# *Localización de Errores Dirigida por la Arquitectura en Sistemas Basados en Eventos*

*Alvaro Soria* [1,2]
[1] *ISISTAN, Facultad de Ciencias Exactas*
*Universidad Nacional del Centro de la Provincia de Buenos Aires*
*Campus Universitario, Paraje Arroyo Seco, B7001BBO Tandil – Argentina*
[2]*ANALYTE – Lab Technology Solutions, Buenos Aires - Argentina*
*Email: asoria@exa.unicen.edu.ar*

El crecimiento de la complejidad de las aplicaciones y con el objetivo de alcanzar la más alta calidad posible, los desarrolladores de software se encuentran continuamente buscando diferentes maneras y alternativas para mejorar la productividad de sus procesos de desarrollo.

La tendencia actual apunta hacia el desarrollo guiado por la arquitectura. Las arquitecturas describen la funcionalidad gruesa del sistema y expresan las principales decisiones de diseño en cuanto a atributos de calidad[Bass98]. Estos aspectos proveen un alto nivel de abstracción común, que la mayoría de las personas involucradas en el proceso de desarrollo puede utilizar como base para crear entendimiento mutuo, formar consenso y utilizarlas como medio de comunicación entre los diferentes participantes. Adicionalmente, los modelos arquitectónicos proporcionan un contexto de alto nivel para el análisis y evolución del sistema.

Básicamente, las arquitecturas de software involucran la descripción de los componentes que constituyen el sistema, las interacciones entre estos componentes, patrones que guían su composición y las restricciones sobre las formas de interacción y/o composición de los componentes. Estas descripciones permiten construir un modelo mental, relativamente manejable, de cómo el sistema esta estructurado y cómo interactúan sus componentes. De esta manera, se pueden manejar sistemas de mayor complejidad y tamaño sin necesidad de contar con un conocimiento detallado sobre cuestiones referidas a la implementación de los componentes tales como: diseño de algoritmos específicos, estructuras de datos, propiedades intrínsecas del lenguaje de programación, etc.

Debido a que el 80% de los costos, de un sistema típico ocurre luego de la instalación inicial, en la etapa de mantenimiento, otro aspecto importante provisto por la arquitectura, es la posibilidad de razonar a un nivel arquitectónico, brindando la información necesaria para la toma de decisiones y planeamiento acerca de los cambios a ser introducidos. Es decir, permite el conocimiento profundo de las relaciones, dependencias, performance y comportamientos de cada uno de los componentes del sistema de software.

Durante el ciclo de vida del software, la búsqueda, localización y reparación de fallas es una de las actividades mas tediosas y costosas en tiempo. Aún también, es una de las actividades de mayor importancia que determina la calidad del producto y el éxito del mismo. En la mayoría de las aplicaciones, este proceso consume más del 50% del total del tiempo y esfuerzo del proceso de desarrollo [Brooks95]. Además, es una de las áreas menos desarrollada y mas desafiantes del desarrollo de software. Un número especial de la revista "Communications of ACM" caracteriza el estado actual de las herramientas de debugging como "The Debugging Scandal" [Lieberman97].

En particular, en los Sistemas Basados en Eventos[Shaw96], los problemas asociados con la localización de los errores se ven magnificados por las características propias de este estilo arquitectónico. Esto se debe, a que los componentes abandonan el control sobre la computación

llevada a cabo por el sistema. Cuando un componente anuncia un evento, este no puede asumir que otros componentes responderán ante este anuncio. Mas aún, si este conoce que otros componentes están interesados en los eventos que este anuncia, no puede confiar en el orden en que estos componentes son invocados. Debido a la impredictibilidad a priori de la secuencia exacta de eventos, la tarea de predecir el comportamiento de este tipo de sistemas en tiempo de ejecución es compleja y produce costos significativos en las etapas de testing y debugging.

Generalmente, cuando se habla de encontrar un error o detectar la localización de una falla, los términos *error* y *falla* son usados indiferentemente. La definición de estos términos, provenientes de la disciplina de tolerancia a fallas, especificadas en el Glosario IEEE de Ingeniería de software es la siguiente [IEEE90]: **Mistake:** *una acción humana que produce un resultado incorrecto;* **Fault:** *un paso incorrecto, proceso o definición de datos;* **Failure:** *un resultado incorrecto;* **Error:** *la diferencia entre un valor o condición computado u observado o medido, y el valor o condición verdadero, especificado o teóricamente correcto*

Estos comportamientos incorrectos de los programas, son encontrados mediante el uso de técnicas de verificación o casos de test, como por ejemplo Model Cheking[Clarke94] . Actualmente la mayoría de los enfoques han volcado sus esfuerzos hacia aspectos teóricos, metodologías y algoritmos para testing automático. Sin embargo, a pesar de que la localización de fallas es una de las actividades que mayor parte de los recursos consume, especialmente en la etapa de mantenimiento del sistema donde los desarrolladores originales ya no se encuentran activamente involucrados, pocos resultados han sido publicados sobre el tema.

Varios enfoques que han sido introducidos en el pasado incluyen: program slicing [Weiser84], debugging algorítmico (algoritmic debugging) [Shapiro83], técnicas basadas en dependencias ( dependency-based techniques) [Jackson95, Korel88], métodos basados en probabilidades (probability-based methods) [Burnell95], entre otros. Estos enfoques tradicionales presentan ciertas limitaciones, o son específicos al lenguaje de programación, o usan algoritmos especializados, o requieren demasiada interacción explícita del usuario para la localización del error.

La necesidad de herramientas automáticas, que asistan en la localización de errores, es un área en actual desarrollo. La tendencia en la automatización de localización y reparación de fallas se encuentra basada en la aplicación de técnicas derivadas del diagnóstico basado en modelos (model-based diagnosis) [Reiter87]. El enfoque se basa en la disponibilidad de una representación lógica, es decir un modelo que describa en cierta manera funcionamiento correcto de un sistema dado. En base a la descripción de la estructura del sistema y las funciones de sus componentes, es posible realizar preguntas (queries) sobre un cierto conjunto de componentes, cuyo malfuncionamiento explica la falla del sistema. Estos conjuntos podrían ser vistos como “diagnósticos” del sistema. El diagnóstico basado en modelos ha sido principalmente usado para la detección de fallas en sistemas físicos. Su aplicación al dominio de debugging ha sido propuesto recientemente y probado en varias ocasiones (ver [Stumptner99,Bond94]).

Sin embargo, el mayor problema de las técnicas de debugging nombradas anteriormente es que se encuentran básicamente enfocadas en localizar errores a nivel de código fuente. Este tipo de debuggers, son fáciles de usar y es sencillo buscar problemas, pero cuando se tienen aplicaciones con millones de líneas de código, el proceso tiende a ser tedioso y consume gran cantidad de tiempo y recursos. Por otro lado, fuerza a los desarrolladores a trabajar a un bajo nivel de abstracción perdiendo el foco del problema global. Como consecuencia, cuando es momento de solucionar la falla, la solución se limita a resolver el punto particular en el código que la produce, sin poder aplicar dicha solución a todo el diseño. Por estas razones, esta forma de localización y reparación

de errores no resuelve completamente los problemas en la aplicación. Otro problema de este enfoque, es que el tiempo y ubicación entre la ocurrencia de la falla y su manifestación, puede ser distantes.

**Localización de Errores Guiada por la Arquitectura**

Resulta razonable pensar, en el contexto anteriormente mencionado, que dichos problemas pueden ser minimizados con un enfoque a nivel arquitectónico, que asista en la problemática de localización de errores, particularmente, en el contexto de Sistemas Basados en Eventos.

Un esquema general del enfoque se muestra en la siguiente figura :

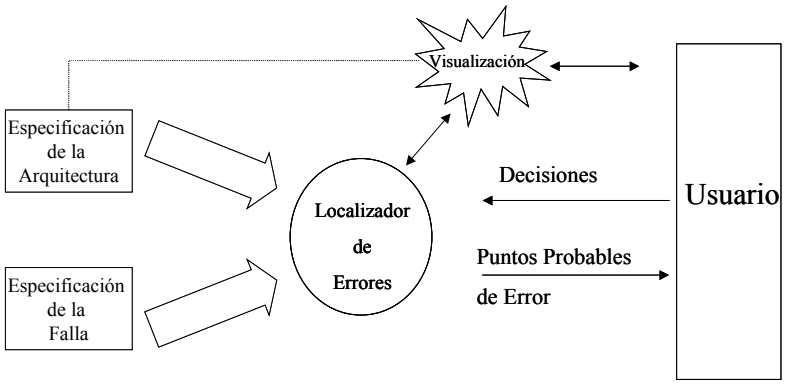


**Figura 1:** Esquema General del Enfoque

Conociendo la arquitectura a ser analizada, se puede generar un modelo a partir de la información que esta provee. Esta información sobre los componentes, cómo estos componentes se encuentran relacionados y las restricciones entre ellos, captura los aspectos estructurales del sistema.

Por otro lado, para el proceso de localización de errores es necesario capturar los aspectos comportamentales del sistema en relación con los componentes de la arquitectura. Estos aspectos pueden ser capturados por medio de la especificación de condiciones temporales, precondiciones y poscondiciones por cada componente para permitir la descripción de fallas. Un lenguaje que puede utilizarse para estas especificaciones es *Object-Z* [Duke95]. Este lenguaje provee un mecanismo para detallar cada uno de los componentes internamente. Otra característica de este tipo de especificación es la capacidad de descripción de las relaciones entre las funciones del sistema y los componentes que lo constituyen.

Con estas entradas, el localizador de errores puede comenzar el proceso de análisis que es el encargado de determinar cuales componentes son potencialmente responsables del malfuncionamiento del sistema. Sin embargo, los caminos posibles que deben ser explorados en el proceso de localización del error, pueden ser demasiados impactando considerablemente en los tiempos de respuesta.

Adicionalmente, el localizador puede incorporar información de ejecución del sistema que le permita construir un contexto mas cercano al problema específico. Esta información es tomada por el localizador y es utilizada para filtrar partes de los cursos posibles de exploración, descartando probables fallas que no tienen sentido, incluyendo dicha información de contexto.

En realidad, la problemática de la localización de errores en nuestro caso, surge de experiencias previas con aplicaciones de workflows basadas en el framework Bubble [Campo02]. Básicamente

un workflow es un proceso que contiene actividades y flujos entre dichas actividades. Un problema recurrente en este tipo de aplicaciones es el desvío erróneo del flujo de trabajo pasando al estado activo actividades que no deberían estar en dicho estado en un momento determinado.

Por ejemplo, supongamos que la finalización de una actividad implica la ejecución excluyente de dos actividades, pero luego su finalización la actividad activa no es la esperada. En este punto, comienza el razonamiento acerca de las posibles causas que produjeron el malfuncionamiento mencionado. Las posibles causas en este contexto serían las siguientes: las condiciones de los flujos se encuentran mal especificadas; o la finalización de la actividad inicial produjo un mal funcionamiento que derivó en la falla; u otro proceso interfirió entre la finalización de la actividad y la evaluación de los flujo modificando propiedades que produjo dicha falla. Estas causas pueden ser provistas por una herramienta que indique al usuario los cursos de acción a tomar en pos de detectar la causa de falla. Una vez elegida la(s) posible(s) causa(s), con información de la arquitectura y la especificación del workflow se puede generar una secuencia acotada de acciones a tomar, filtradas de todo el espacio de búsqueda posible.

Este proyecto se propone utilizar como enfoque principal, el uso de la información provista por el modelo arquitectónico conjuntamente con técnicas de diagnóstico basadas en modelos [Reiter87,Kleer87] para proveer una guía en la localización de errores, a un mayor nivel de abstracción que las herramientas actuales, mejorando la calidad de las soluciones y disminuyendo los tiempos gastados en este proceso.

Actualmente, se están analizando diferentes modelos que permitan una mayor efectividad del razonador. A partir de la especificación arquitectónica y su descripción en el lenguaje *Object-Z* [Duke95], se pueden derivar diferentes modelos lógicos. Estos modelos lógicos son utilizados para razonar acerca de las posibles causas que producen la falla. Utilizando estos modelos como base, se están realizando diferentes investigaciones en la aplicación de técnicas de diagnóstico basadas en modelos (Model-Based Diagnosis[Reiter87]) para la localización de errores a partir de la información arquitectónica, que permita identificar un conjunto mínimo de componentes que expliquen el mal funcionamiento del sistema.

Con este enfoque, se pretende lograr las siguientes mejoras en el proceso de localización de errores: (1)Reducción de los esfuerzos asociados a las actividades de localización de errores en Sistemas Basados en Eventos; (2) Provisión de un soporte automatizable para el enfoque;(3) Elevar el nivel de abstracción de las tareas de localización de errores a fin de apuntar a la detección de errores de diseño mas que a errores puntuales de código.

## 1. REFERENCIAS


[Bass98] L. Bass, P. Clements, R. Kazman .*Software Architecture in Practice.* Addison Wesley,Inc . 1998

[Bond94] G. W. Bond. *Logic Programs for Consistency-Based Diagnosis.* PhD Tesis. Canadá 1994.

[Brooks95] F. Brooks. *The Mythical Man-Month, 2nd edition.* Addison-Wesley. 1995

[Burnell95] L. Burnell y E. Horvitz, .*Structure and chance: Melding logic and probability for software debugging* Communications de ACM, Vol 38 No 3 ,31-41,1995.

[Campo02] M Campo, A. Diaz Pace y M. Zito. *Developing Object-Oriented Enterprise Quality Frameworks using Proto-frameworks*. Software Practice and Experience. 32:1-7, Julio 2002.

[Clarke 94] E M. Clarke., O. Grumberg, y D. E. Long.. *Model Checking and Abstraction*. ACM Transactions on Database Systems, 16(5):1512-1542, Sept. 1994.

[Duke95] R. Duke, G. Rose y G. Smith. *Object-Z: A Specification language advocated for description of Standards.* Computer Standards & Interfaces, 17:511-533, 1995.

[Lieberman97] H. Lieberman. *The Debugging Scandal* - Communications of the ACM, Vol 34 No 4, abril 1997.

[IEEE90] IEEE. *IEEE Standard Glossary of Software Engineering Terminology(IEEE Std. 610,12-1990)*, Technical Report, IEEE 1990.

[Jackson95] D. Jackson. *Aspect: Detecting Bugs with Abstract Dependences.*ACM TOSEM, 4(2):109-145, Abril 1995.

[Korel88] B. Korel. *PELAS-Program Error-Locating Assistant System.*IEEE TSE, 14(9):1253-1260,1988.

[Reiter87] R. Reiter. *A Theory of diagnosis from first principles.* En Artificial Intelligence, 32(1):57-95, 1987.

[Shaw96] M. Shaw y D. Garlan . *Software Architecture, perspectives on an emerging discipline*, Chapter 2: Architectural Styles, Prentice-Hall, 1996

[Shapiro83] E. Shapiro. *Algorithmic Program Debugging.* MIT Press, Massachusetts, 1983.

[Stumptner99] M. Stumptner y F. Wotawa. *Debugging Functional Programs.* En Proc. de 16th Inter. Joint Conf.On Artificial Intelligence, pág. 1074-1079, Suecia 1999.

[Weiser84] M. Weiser. *Program Slicing*. IEEE Transactions on Software Engineering. 10(4):352-357. Julio 1984.