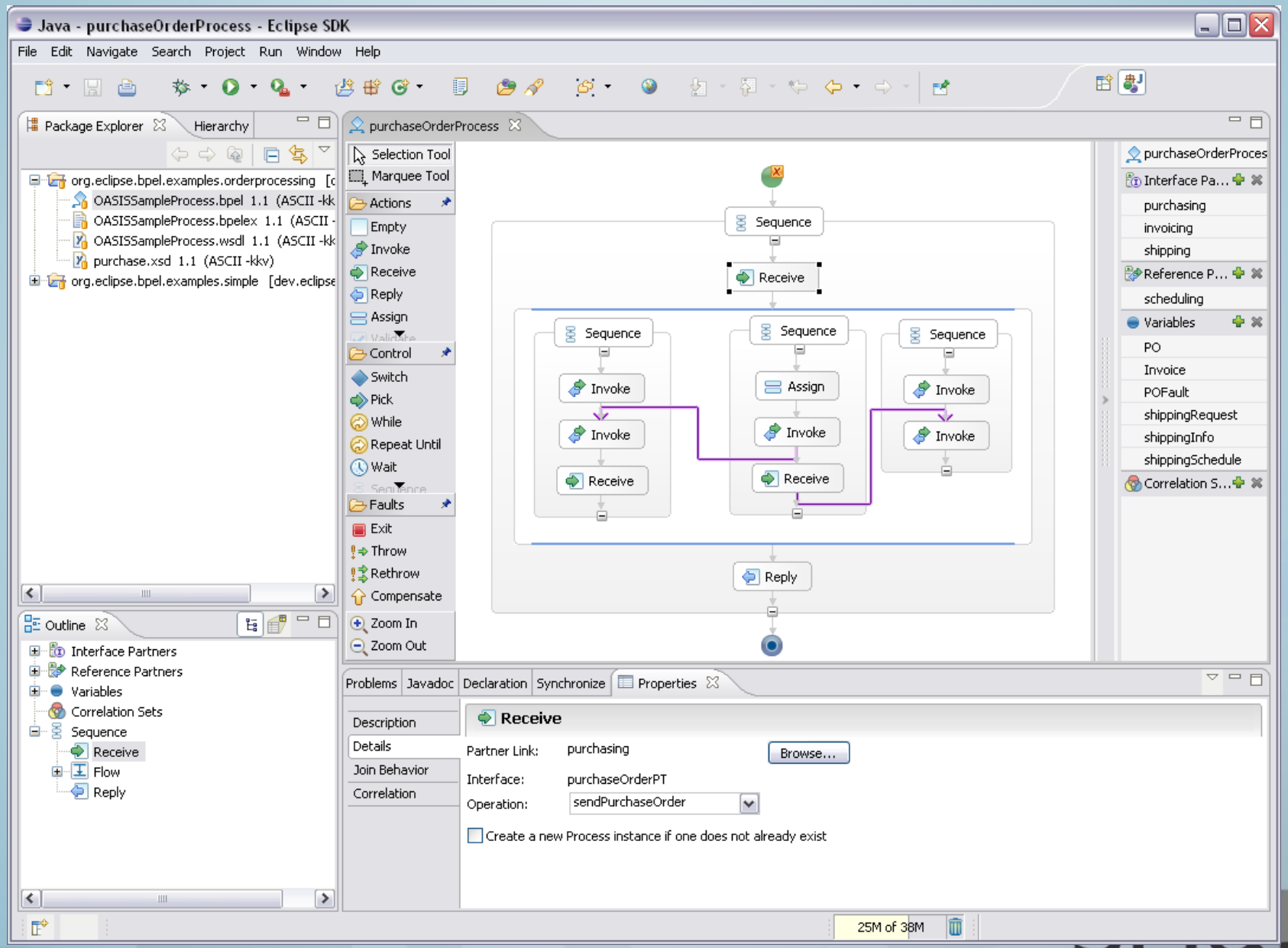
WS-BPEL



WS-BPEL

- BPEL es un lenguaje de programación a gran escala (*programming-in-the-large*), con:
  - Características propias de lenguajes *tradicionales*:
    - Variables (XML Schema)
    - Asignaciones
    - Bucles
  - Otras propias:
    - Instrucciones “poco comunes”: manejo de errores, tratamiento de eventos, etc.
    - En vez de llamar funciones, llama a WS
    - Los WS pueden no estar disponibles o funcionar mal

WS-BPEL



WS-BPEL

- Las composiciones de WS en BPEL:
  - Se programan en un editor que genera código XML
  - Se despliegan en un servidor web, quedando a la espera de peticiones de servicio
  - Cuando llega un mensaje, el servidor mira si es:
    - Un mensaje de petición nueva, y crea un proceso
    - Un mensaje para un proceso en ejecución, y se lo entrega
  - Cuando no deseamos ofrecer más el servicio, se repliega

Recopilación: SOA

- Las tecnologías SOA están aumentando la interoperabilidad
  - La base son los WS
- Con WS-BPEL se puede programar usando WS fácilmente
  - Incorpora todas las instrucciones tradicionales
  - También otras no tradicionales
  - Está sujeto al entorno (caída de red, etc)
- La prueba de WS-BPEL no parece sencilla

Prueba de software

- La prueba funcional se divide en dos tipos:
  - Caja negra: se prueba un sistema considerando sólo sus entradas y salidas
  - Caja blanca: se prueba el sistema analizando su lógica interna. Más potente
- Al programar en BPEL:
  - Tenemos el código fuente de la composición
  - No tenemos el de los WS
  - Se adecua a la prueba de caja blanca

Prueba con invariantes

- Un invariante es una propiedad que cumple un programa:
  - Pre y post-condiciones
  - Invariantes de bucles
  - Etc.
- Sirven para probar un programa:
  - Verificación (encontrar errores)
  - Ayudar al ampliar un programa

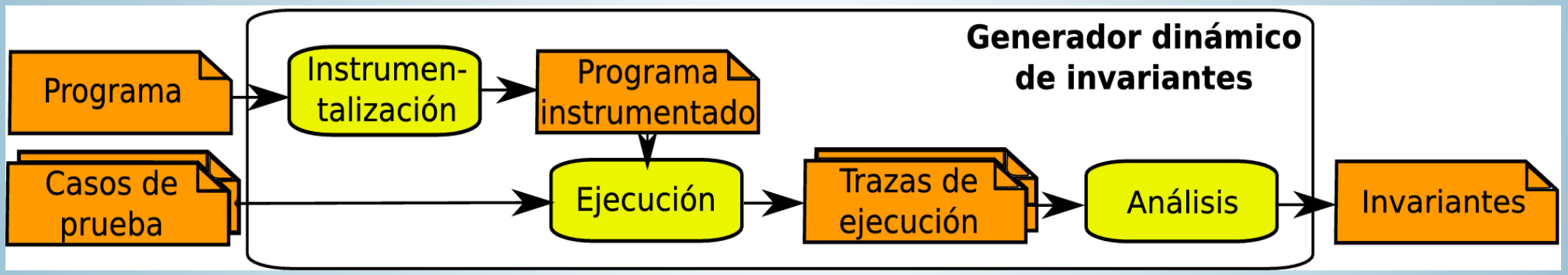
Generación de invariantes

- Tres formas de generarlos:
  - Manualmente:
    - No abordable (sólo en ADA ;-)
  - Automáticamente
    - Analizan el programa (dependientes del lenguaje)
    - Capacidad limitada
  - Dinámicamente
    - Genera invariantes de información recopilada en una serie de ejecuciones del programa
    - Cuidado con la fiabilidad

Generación Dinámica de invar.

- Se hace en tres fases:
  - Instrumentalización: se prepara un entorno de ejecución que genere información del programa
    - Añadir *prints* al programa, modificar la *JVM*, etc
  - Ejecución: se ejecuta el programa con una serie de casos de prueba
    - Se obtienen *trazas* de ejecución de cada caso
  - Inferencia: se analizan las trazas
    - Comprobación de propiedades

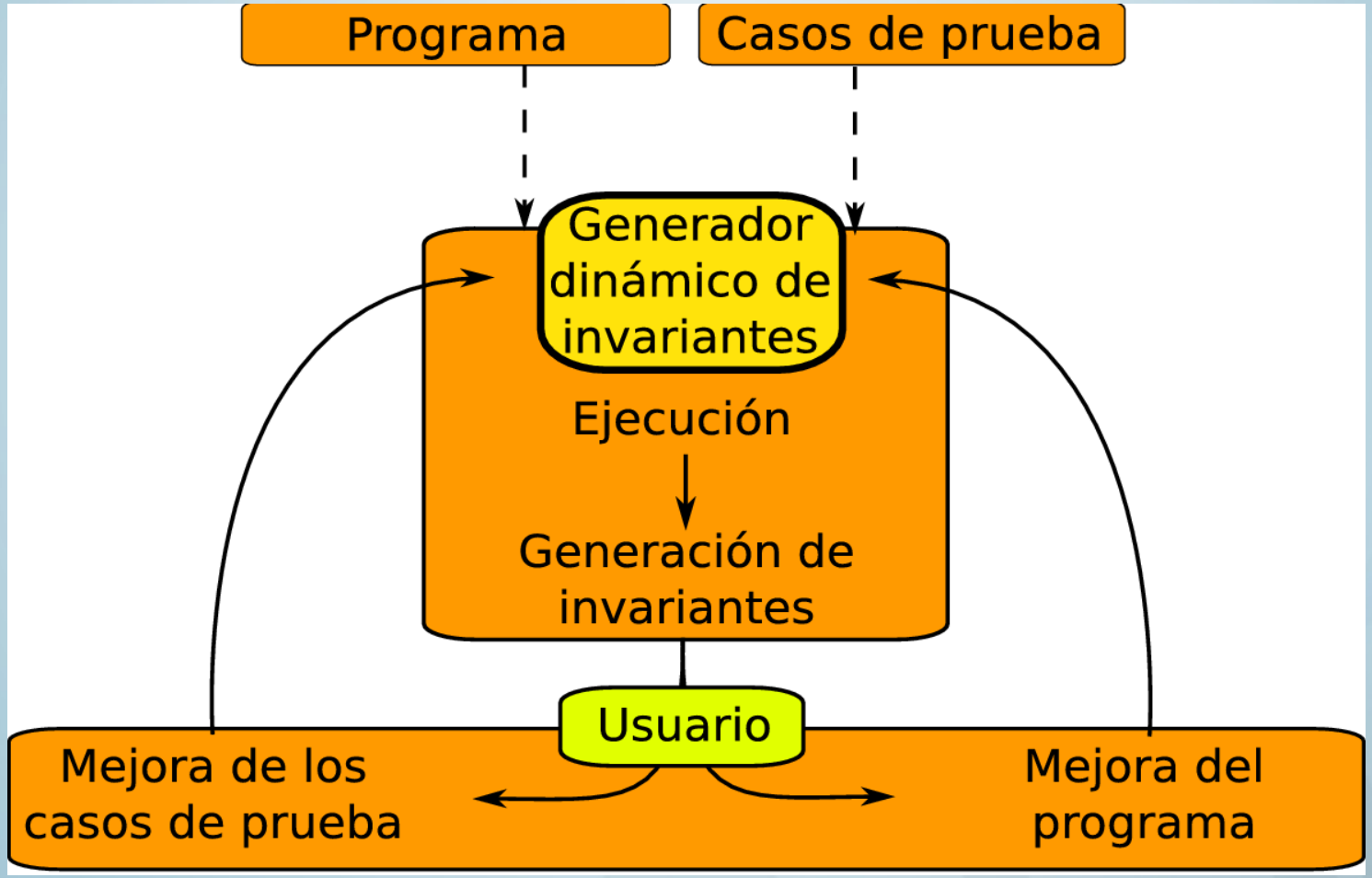
Generación Dinámica de invar.



Prueba con invariantes dinámicos

- Cuidado con los invariantes:
  - Si uso un conjunto de casos de prueba *bueno* obtendré invariantes interesantes
  - Si uso un conjunto de casos de prueba limitado o sesgado puedo obtener invariantes falsos
    - Esos invariantes me ayudarían a mejorar el conjunto
- La generación dinámica de invariantes permite:
  - Evaluar y mejorar conjuntos de casos de prueba (además de la prueba de software)

Prueba con invariantes dinámicos



Introducción - recopilación

- Las tecnologías SOA (WS + WS-BPEL) permiten programar a gran escala
  - Pero también complican la prueba de software
- La generación dinámica de invariantes es una técnica de prueba de software basada en la ejecución de programas
  - ¿Es aplicable a WS-BPEL?

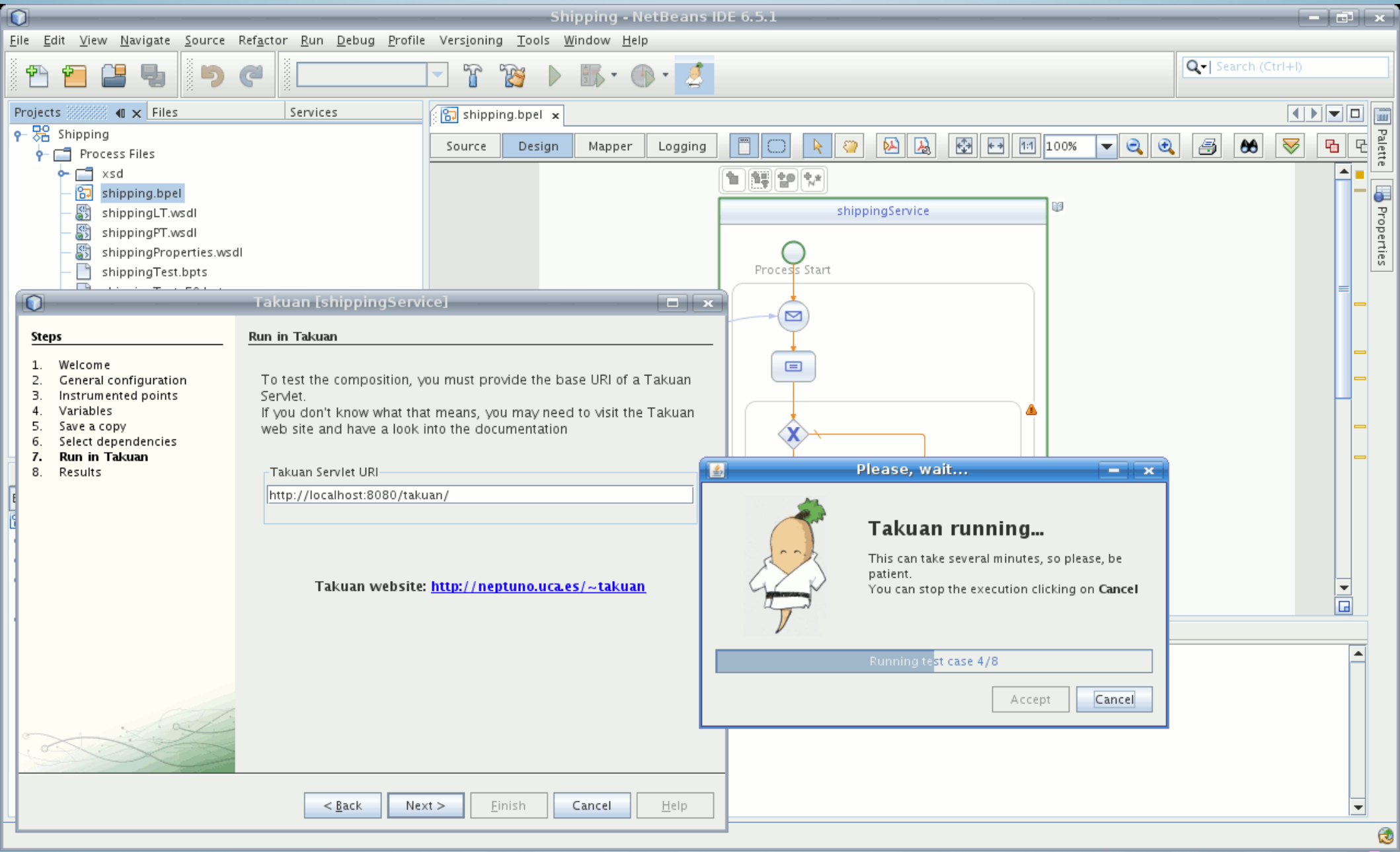
Generación dinámica de invariantes en WS-BPEL

- La generación dinámica de invariantes parece ser adecuada para la prueba de WS-BPEL por:
  - No realizar traducción alguna del código
  - Basarse en información recopilada en ejecuciones real del código
    - El entorno y complejidad interna se refleja en las trazas
- Sería interesante que se pudieran simular WS:
  - Que no estén disponibles, sean caros, etc
  - Porque deseemos definir un “escenario” para su prueba

Takuan

- Takuan es un generador dinámico de invariantes para WS-BPEL. Internamente usa:
  - ActiveBPEL: motor WS-BPEL 2.0
  - BPELUnit: biblioteca de pruebas unitarias
    - Ejecuciones masivas y simulación de WS
  - Daikon: generador dinámico de invariantes para C, C++ y Java
- Todos son sistemas libres, al igual que Takuan:
  - http://neptuno.uca.es/~takuan

SPI&FM
Software Process Improvement and Formal Methods group



SPI&FM
Software Process Improvement and Formal Methods group

Opciones de Takuan

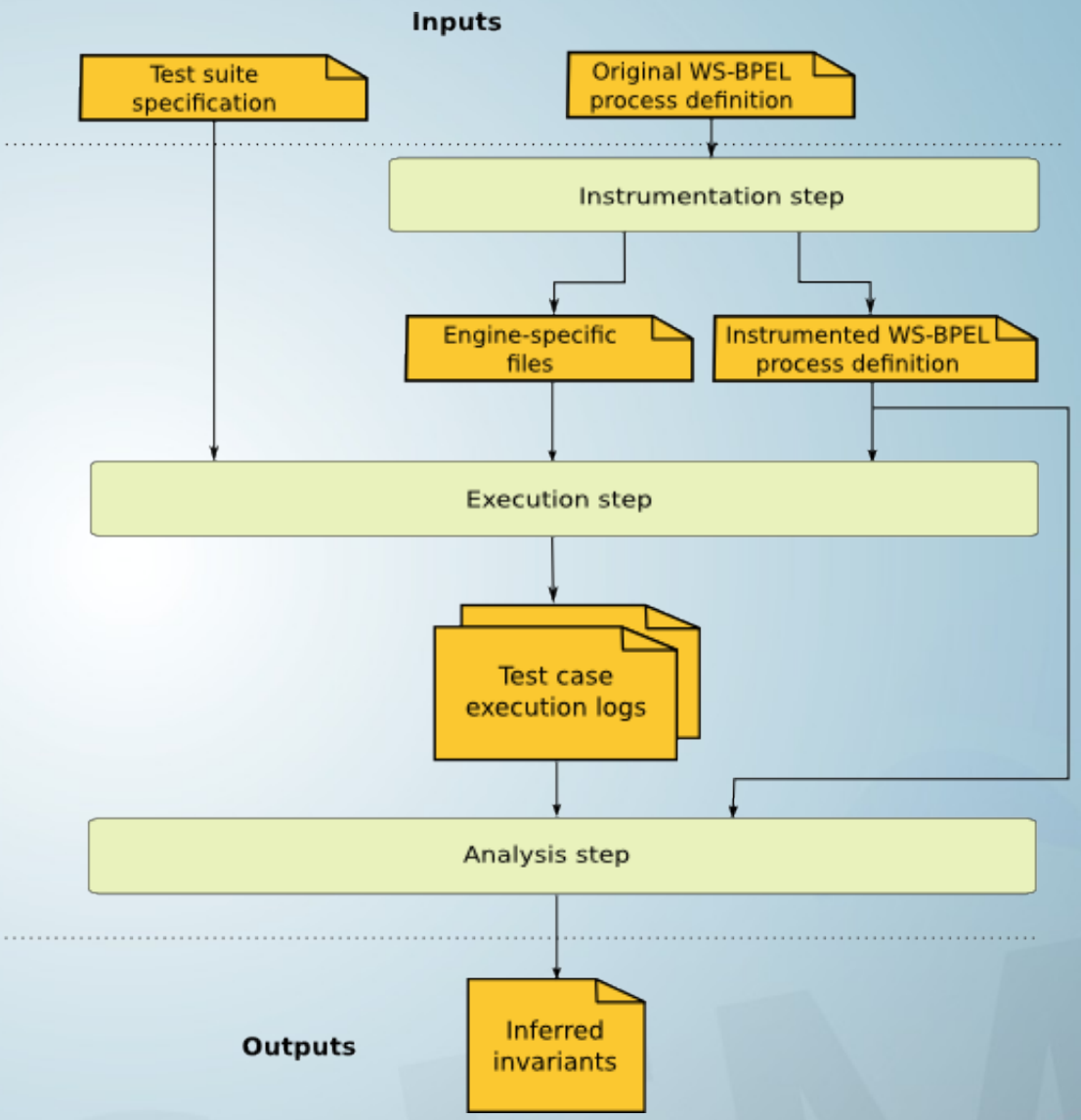
- Takuan por defecto instrumenta las instrucciones que estructuran el WS-BPEL:
  - *Sequence*
  - *Flow*
  - Se pueden seleccionar cuales
  - Se denominan puntos del programa (PP)
- Por defecto:
  - Registra todos los campos de todas las variables
  - Se pueden indicar cuáles en cada punto

Opciones de Takuan

- Takuan por defecto instrumenta las instrucciones que organizan el WS-BPEL:
  - Sequence
  - Flow
  -

Architectura de Takuan

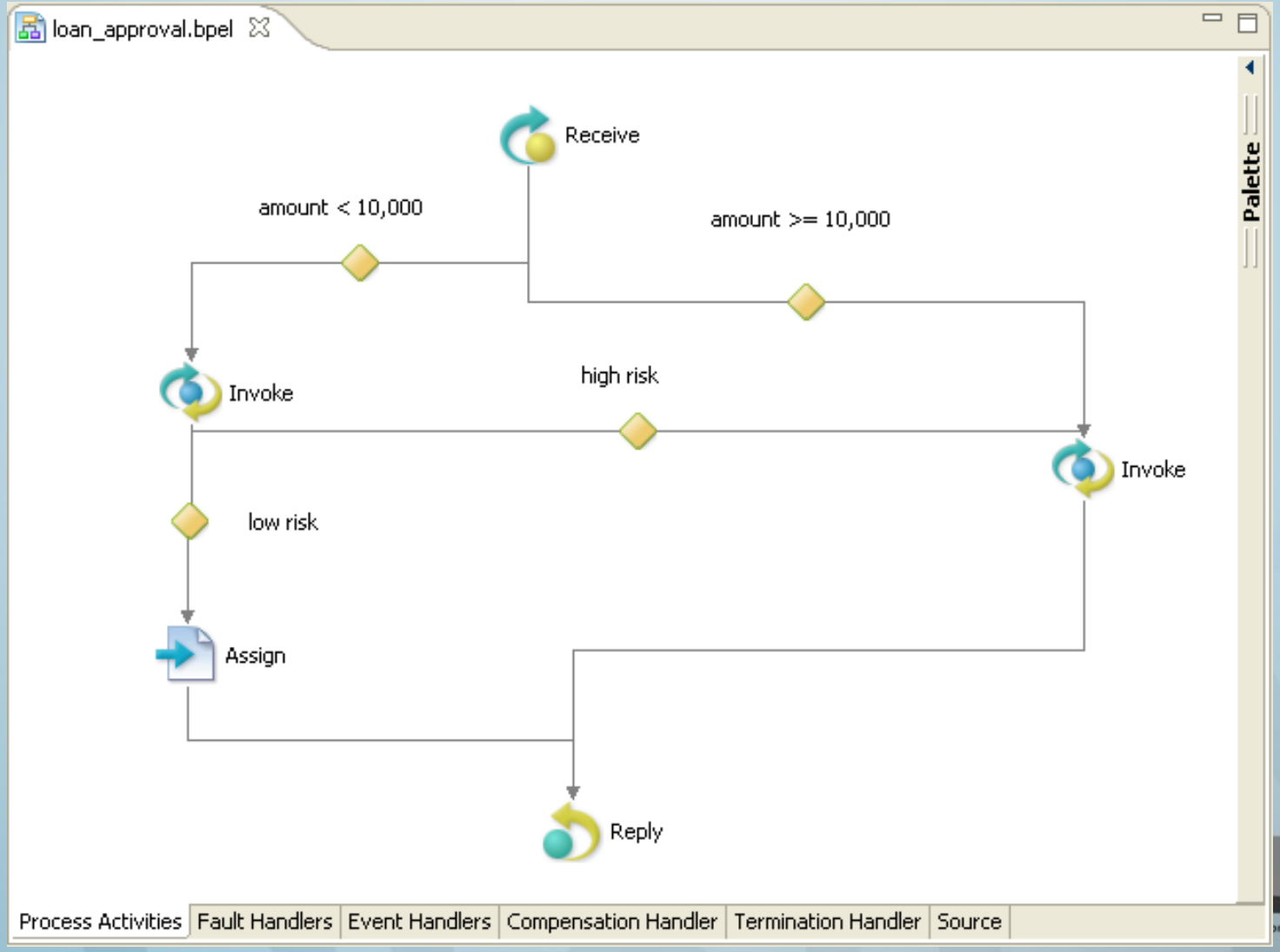
- Basada en las tres fases:
  - Instrumentalizaión
  - Ejecución
  - Análisis



SPI&FM
Software Process Improvement and Formal Methods group

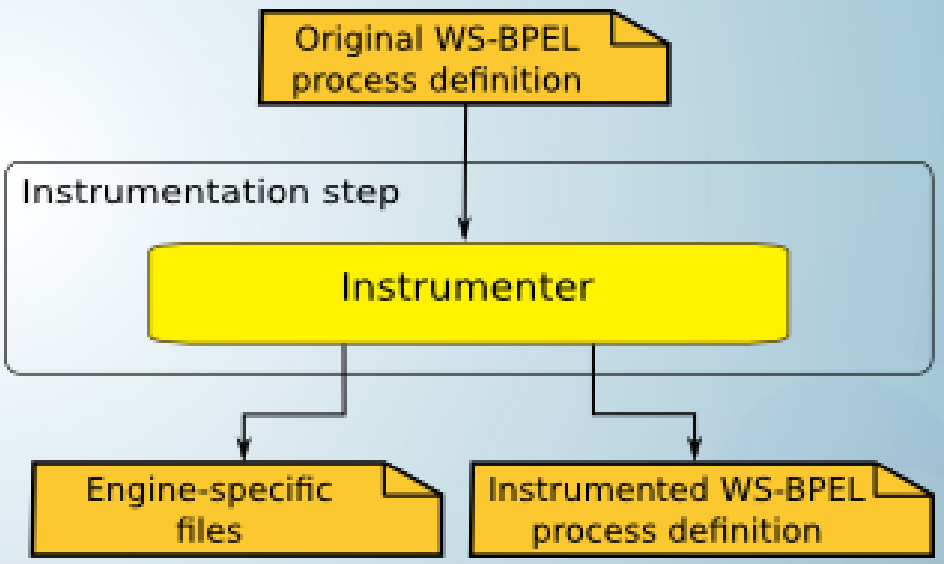
Arquitectura de Takuan

- La veremos con el ejemplo del préstamo:



Arquitectura de Takuan

- Instrumentalización:
  - Se añaden funciones XPath de impresión de valores de variables



SPI&FM
Software Process Improvement and Formal Methods group

Ejemplo

- Código sin instrumentar:

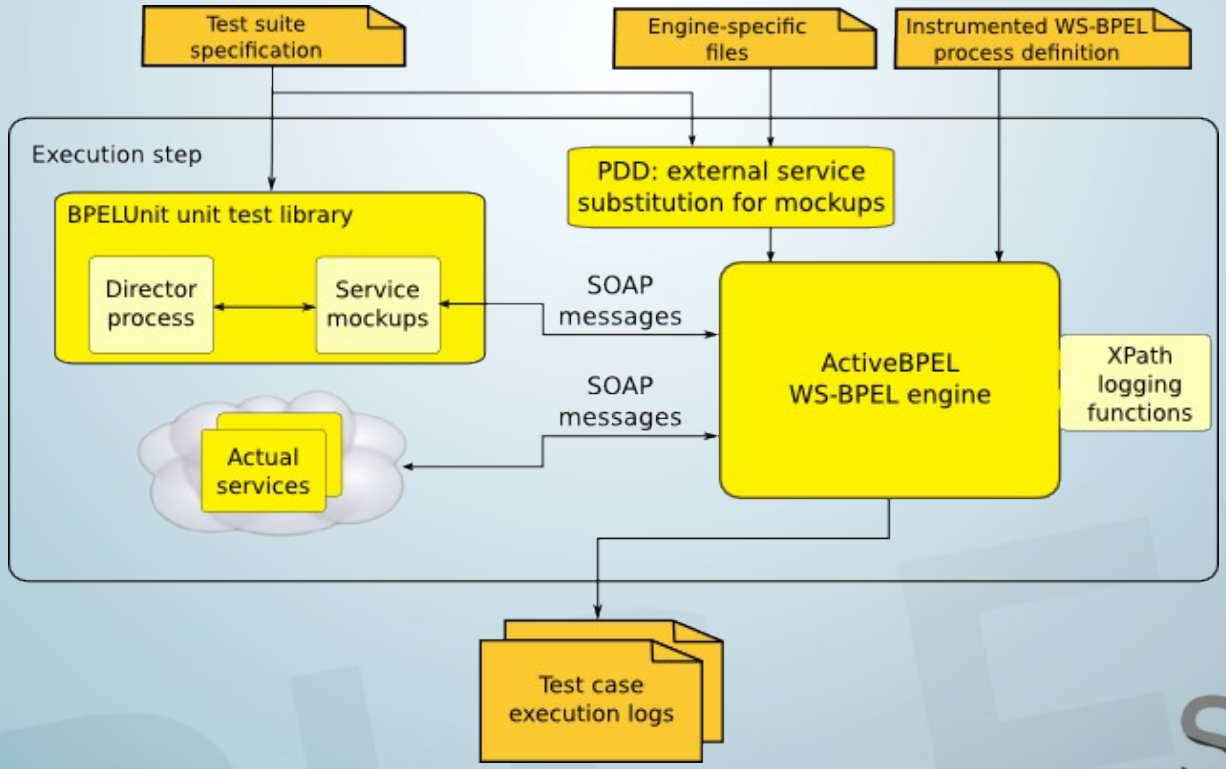
```
<assign name="approveLoan"> <copy>
    <from>true()</from>
    <to>$processOutput.output/accept</to>
  </copy> </assign>
```

Ejemplo

```
<sequence>
  <assign> <copy>
    <from>reg:inspect('$processOutput.output')</from>
    <to>$dummy_processOutput.output</to>
  </copy> </assign>
  <assign name="approveLoan"> <copy>
      <from>true()</from>
      <to>$processOutput.output/accept</to>
    </copy> </assign>
  <assign> <copy>
    <from>reg:inspect('$processOutput.output')</from>
    <to>$dummy_processOutput.output</to>
  </copy> </assign>
</sequence>
```

Takuan architecture

- Ejecución:
  - *BPELUnit* simula WS y ejecuta los casos de prueba en *ActiveBPEL*. Cada caso genera su traza



Ejemplo

- Ejemplo de caso de prueba:

```
<testCase name="SmallAmount" ...>
  <clientTrack> <sendReceive operation="approveLoan" ...>
    <send>
      <data> <ex:ApprovalRequest>
        <ex:amount>150000</ex:amount>
      </ex:ApprovalRequest> </data>
    </send>
  </sendReceive> </clientTrack>
  <!-- ... partner tracks for external services ... -->
</testCase>
```

Ejemplo

- Ejemplo de registro de ejecución (*log*):

Executing [(...)/sequence/assign]

  INSPECTION($processOutput.output/accept) = false

Completed normally [(...)/sequence/assign]

Executing [(...)/sequence/assign]

Completed normally [(...)/sequence/assign]

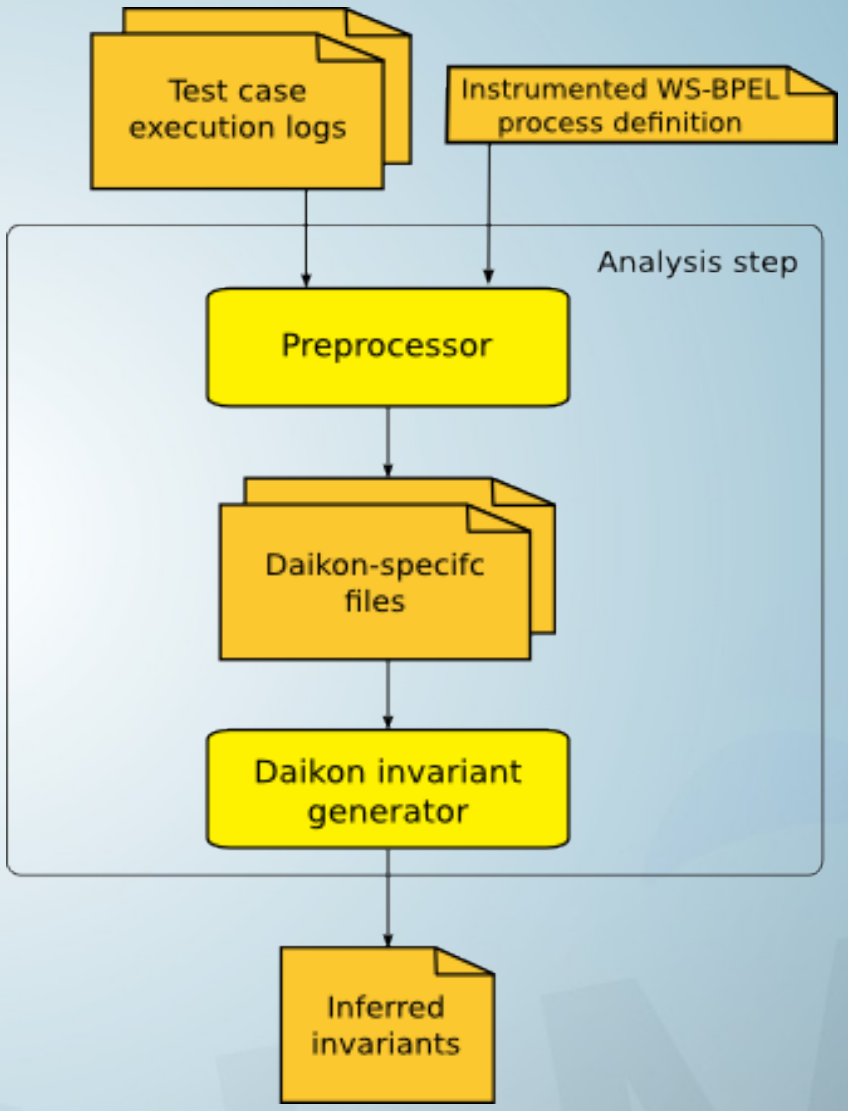
Executing [(...)/sequence/assign]

  INSPECTION($processOutput.output/accept) = true

Completed normally [(...)/sequence/assign]

Arquitectura de Takuan

- Análisis:
  - Un script adapta los logs al formato de entrada de *Daikon*
  - *Daikon* genera los invariantes



SPI&FM
Software Process Improvement and Formal Methods group

Ejemplo

- Invariantes al final de la composición:

approverInput.input.amount ==
processInput.input.amount

approverOutput.output.accept ==
processOutput.output.accept

approverOutput.output.accept one of { 0,1 }

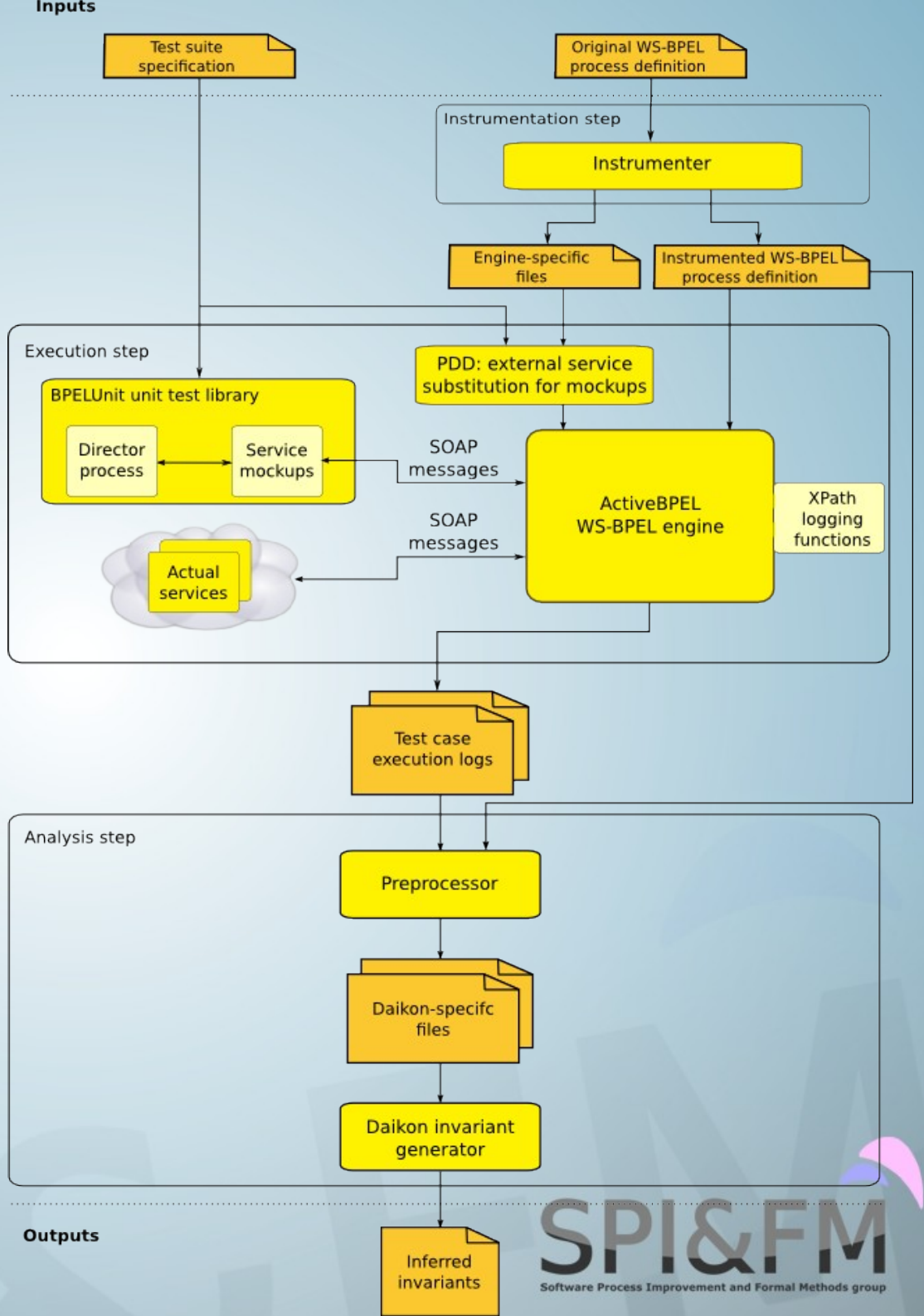
approverInput.input.amount == 150000

size (approverOutput.output.accept) == 1

Ejemplo

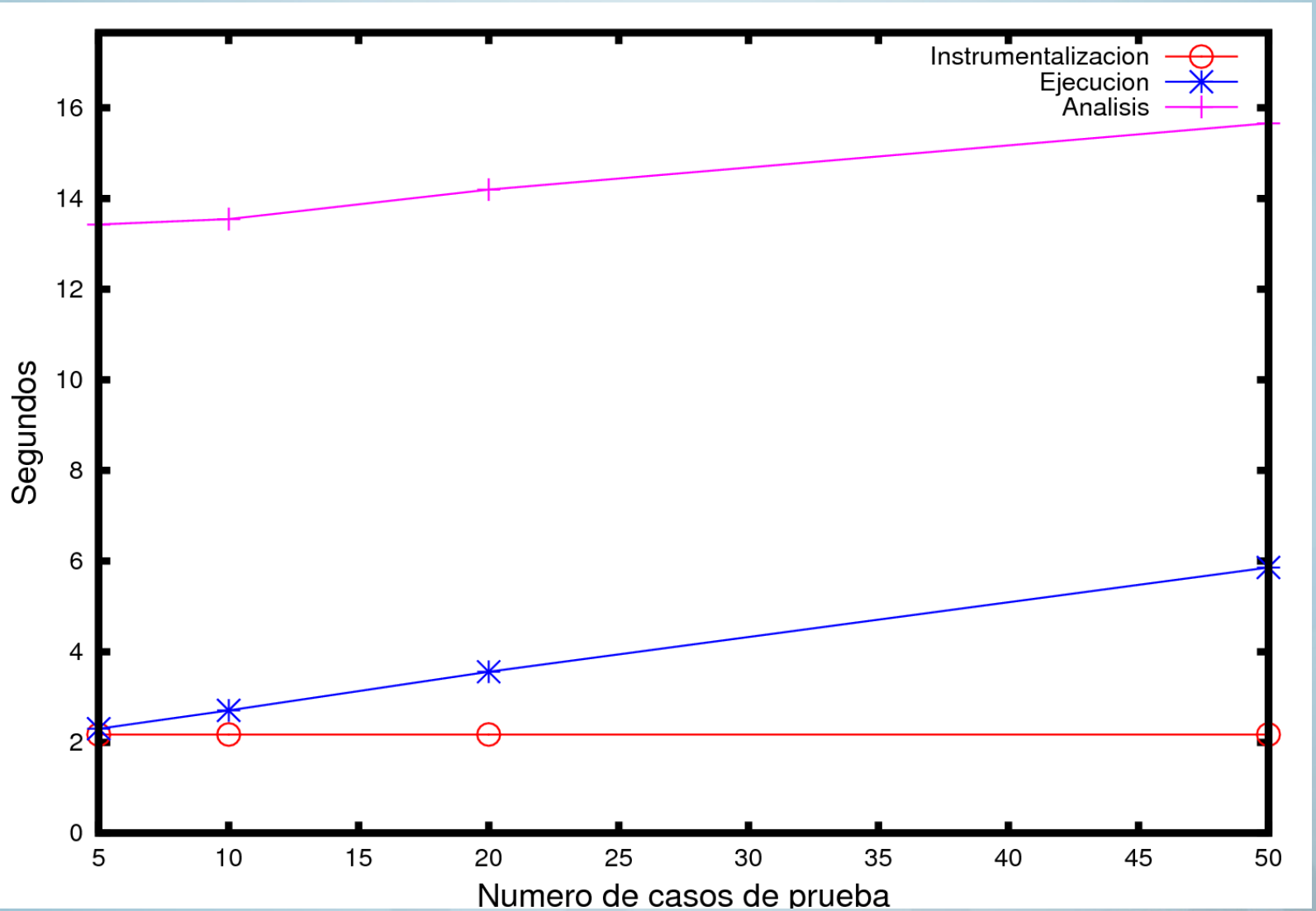
- Invariantes interesantes:
  - approverInput.input.amount == processInput.input.amount
  - approverOutput.output.accept == processOutput.output.accept
  - approverOutput.output.accept one of { 0,1 }
- Invariantes falsos (por casos de prueba malos)
  - approverInput.input.amount == 150000
- Invariantes incluidos en el XML Schema:
  - size (approverOutput.output.accept) == 1

Arquitectura de Takuan



SPI&FM
Software Process Improvement and Formal Methods group

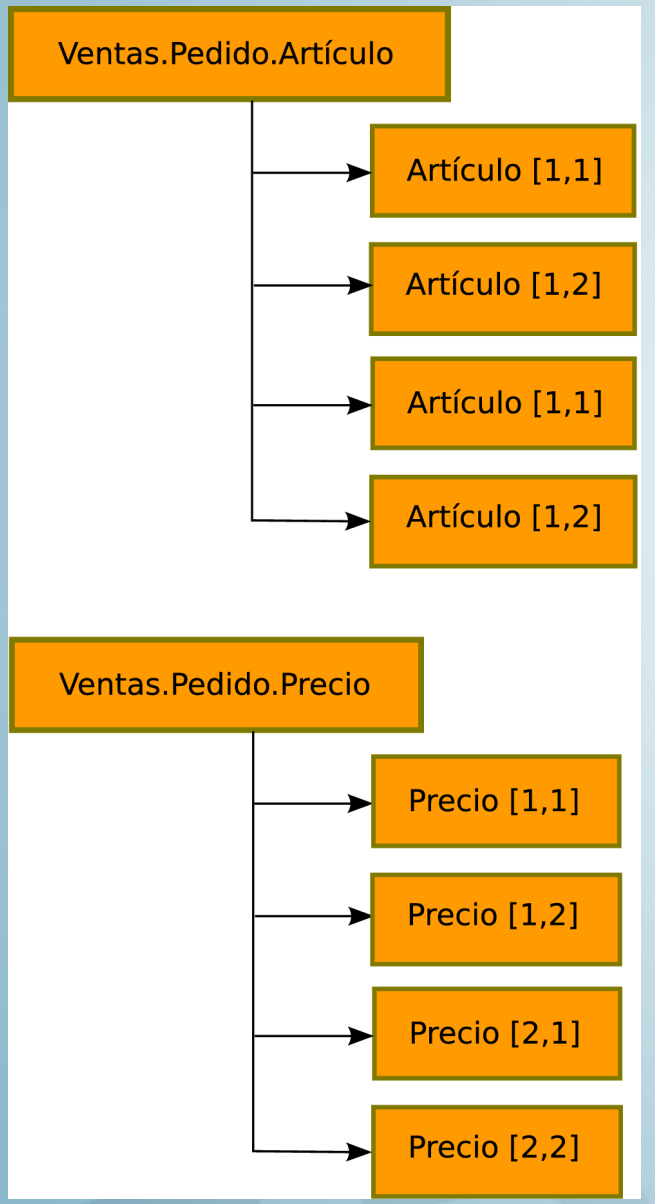
Rendimiento de Takuan

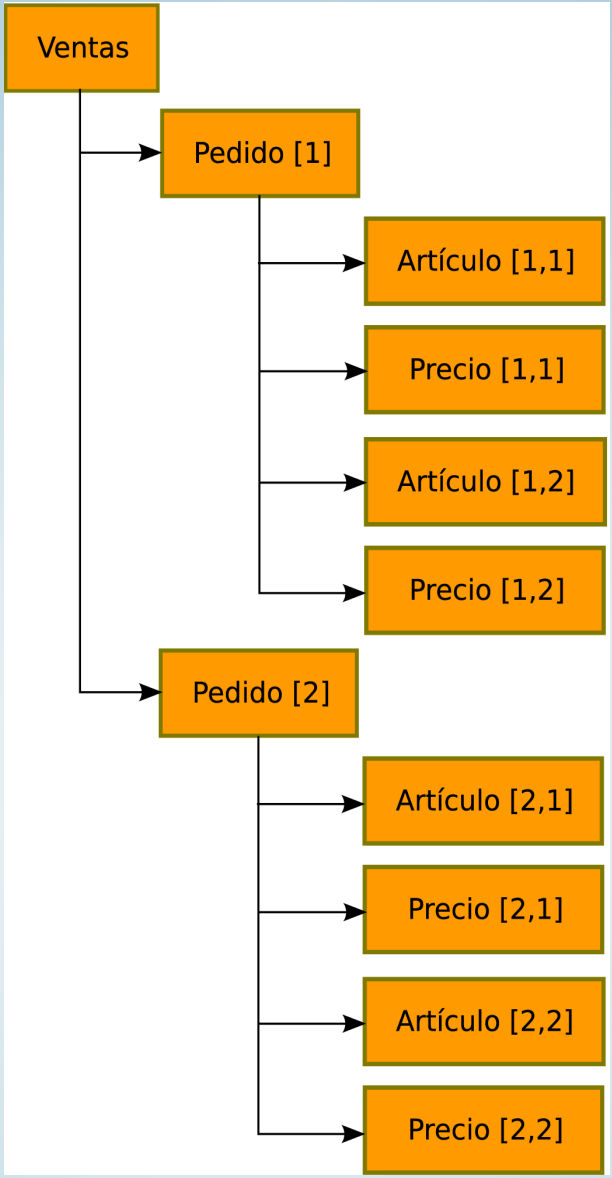


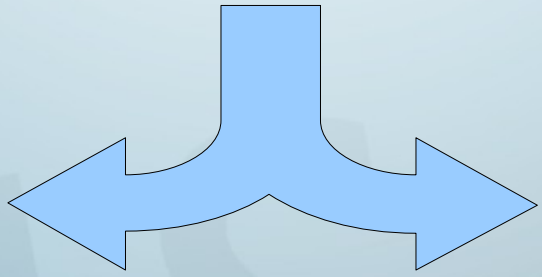
Mapeo de árboles XML Schema

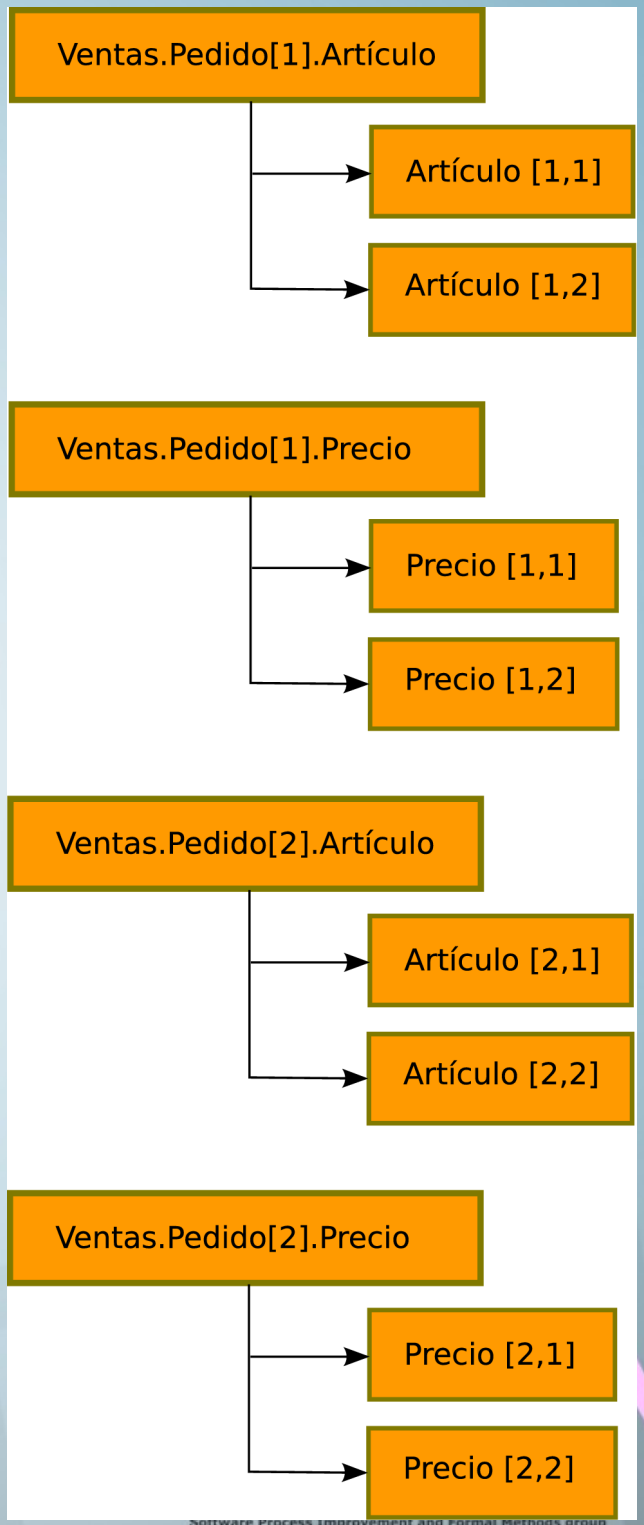
- Los datos XML Schema suelen ser árboles
  - Daikon sólo maneja vectores unidimensionales
- Técnicas de mapeo:
  - División: se crean vectores con las hojas agrupadas por padres
    - Como Kvasir, un mapeador de C++ a Daikon
  - Aplanado: se crean vectores con las hojas de un mismo tipo sin importar el padre
    - Como el operador // de XPath

Mapeo de árboles XML Schema









Mapeo de árboles XML Schema

Variable XML Schema original

```
<MetaSearchProcessResponse>
    <noResult>4</noResult>
    <noFromGoogle>1</noFromGoogle>
    <noFromMSN>3</noFromMSN>
    <result>
        <url>http://url1google</url>
        <title>Title1google</title>
        <snippet>Snippet1google</snippet>
        <from>Google</from>
    </result>
    ... <!-- (three results from MSN) -->
</MetaSearchProcessResponse>
```

Mapeo de árboles XML Schema

Resultado con división

Integer noResult = 4

Integer noFromGoogle = 1

Integer noFromMSN = 3

String result1_url = “http://url1google”

String result1_title = “Title1google”

String result1_snippet = Snippet1google

String result1_from = Google

String result2_url = “http://url1msn”

...

Mapeo de árboles XML Schema

Resultado con aplanado

Integer noResult = 4

Integer noFromGoogle = 1

Integer noFromMSN = 3

String result_url[1] = “http://url1google”

String result_url[2] = “http://url1msn”

String result_url[3] = “http://url2msn”

String result_url[4] = “http://url3msn”

String result_title[1] = “Title1google”

...

Resumen de resultados

Mapeo	P.P	Variables	Tiempo
División	64	17.404 (4720)	7:18 (1:19)
División	127	2.240 (704)	0:43 (0:42)
Aplanado	64	11.412 (3888)	3:46 (1:01)
Aplanado	12	1.560 (624)	0:28 (0:19)

Mapeo de árboles XML Schema

- Comparación de rendimiento:
  - La división tarda más que el aplanado
    - Esa diferencia aumenta al aumentar el número de variables y puntos a inspeccionar
- Tipos de invariantes:
  - Se obtiene distintos tipos de invariantes, sin ser un tipo mejor que otro en términos absolutos:
    - La división da invariantes más concretos, de campos simples
    - El aplanado los da más generales, de un tipo de dato en toda la variable

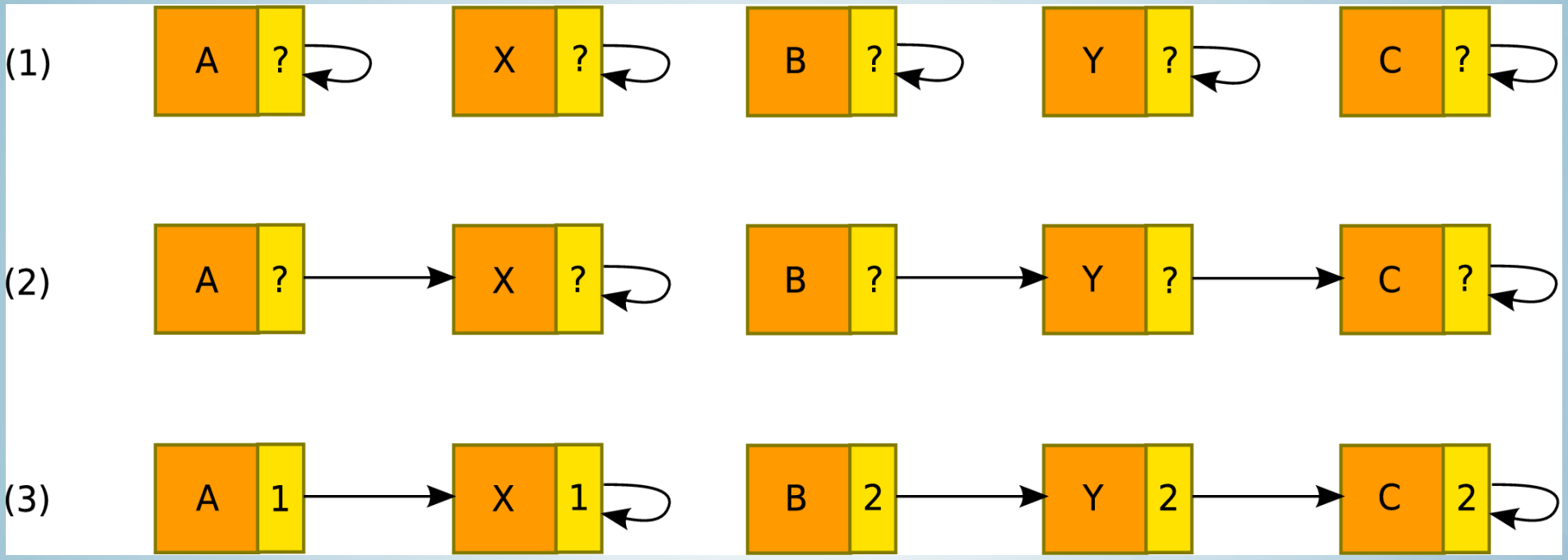
Optimizaciones en Takuan

- Takuan funciona correctamente, pero:
  - Puede tardar mucho tiempo en determinadas pruebas
  - Puede proporcionar mucha información redundante o no interesante (variables no relacionadas)
- Es necesario optimizarlo:
  - Comparabilidad entre variables: relacionar sólo variables del mismo tipo abstracto
  - No inferir invariantes en el XML Schema

Comparabilidad en Takuan

- Daikon admite atributos en las variables que las clasifiquen en “tipos abstractos”
  - Ejemplo: X = A *and* Y = B + C
  - Dos tipos: uno para X y A, otro para Y, B y C
- Pero esa información sólo se puede obtener de un proceso WS-BPEL tras su ejecución:
  - Hay que hacer un análisis con grafos
  - Información muy interesante porque en WS-BPEL hay “variables globales” (filtrado de no usadas)

Comparabilidad en Takuan



- Grafo para *if (A > X) or (B + Y = C)*

Restricciones XML Schema

- Hay que evitar que Daikon genere invariantes incluidos en el XML Schema. Ejemplos:
  - Longitudes máximas y mínimas de vectores
  - Valores máximos o mínimos de variables
- Para ello modificamos Daikon y le indicamos que no compruebe determinadas restricciones
  - Problema: no siempre el XML Schema es todo lo estricto que debiera

Resultados (división)

Técnicas	P.P.	Var.	Mem.	Time	Invs
-	64	17.404	656	409	30.399
XML (X)	64	17.404	646	400	21.793
Comp (C)	48	14.148	561	416	27.089
CX	48	14.148	579	401	18.358
CFiltro (F)	48	1.398	25	72	2.135
CXF	48	1.398	24	75	1.559

Resultados (aplanado)

Técnicas	P.P.	Var.	Mem.	Time	Invs
-	64	11.412	291	162	18.658
XML (X)	64	11.412	280	173	18.654
Comp (C)	48	9.036	261	179	16.718
CX	48	9.036	264	163	16.714
CFiltro (F)	48	710	11	52	942
CXF	48	710	12	55	940

Análisis de resultados

- La comparabilidad siempre interesa
- En división, el porcentaje de invariantes en el XML Schema es significativo
  - No así en aplanado
- El filtrado de variables es interesante
  - Pero puede ser agresivo

Experimentos

- Takuan funciona bien si (y sólo si) tiene un buen conjunto de casos de prueba
  - ¿Cuándo es bueno un conjunto?
- Dos aproximaciones:
  - Cuantitativa: ¿cuántos casos de prueba aleatorios hacen falta?
  - Cualitativa: ¿qué características pueden tener unos conjuntos para ser mejores que otros de igual tamaño?

Estudio de casos de prueba

- Objetivo del experimento:
  - ¿Qué influencia tiene el número de casos de prueba de entrada en la salida de Takuan?
  - Más casos nunca empeorarán los resultados, pero necesitan tiempo ¿hay algún límite?
  - ¿Hay algún mínimo fiable?
- Estudiamos dos composiciones:
  - Préstamo bancario del estándar “linealizado”
  - Mercado compraventa (Marketplace - ActiveBPEL)

Experimento

- Se crea un conjunto de 5 casos aleatorios:
  - Se ejecuta Takuan y se almacenan resultados
- Se añaden otros 5 casos aleatorios al conjunto
  - Se ejecuta Takuan y se almacenan resultados
- Se repite el proceso con conjuntos de 20, 50, 100 y 200 casos.
- Se analizan los invariantes distintos, dónde se encuentran esas diferencias y el tiempo de ejecución

Resultados préstamo

Número de casos	5	10	20	50	100	200
Total de invariantes	106	164	160	163	167	167
Invariantes interesantes distintos	116	45	18	4	0	
Invariantes (incl. no inter.) distintos	124	62	38	14	10	
Total de puntos de programa (TPP)	8	10	10	10	10	10
TPP con invariantes interesantes distintos	10	7	7	4	0	
TPP con invariantes distintos (incl. no inter.)	10	8	8	6	4	

Resultados mercado

Número de casos	5	10	20	50	100
Total de invariantes	8	8	6	6	6
Invariantes interesantes distintos	6	2	0	0	
Invariantes (incl. no inter.) distintos	14	14	8	8	
Total de puntos de programa (TPP)	3	3	3	3	3
TPP con invariantes interesantes distintos	2	2	0	0	
TPP con invariantes distintos (incl. no inter.)	2	2	2	2	

Tiempos

Número de casos (préstamo)	5	10	20	50	100	200
Tiempo total	22	57	47	1:14	2:07	3:00
Tiempo CPU usuario	6	8	8	12	19	24
Tiempo CPU sistema	10	14	15	21	26	46

Número de casos (mercado)	5	10	20	50	100
Tiempo total	20	32	54	2:09	3:30
Tiempo CPU usuario	3	3	4	8	12
Tiempo CPU sistema	8	10	12	21	32

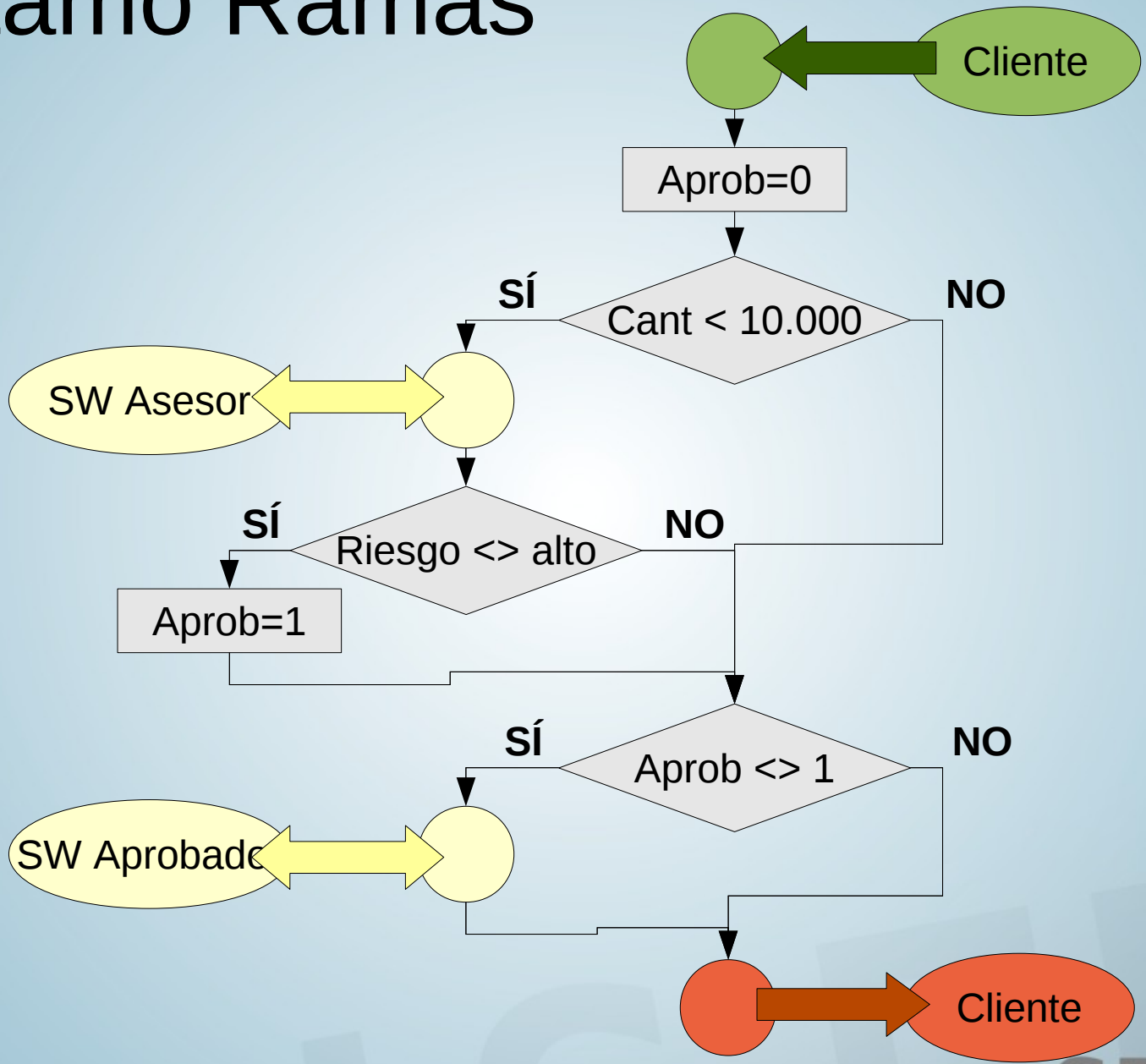
Análisis de resultados

- Analizando invariantes
  - Es necesario un “mínimo” de casos para fiarnos: aprox. 10 para el mercado y 50 para préstamo
  - Muy pocos casos dejan puntos sin invariantes
  - Existe un “tope” de eficiencia: aprox. 20 para el mercado y 50-100 para préstamo
  - Para “no interesantes” algo más
- Tiempos “significativos” para compo. pequeñas
  - En el mercado hay bastantes tiempos de espera

Estudio por coberturas

- Objetivo del experimento:
  - ¿Qué influencia tiene la cobertura del conjunto de casos de prueba de entrada en la salida?
- Se busca una composición con conjuntos de entrada con coberturas independientes:
  - Cobertura de instrucciones
  - Cobertura de ramas
  - Cobertura de caminos
- Modificamos el ejemplo del préstamo

Préstamo Ramas



Experimento

- Se definen cinco conjuntos de 15 casos:
  - Aleatorio (ninguna cobertura)
  - Cobertura de instrucciones (x2)
  - Cobertura de ramas
  - Cobertura de caminos
- Se ejecutan en Takuan y se comparan con la coberturas más completa (caminos)
  - Invariantes distintos
  - Dónde se encuentran esas diferencias

Resultados

Cobertura	A.	In1	In2	Ra	Ca
Total de invariantes	67	63	63	64	66
Invariantes interesantes distintos	7	5	3	2	
Invariantes (incl. no inter.) distintos	13	7	5	4	
Total de puntos de programa (TPP)	7	7	7	7	7
TPP con invariantes interesantes distintos	4	1	3	1	
TPP con invariantes distintos (incl. no inter.)	4	2	4	2	

Análisis de los resultados

- Número de invariantes similares
  - Pero existen bastantes diferencias entre ellos
  - Las diferencias se reparten por varios puntos
- Se confirma que a mejor cobertura, mejores invariantes se obtienen
- Poca diferencia entre ramas y caminos
  - Nos podemos “conformar” con ramas
- Tiempos de ejecución similares
  - La calidad “no penaliza”

Trabajo actual

- ¿Cómo demostrar que los invariantes que saca Takuan “descubren” fallos en una compos.?
- Propuesta:
  - Crear un conjunto de prueba fiable (muy grande)
  - Generar invariantes
  - Mutar la composición
  - Generar invariantes de los mutantes
  - Comparar invariantes obtenidos
- Buenos resultados

Líneas futuras

- Hasta aquí hemos llegado, pero hay más ideas:
  - Pruebas no funcionales: tiempo, costes
  - Caracterización más fina del tamaño de conjunto aleatorio necesario para sacar invariantes
  - Investigación sobre los mutantes no muertos: buscar invariantes que los maten
  - Analizar comportamiento de Takuan ante determinadas situaciones y/o instrucciones
  - Pasar la salida de Takuan a otros formatos (XML) y buscarle utilidad (incluirlo en el WS-BPEL, en una metodología SOA, …)

¿Es difícil trabajar en Takuan?

- Hasta ahora hemos trabajado:
  - Manuel Palomo Duarte: tesis (en breve ;-)
  - Antonio García Domínguez: arquitectura y primeros desarrollos
  - Alejandro Álvarez Ayllón: desarrollos posteriores (cobertura, etc). PFC de Ingeniería en Informática
  - Javier Santacruz Lopez-Cepero: framework de trabajo (pruebas masivas, bibliotecas de experimentos, comparativas, etc). PFC de ITIS
  - ¿Quién quiere ser el siguiente?

¿Y con esto se publica?En 2 años:

SPI&FM
Software Process Improvement and Formal Methods group

Gracias por su atención

¿Preguntas?

http://neptuno.uca.es/~takuan

Trabajo financiado por el Programa Nacional de I+D+I del Ministerio de Educación y Ciencia y fondos FEDER mediante el proyecto SOAQSim (TIN2007-67843-C06-04)

SPI&FM
Software Process Improvement and Formal Methods group