DEPURADOR DECLARATIVO SOBRE ECLIPSE PARA PROGRAMADORES ERLANG 2.0

RUBÉN MUÑOZ GÓMEZ

MÁSTER EN INVESTIGACIÓN EN INFORMÁTICA, FACULTAD DE INFORMÁTICA,
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin Máster en Ingeniería Informática

Madrid, Julio de 2017

Directores:

Adrián Riesco Rodríguez / Salvador Tamarit Muñoz

Autorización de Difusión

RUBÉN MUÑOZ GÓMEZ

Madrid, Julio de 2017

El abajo firmante, matriculado en el Máster en Ingeniería Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "DEPURADOR DECLARATIVO SOBRE ECLIPSE PARA PROGRAMADORES ERLANG 2.0", realizado durante el curso académico 2015-2016 bajo la dirección de Adrián Riesco Rodríguez y Salvador Tamarit Muñoz; en el Departamento de Sistemas Informáticos y Computación, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Resumen en castellano

El presente trabajo tiene como finalidad la ampliación de la funcionalidad ofrecida por el *plugin* de Eclipse conocido como E-EDD (Eclipse - Erlang Declarative Debugger), el cual fue desarrollado por Joel Sánchez en su trabajo fin de Máster en el curso académico 2014/2015. El proyecto E-EDD proporciona un interfaz gráfico al depurador declarativo de línea de comandos para Erlang, denominado EDD (Erlang Declarative Debugger), permitiendo al usuario depurar módulos Erlang aplicando depuración declarativa con todas las facilidades que ofrece el entorno de desarrollo Eclipse mientras le muestra las estructuras de datos subyacentes a la ejecución y enlaza el proceso de ejecución con el código original.

Este trabajo tiene como objetivo mejorar la funcionalidad y la usabilidad de este *plugin* añadiéndole más interactividad con el usuario, con lo que conseguimos agilizar y facilitar la realización del proceso original de depuración; y al introducir una nueva función por la cual el usuario podrá realizar una depuración declarativa sobre programas Erlang concurrentes, esto es especialmente útil, debido a que la concurrencia es una de las características más importantes de Erlang.

Palabras clave

Eclipse, Erlang, Depuración Declarativa, E-EDD, EDD, plugin

Resumen en inglés

The finality of this project is to improve the functionality of Eclipse's plugin E-EDD (Eclipse - Erlang Declarative Debugger), that was developed by Joel Sánchez in his Master's thesis in 2014/2015. E-EDD project provides a graphic interface for command line declarative debugger for Erlang, EDD (Erlang Declarative Debugger), that allowing the user to debug Erlang modules by applying declarative debugging with all the features offered by the Eclipse framework, while is showing the data structures of the debugging execution and linking the execution process to the original code.

This project has the objective to improve the functionality and usability of this plugin adding more interactivity with the user, with that we managed to speed up and facility the execution of the original debugging process; and introducing a new function with the user can execute a declarative debugging process for concurrent Erlang programs, this is very useful because the concurrency is one of the most important Erlang features.

Keywords

Eclipse, Erlang, Declarative Debugging, E-EDD, EDD, plugin

Índice de contenidos

Agradecimientos

En primer lugar, me gustaría dar las gracias a mi familia quienes siempre han estado dándome su apoyo y animándome a continuar.

También me gustaría agradecer tanto a Salvador Tamarit como Adrián Riesco, mis tutores, por el esfuerzo y dedicación que me han ofrecido durante todo el desarrollo del presente trabajo, y sin los cuales no hubiera sido posible el presente proyecto.

Por último, me gustaría agradecer a todos los profesores que me han enseñado todo lo que sé, y a todas las personas que me han apoyado en este proceso.

1 Introducción

El presente trabajo tiene como finalidad la ampliación de la funcionalidad ofrecida por el *plugin* de Eclipse [1] conocido como E-EDD (Eclipse - Erlang Declarative Debugger) [14], que fue desarrollado por Joel Sánchez en su trabajo fin de Máster [D1] en el curso académico 2014/2015. Este *plugin* proporciona un interfaz gráfico al depurador declarativo de línea de comandos para Erlang, denominado EDD [13] explicado en [3.4 EDD (Erlang Declarative Debugger)]. E-EDD permite al usuario depurar módulos Erlang aplicando depuración declarativa con todas las facilidades que ofrece el entorno de desarrollo Eclipse mientras le muestra las estructuras de datos subyacentes a la ejecución y enlaza el proceso de ejecución con el código original.

Durante el presente documento empezaremos viendo en el punto [3. Conceptos] el concepto de depuración declarativa, junto con una explicación del lenguaje de programación Erlang, el entono de desarrollo Eclipse y el depurador declarativo EDD.

Tras esto veremos la funcionalidad del *plugin* E-EDD que se explicará a lo largo del presente documento, primero en el punto [4 E-EDD (Eclipse - Erlang Declarative Debugger)] se explicará el funcionamiento original de la herramienta creada por Joel Sánchez. Y en el siguiente apartado [5 E-EDD 2.0: Nueva funcionalidad] se explicarán los añadidos realizados sobre este en el actual proyecto fin de Máster, esta ampliación del *plugin* consta de dos partes:

- El desarrollo de la mejora el proceso de depuración de tal manera que el usuario pueda interactuar con la aplicación para que conseguir con ello que pueda responder directamente a cualquier pregunta realizada por el depurador, sin tener así que esperar a que este se posicione sobre esta, consiguiendo así que el proceso de depuración se realice de manera más rápida y eficaz. Esta funcionalidad se explica con más detalle en el apartado [5.1. Árbol de Operaciones].
- El desarrollo de las operativas y el interfaz gráfico necesarios para permitir que el usuario a través del entorno generado por el *plugin* pueda realizar un proceso de depuración de aplicaciones Erlang concurrentes, teniendo esto mucha importancia ya que la concurrencia es uno de los puntos claves de Erlang. Esta funcionalidad se explica con más detalle en el apartado [5.2. Depuración Concurrente].

Tras ver la funcionalidad del *plugin* explicaremos en el punto [6 Manual de uso] cómo utilizar el plugin la herramienta, tras lo que dejaremos paso a las conclusiones y las posibles mejoras que se pueden realizar sobre el proyecto en el punto [7 Conclusiones].

Por último, en los anexos podremos ver la información técnica del proyecto como son las principales clases de las que está formado [Anexo A - Clases], la descripción técnica de los procesos que se realizan dentro del *plugin* [Anexo B - Descripción técnica de los procesos], la comunicación que se realiza entre el *plugin* y el depurador declarativo EDD [Anexo C – Comunicación Java-Erlang] y el manual de instalación del plugin y sus prerrequisitos [Anexo D – Guía de Instalación].

1.1 Motivación

A la hora de empezar el proyecto Fin de Máster realicé varias búsquedas de información e ideas entre los diferentes profesores disponibles, teniendo siempre en cuenta que mi idea era realizar un trabajo enfocado en desarrollo de Software que me sirviera para utilizar los conocimientos adquiridos durante el Máster, así como para ampliar mis conocimientos.

Durante la búsqueda de información para el proyecto hablé con el Dr. Adrián Riesco, a quién conocía con anterioridad por haberme impartido clase en la asignatura de *Auditoría, calidad y fiabilidad informáticas* en la Facultad de Informática de la Universidad Complutense de Madrid para el Master en Ingeniería del Software. En dicha conversación, me comento la idea de realizar una ampliación del *plugin* de Eclipse E-EDD, desarrollado anteriormente por Joel Sánchez en su trabajo fin de Máster en el curso académico 2014/2015.

Después de hablar con varios profesores de la facultad para realizar el proyecto fin de Máster, me decanté por esta opción, debido a que este proyecto cumplía con mis expectativas, ya que está enfocado en el desarrollo Software y me permite poner en práctica y ampliar mis conocimientos adquiridos, principalmente del lenguaje de programación Java con el que se realiza el desarrollo. Además, con este proyecto consigo adquirir nuevos conocimientos, que me pueden servir para el futuro, sobre el desarrollo de plugins en Eclipse, los lenguajes de programación declarativos como es Erlang, así como el método de depuración declarativa que realiza EDD, etc.

2 Introduction

The finality of this project is to improve the functionality of Eclipse's plugin [1] E-EDD (Eclipse - Erlang Declarative Debugger) [14], that was developed by Joel Sánchez in his Master's thesis [D1] in academic course 2014/2015. E-EDD provides a graphic interface for the command line declarative debugger for Erlang, EDD [13] (Erlang Declarative Debugger), explained in the Section [3.4 EDD (Erlang Declarative Debugger)]. E-EDD allow the user to debug Erlang modules by applying declarative debugging with all the features offered by the Eclipse framework, showing the data structures of the debugging execution and linking the execution process to the original code.

In the Section [3. Conceptos] we describe the declarative debugging, the programming language Erlang, the framework Eclipse, and the declarative debugger EDD.

After that we explain in the next sections the functionality of the plugin E-EDD. First in Section [4 E-EDD (Eclipse - Erlang Declarative Debugger)] we explain the original functionality of the plugin. In the next Section [5 E-EDD 2.0: Nueva funcionalidad] we present the improvements made it in this project, that is divided in two parts:

- The development of an improvement of the sequential debugging process, that allow the user interact with the questions tree of the plugin to answer to any question of the debugger in the order he wants, without waiting the debugger to select that question. With this functionality, the user can make the debugging process more quickly and efficiently. This functionality will be explained in the Section [5.1. Árbol de Operaciones].
- The development of the processes and the graphic interfaces that allow the user to perform declarative debugging in concurrent Erlang modules. This is very useful because the concurrency is one of the most important Erlang features. This functionality will be explained in the Section [5.2. Depuración Concurrente].

After seeing the functionality of the plugin, we will explain in Section [6 Manual de uso], how use the plugin. After that we will see the conclusions and future work of this project in Section [7 Conclusiones].

At last, in the annexes we will explain the technique information of the plugin, like the main classes that the project uses [Anexo A - Clases], the technical description of the process that compose the plugin [Anexo B - Descripción técnica de los procesos], the communication of the

plugin with the declarative debugger EDD [Anexo C – Comunicación Java-Erlang], and the plugin instalation guide with his requirements [Anexo D – Guía de Instalación].

2.1 Motivation

At the beginning of the Master's thesis, I searched for information between the different teachers of the faculty, with the idea of made a software development project where I could use the knowledge acquired in the Master, and to improve my knowledge about software development.

In the searching of information for the project I spoke with Dr. Adrián Riesco, who I met before to this reunion because he tought in the Master subject of *Auditoría, calidad y fiabilidad informáticas* in the computing faculty of the *Universidad Complutense de Madrid*. In this talk he gave me the idea to do the project about make an improvement of the Eclipse plugin E-EDD, which it was develop by Joel Sanchez in his Master thesis in the academic course 2014/2015.

After talk with a lot of teachers of the faculty to make my End of Master Project, I decided to choose this option, because this project meet my requirements, it is Software development project that allow me to use and improve my knowledge about Software development, mainly in the programming language of Java that is used to develop this plugin. Also, in this project I could acquire more knowledge, that can be useful in my future, about the plugin development in Eclipse, the declarative programming languages like Erlang, the method of declarative debugging using in EDD, etc.

3 Conceptos

En este apartado se explicarán los conceptos básicos para el correcto entendimiento del trabajo.

3.1 Erlang

Erlang [3, 18] es un lenguaje de programación funcional desarrollado en Ericsson Computer Science Laboratory en 1986, aunque en un principio era un lenguaje propietario en 1998 fue cedido como software de código abierto. El cual se caracteriza por:

- **Concurrencia:** permite la ejecución de docenas de procesos ligeros ejecutándose simultáneamente para la realización de una tarea, los cuales se comunicarán mediante paso de mensajes, con lo que se conseguirá agilizar la ejecución al poderse realizar varias tareas simultaneas en varios procesos que pueden realizar sus operaciones en procesadores distintos.
- **Programación distribuida:** permite la realización de programas que hacen uso de una red de ordenadores que entre ellos se reparten la ejecución de un programa Erlang. Con esto se conseguirá agilizar la ejecución al poderse realizar varias tareas simultaneas en diferentes ordenadores sin tener por lo tanto el cuello de botella que tiene la programación concurrente por el acceso a la memoria del ordenador.
- **Tolerancia a fallos:** sus aplicaciones pueden recuperarse de un error y seguir funcionando normalmente.
- **Tiempo real**: sus programas responden rápidamente, y su código puede ser actualizado en tiempo real, con lo que al realizar un cambio en el código mientras se está ejecutando el cambio se vería en la ejecución sin tener que reiniciar el proceso.

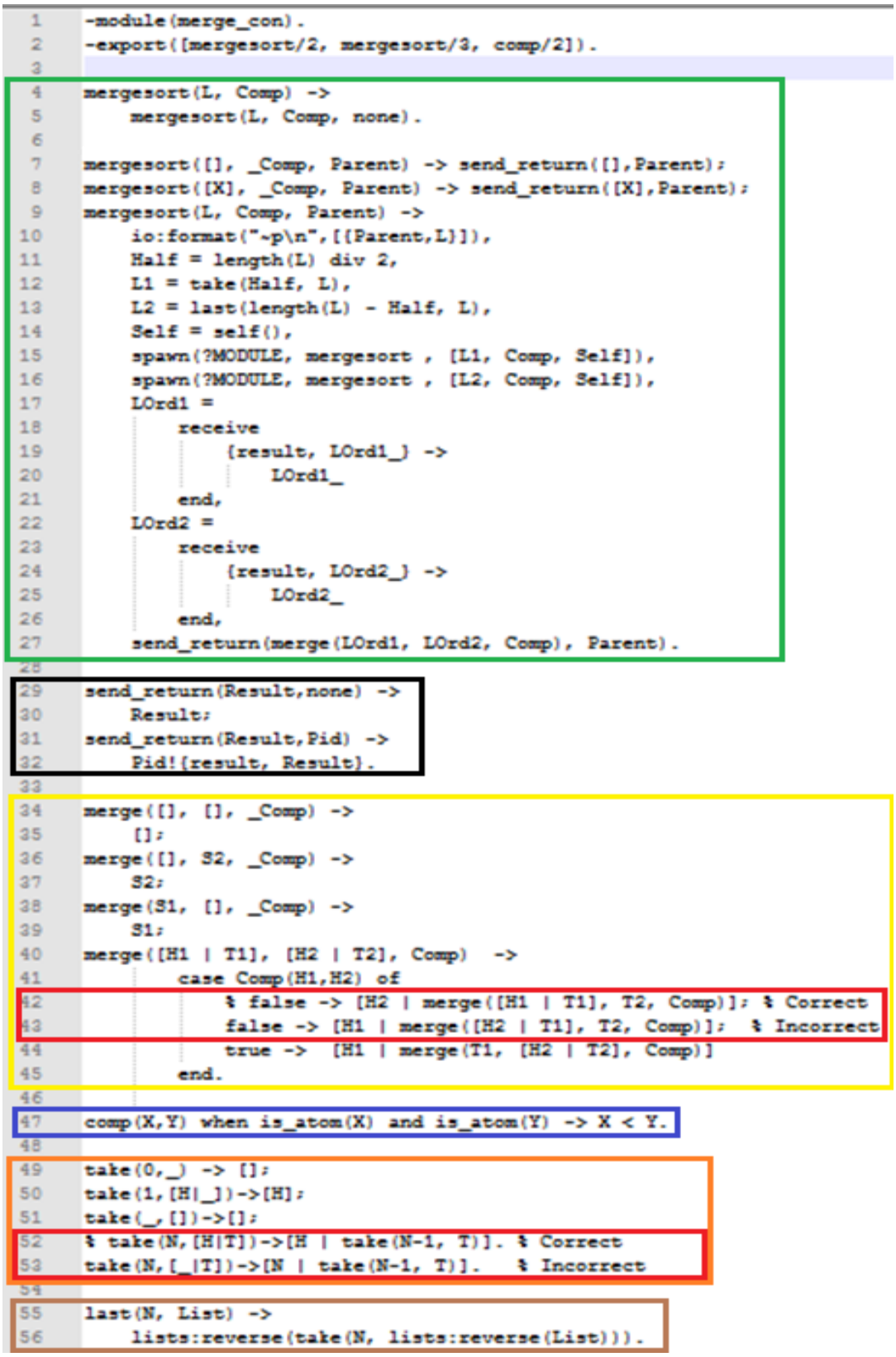
3.1.1 Ejemplo

A lo largo del presente documento se mostrarán varias imágenes del proceso de depuración con E-EDD, para realizar estos procesos de depuración utilizaremos el módulo Erlang **merge**, tanto en versión secuencial, cuyo código se puede observar en la *Ilustración 3-1*, como en concurrente, cuyo código se puede observar en la *Ilustración 3-1*. Estos módulos definen un método de ordenación de arrays por el algoritmo de “divide y vencerás” y cuyos códigos se

encuentran disponibles dentro de la carpeta Examples del EDD [13], estos módulos vienen con errores, para realizar el proceso de depuración, junto con su versión correcta comentada, estas líneas se pueden identificar fácilmente ya que tienen a su derecha un comentario indicando si son correctas o no.

```
1  -module(merge).
2  -export([mergesort/2, comp/2]).
3
4  mergesort([], _Comp) -> [];
5  mergesort([X], _Comp) -> [X];
6  mergesort(L, Comp) ->
7      Half = length(L) div 2,
8      L1 = take(Half, L),
9      L2 = last(length(L) - Half, L),
10     LOrd1 = mergesort(L1, Comp),
11     LOrd2 = mergesort(L2, Comp),
12     merge(LOrd1, LOrd2, Comp).
13
14 merge([], [], _Comp) ->
15     [];
16 merge([], S2, _Comp) ->
17     S2;
18 merge(S1, [], _Comp) ->
19     S1;
20 merge([H1 | T1], [H2 | T2], Comp)  ->
21         case Comp(H1,H2) of
22 %             false -> [H2 | merge([H1 | T1], T2, Comp)]; % Correct
23             false -> [H1 | merge([H2 | T1], T2, Comp)];  % Incorrect
24             true ->  [H1 | merge(T1, [H2 | T2], Comp)]
25         end.
26
27
28 comp(X,Y) when is_atom(X) and is_atom(Y) -> X < Y.
29
30
31 take(0,_) -> [];
32 take(1,[H|_])->[H];
33 take(_,[])->[];
34 % take(N,[H|T])->[H | take(N-1, T)]. % Correct
35 take(N,[_|T])->[N | take(N-1, T)].   % Incorrect
36
37 last(N, List) ->
38     lists:reverse(take(N, lists:reverse(List))).
```

Ilustración 3-1 Código del programa ***merge.erl***



```
1  -module(merge_con).
2  -export([mergesort/2, mergesort/3, comp/2]).
3
4  mergesort(L, Comp) ->
5      mergesort(L, Comp, none).
6
7  mergesort([], _Comp, Parent) -> send_return([],Parent);
8  mergesort([X], _Comp, Parent) -> send_return([X],Parent);
9  mergesort(L, Comp, Parent) ->
10     io:format("~p\n",[{Parent,L}]),
11     Half = length(L) div 2,
12     L1 = take(Half, L),
13     L2 = last(length(L) - Half, L),
14     Self = self(),
15     spawn(?MODULE, mergesort , [L1, Comp, Self]),
16     spawn(?MODULE, mergesort , [L2, Comp, Self]),
17     LOrd1 =
18         receive
19             {result, LOrd1_} ->
20                 LOrd1_
21         end,
22     LOrd2 =
23         receive
24             {result, LOrd2_} ->
25                 LOrd2_
26         end,
27     send_return(merge(LOrd1, LOrd2, Comp), Parent).
28
29 send_return(Result,none) ->
30     Result;
31 send_return(Result,Pid) ->
32     Pid!{result, Result}.
33
34 merge([], [], _Comp) ->
35     [];
36 merge([], S2, _Comp) ->
37     S2;
38 merge(S1, [], _Comp) ->
39     S1;
40 merge([H1 | T1], [H2 | T2], Comp)  ->
41         case Comp(H1,H2) of
42             % false -> [H2 | merge([H1 | T1], T2, Comp)]; % Correct
43             false -> [H1 | merge([H2 | T1], T2, Comp)];  % Incorrect
44             true ->  [H1 | merge(T1, [H2 | T2], Comp)]
45         end.
46
47 comp(X,Y) when is_atom(X) and is_atom(Y) -> X < Y.
48
49 take(0,_) -> [];
50 take(1,[H|_])->[H];
51 take(_,[])->[];
52 % take(N,[H|T])->[H | take(N-1, T)]. % Correct
53 take(N,[_|T])->[N | take(N-1, T)].   % Incorrect
54
55 last(N, List) ->
56     lists:reverse(take(N, lists:reverse(List))).
```

Ilustración 3-2 Código del programa ***merge_con.erl***

A continuación, explicaremos el código de Erlang del programa concurrente **merge_con**, el cual no difiere mucho de su versión secuencial.

La segunda línea nos indica los métodos a los que se puede llamar desde fuera del módulo. En nuestro caso se pueden utilizar los métodos **mergesort**, con dos y tres parámetros de entrada, y **comp** con dos parámetros de entrada.

A partir de la segunda línea se definen los diferentes métodos de los que está compuesto **merge_con**:

- **mergesort**: Este es el método principal de la clase que realiza la ordenación de los elementos que se pasan por parámetro. Su definición corresponde con el bloque verde.
    - El primer parámetro son los elementos que se van a enviar, el segundo es el método de comparación a utilizar el tercero es el identificador del proceso padre.
    - Este método tiene cuatro definiciones:
        - La primera definición que viene con dos parámetros, sirve para iniciar el proceso llamando al mismo método solo que añadiéndole el tercer parámetro el cual será un valor vacío.
        - La segunda definición describe las acciones a realizar cuando hay tres parámetros de entrada y el array de elementos del primer parámetro no tiene ningún elemento, el cual se comunicará al proceso padre.
        - La tercera define las acciones a tomar cuando el array de elementos solo tiene un único elemento, el cual se comunicará al proceso padre.
        - El cuarto define las tareas a realizar por el programa cuando hay varios elementos en el array.
            - Se divide el array por la mitad con los métodos **take** y **last**.
            - Se crean dos nuevos procesos con la palabra reservada "**spawn**", en el que pedimos que se realice el método **mergesort** sobre la mitad correspondiente del array creados en el paso anterior, utilizando el método de comparación indicado en el segundo parámetro y pasándose a sí mismo como tercer parámetro para que el nuevo proceso pueda comunicarse directamente con este.
            - Se recogen el resultado devuelto por los procesos creados en el paso anterior.
            - Devuelve el resultado de combinar y ordenar los dos arrays al proceso padre.

- **send_return**: Es el método encargado de la comunicación. Este método devolverá el valor pasado en el primer parámetro al proceso que está definido en el segundo parámetro, o lo mostrará al usuario en caso de que no haya ningún proceso definido. Su definición corresponde con el bloque negro.
- **merge**: Combina los arrays que se devuelven como primer y segundo parámetro, ordenándolos con el método de ordenación que se pasa en el tercer parámetro. Su definición corresponde con el bloque amarrillo.
- **take**: obtiene los primeros N elementos de un array. Siendo el primer parámetro el número de elementos a coger y el segundo la lista de elementos. Su definición corresponde con el bloque naranja.
- **last**: obtiene los últimos N elementos de un array. Siendo el primer parámetro el número de elementos a coger y el segundo la lista de elementos. Su definición corresponde con el bloque marrón.

La versión secuencial es idéntica a esta solo varía en el método **mergesort** en su cuarta definición, que en vez de crear un nuevo proceso que para hacer la siguiente llamada a **mergesort**, este módulo ejecuta directamente el método en el mismo proceso, por lo que no hace falta paso de mensajes y por lo tanto el método **send_return**.

Los errores que se han metido dentro del código son los mismos en ambas versiones y se corresponden con los bloques en rojo:

- En el método **merge** de las líneas 23 y 40 de los módulos **merge** y **merge_con** respectivamente. Esto es incorrecto debido a que no estamos ordenando en ningún momento el array puesto que siempre nos devolverá como resultado un array con la cabecera del primer array seguido de los valores del segundo array y terminando con los valores restantes del primer array, por ejemplo, para los arrays [a, b, c] y [d, e, f] nos devolvería el array [a, d, e, f, b, c], que obviamente no esta ordenado. Esto se debe a que devolvemos en el array como primer elemento el valor de la cabecera del primer array y ponemos la cabecera del segundo array como la cabecera del primer array a enviar a la siguiente llamada al método **merge**.
- En el método **take** de las líneas 35 y 53 de los módulos **merge** y **merge_con** respectivamente. Esta definición es incorrecta puesto que en vez de devolver los n primeros elementos del array nos devuelve un array con una secuencia numérica de

n a 2, y tras esto el elemento en la posición n, por ejemplo, si pedimos los tres primeros elementos del array [a, b, c, d, e, f, g] nos devolverá el array [3, 2, c]. Esto se debe a que se está poniendo en la cabecera el número de elementos a coger y no el primer elemento del array.

Una vez explicado el código procederemos a realizar un ejemplo de cómo funcionaría la ejecución de **merge_con:mergesort([b,a], fun merge_con:comp/2)**, que se usurará como ejemplo en las explicaciones de la herramienta.

Cuando ejecutamos un módulo Erlang este busca de arriba abajo por la primera ocurrencia que cumpla con lo pedido y ejecuta este código, en nuestro caso el programa encontrará esta ocurrencia en la línea 4, por lo que realizara el código que se encuentra dentro con lo que llamaremos al método **mergesort** con tres parámetros indicando como tercer parámetro el valor none.

En esta llamada obtendremos el valor de la mitad de la longitud del array y lo guardaremos en la variable **Half**, y con ello dividir por la mitad el array utilizando los métodos **take**, línea 50, y **last**, línea 55 que hará también una llamada a **take** con la definición de la línea 50, obteniendo con ello los arrays **L1=[b]** y **L2=[a]**. Tras esto crearemos dos nuevos procesos que ejecutarán la llamada al método **mergesort** pasándole a cada uno una de las mitades del array junto con la misma función de comparación y pasándole la id del proceso padre para que nos envíen sus resultados, los cuales esperaremos a obtener en las siguientes líneas metiéndolos en las variables **LOrd1=[b]** y **LOrd2=[a]**, en nuestro caso está llamada a **mergesort** irá por la definición de la línea 8 ya que el array sólo tiene un componente.

Una vez obtenidos los resultados se procederán a juntarlos de manera ordenada con el método **merge**, al que le pasaremos los dos arrays y el método de comparación. En nuestro caso iremos a la línea 40 donde realizaremos una comparación con nuestro método de comparación de los primeros valores de los arrays, y devolviendo en este caso falso, puesto que **b** es mayor que **a**, yendo por lo tanto a la línea 43, que es incorrecta, donde pondremos en la cabecera del array **b** devolver el valor del primer array y tras este elemento el resultado de la función merge con los elementos que quedan en los arrays, en nuestro caso este merge irá por la línea 28, y nos devolverá el valor **[a]** por lo que la salida final será el valor **[b,a]** y el cual será el resultado final del proceso**.** Este resultado es incorrecto puesto que el array no está ordenado.

3.2 Eclipse

Eclipse [1] es un entorno de desarrollo de código abierto desarrollado en Java [18] que permite y facilita a cualquiera de sus usuarios ampliar su funcionalidad a través de la instalación y el desarrollo de *plugins*, componentes software que añaden nuevas características a un programa, los cuales se añaden a través de los puntos de extensión.

Los principales componentes de Eclipse, los cuales se muestran en la *Ilustración 3-3*, son:

- **Plataform Runtime**: Es el encargado de gestionar y administrar los diversos *plugins* que se encuentran instalados.
- **WorkBench**: Implementa el interfaz gráfico de Eclipse, y es quien define los diferentes puntos de extensión para añadir componentes gráficos. El desarrollo de los componentes gráficos que se añaden al componente son desarrollados a través de los paquetes SWT [6] y Jface [5]. La información de todas las clases con las que desarrollar el apartado gráfico de Eclipse se pueden encontrar dentro de su API [2].
- **Workspace**: Encargado de gestionar los diversos recursos que son desarrollados por el usuario, como son los diferentes proyectos con sus clases, XML, imágenes, etc.
- **Help system**: Define los diferentes puntos de extensión a partir de los cuales se introduce la ayuda.
- **Team support**: Permite la funcionabilidad para el control de versiones, como son CVS (*Concurrent Versions System*) y SVN (*Subversion*), que facilitan el desarrollo de aplicaciones realizadas en equipo.
- **Debug Suport**: Define un modelo de depuración independiente del lenguaje y el interfaz gráfico para la realización de depuradores y lanzadores.

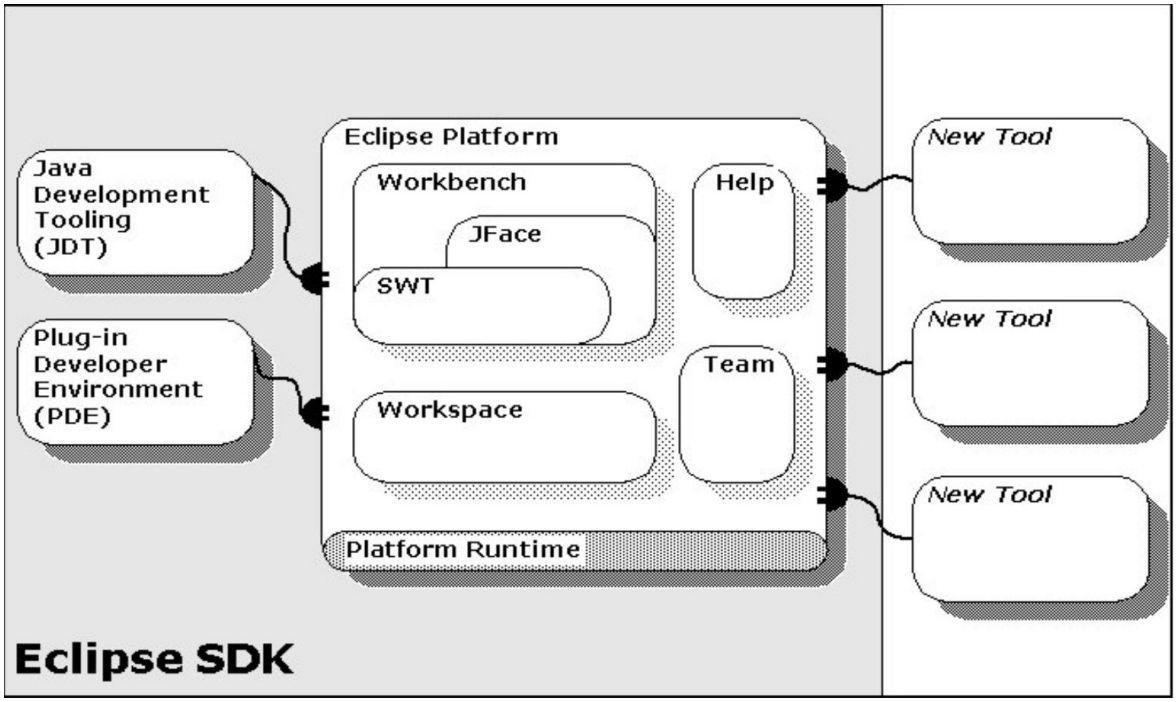


Ilustración 3-3 Arquitectura de Eclipse [2]

3.3 Depuración declarativa

La depuración declarativa [D5] es un tipo de depuración semi-automática, la cual a partir de una ejecución errónea genera un árbol de depuración, por el que guía al usuario, a través de una serie de preguntas sobre los resultados almacenados en los nodos respecto al esperado, hasta que se encuentre el error.

Si depuramos el programa merge.erl para la ejecución de "*merge:mergesort([b,a], fun merge:comp/2)*", la cual hemos visto con anterioridad en el apartado [3.1.1 Ejemplo], EDD creará el árbol de depuración que se observa en la *Ilustración 3-4*, este árbol está compuesto de varios nodos que se corresponden con las diferentes llamadas a métodos que se realiza en la ejecución junto a su resultado, siendo los nodos hijos métodos que se llaman dentro del método padre. En cada nodo se pregunta si el resultado es correcto para deducir dónde se localiza el error:

- Si un nodo está mal y sus hijos son correctos dentro de este método estará el error, este nodo incorrecto se denomina buggy.

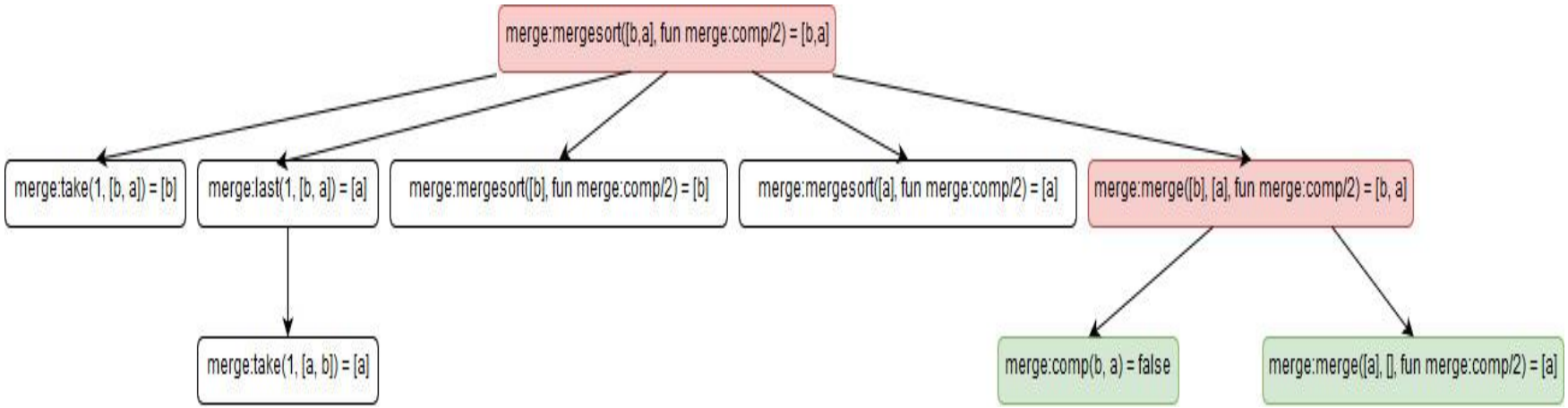


Ilustración 3-4 Árbol Declarativo generado por la depuración declarative de la ejecución de merge:mergesort([b,a], fun merge_con:comp/2)

3.4 EDD (Erlang Declarative Debugger)

EDD [13, D2, D3, D4] es un depurador declarativo en modo consola para programas Erlang, escrito en el mismo lenguaje, que soporta depuración de programas tanto secuenciales como concurrentes. Esta herramienta está a cargo de Rafael Caballero, Enrique Martin-Martin, Adrián Riesco y Salvador Tamarit.

3.4.1 Funcionamiento

En este punto explicaremos como funciona EDD, utilizando como ejemplo los módulos vistos en el apartado [3.1.1 Ejemplo], las llamadas usadas para realizar la depuración son las mismas en todo el documento y su proceso de depuración será explicado en el apartado [6 Manual de Uso].

Para poder hacer funcionar correctamente EDD se necesitará compilar todos los archivos Erlang que constituyen el proyecto. Una vez compilado se deberán seguir los siguientes pasos mostrados en la *Ilustración 3-5* para ejecutalo:

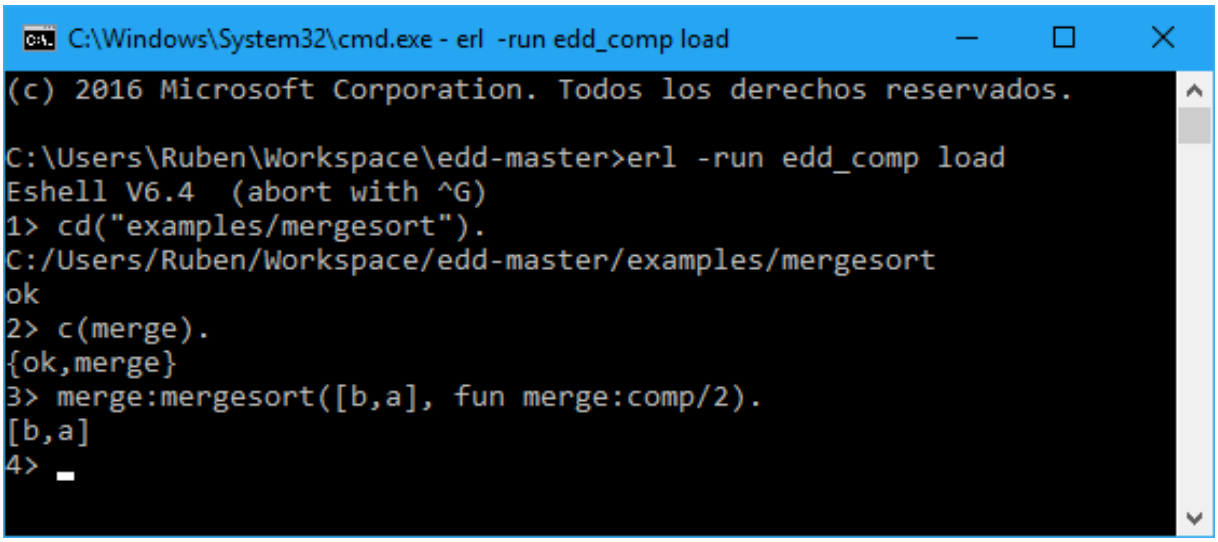


Ilustración 3-5 Ejemplo de compilación y ejecución de archivos en EDD

- Para ejecutar EDD tendremos que hacerlo desde la línea comandos, posicionándonos dentro de la carpeta del proyecto y en ella mandamos ejecutar el comando "**erl -run edd_comp load**".

- Tras esto nos tendremos que posicionar dentro de la carpeta donde se encuentra el programa Erlang que queremos depurar.
  Ej.: **cd("examples/mergesort").**
- Compilamos el archivo a depurar.
  Ej.: **c(merge_con).**
  Ej.: **c(merge).**
- Una vez hecho esto podremos ejecutar la función que se quiere probar para observar sus resultados.
  Ej.: **merge_con:mergesort([b,a], fun merge_con:comp/2).**
  Ej.: **merge:mergesort([b,a], fun merge:comp/2).**
- Para empezar la depuración se utilizarán distintas funciones dependiendo de si se quiere realizar una depuración secuencial o concurrente, las cuales veremos en los apartados siguientes.

3.4.1.1 Depuración Secuencial

Para empezar la depuración secuencial se utilizará la función "**edd:dd**" en la que se pasa como primer parámetro la función que se quiere depurar.

Ej.: **edd:dd("merge:mergesort([b,a], fun merge:comp/2)", tree).**

Tras esto el depurador nos irá realizando una serie de preguntas que habrá que responder hasta encontrar el error, estas preguntas se corresponden con el recuadro de color verde de la *Ilustración 3-6,* siendo la parte entre corchetes las posibles respuestas a dar y la letra al final de las preguntas la respuesta dada. En esta imagen vemos todo el proceso básico de la depuración declarativa hasta llegar a la cláusula problemática tras responder una serie de preguntas, a las cuales se responderán con los siguientes valores:

- **Yes (y):** La evaluación es correcta.
- **No (n)**: La evaluación es incorrecta.
- **Trusted (t):** La evaluación es correcta y la función es confiable, por lo cual todas las ejecuciones de esta función serán correctas.
- **No + Value (v)**: La evaluación no es correcta y el usuario aporta el valor esperado.

- **Don't know (d):** El usuario no sabe cómo responder a la pregunta.
- **Inadmissible (i):** La función no puede llevar esos parámetros.
- **Undo (u):** Se vuelve a la pregunta anterior.
- **Abort (a)**: Se termina el proceso de depuración.

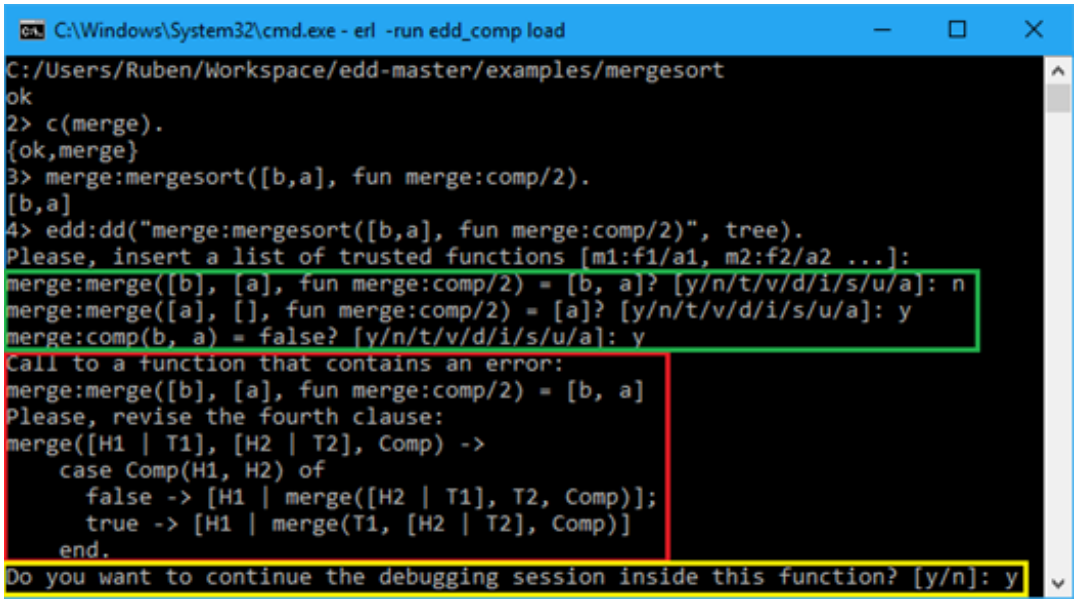


Ilustración 3-6 Ejemplo del proceso de depuración declarativa secuencial con EDD

Una vez que el depurador haya encontrado el origen del error nos informará como vemos en el recuadro de color rojo de la *Ilustración 3-6,* tras lo que nos preguntará si queremos realizar una depuración más exhaustiva, como podemos ver en el recuadro amarrillo de la misma imagen. Si respondemos que no se terminará la ejecución, y si se responde *sí* se continuará con una nueva serie de preguntas en caso afirmativo hasta encontrar la fuente del error, la cual se nos mostrará e informará cuando se encuentre, terminando ahí el proceso de depuración.

Todo este proceso de depuración por zoom lo podemos observar en la *Ilustración 3-7.* Una vez que respondemos que deseamos continuar se nos mostrará la información generada por EDD para la depuración en un árbol, parte de este árbol lo podemos observar al principio de la Ilustración 3-7 no estando este entero debido a su inmensa amplitud.

Tras generar sus datos se nos realizará una serie de preguntas, que corresponden con el recuadro verde de la *Ilustración 3-7*, donde las seis primeras líneas corresponden con la pregunta

que se desea realizar y las siguientes líneas con las posibles respuestas que se pueden dar a la pregunta siendo la última la respuesta dada a esta.

Una vez respondidas las preguntas necesarias el depurador llegará a la fuente del error y se nos informará de esto terminando con ello el proceso de depuración, esto lo podemos observar en el recuadro rojo de la *Ilustración 3-7*.

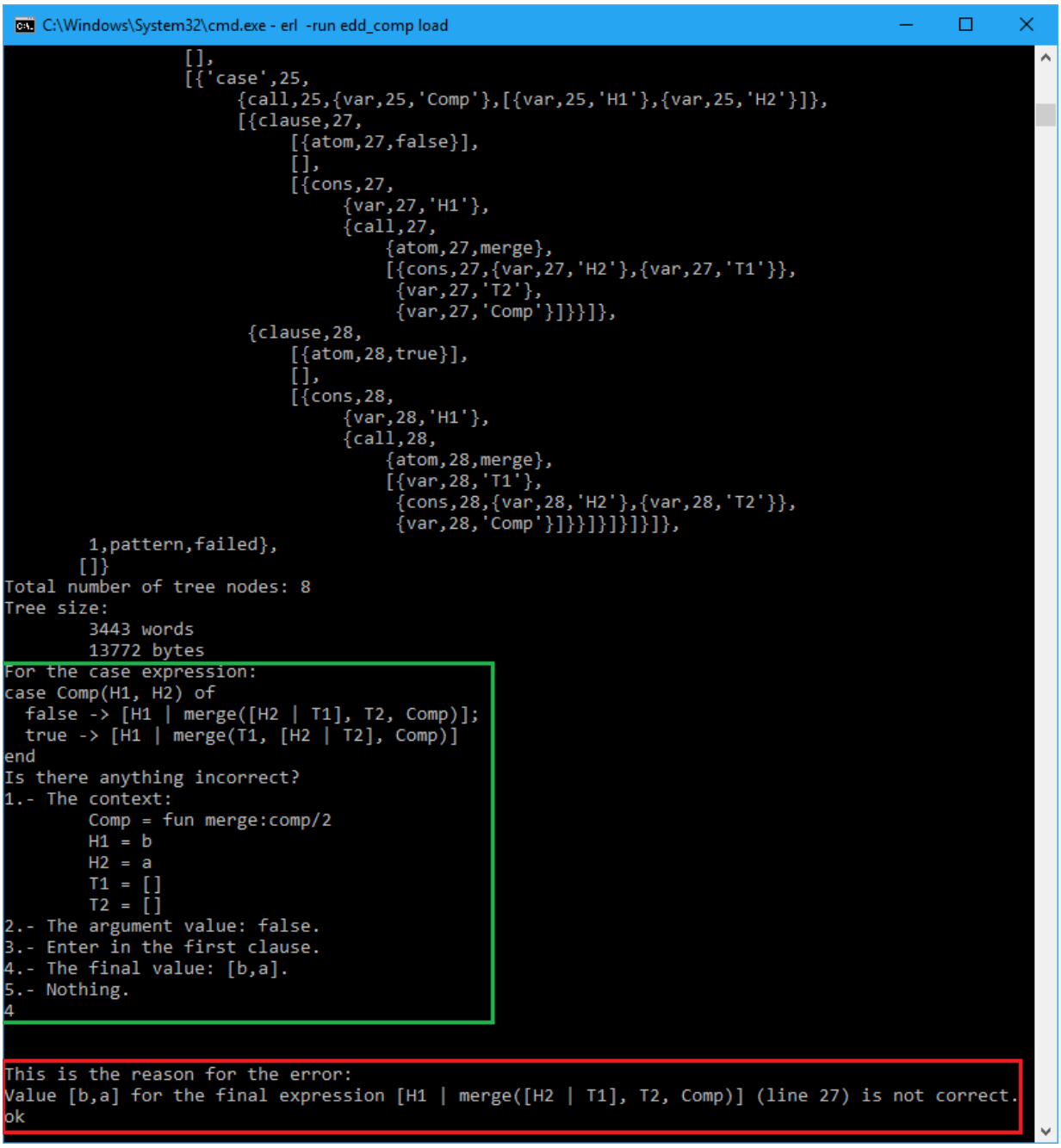


Ilustración 3-7 Ejemplo del proceso de depuración declarativa secuencial exahustiva con EDD

3.4.1.2 Depuración Concurrente

Para empezar la depuración concurrente se utilizará la función "**edd:cdd**" en la que se pasa como primer parámetro la función que se quiere depurar y como segundo el tiempo máximo en segundos que tarda la ejecución de la función.

Ej: **edd:cdd("merge_con:mergesort([b,a], fun merge_con:comp/2)", 100000).**

Una vez que realicemos la llamada al proceso de depuración concurrente se nos mostrará la información sobre la ejecución de la llamada a depurar, la cual podemos observar en los distintos bloques de la *Ilustración 3-8*:

- En el recuadro verde podemos ver la llamada que inicia el proceso depuración junto con el resultado devuelto.
- En el recuadro amarillo se nos muestra la información de los diferentes procesos creados durante la ejecución, esta se conforma del pid del proceso, la llamada con la que se generó el proceso y el resultado devuelto.
- En el recuadro azul vemos como el depurador nos pregunta por el punto por el que pensamos que se encuentra el error para enfocar las preguntas en ese punto para encontrar así el error con más facilidad.

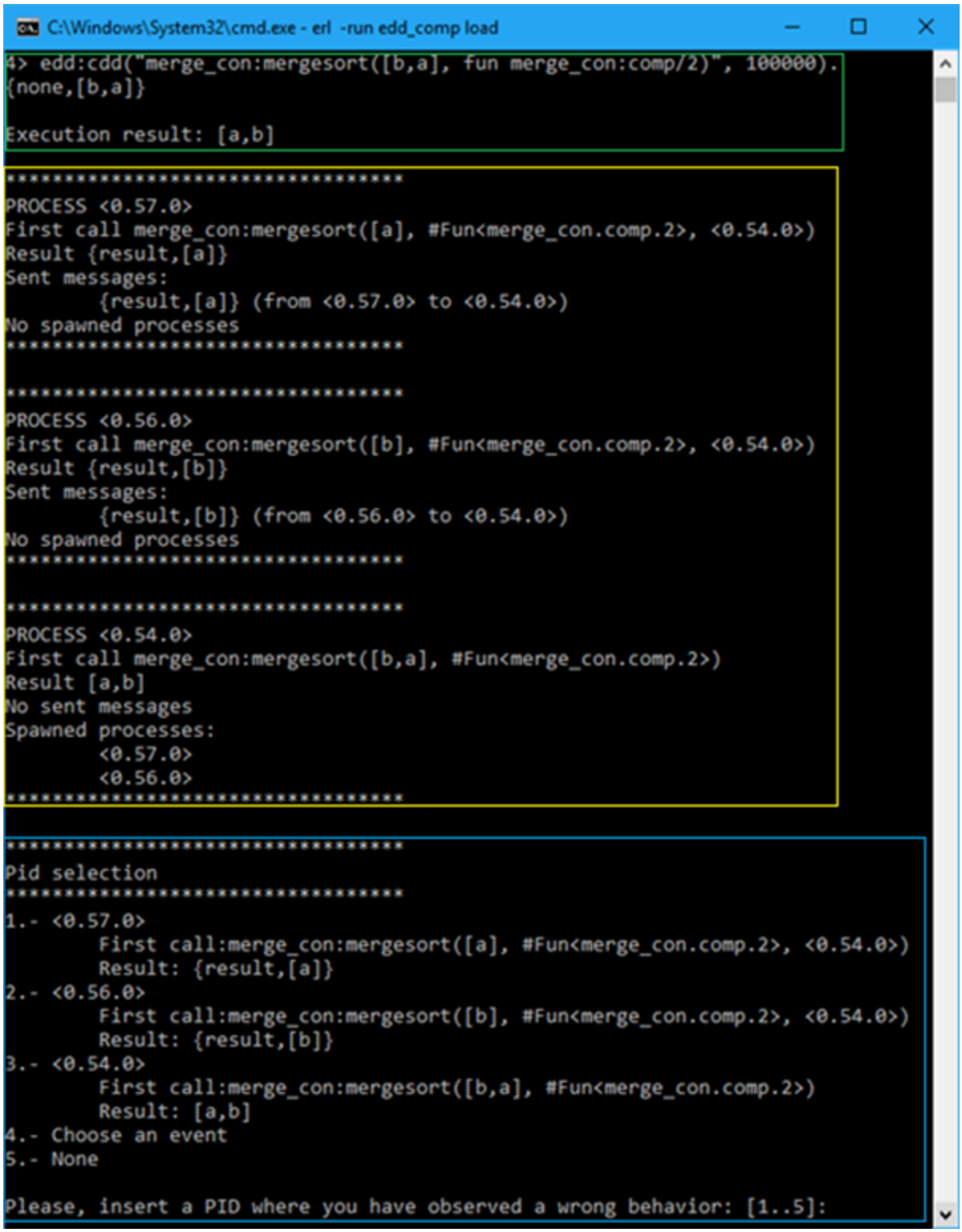


Ilustración 3-8 Ejemplo del inicio del proceso de depuración declarativa concurrente con EDD

Tras seleccionar el punto por el que se quiere empezar el proceso de depuración se nos realizarán una serie de preguntas, un ejemplo de una pregunta que nos puede hacer el depurador la podemos observar en el recuadro verde de la *Ilustración 3-9*, donde las primeras cuatro líneas corresponden con la pregunta a realizar seguido de las posibles respuestas que se puede dar a la pregunta, las dos últimas líneas de este recuadro se corresponden a las diferentes respuestas que se pueden dar, siendo la última la respuesta dada.

Tras contestar una serie de preguntas el depurador encontrará la causa del error informándonos de ello y terminando así el proceso, como podemos ver en el recuadro rojo de la *Ilustración 3-9,* tras responder la última pregunta del proceso de depuración, donde se nos indica el PID del proceso donde está el error y la llamada realizada donde ocurrió el error.

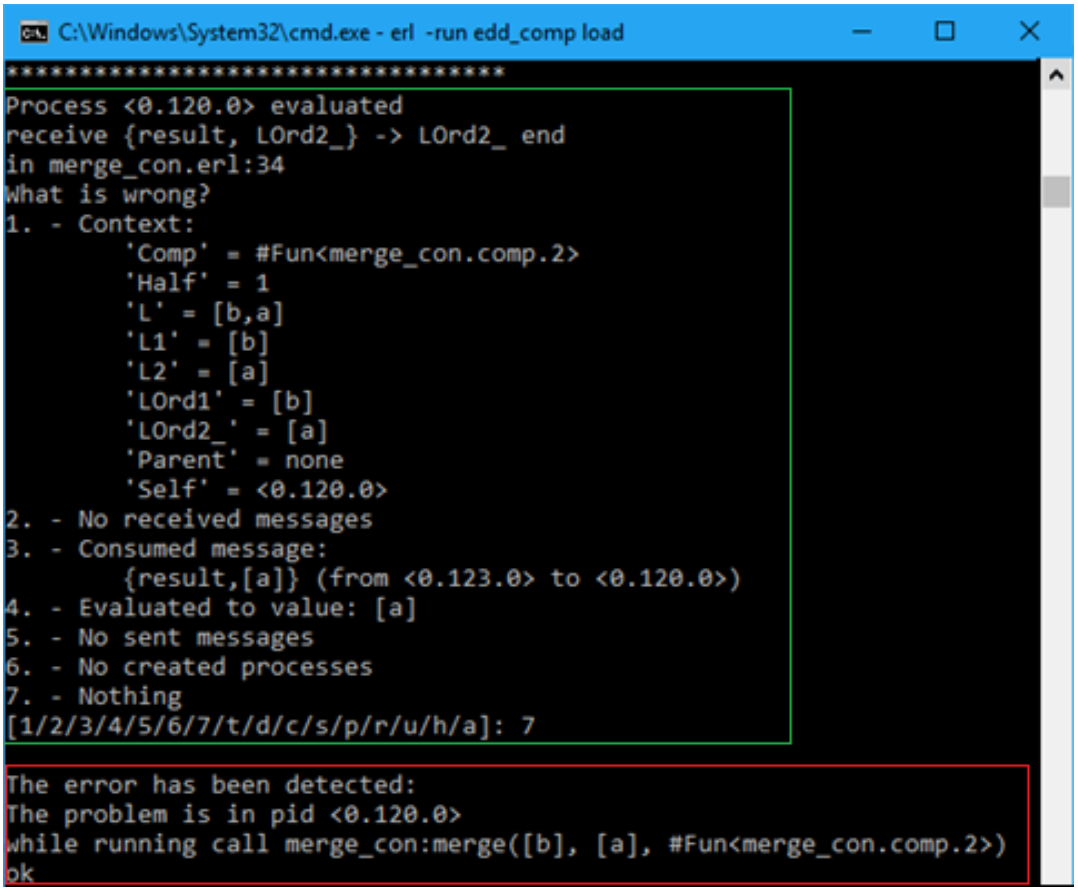


Ilustración 3-9 Ejemplo del fin del proceso de depuración declarativa concurrente con EDD

4 E-EDD (Eclipse - Erlang Declarative Debugger)

En este apartado se describirá la funcionalidad original de este *plugin*, el cual se amplía en este proyecto.

4.1 Descripción

E-EDD es un *plugin* para Eclipse, desarrollado en el Trabajo Fin de Máster de Joel Sánchez en el curso académico 2014/15 [D1], que nos aporta una interfaz gráfica para el depurador declarativo EDD, explicado en el apartado [3.4 EDD (Erlang Declarative Debugger)], a la vez que nos muestra otra serie de datos relevantes de su proceso de depuración.

Durante dicho proyecto se crearon las vistas y el código necesario para obtener un interfaz gráfico para EDD, con el cual realizar el proceso de depuración secuencial mostrándonos a su vez las diferentes preguntas a realizar, así como el estado de las preguntas de manera actualizada, mientras respondemos una a una las preguntas realizadas por el depurador EDD. El objetivo de este proyecto consiste en mejorar y aumentar la funcionalidad de este *plugin* realizando los cambios descritos en el apartado [5. E-EDD v2 Nueva funcionalidad].

4.2 Funcionalidad

A continuación, vamos a explicar cómo funcionaba la herramienta originalmente, la cual sólo contenía el proceso de depuración secuencial. Para lanzar este proceso de depuración declarativa secuencial sobre una aplicación Erlang se tendrá que utilizar la ventana principal de este *plugin*, la cual se muestra en la *Ilustración 4-1*.

En esta ventana tendremos que comunicar el archivo Erlang que queremos depurar rellenando la primera caja de texto utilizando para ello el explorador de archivos que se muestra al pulsar el botón “Browse...”, una vez hecho esto en la segunda caja de texto escribimos la expresión que queremos utilizar para llevar a cabo la depuración. Una vez rellenados estos campos sólo habrá que pulsar el botón de empezar. En la *Ilustración 4-1* podemos observar la ventana rellena con una serie de datos de ejemplo y con el botón que lanza el proceso señalado.

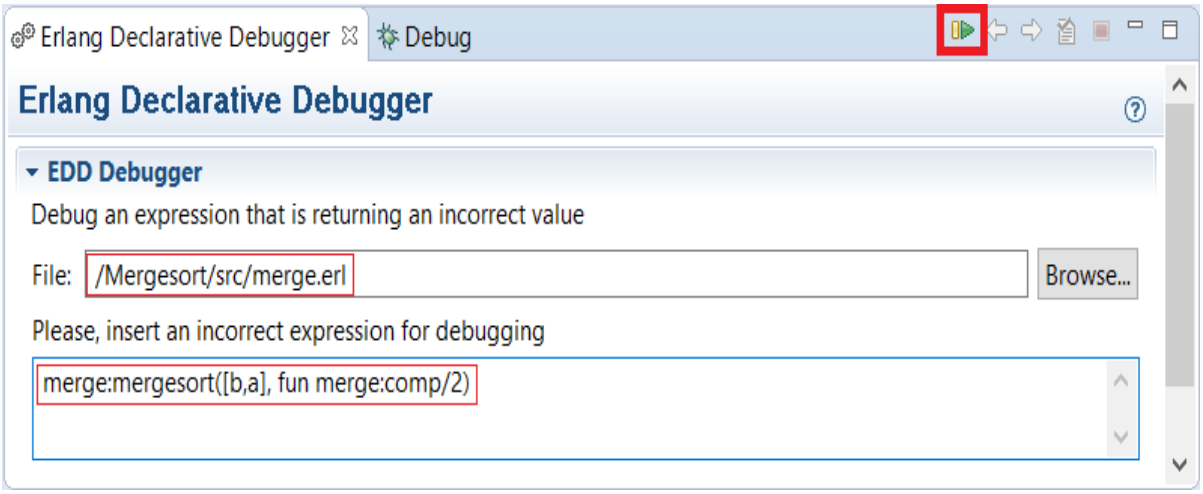


Ilustración 4-1Ventana Principal E-EDD con datos de ejemplo.

Una vez empezada la ejecución se mostrará distinta información del proceso en diferentes ventanas del sistema, las cuales veremos a continuación:

- En la ventana "Graphviz View" se mostrará la imagen de un diagrama de árbol, generado con Graphviz [16], con las diferentes preguntas a realizar, ordenadas jerárquicamente, durante el proceso de depuración, encontrándose los nodos de las preguntas contestadas coloreadas difiriendo de color dependiendo si la respuesta es cierto, que se encontrará en color verde, o falso, en cuyo caso se coloreará de rojo. Un ejemplo de esta ventana que se muestra en el proceso, junto con los nodos coloreados de las respuestas contestadas se encuentra en la imagen de la *Ilustración 4-2*.

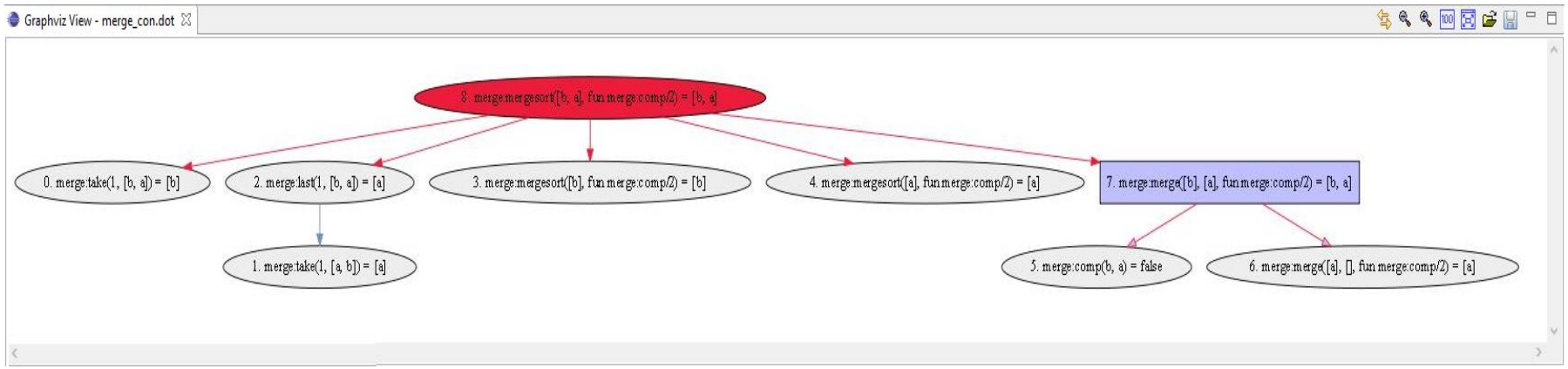


Ilustración 4-2 Ventana Graphviz View con el árbol que se muestra durante el proceso de depuración.

- En la ventana “EDD Tree View” se mostrará en un árbol de ficheros las diferentes preguntas a realizar durante el proceso de depuración, con las que no se podrá interactuar en esta ventana, estando las preguntas contestadas señaladas con un icono representativo de la respuesta que se ha dado a la pregunta. La *Ilustración 4-3* muestra un ejemplo de lo que se ve en esta ventana durante el proceso de depuración.

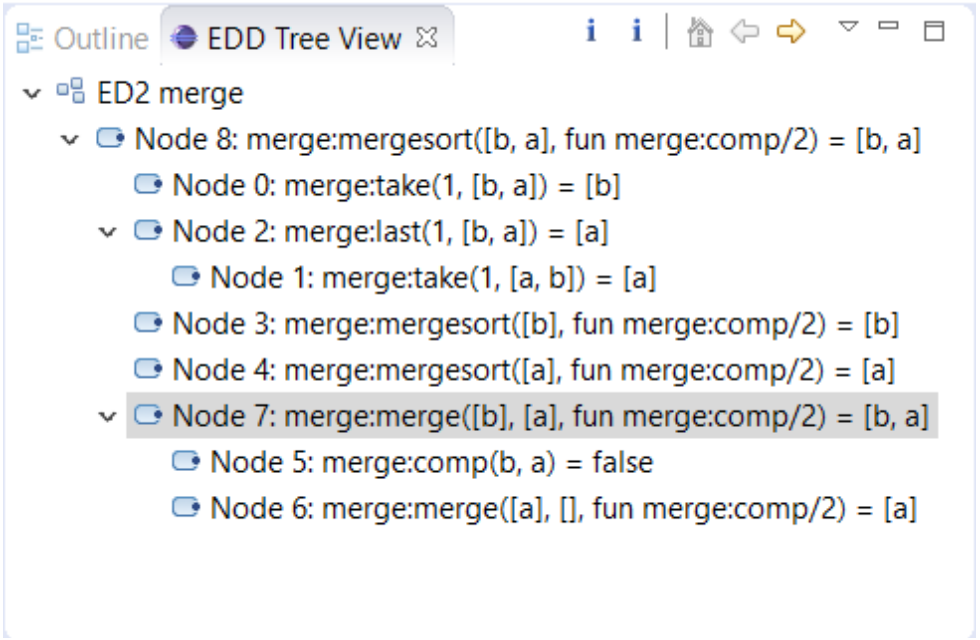


Ilustración 4-3 Ventana "EDD Tree View" con el árbol de preguntas que se muestra durante el proceso de depuración.

- En la ventana principal “Erlang Declarative Debugger” del proceso se mostrará una nueva sección con la pregunta a contestar por el sistema, así como una serie de botones que corresponden a las distintas preguntas a realizar, mientras en la ventana de edición se encontrará el código que estamos depurando con la parte de código que se relaciona con esa pregunta seleccionada. La *Ilustración 4-4* muestra un ejemplo de lo que se ve en esta ventana durante el proceso de depuración.

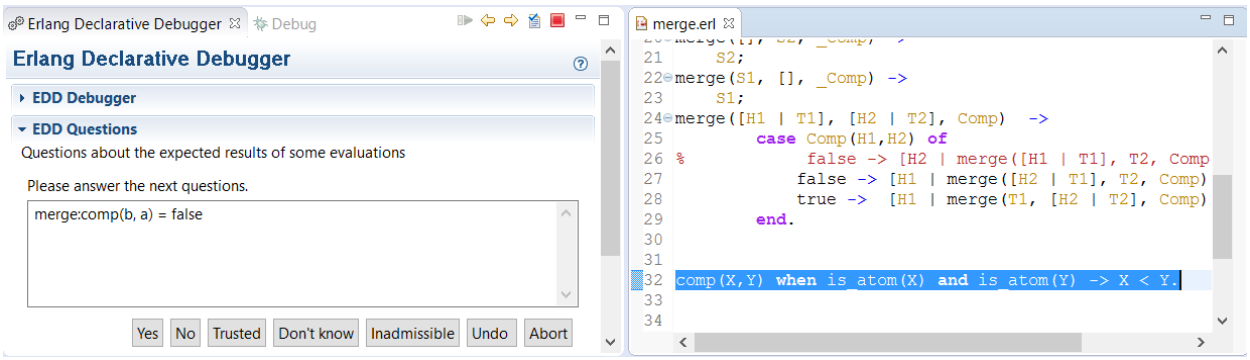


Ilustración 4-4 Ventana principal con la pregunta a responder junto a la ventana de edición con el código seleccionado relativo a la pregunta.

Si vamos respondiendo a las diferentes preguntas que se nos realizan en la ventana principal llegaremos a un punto donde se nos indicará que se ha encontrado el problema, comunicándonos cuál es y se nos preguntará si deseamos realizar una búsqueda más exhaustiva, tal y como podemos observar en la *Ilustración 4-5*, en caso de responder *no* se terminará el proceso de depuración y en caso de elegir la opción de continuar se procederá a la depuración con zoom, con lo que se generarán un nuevo set de preguntas a responder y se refrescarán las ventanas con las nuevas preguntas.

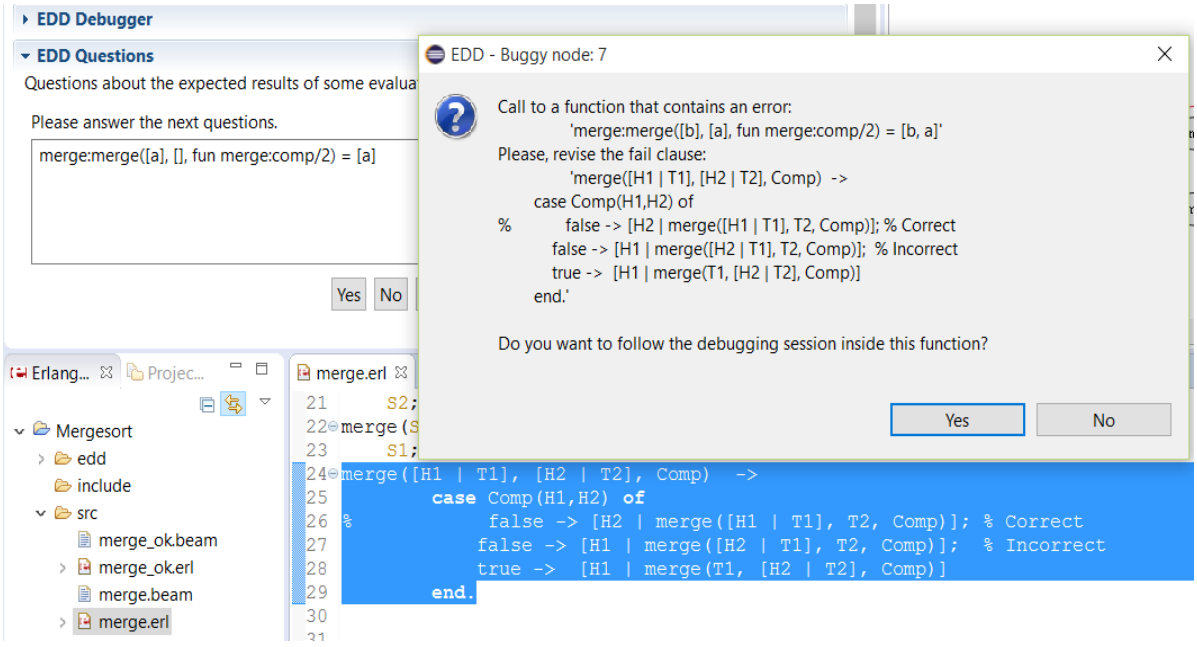


Ilustración 4-5 Mensaje que se muestra al terminar el proceso de depuración.

Una vez respondidas suficientes preguntas del proceso de depuración con zoom se nos mostrará un mensaje comunicándonos el motivo del error y terminando la ejecución del proceso de depuración, como se nos muestra en la imagen de la *Ilustración 4-6*.

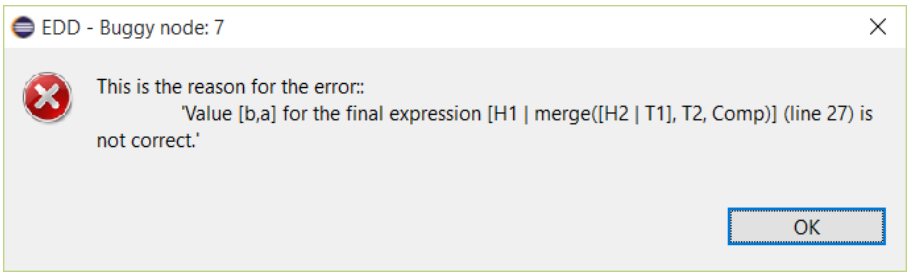


Ilustración 4-6 Mensaje que se muestra al terminar el proceso de depuración por Zoom.

5 E-EDD 2.0: Nueva funcionalidad

En este apartado se describirán las diferentes mejoras que se han realizado sobre el *plugin* original de E-EDD, descrito en el capítulo anterior. Estas mejoras consisten en la inserción de nueva funcionalidad a la ventana "EDD Tree View", que se describe en el apartado [5.1 Árbol de operaciones], y la adición de la operativa de depuración concurrente, descrito en el apartado [5.2 Depuración concurrente].

5.1 Árbol de operaciones

Una de las mejoras realizadas ha sido añadir interactividad a la ventana "EDD Tree View", la cual originalmente sólo mostraba mediante un árbol de ficheros las diversas preguntas que puede realizar el depurador en el proceso de depuración.

Esta mejora consiste en añadir interactividad al árbol de ficheros, donde se muestran las preguntas, consiguiendo con ello agilizar el proceso de depuración secuencial al permitir al usuario responder directamente las preguntas que desee sin tener que esperar a que el depurador se posicione sobre estas. Las diferentes maneras con las que se puede interactuar con el árbol son las siguientes:

- Se podrá responder cualquiera de las preguntas mostradas a través de una serie de botones, que se han añadido a la vista y que corresponden con las posibles respuestas que se pueden realizar sobre una pregunta en la depuración secuencial (yes, trusted, no, inadmissible, don't know), los cuales se pueden observar en la imagen de la *Ilustración 5-1*. Para responder sólo hará falta seleccionar la pregunta que se desea responder y pulsar el botón correspondiente a la respuesta que se quiera dar.

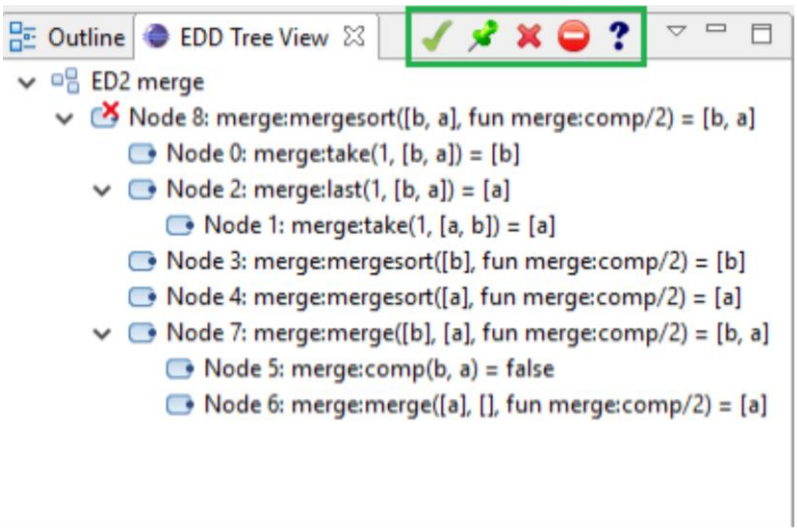


Ilustración 5-1 Ventana del árbol de preguntas, señalando los botones con los que se responden.

- Se podrá responder cualquiera de las preguntas mostradas a través de un menú contextual que aparecerá al pulsar el botón derecho sobre la pregunta y que contendrá las diferentes respuestas que se pueden dar a la pregunta en la depuración secuencial. Este menú se puede observar en la imagen de la *Ilustración 5-2*. Para responder a la pregunta se necesitará pulsar con el botón derecho sobre la pregunta que se desea responder con lo que se mostrará el menú contextual y en él se seleccionará la respuesta que se quiera dar.

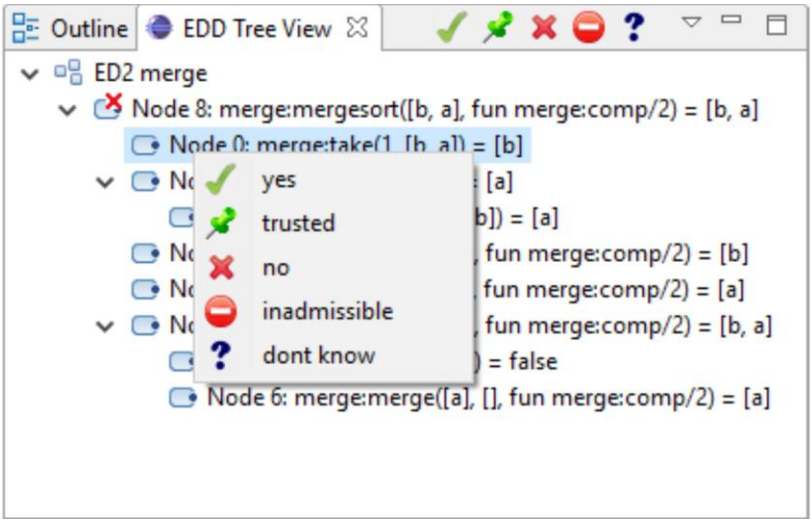


Ilustración 5-2 Menú contextual del árbol de preguntas

- Se podrá ir directamente a cualquiera de las preguntas haciendo doble-click sobre dicha pregunta de tal modo que esta se mostrará en la ventana principal, para ser respondida.

Esta nueva funcionabilidad solo se encuentra disponible para la realización de la depuración secuencial y no se mostrará en la nueva operativa de depuración concurrente debido a una serie de impedimentos:

- El gran tamaño que tienen tanto las preguntas como las respuestas que se realizan en la depuración concurrente, lo que dificulta la visualización de las mismas
- El hecho de que al contrario que la depuración secuencial las respuestas no son fijas, y puede cambiar el número de estas entre una pregunta y otra.

5.2 Depuración concurrente

Esta funcionalidad es el mayor añadido realizado en este proyecto, a través de la cual el usuario puede depurar una aplicación Erlang concurrente. Dado que la concurrencia es uno de los aspectos clave de Erlang, así como una de las características más difíciles de depurar, esta extensión ofrece una importante mejora respecto al anterior interfaz.

Para la realización de este apartado se ha intentado reutilizar lo máximo posible el código y las ventanas ya existentes, las cuales se verán en los siguientes apartados.

5.2.1 Vista principal

Para la ejecución de este proceso de depuración se reutiliza el mismo panel que para la depuración secuencial, utilizando los mismos campos para los valores de entrada, cambiando solo el botón con el que se inicia la operativa, el cual se ha añadido a la vista y se puede observar destacado en la *Ilustración 5-3*.

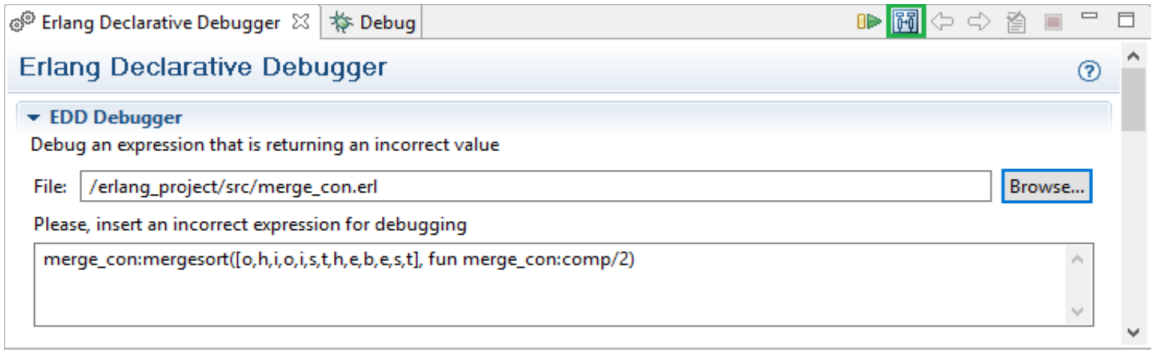


Ilustración 5-3 Ventana principal con los datos para realizar la depuración concurrente.

Una vez que se ha pulsado el botón se informará si faltan datos, y en caso contrario se levantará el depurador EDD, y se le comunicará que debe de realizar un proceso de depuración concurrente. Tras comunicar el inicio de la depuración a EDD se empezará el proceso de depuración, mostrando su información que ha sido pasada por EDD en los mismos paneles que con la depuración secuencial, además de otras vistas con información exclusiva de este proceso de depuración, que se verán más adelante.

Tras la inicialización del proceso de depuración, en la vista principal donde se han puesto los datos, se desplegará al igual que con la depuración secuencial un apartado de preguntas. En este apartado veremos un cuadro de texto con la pregunta actual gestionada por EDD, y sus posibles respuestas en una serie de botones numerados que indican la respuesta a elegir, como se puede observar en la *Ilustración 5-4*.

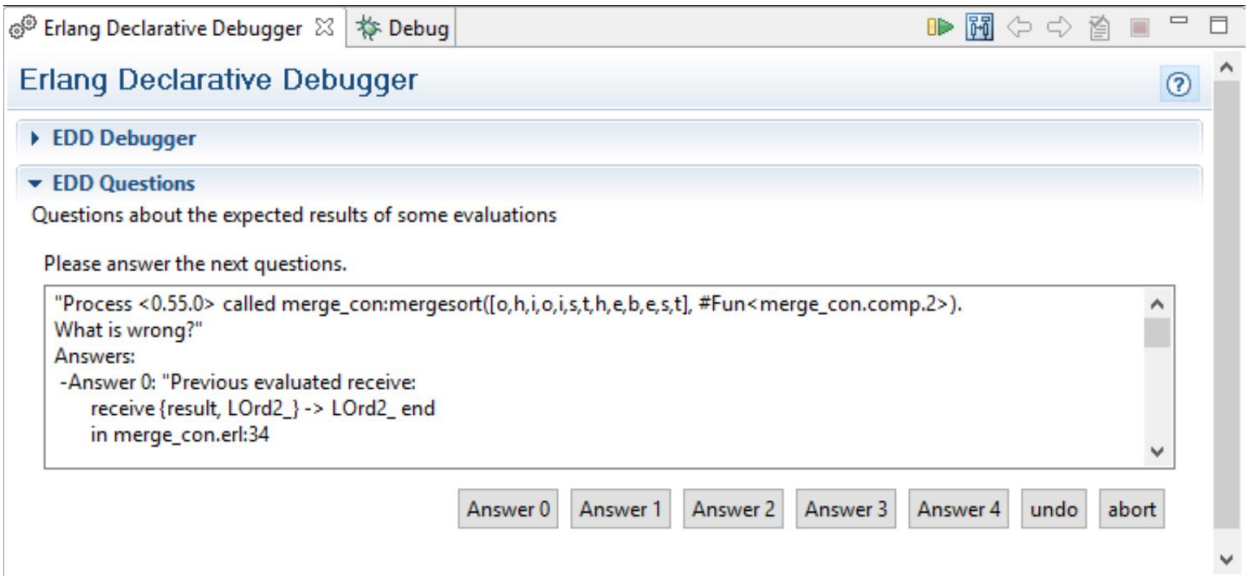


Ilustración 5-4 Ventana principal con una pregunta del proceso de depuración concurrente.

Al contrario que con la depuración secuencial el texto de las respuestas se encuentra en el cuadro de texto. Esto es debido a que la cantidad de estas no es constante y suelen tener textos largos, por lo que no se verían bien como botones y habría que cambiar el diseño completamente, o realizar la operativa en otro apartado, dificultando con ello la visualización.

5.2.2 Árbol de creación de procesos

En la ventana de "Graphiz View", donde en el proceso de depuración secuencial se mostraba un árbol con las diferentes preguntas del depurador y su estado, cuando se ejecuta la

depuración concurrente se mostrará en esta ventana un diagrama de árbol, generada al igual que el anterior a través de Graphviz [16]. Esta ventana muestra los diferentes procesos que se han ido creando durante la ejecución, enlazados de manera jerárquica, siendo la raíz el primero. Un ejemplo del árbol resultante lo podemos ver en la *Ilustración 5-5*. Gracias a este árbol el usuario podrá ver todos los procesos generados durante la depuración, sirviendo esto para facilitar encontrar y centrarse en los procesos erróneos.

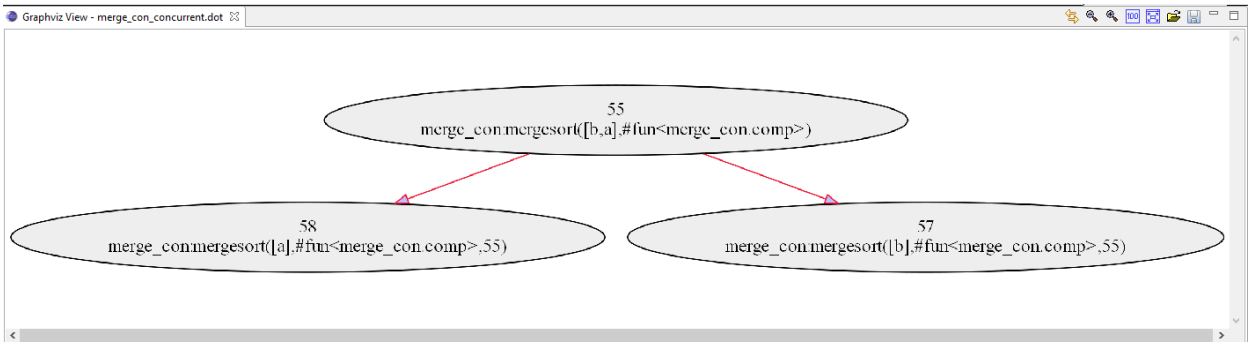


Ilustración 5-5 Ventana de Graphiz con el árbol de creación de procesos.

5.2.3 Diagrama de secuencia

Esta nueva vista, que podemos ver en la *Ilustración 5-6*, mostrará el diagrama de secuencia de la ejecución del programa realizado en la depuración concurrente, con el cual el usuario podrá interaccionar. En él se muestra los diferentes procesos que se han creado, que se corresponden con las líneas verticales, siendo las cabeceras su identificador y el método con el que han sido llamadas, y el cambio de mensajes que ha habido entre ellos durante la ejecución de la depuración, que corresponden a las flechas horizontales que unen los procesos emisor y receptor del mensaje y que pueden ser de tres tipos identificados por su color y texto: creación de procesos (líneas rojas), envío de mensajes (líneas azules) y recepción de mensajes (líneas verdes).

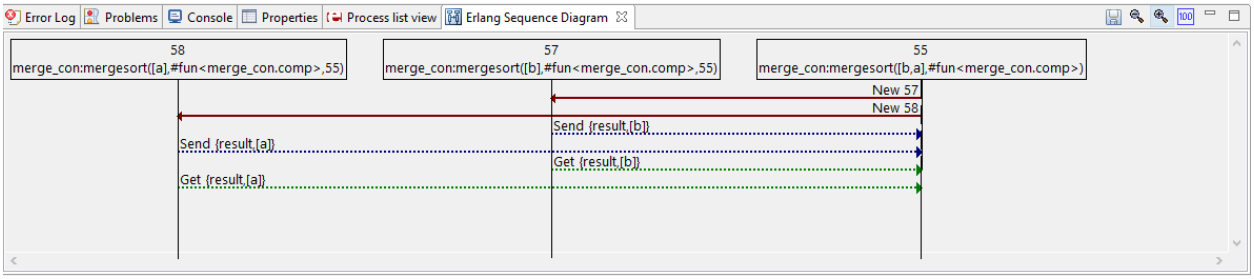


Ilustración 5-6 Ventana del diagrama de secuencia de la depuración concurrente.

El desarrollo del diagrama ha sido realizado a través de los objetos básicos de Eclipse, realizando una serie de patrones para realizarlo, explicados en más detalle en el apartado [B.2.2 Diagrama de Secuencia], ya que no se han encontrado ningún *plugin* que permitiera dibujar un diagrama de secuencia de manera programática con el que el usuario pudiera interactuar, encontrando en su lugar generadores de imágenes de diagramas de secuencia, herramientas gráficas de desarrollo, y obtención de diagramas de un proyecto a través de Ingeniería inversa.

Esta vista contiene una barra de herramientas con las siguientes funciones:

- Una opción que permite guardar como una imagen el diagrama de secuencia.
- Una opción para reducir el tamaño del diagrama.
- Una opción para aumentar el tamaño del diagrama.
- Una opción para devolver el diagrama al tamaño original.

Como hemos comentado antes el usuario puede interactuar con el diagrama haciendo click en diversas zonas del diagrama, en las cuales el icono del ratón cambiará por el de selección para facilitar el manejo de la herramienta. Las áreas del diagrama con las que el usuario podrá interaccionar son:

- **Las cabeceras de los procesos**: al presionar sobre cualquiera de estas se mostrará una ventana de dialogo, como la que se muestra en la *Ilustración 5-7*, con la información relativa del Proceso:
  - Proceso que creó el mensaje.
  - Mensajes enviados por el proceso.
  - Mensajes recibidos por este.

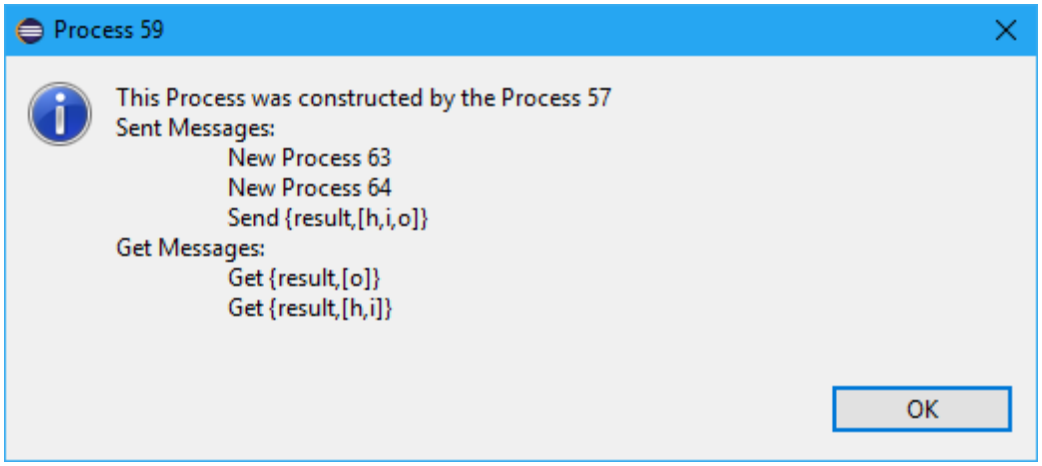


Ilustración 5-7 Dialogo con la información del proceso.

- **Mensajes**: Al hacer click sobre un mensaje se nos mostrará un dialogo con una serie de preguntas relativas a dicho mensaje, que el usuario podrá responder, como se muestra en la *Ilustración 5-8*.

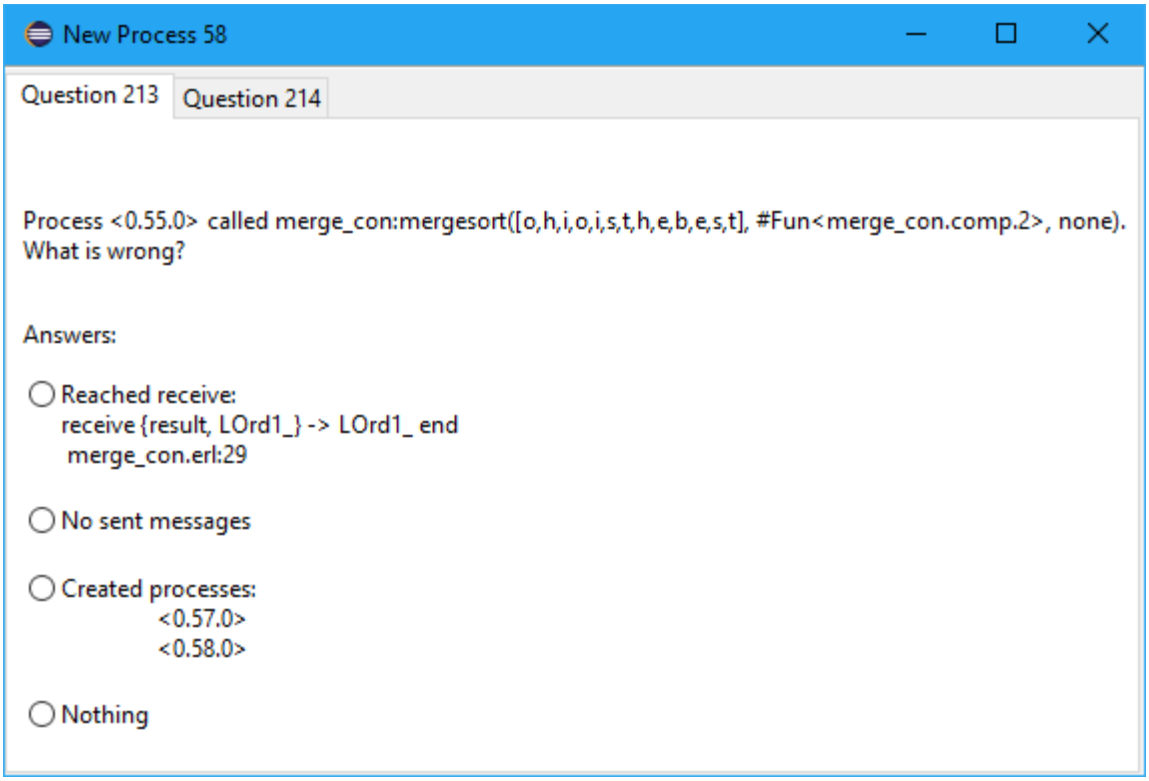


Ilustración 5-8 Dialogo de preguntas de un mensaje.

6 Manual de Uso

En este apartado explicaremos mediante un ejemplo, que ya ha sido explicado en el punto [3.1.1 Ejemplo] junto con el código de los módulos que utiliza, cómo se maneja el *plugin*.

6.1 Creación de un proyecto

Para poder depurar un programa Erlang este debe encontrarse dentro de un proyecto Erlang que se encuentre en el workspace actual de Eclipse. Para crear un proyecto Eclipse tendremos que hacer click en la opción del menú "File/New/Erlang Proyect", tal y como se ve en la *Ilustración 6-1*.

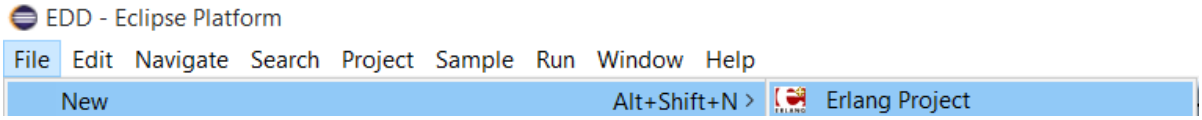


Ilustración 6-1 Opción del menú para crear un nuevo proyecto Erlang

A continuación, se nos preguntará por el nombre del proyecto, como se observa en la *Ilustración 6-2*.

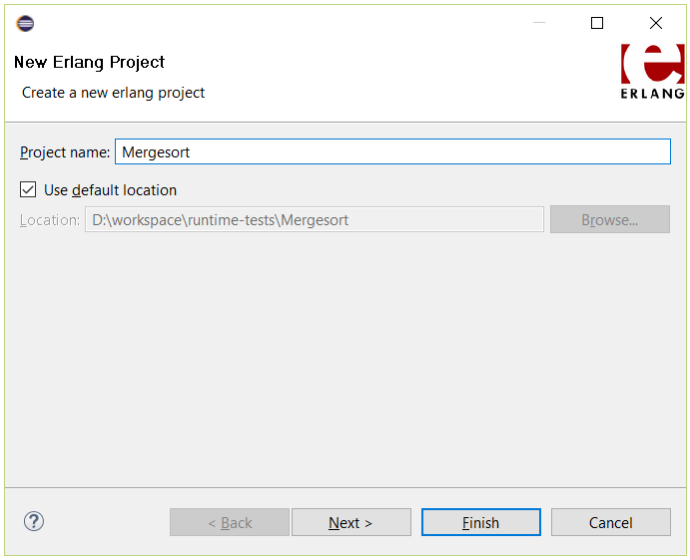


Ilustración 6-2 Ventana de inserción del nombre del nuevo proyecto Erlang

Tras ello se nos pedirá la versión de Erlang que se está utilizando, tal y como se muestra en la *Ilustración 6-3*.

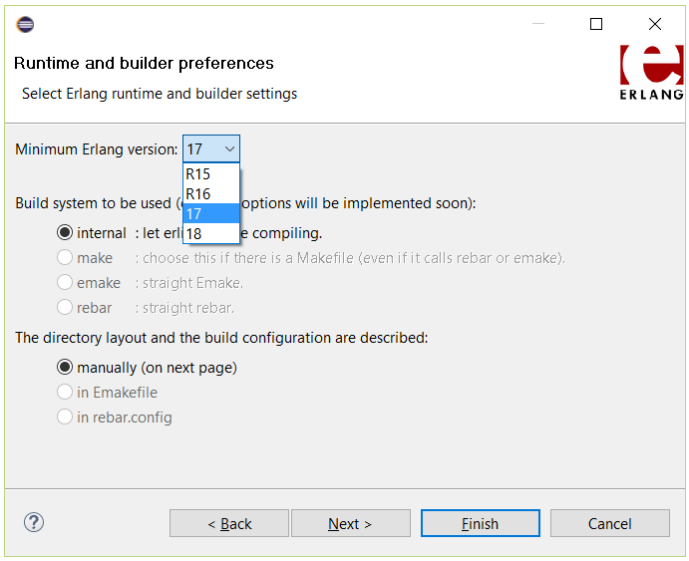


Ilustración 6-3 Ventana de petición de datos de Erlang.

En la siguiente página se pedirá que se indique la localización del código fuente y de los archivos compilados. Se recomienda modificar el directorio de salida de los archivos ebin a src tal y como se muestra en la *Ilustración 6-4*, dado que la ubicación de los binarios puede dar lugar a errores de ejecución.

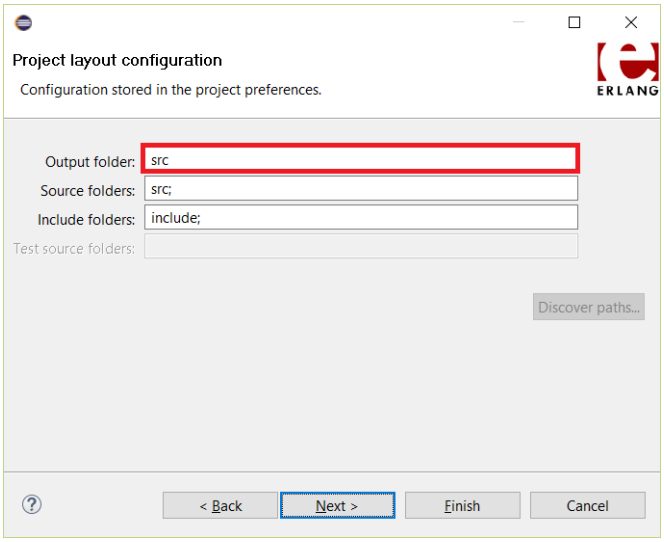


Ilustración 6-4 Ventana de localización de los diferentes tipos de archivos a gestionar por el proyecto

Una vez terminado la creación del proyecto solo hará falta poner los programas Erlang que se quieren depurar dentro de la carpeta “scr” del proyecto.

6.2 Depuración secuencial

Para realizar una depuración secuencial se deben de rellenar los datos de entrada de la ventana "**Erlang Declarative Debugger**", que corresponden al programa que se quiere probar y la llamada que se va a depurar, como podemos observar en la *Ilustración 6-5*.

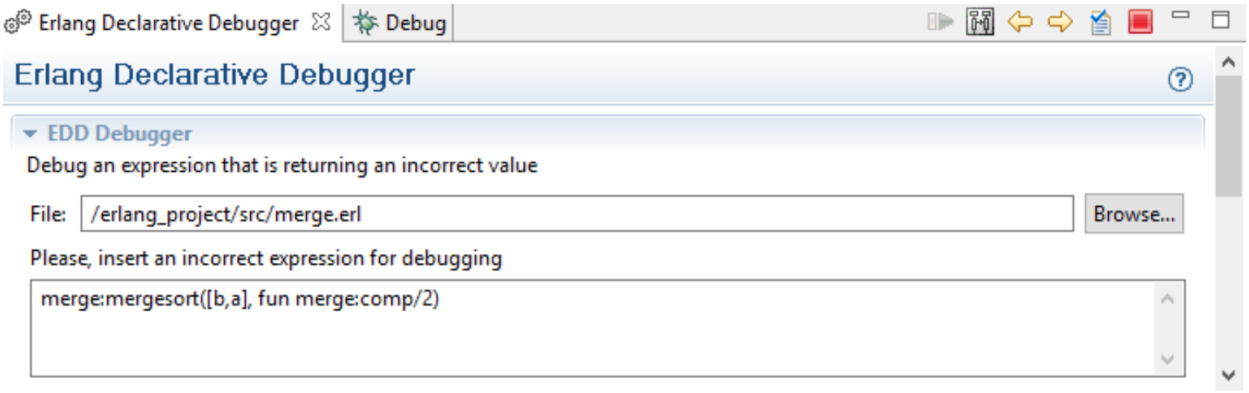


Ilustración 6-5 Ventana principal con datos de ejemplo para el proceso de depuración secuencial.

Una vez introducidos los datos se debe de pulsar el botón de inicialización de la depuración, con la cual comenzará el proceso de depuración mostrando en la ventana anterior una sección con una pregunta y se señalará en el código la parte referenciada en la pregunta, tal y como se muestra en la *Ilustración 6-6*.

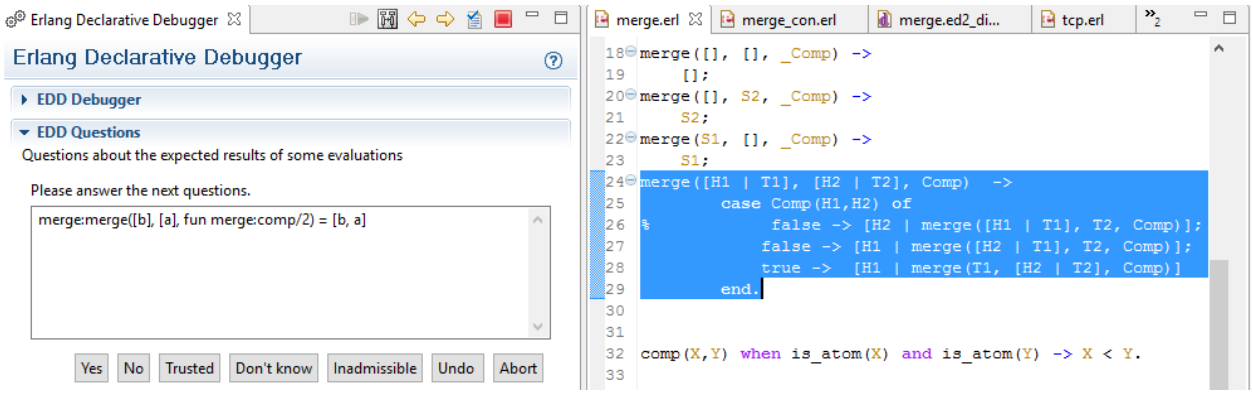


Ilustración 6-6 Primera pregunta del proceso de depuración secuencial

A parte de esto se mostrará el árbol de preguntas tanto en la ventana "**Graphviz View**", como un diagrama de árbol como se muestra en la *Ilustración 6-7*, como en la ventana "**Edd Tree View**", mediante un árbol de ficheros como podemos observar en la *Ilustración 6-8*.

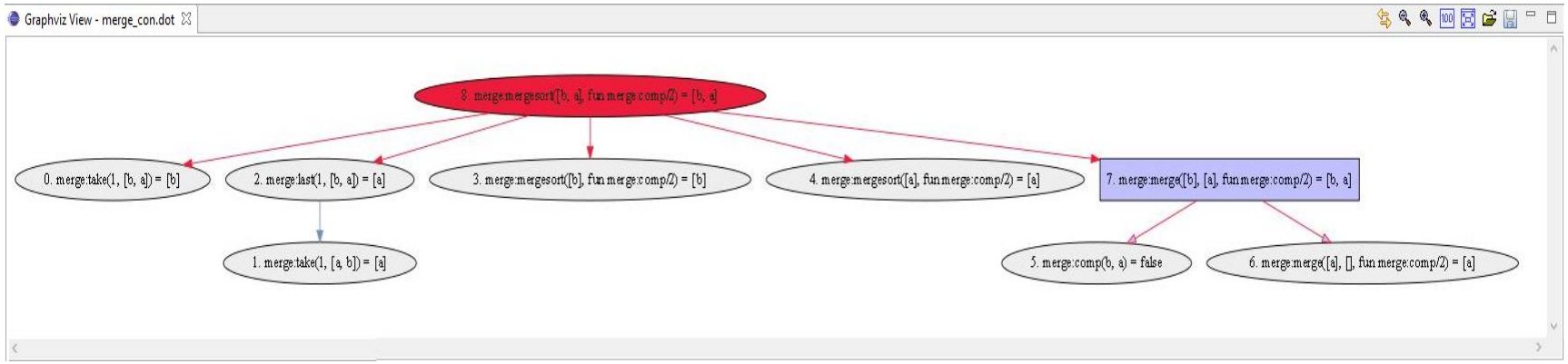


Ilustración 6-7 Árbol de depuración obtenido en la primera pregunta del proceso de depuración secuencial.

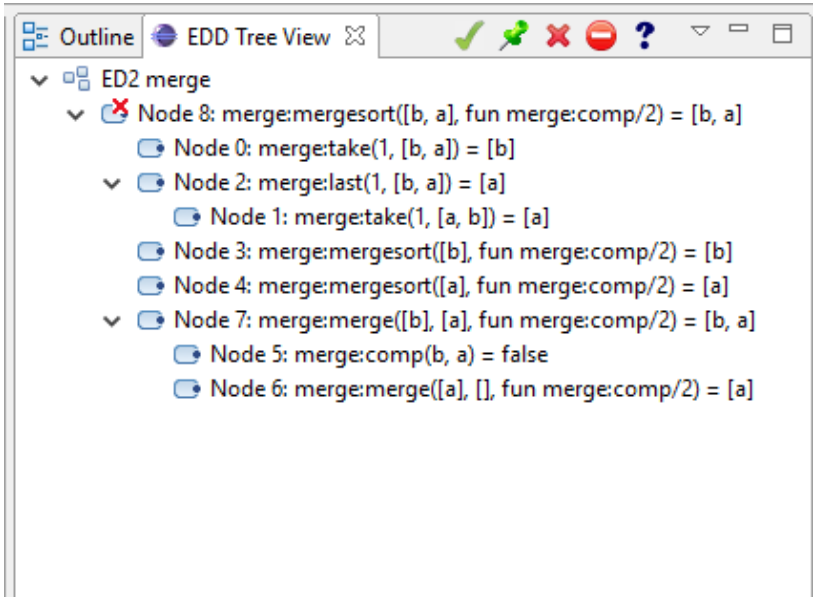


Ilustración 6-8 Ventana EDD Tree View al iniciar el proceso de depuración secuencial.

Para proseguir con la depuración se habrá de contestar a las diferentes preguntas que envía el depurador. Esto se podrá hacer a través de las ventanas:

- **Erlang Declarative Debugger**: Contestando la pregunta que se muestra en la ventana, pulsando el botón correspondiente a la respuesta que se quiere contestar, que en nuestro caso será el botón destacado que podemos ver en la *Ilustración 6-9*.

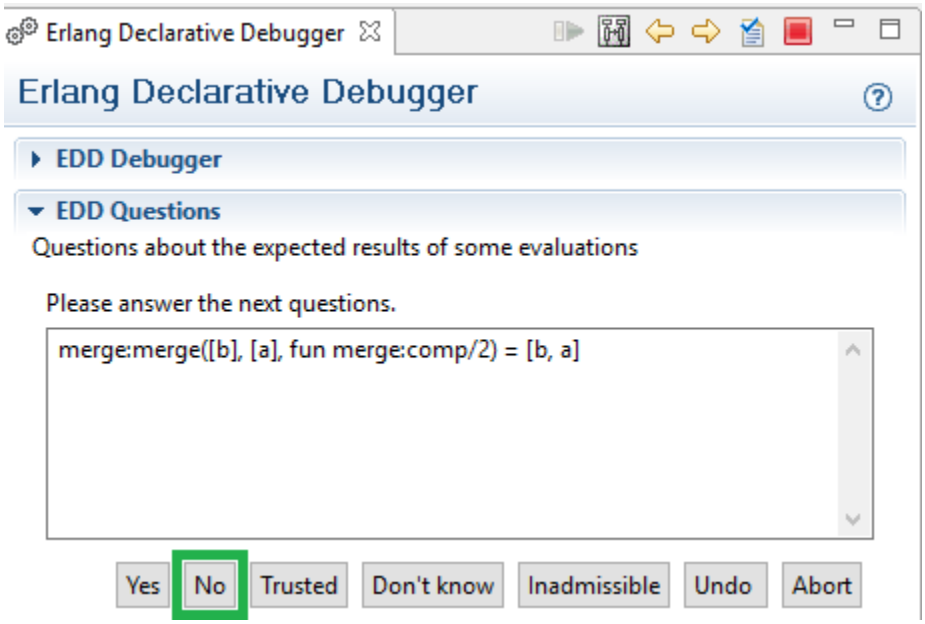


Ilustración 6-9 Ventana principal con la respuesta dada a la primera pregunta.

- **Edd Tree View**: Contestando cualquiera de las preguntas que se muestran en el árbol, lo cual se puede hacer de las siguientes maneras:
  - Seleccionando la pregunta que se quiere responder y pulsando el botón correspondiente a la respuesta que se quiere dar. En la *Ilustración 6-10* podemos observar esta ventana con los botones que podemos pulsar para responder a las preguntas destacados.

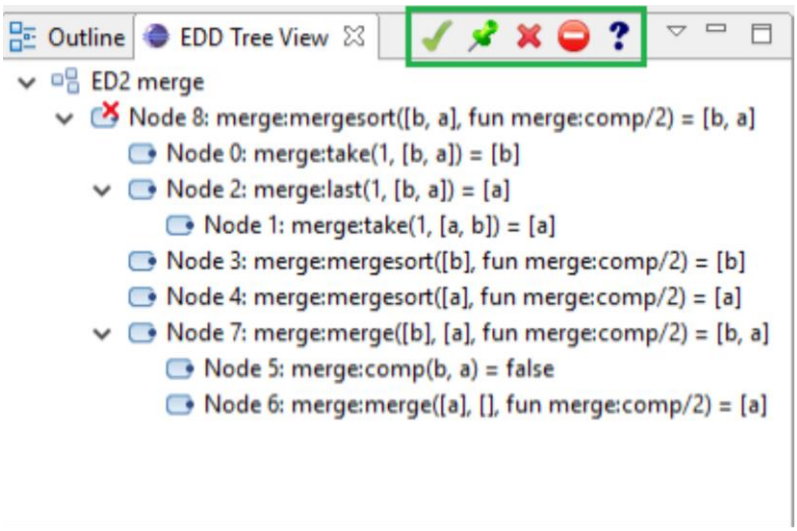


Ilustración 6-10 Ventana del árbol de preguntas, señalando los botones con los que se responden.

- Pulsando el botón derecho sobre la pregunta a contestar y eligiendo del menú desplegable que se muestra la opción a contestar, como podemos ver en la imagen de la *Ilustración 6-11*.

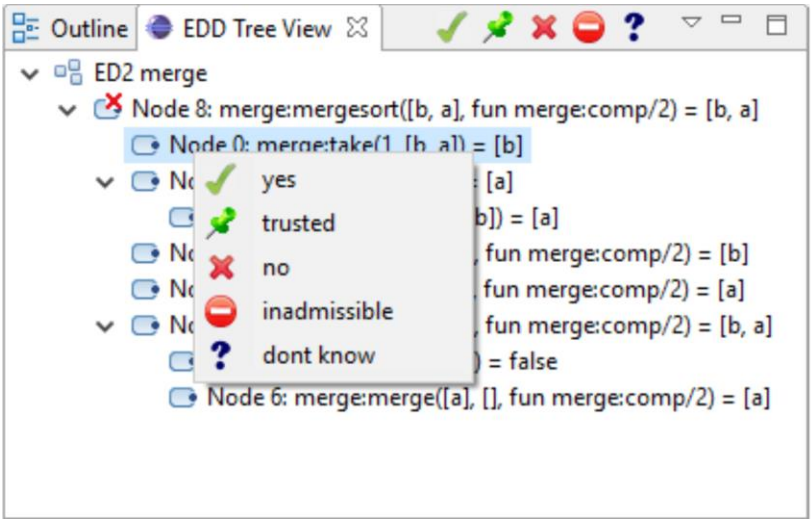


Ilustración 6-11 Menú contextual del árbol de preguntas

- Dando doble click sobre la pregunta que se quiere contestar, lo que hará que esta se muestre en la ventana "**Erlang Declarative Debugger**", donde el usuario podrá responderla.

En nuestro caso para encontrar el error bastará con contestar las preguntas con la misma repuesta que se muestran en las imágenes que muestran en la *Ilustración 6-10*, la *Ilustración 6-12* y la *Ilustración 6-13* de cualquiera de las maneras que se han explicado anteriormente.

En la primera pregunta, que podemos ver en la *Ilustración 6-10*, nos preguntan si el resultado esperado de la operativa es la lista [b, a], siendo esto incorrecto, puesto el resultado de la ordenación del array [b, a] debería ser [a, b].

En la segunda pregunta, que podemos ver en la *Ilustración 6-12*, nos preguntan si el resultado de la operación merge([a], [], fun merge:comp/2) es [a], siendo esto cierto, pues cuando se pasa un solo valor entre los dos arrays se debe de devolver un array con sólo ese valor.

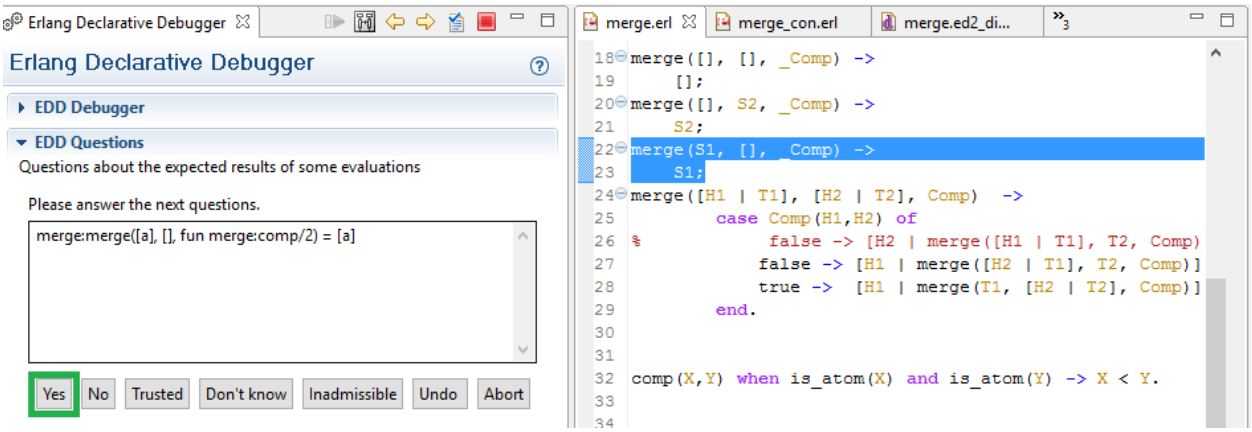


Ilustración 6-12 Segunda pregunta del proceso de depuración secuencial con la respuesta a dar

En la tercera pregunta, que podemos ver en la *Ilustración 6-12*, nos preguntan si el resultado de la operación merge:comp(b,a) es false, siendo esto cierto.

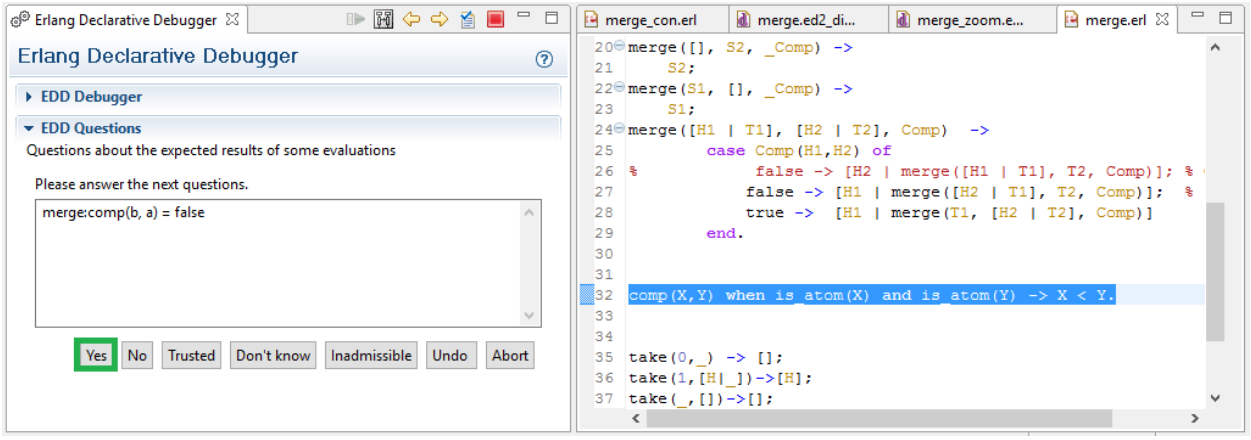


Ilustración 6-13 Tercera pregunta del proceso de depuración secuencial con la respuesta a dar

Tras responder las suficientes preguntas, el depurador encontrará la causa del error y se nos informará a través de un pop-up, tal y como se muestra en la *Ilustración 6-14*, dónde se nos comunicará donde se encuentra el error, y se preguntará si se quiere inspeccionar más a fondo.

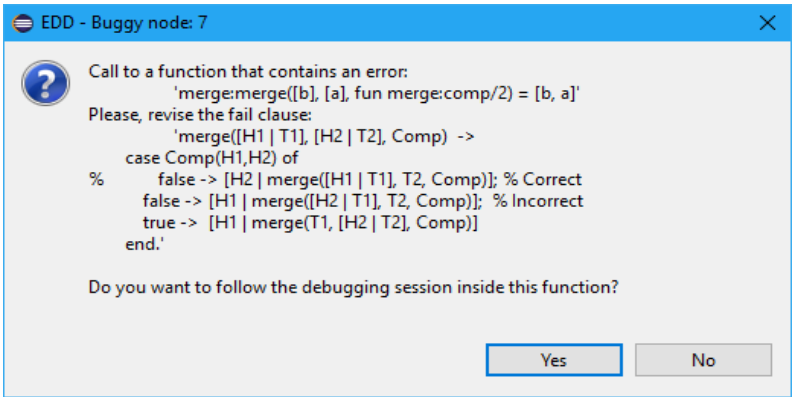


Ilustración 6-14 Mensaje de comunicación de que el depurador a encontrado el error.

Si decidimos pulsar la opción de "No" se terminará la depuración, y en caso contrario se generarán una nueva serie de preguntas más exhaustivas y se modificará toda la información mostrada en las vistas "**Graphviz View**", que se muestra en la *Ilustración 6-16* y "**Edd Tree View**", que se muestra en la *Ilustración 6-17*, tras lo que se realizarán más preguntas que se responderán de la misma manera que se muestra en la *Ilustración 6-15* junto con el código relativo a la pregunta actual.

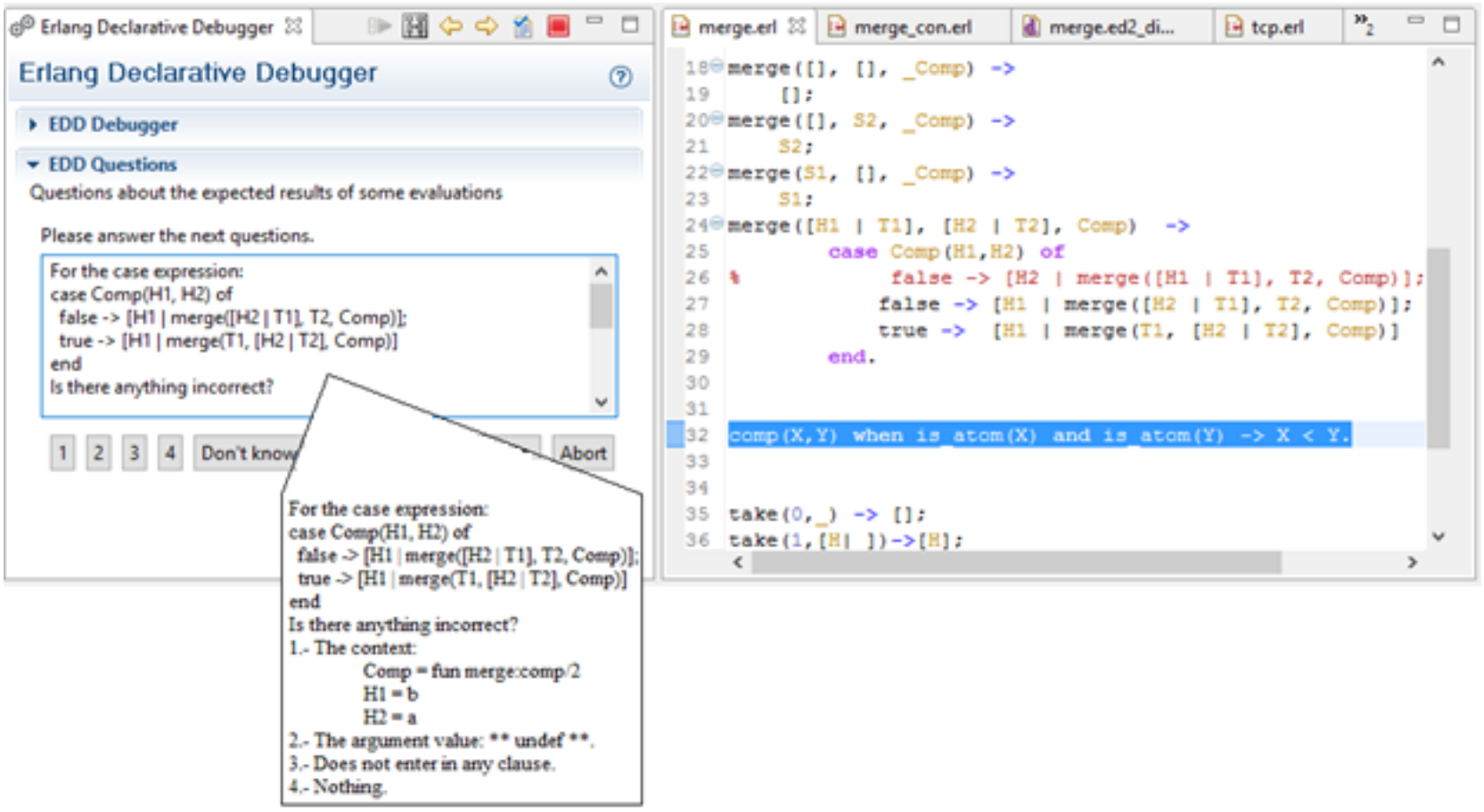


Ilustración 6-15 Primera pregunta del proceso de depuración secuencial por Zoom

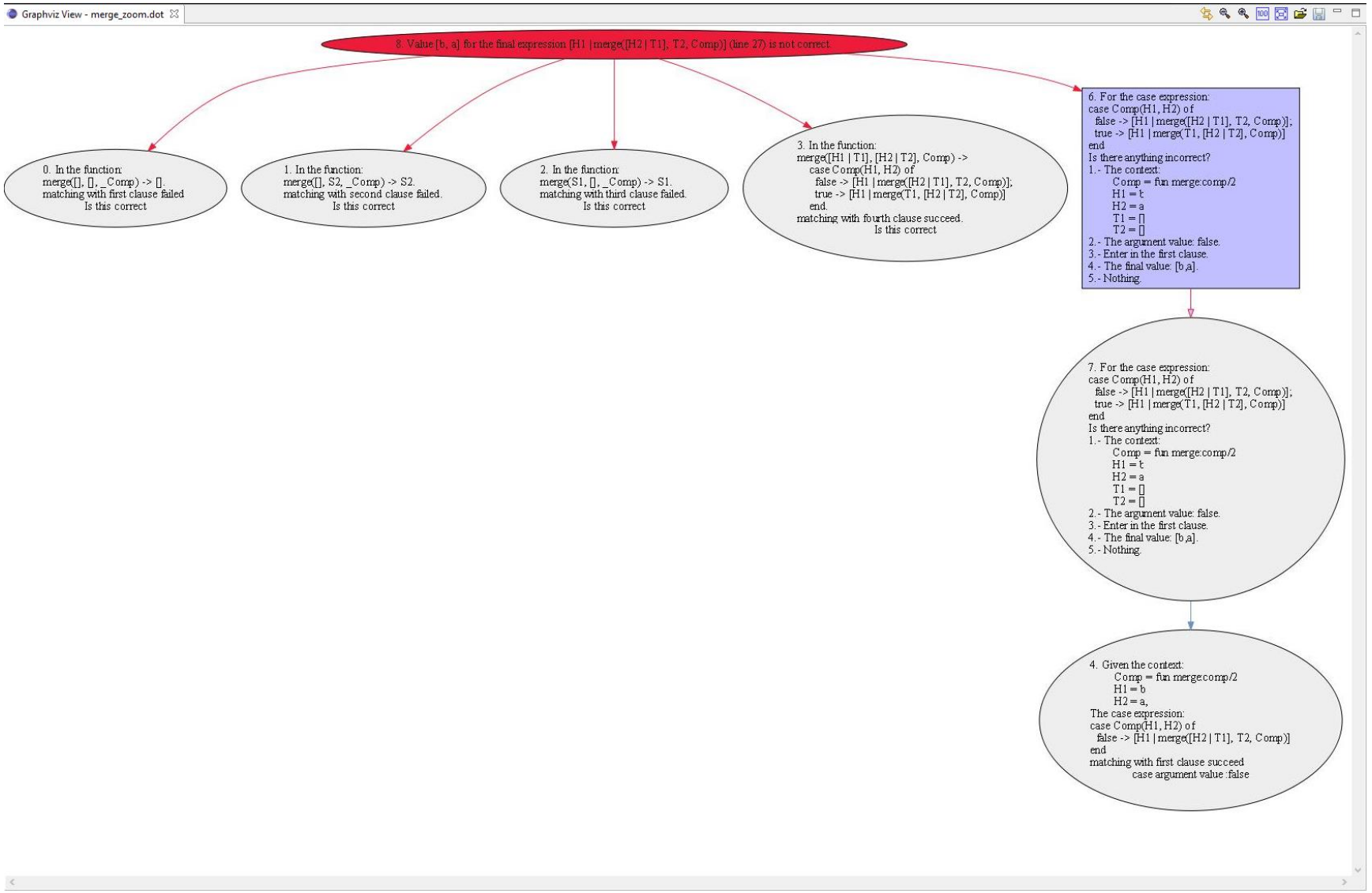


Ilustración 6-16 Árbol de depuración del proceso de depuración secuencial por Zoom

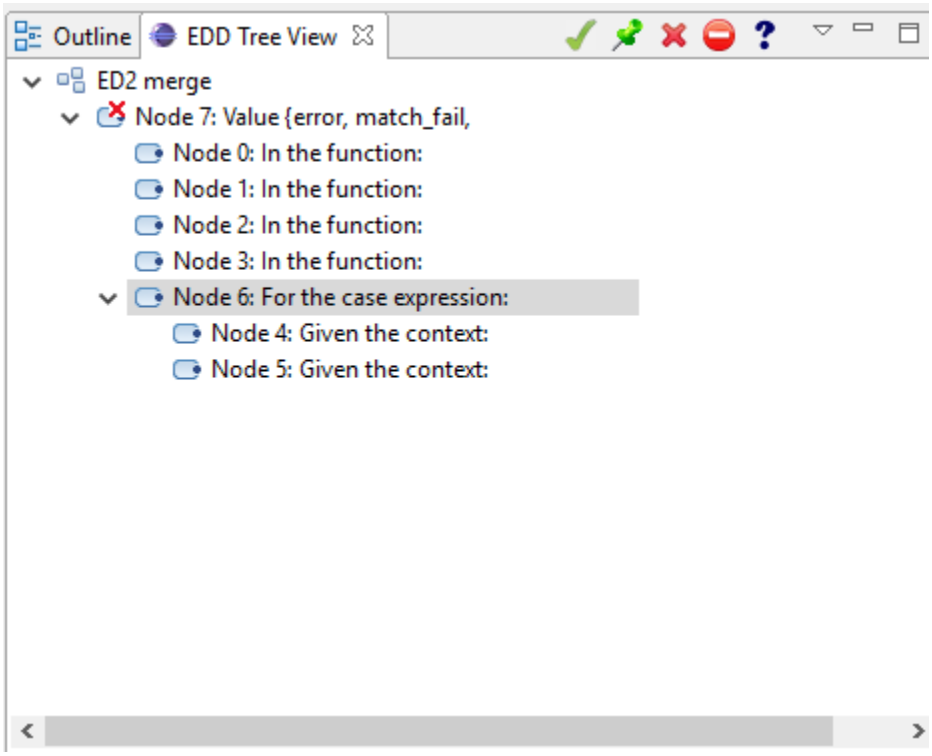


Ilustración 6-17 Ventana EDD Tree View al inicio de la depuración secuencial por Zoom.

Tras esto tendremos que responder las preguntas pertinentes, en nuestro caso bastará con responder las preguntas con las respuestas que se pueden observar las imágenes que muestran en la *Ilustración 6-18*, la *Ilustración 6-19*, la *Ilustración 6-20* y la *Ilustración 6-21*.

En la primera pregunta, que podemos ver en la *Ilustración 6-18*, nos preguntan para la expresión mostrada si alguna de las respuestas indica algo erróneo dentro de la expresión, en nuestro caso ninguna de las repuestas es causante del problema, por lo que elegiremos la opción 4, que ninguna es errónea.

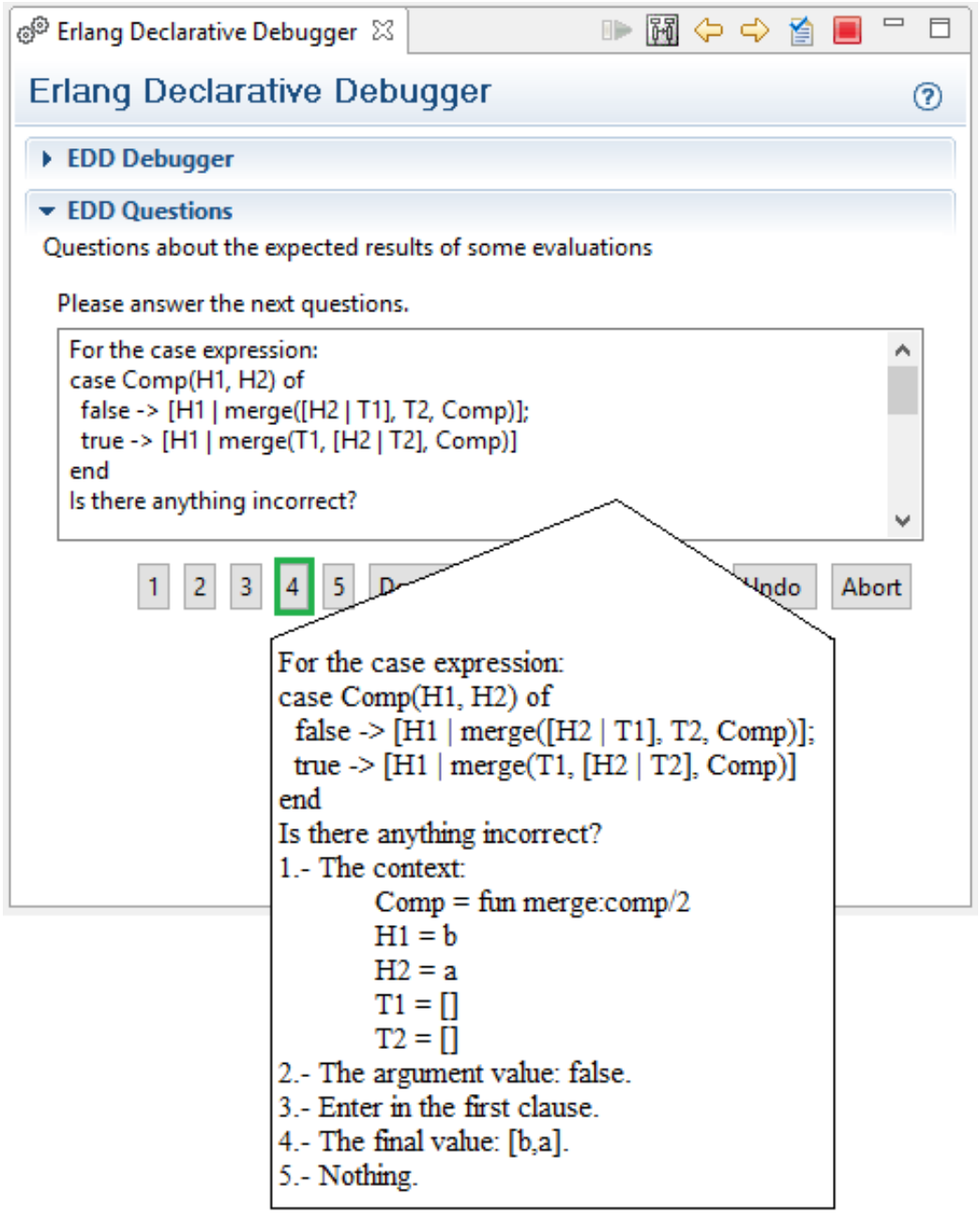


Ilustración 6-18 Primera pregunta del proceso de depuración secuencial por Zoom con la respuesta a dar

En la segunda pregunta, que podemos ver en la Ilustración 6-19, nos preguntan si la expresión mostrada es correcta, en nuestro caso lo es, por lo que respondemos con Yes.

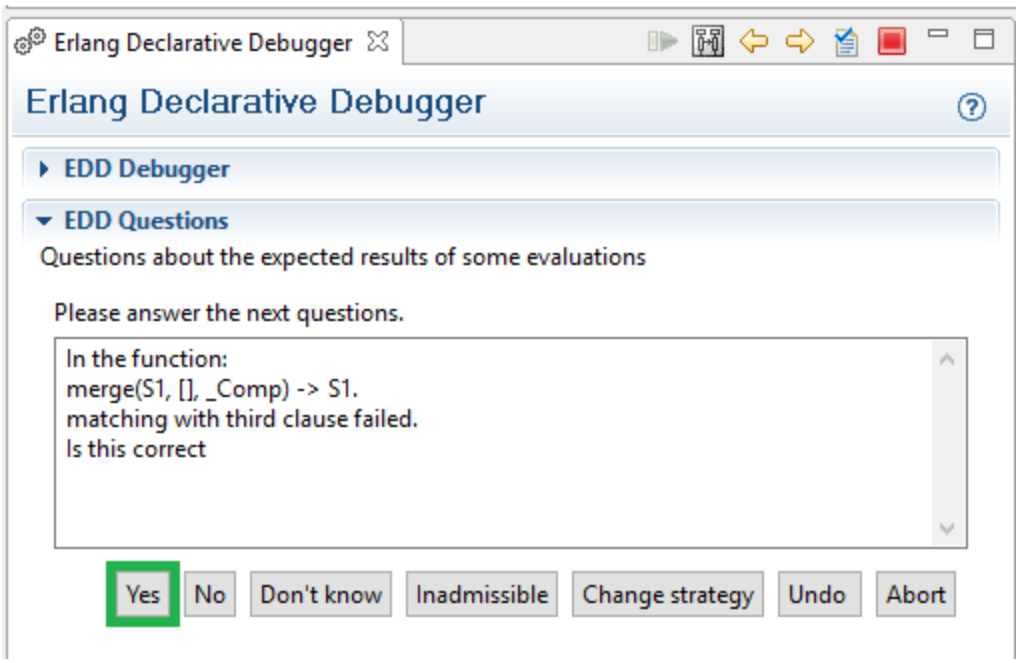


Ilustración 6-19 Segunda pregunta del proceso de depuración secuencial exhaustivo con la respuesta a dar.

En la tercera pregunta, que podemos ver en la Ilustración 6-20, nos preguntan si la expresión mostrada es correcta, en nuestro caso lo es, por lo que respondemos con Yes.

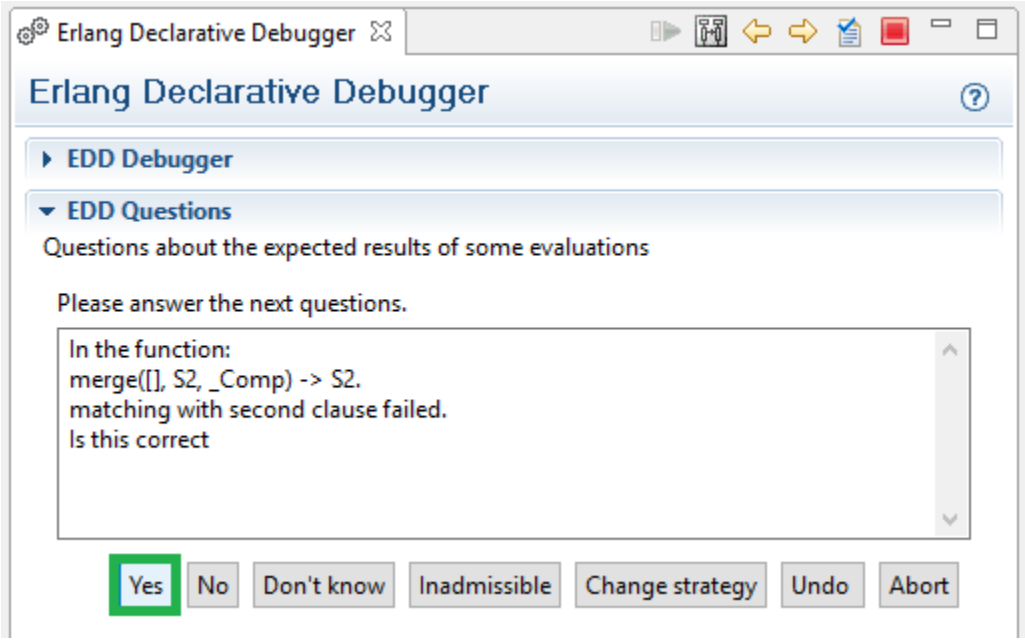


Ilustración 6-20 Tercera pregunta del proceso de depuración secuencial exhaustivo con la respuesta a dar

En la cuarta pregunta, que podemos ver en la Ilustración 6-21, nos preguntan si la expresión mostrada es correcta, en nuestro caso no lo es, puesto que la opción para el valor falso es errónea.

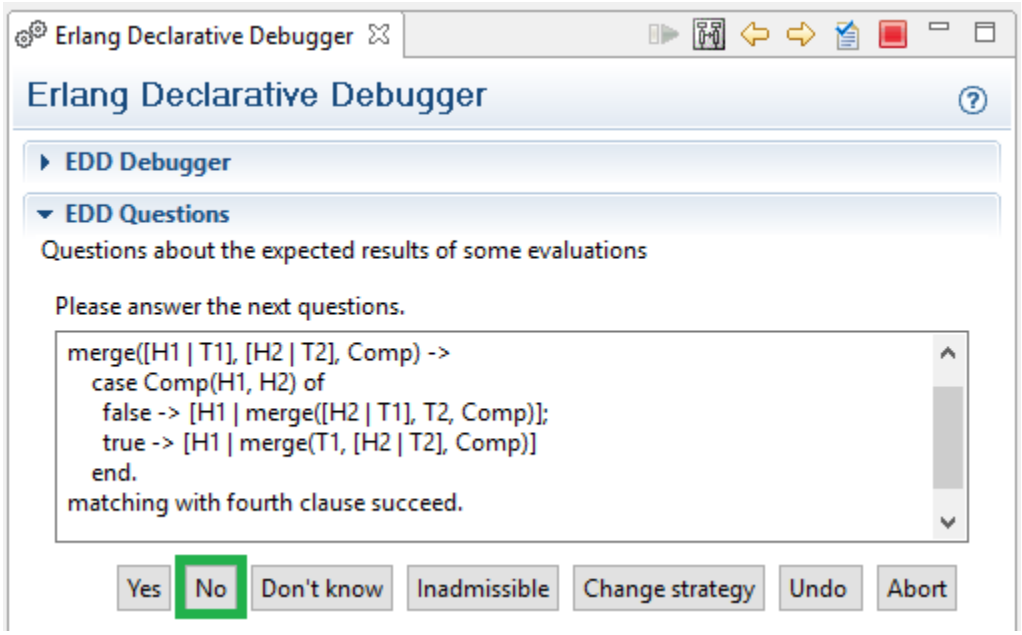


Ilustración 6-21 Cuarta pregunta del proceso de depuración secuencial exhaustivo con la respuesta a dar.

Tras responder a las preguntas realizadas por el depurador este encontrará la causa del error y se nos informará a través de un pop-up, tal y como se muestra en la *Ilustración 6-22*, donde se nos comunicará donde se produce el error y se terminará el proceso.

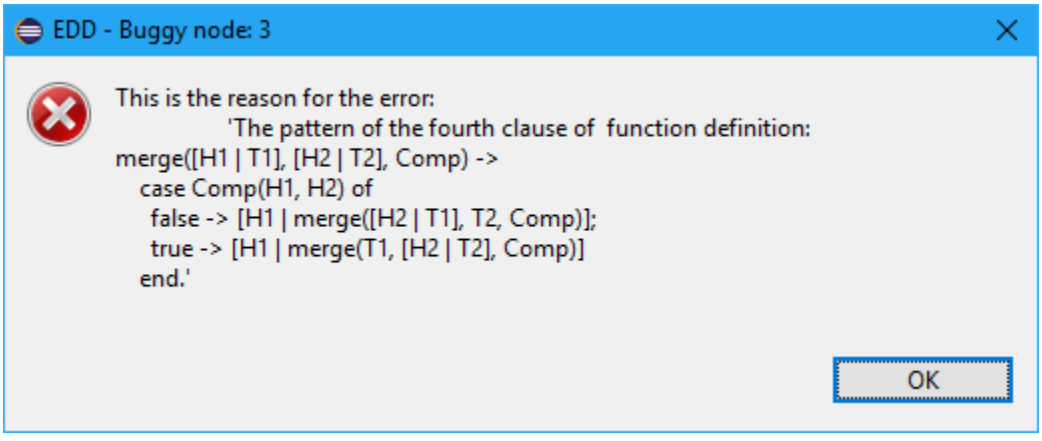


Ilustración 6-22 Diálogo que informa que se ha encontrado el error en la depuración secuencial por zoom

6.3 Depuración concurrente

Para realizar una depuración concurrente se habrán de rellenar los datos de entrada de la ventana, que corresponden al programa que se quiere probar y la función con la que se va a depurar, en la *Ilustración 6-23* podemos observar la ventana con los datos de ejemplo rellenos.

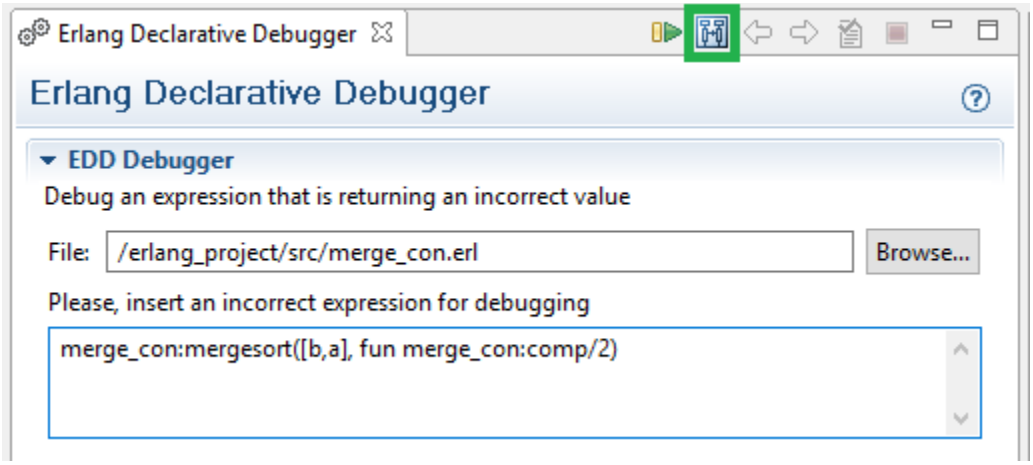


Ilustración 6-23 Ventana principal con datos de ejemplo para el proceso de depuración concurrente.

Una vez introducidos los datos se debe pulsar el botón de inicialización de la depuración concurrente, destacado en la *Ilustración 6-23*, que iniciará el proceso de depuración:

- Al igual que en la depuración secuencial se mostrará en la ventana principal "**Erlang Declarative Debugger**" una sección con una pregunta a responder tal y como podemos ver en la *Ilustración 6-24*.

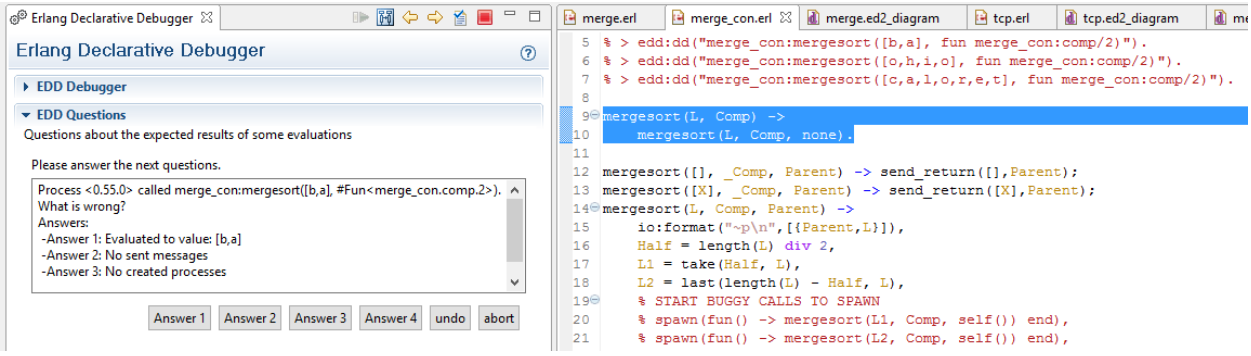


Ilustración 6-24 Ventana principal con la primera pregunta dada por EDD en la depuración concurrente, junto con el código destacado relativo a esta..

- También se mostrará el diagrama de secuencia generado por el proceso en la ventana "**Erlang Sequence Diagram**", que se puede observar en la *Ilustración 6-25*.

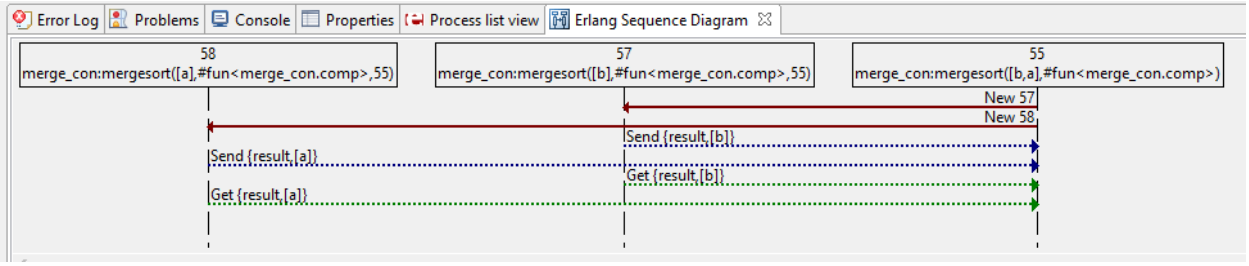


Ilustración 6-25 Diagrama de secuencia generado en el proceso de depuración concurrente.

- Se mostrará el árbol de creación de Procesos en la ventana "**EDD Tree View**", como se muestra en la *Ilustración 6-26*.

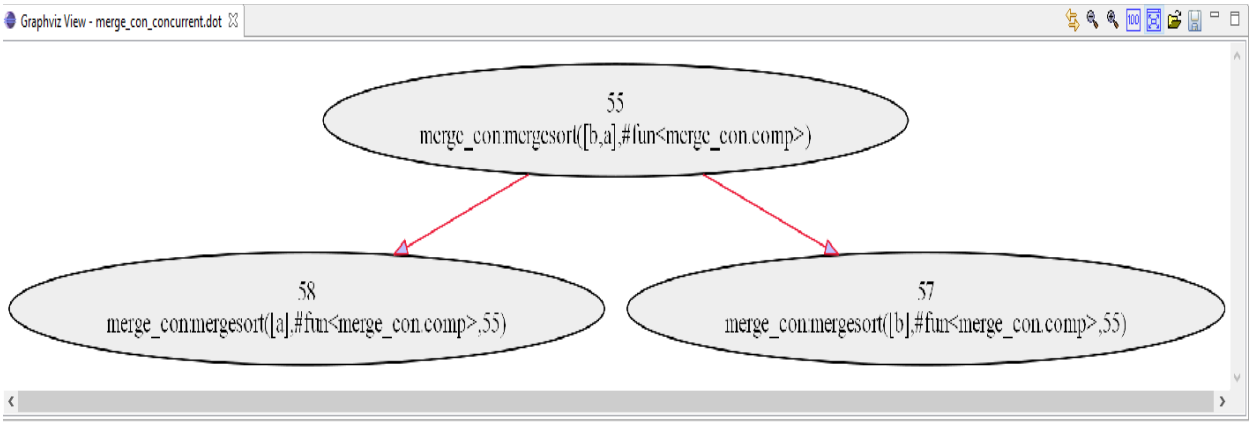


Ilustración 6-26 Árbol de procesos generado en el proceso de depuración concurrente.

Para proseguir con la depuración se habrá de contestar a las diferentes preguntas que envía el depurador, las cuales se podrán responder a través de dos ventanas:

- **Erlang Declarative Debugger**: Al igual que en la depuración secuencial, se mostrará una pregunta en la ventana principal del depurador, junto con las posibles respuestas a estas y los botones para responderlas. Esta pregunta irá cambiando a medida que se van respondiendo, hasta el momento en que se encuentre el error. En la *Ilustración 6-27*, podemos ver esta ventana con una pregunta a realizar que podemos contestar pulsando el botón correspondiente a la respuesta a dar.

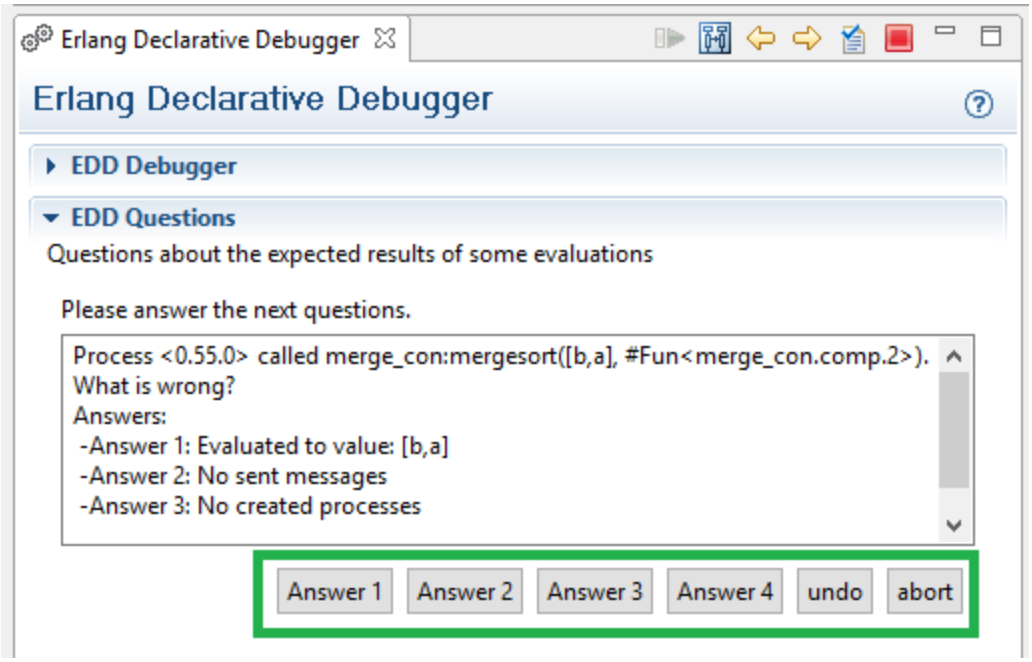


Ilustración 6-27 Ventana principal durante el proceso de depuración concurrente.

- **Diagrama**: Al hacer click sobre un mensaje del diagrama de secuencia se nos abrirá una ventana de diálogo con una serie de preguntas, a las cuales el usuario podrá contestar si estas no han sido contestadas con anterioridad. Las preguntas que han sido respondidas tendrán el icono de check en sus pestañas y los botones para responder a su pregunta bloqueados. En la *Ilustración 6-28* podemos ver un ejemplo de dialogo de preguntas mostrado al pulsar sobre el mensaje *Send{result,[b]}*, que podemos observar en el rótulo.

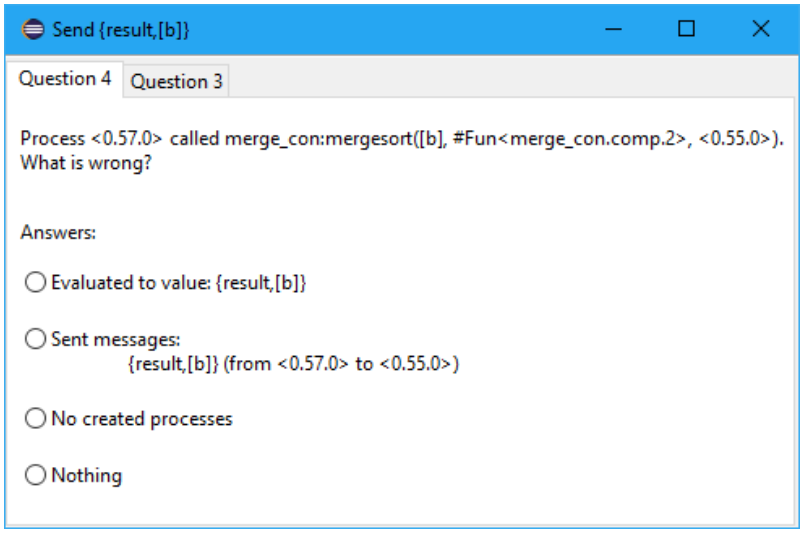


Ilustración 6-28 Ejemplo de ventana de dialogo de preguntas del diagrama de secuencia.

En nuestro caso para encontrar el error bastará con contestar las preguntas con la misma repuesta que se muestran en la *Ilustración 6-29*, la *Ilustración 6-30*, la *Ilustración 6-31*, la *Ilustración 6-32*, la *Ilustración 6-33*, la *Ilustración 6-34* y en la *Ilustración 6-35* de cualquiera de las maneras que se han explicado con antelación.

En la primera pregunta, que podemos ver en la *Ilustración 6-29*, nos preguntan cuál de los datos que se muestran para la ejecución inicial es incorrecto, siendo en nuestro caso la segunda opción, puesto el resultado de la ordenación del array [b, a] debería ser [a, b].

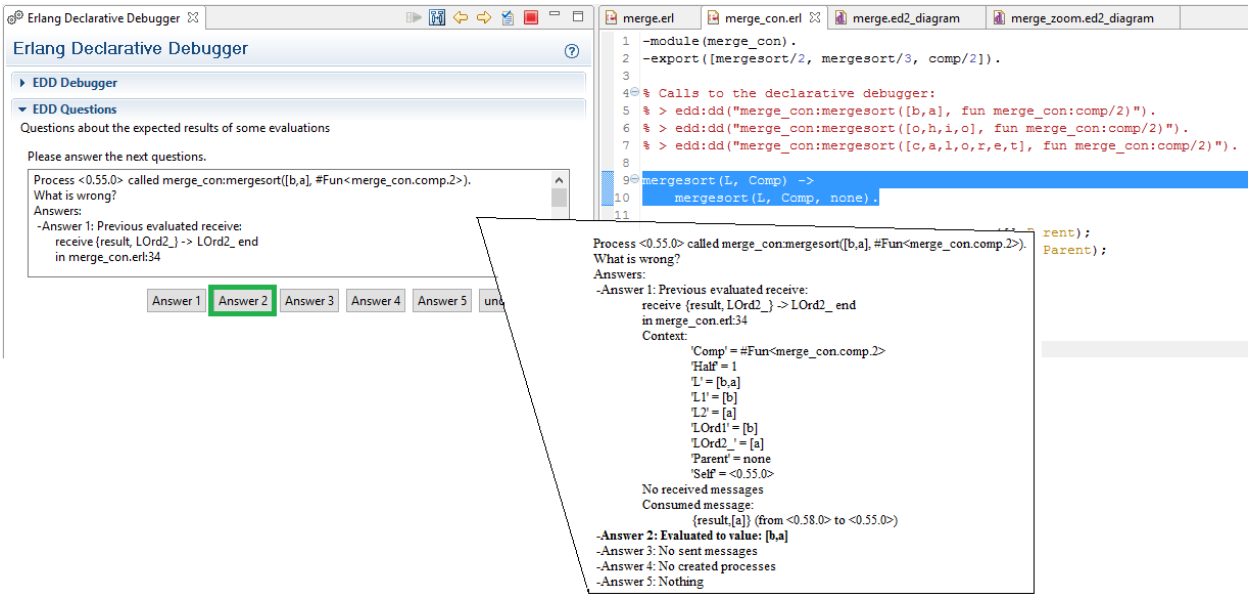


Ilustración 6-29 Primera pregunta del proceso de depuración concurrente con la respuesta a dar.

En la segunda pregunta, que podemos ver en la *Ilustración 6-30*, nos preguntan cuál de la siguiente información para la ejecución de la función **merge([a], [], #Fun<merge_con.comp 2>)** es incorrecta, siendo en nuestro caso la quinta opción, puesto que tanto los datos, como el proceso y resultado son correctos.

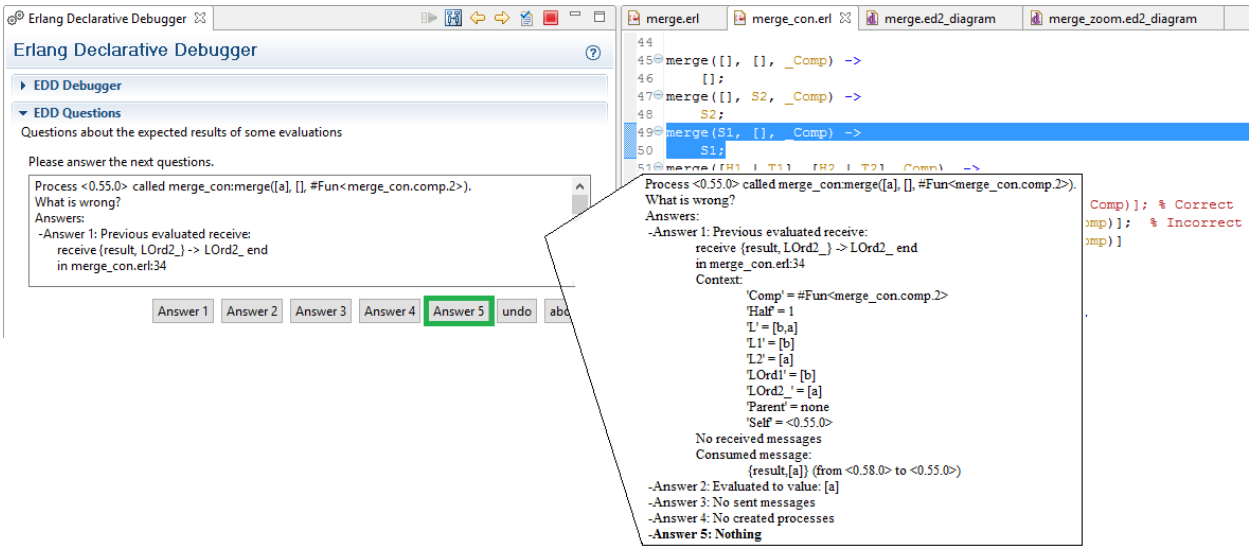


Ilustración 6-30 Segunda pregunta del proceso de depuración concurrente con la respuesta a dar.

En la tercera pregunta, que podemos ver en la *Ilustración 6-31*, nos preguntan cuál de la siguiente información para la ejecución de la función **send_return([b, a], none)** es incorrecta, siendo en nuestro caso la quinta opción, puesto que tanto los datos, como el proceso y resultado son correctos.

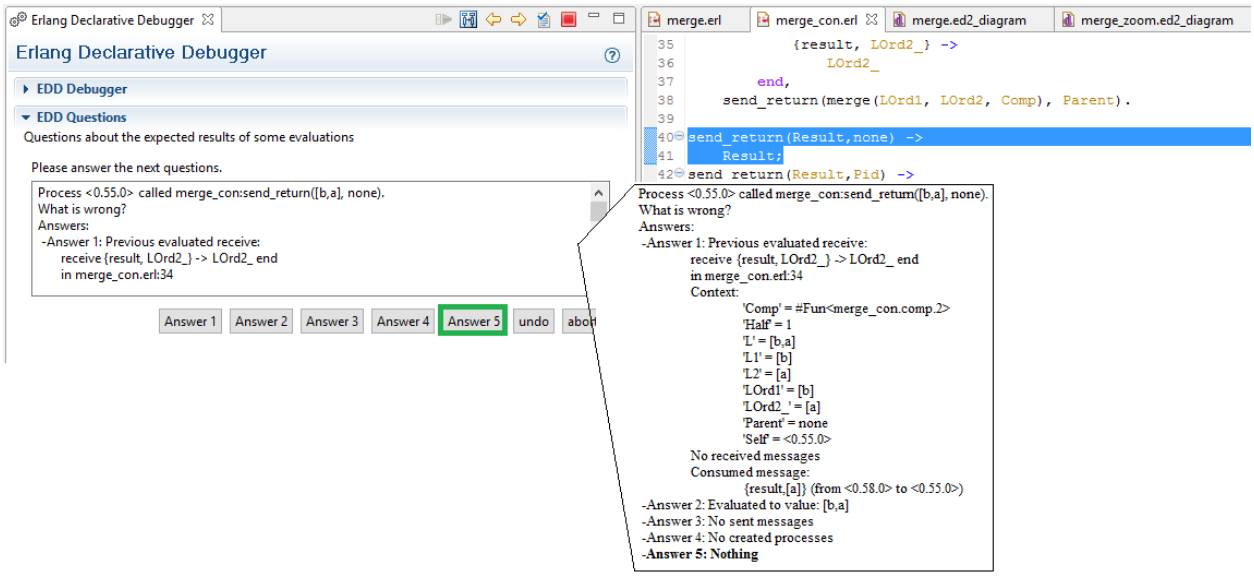


Ilustración 6-31 Tercera pregunta del proceso de depuración concurrente con la respuesta a dar.

En la cuarta pregunta, que podemos ver en la *Ilustración 6-32*, nos preguntan cuál de la siguiente información para la ejecución de la función **mergesort([b, a], #Fun<merge_con.comp 2>, none)** es incorrecta, siendo en nuestro caso la segunda opción, puesto que el resultado esperado de ordenar el arraya [b, a] debería ser [a, b].

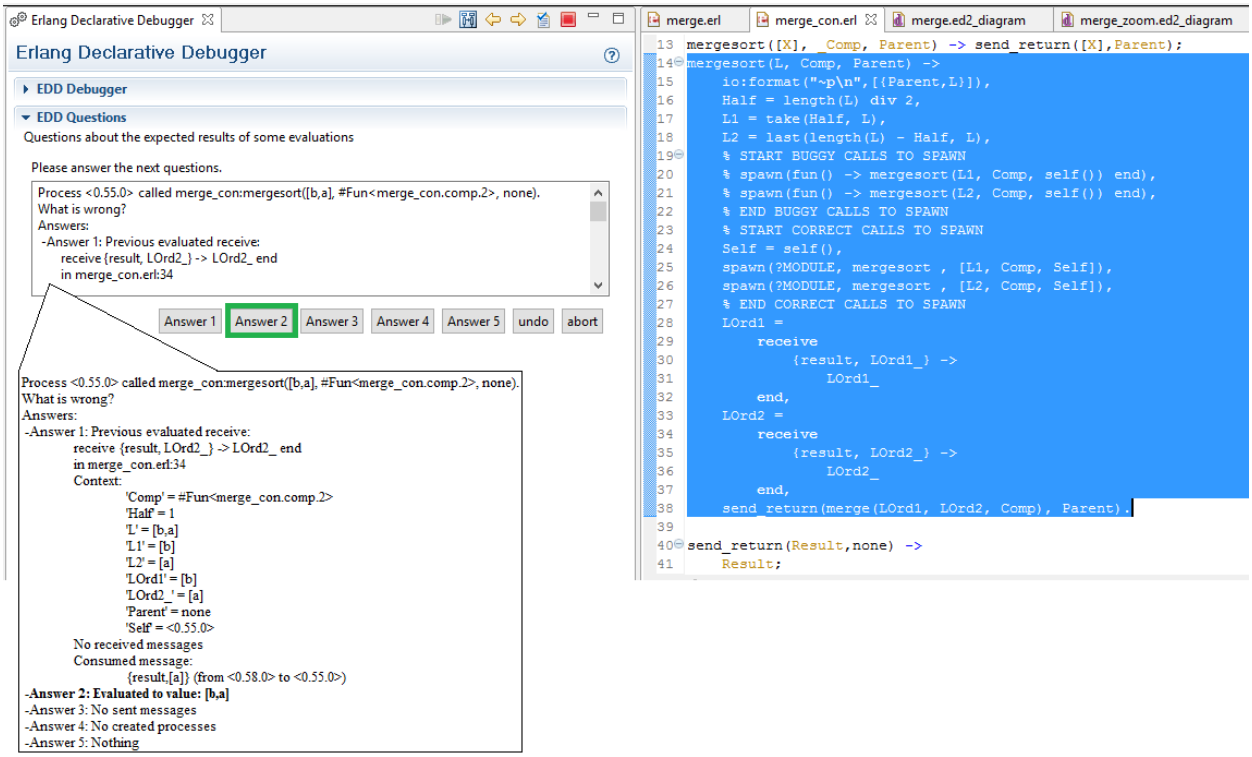


Ilustración 6-32 Cuarta pregunta del proceso de depuración concurrente con la respuesta a dar.

En la quinta pregunta, que podemos ver en la *Ilustración 6-33*, nos preguntan cuál de la siguiente información para la ejecución de la función **comp(b, a)** es incorrecta, siendo en nuestro caso la quinta opción, puesto que tanto los datos, como el proceso y resultado son correctos.

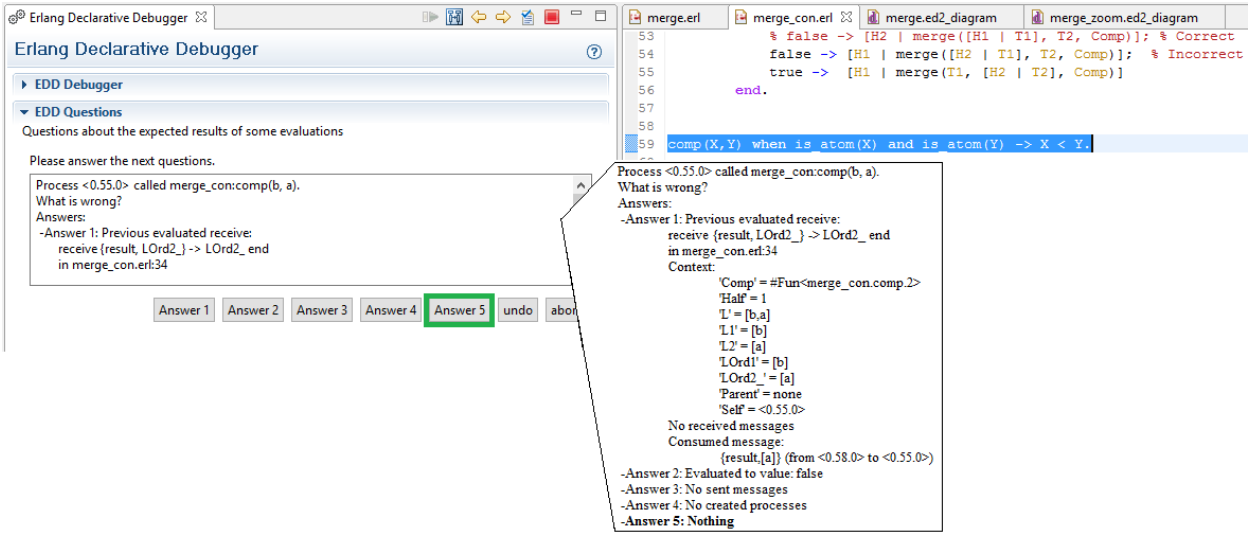


Ilustración 6-33 Quinta pregunta del proceso de depuración concurrente con la respuesta a dar.

En la sexta pregunta, que podemos ver en la *Ilustración 6-34*, nos preguntan cuál de la siguiente información para la ejecución de la función **merge([b], [a], #Fun<merge_con.comp 2>)** es incorrecta, siendo en nuestro caso la segunda opción, puesto que la unión ordenada de los arrays [b] y [a] debería ser [a ,b].

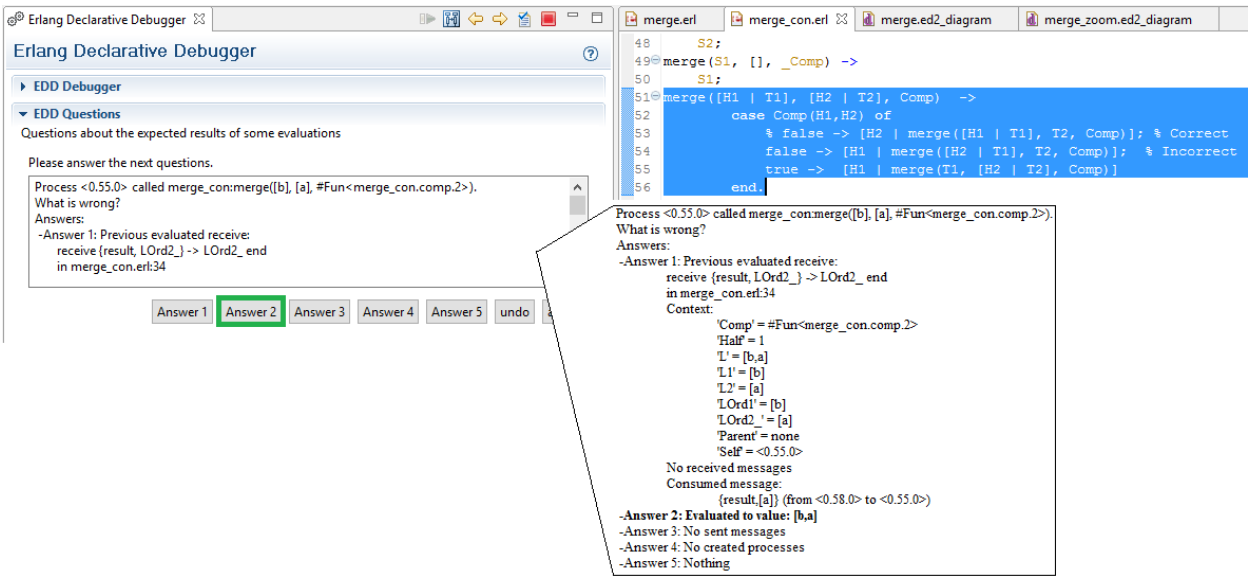


Ilustración 6-34 Sexta pregunta del proceso de depuración concurrente con la respuesta a dar.

En la séptima pregunta, que podemos ver en la Ilustración 6-35, nos preguntan cuál de la siguiente información para la recepción de datos es incorrecta, siendo en nuestro caso la séptima opción, puesto que tanto los datos, como el proceso y resultado son correctos.

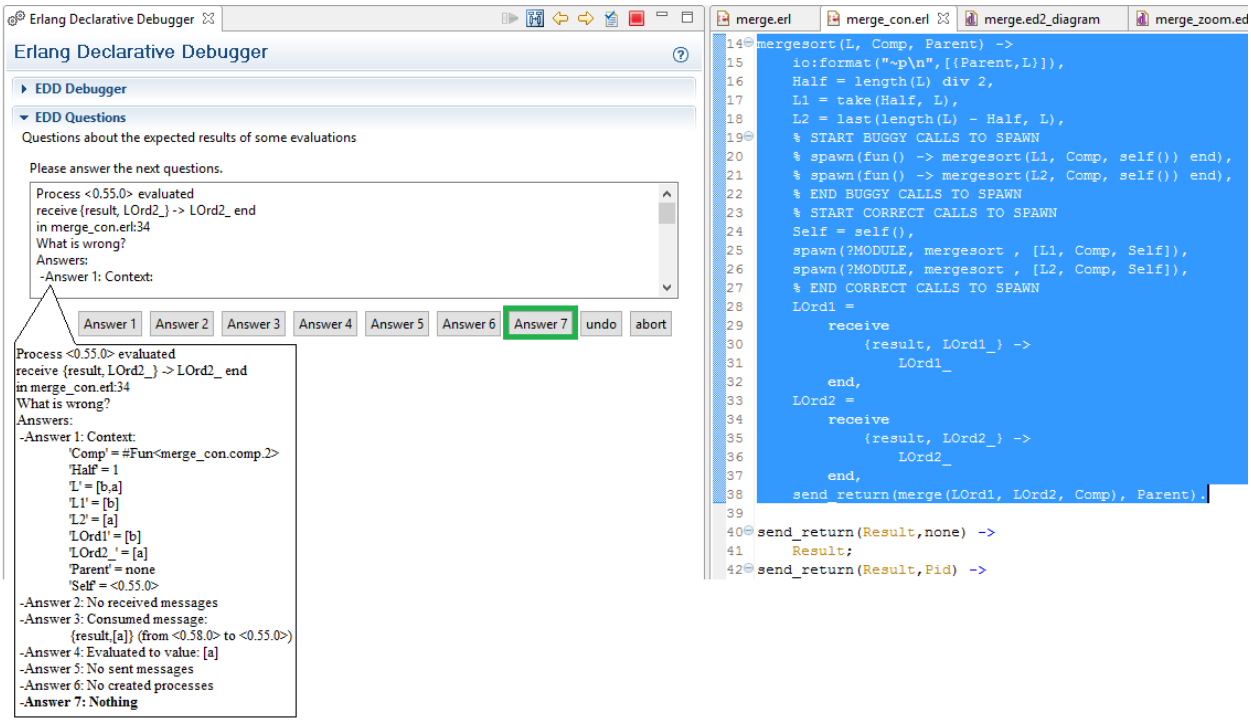


Ilustración 6-35 Septima pregunta del proceso de depuración concurrente con la respuesta a dar.

Tras responder las preguntas se mostrará el mensaje de la *Ilustración 6-36*, donde se nos comunica que se ha encontrado el punto donde ocurre el error.

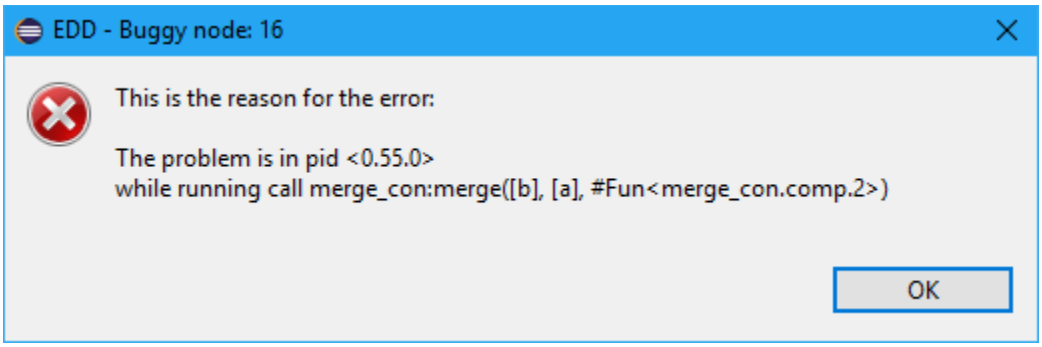


Ilustración 6-36 Dialogo que informa que se ha encontrado el error en la depuración concurrente.

7 Conclusiones y trabajo futuro

A lo largo del presente documento hemos visto las diferentes mejoras realizadas sobre la herramienta E-EDD, y el uso que esta hace sobre EDD.

Con este proyecto hemos conseguido mejorar la usabilidad de E-EDD al incorporar la interactividad con el usuario con el árbol de preguntas en la depuración declarativa secuencial. Con esta mejora logramos agilizar el proceso de depuración al poder el usuario ir directamente a las preguntas que piense que son más significativas para encontrar la causa del error.

También hemos añadido a E-EDD la opción de realizar un proceso de depuración declarativa sobre programas Erlang concurrentes, que ha resultado en un valioso añadido pues la concurrencia es una de las características más importantes de Erlang. Con ello hemos facilitado el uso y mejorando el entendimiento de la información del proceso de la depuración concurrente declarativa, que se realizaba antes por línea de comandos en el depurador EDD, gracias a la información que se muestran en los diferentes gráficos del proceso, así como el interfaz con el que se responden las distintas preguntas.

7.1 Trabajo futuro

Como toda aplicación todavía tiene espacio para mejorar, ya sea mejorando sus procesos, aumentando su usabilidad e interactividad con el usuario o extendiendo su funcionabilidad. Entre las diversas mejoras que se pueden realizar al *plugin* están:

- Añadir apartados de ayuda a cada ventana del proyecto, con la explicación de su funcionamiento y su utilidad. Con ello conseguiríamos aumentar la usabilidad de la herramienta, pues ante una duda sobre la herramienta el usuario podría simplemente acceder a la ayuda para resolverla.
- Encontrar una manera de mostrar el árbol de preguntas desarrollado por la depuración concurrente de una manera sencilla, amigable y la cual permita interactuar con el usuario para responder a estas preguntas, tal y como sucede con el árbol de preguntas de la depuración secuencial. Esto permitiría agilizar el proceso de depuración a la par que el usuario puede observar el progreso realizado en la depuración.
- Mezclar el proceso de depuración declarativa con test unitarios, permitiendo con ello probar los programas de manera declarativa.

8 Conclusions and future work

Throughout this paper we have seen the different improvements made in the plugin E-EDD, and the use that this tool does of EDD.

With this project, we have improved the usability of E-EDD with the incorporation of user interaction in the debugging tree of the sequential declarative debugging process. With this feature, we speed up the debugging process, because the user can go directly to the questions that he thinks are more significant to find the error.

Also, we have added to E-EDD the option to perform declarative debugging process in concurrent Erlang programs, which is a great improvement because the concurrency is one of the more important features of Erlang. With that we have facilitated the use and improved the way to understand the information of the concurrent declarative debugging process, thanks to the different graphs of the process, which show the information of the process, as well as the graphic interface that this plugin give us to answer the different questions of the debugger.

8.1 Future work

As all applications, this plugin can be improved, improving his process, increasing its usability and user interactivity or expanding his functionality. The different improvements we highlight the following:

- Adding help subjects to all project views, with the explication of its function and its utility. With this we improve the usability of this plugin, because if the user has doubts about the tool, he can access to this help for resolve it.
- Finding a way to show all the information of the questions tree generated in the concurrent debugging process in a friendlier way, that allows the user to interact with the tool to answer the different questions of the debugger, like in the questions tree of sequential debugging process. With this improvement, we speed up the debugging process and the user can see the progress of the debugging.
- Mixing the declarative debugging process with unit test, allowing for testing a program through the answers giving during the debugging process.

Bibliografía

[1] Eclipse:
https://eclipse.org

[2] Eclipse API:
http://help.eclipse.org

[3] Eclipse EMF
https://eclipse.org/modeling/emf/

[4] Eclipse GMF
http://www.eclipse.org/gmf-tooling/

[5] Eclipse JFace
https://wiki.eclipse.org/JFace

[6] Eclipse SWT
https://www.eclipse.org/swt/

[7] EMF2GV plugin
http://emftools.tuxfamily.org/update/

[8] Erlang:
https://www.erlang.org/

[9] Ericsson AB. The Jinterface Package (2015)
http://www.erlang.org/doc/apps/jinterface/jinterface_users_guide.html

[10] Erlide
http://erlide.org/

[11] Erlide plugin
http://download.erlide.org/update

[12] GitHub EclipseGraphViz
https://github.com/abstratt/eclipsegraphviz

[13] GitHub EDD
https://github.com/tamarit/edd

[14] GitHub E-EDD
https://github.com/jsanchezp/e-edd

[15] GitHub E-EDD 2.0
https://github.com/RubenM13/E-EDD-2.0

[16] Graphviz:
http://www.graphviz.org/

[17] Graphviz Plugin
http://download.erlide.org/update/graphviz

[18] Java
http://www.oracle.com/technetwork/java/index.html

[19] Learn you some Erlang
http://learnyousomeerlang.com/content

[20] UMLGraph Página oficial
http://www.umlgraph.org/

Documentación

[D1] Sánchez Pedroza, Joel (2015) E-EDD: integración en Eclipse del depurador declarativo para Erlang EDD. [Trabajo fin de Máster]
http://eprints.ucm.es/34526

[D2] R. Caballero, E. Martin-Martin, A. Riesco, and S. Tamarit. EDD: A Declarative Debugger for Sequential Erlang Programs (2014) [en línea] Informe técnico SIC02/14. Dpto. Sistemas Informáticos y Computación, Universidad Complutense de Madrid.
http://maude.sip.ucm.es/~adrian/files/tacas14.pdf

[D3] R. Caballero, E. Martin-Martin, A. Riesco, and S. Tamarit: A Zoom-Declarative Debugger for Sequential Erlang Programs.Science of Computer Programming 110:104-118.
http://maude.sip.ucm.es/~adrian/files/zoom_debugging.pdf

[D4] R. Caballero, E. Martin-Martin, A. Riesco, and S. Tamarit. Declarative Debugging of Concurrent Erlang Programs (Extended version). Technical Report SIC-03/16, Dpto. Sistemas Informáticos y Computación, Universidad Complutense de Madrid, July 2016.
http://maude.sip.ucm.es/~adrian/files/tr_03_16.pdf

[D5] Riesco Rodríguez, Adrián. Depuración declarativa y verificación heterogénea en Maude. Tesis (Doctor en Informática). Madrid, España. Universidad Complutense de Madrid, Facultad de Informática, Depto. de Sistemas Informáticos y Computación,
http://maude.sip.ucm.es/~adrian/files/thesis.pdf

Anexo A - Clases

En este apartado se detallará la información de las clases más importantes que se utilizan para el funcionamiento del *plugin*, separadas en diversos apartados donde se mostrarán en cada uno con un diagrama de clases con las clases a exponer en el apartado. Para cada clase se mostrará una pequeña descripción de su funcionabilidad, e información de los métodos y atributos más importantes de estos.

A.1 Proceso de depuración

En este apartado se describirán las principales clases que se utilizan en el proceso de depuración, y las cuales se muestran en el diagrama de clases de la *Ilustración A-1*.

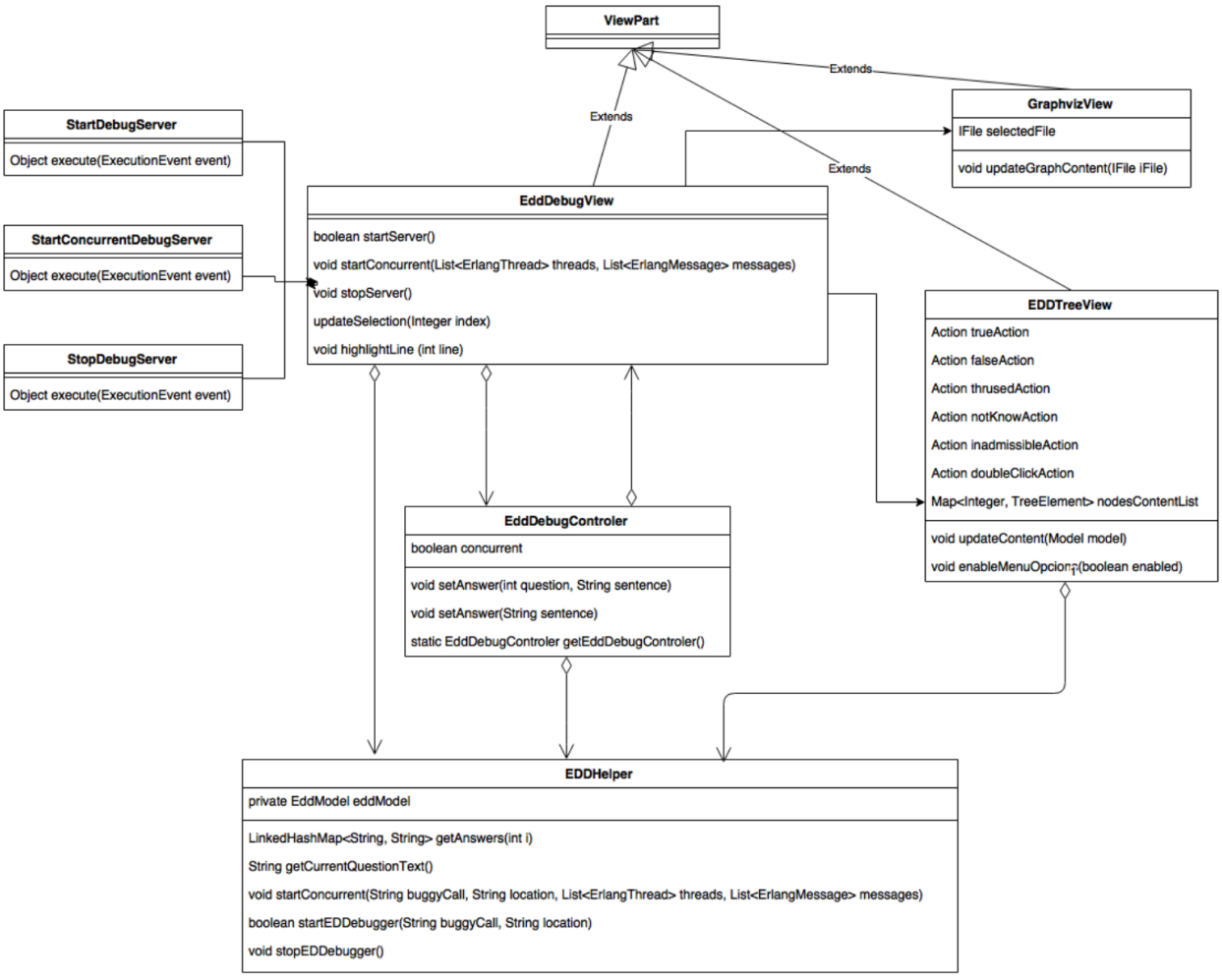


Ilustración A-1 Diagrama de clases del proceso de depuración.

A.1.1 EddDebugView

Esta clase se encarga de la visualización de la vista principal. A esta clase se le ha quitado la operativa encargada de responder una pregunta y actualizar la información del panel, extrayéndolo en la clase **EddDebugControler**, para conseguir que se pueda responder desde diferentes sitios y que todas las vistas se actualicen.

Métodos:

- **boolean startServer()**: Se encarga de inicializar la depuración secuencial a través de los parámetros recogidos en la vista.
- **void startConcurrent(List<ErlangThread> threads, List<ErlangMessage> messages)**: Inicializa el proceso de depuración concurrente a través de los parámetros que recibe de los campos mostrados en la vista, y devuelve en los parámetros de entrada la información de los procesos creados y el paso de mensajes que ha habido entre ellos.
- **boolean stopServer()**: Detiene el proceso de depuración que se está llevando a cabo.
- **void updateSelection(Integer index)**: Actualiza la información de la ventana mostrando la información de la pregunta con el identificador que se pasa al método como parámetro de entrada.
- **void highlightLine (int line)**: Se encarga de destacar la línea, con el número del parámetro que se pasa en la entrada, al método del programa Erlang que se está depurando.

Atributos:

- **EddDebugControler controler**: Objeto que se encarga de gestionar la respuesta dada por el usuario durante proceso de depuración.
- **EDDHelper helper**: Objeto que se utiliza para inicializar, terminar y obtener toda la información sobre el proceso de depuración.

A.1.2 EddDebugControler

Esta clase define un objeto único, patrón Singleton, que se encarga de realizar todo el proceso común a realizar por las diferentes vistas cuando el usuario interactúa con estas para responder a una pregunta. Comunicando al depurador de si se debe cambiar de pregunta, en caso de que se haya respondido a una pregunta distinta de la actual, enviando la respuesta dada, comprobando si se ha terminado el proceso y llamando al proceso de refresco de las ventanas para que muestren la información actual.

Métodos:

- **void setAnswer(String sentence)**: Responde a la pregunta actual con la respuesta que se pasa como parámetro de entrada, comprueba si se ha terminado el proceso de depuración y comunica a las ventanas que se refresquen.
- **void setAnswer(int question, String sentence)**: Responde a la pregunta con el identificador y con la respuesta que se pasan como parámetros de entrada, comprueba si se ha terminado el proceso de depuración y comunica a las ventanas que se refresquen.

Atributos:

- **EDDHelper helper**: Objeto con el que se comunica la clase para comunicarse con el servidor Erlang, enviando a través de él las respuestas a las preguntas realizadas.
- **EDDebugView debugView**: Ventana principal de la aplicación que se utiliza para actualizar su información cuando se realiza una respuesta a una pregunta del proceso de depuración.
- **boolean concurrent**: Indica si el proceso de depuración es concurrente o no.
- **EddDebugControler controler**: Objeto de la misma clase que sigue el patrón Singelton para que sólo exista un objeto del mismo en la aplicación y pueda ser fácilmente obtenido por todos los objetos que necesiten.

A.1.3 EddHelper

Clase encargada de gestionar la comunicación entre el apartado Erlang y las vistas de Eclipse.

Métodos:

- **boolean startEDDebugger(String buggyCall, String location)**: Inicializa el proceso de depuración secuencial con la llamada y el archivo que reciben los parámetros de entrada.
- **void startConcurrent(String buggyCall, String location, List<ErlangThread> threads, List<ErlangMessage> messages)**: Inicializa el proceso de depuración concurrente, y devuelve en las listas que recibe en los parámetros de entrada la información de los procesos creados y el paso de mensajes que ha habido entre ellos.
- **void stopEDDebugger():** Para la ejecución del proceso de depuración actual.

- **String buildDOT(boolean isZoom, Integer highlightNode, boolean cw)**: Crea la descripción de un diagrama de árbol de la relación que tienen las preguntas entre sí en la depuración secuencial.
- **String buildDOT(List<ErlangThread> list)**: Crea la descripción de un diagrama de árbol de parentesco de los procesos que se pasan en la entrada.

  Atributos:
- **EddModel eddModel**: Objeto que contiene la información del proceso de depuración actual.
- **Erlang2Java e2j**: Objeto que se encarga de gestionar la comunicación entre Java y Erlang.

A.1.4 EddTreeView

Clase encargada de la visualización en un árbol de ficheros de las preguntas realizadas en el proceso de depuración secuencial, y que permite responderá estas a través del mismo.

Métodos:

- **void updateContent(Model model)**: Pasa el modelo con las preguntas a mostrar en el árbol.
- **void enableMenuOpcions(boolean enabled)**: Habilita/Deshabilita las opciones de interactividad de la ventana.

  Atributos:
- **Action trueAction**: Acción a realizar cuando se da en el botón o en la opción del menú contextual para responder a una pregunta del depurador secuencial con valor “Cierto”.
- **Action falseAction**: Acción a realizar cuando se da en el botón o en la opción del menú contextual para responder a una pregunta del depurador secuencial con valor “Falso”.
- **Action thrusedAction**: Acción a realizar cuando se da en el botón o en la opción del menú contextual para responder a una pregunta del depurador secuencial con valor “Seguro”.
- **Action notKnowAction**: Acción a realizar cuando se da en el botón o en la opción del menú contextual para responder a una pregunta del depurador secuencial con valor “No Sabe”.
- **Action inadmissibleAction**: Acción a realizar cuando se da en el botón o en la opción del menú contextual para responder a una pregunta del depurador secuencial con valor “Inadmisible”.

- **Action doubleClickAction**: Acción a realizar cuando se da doble click sobre una pregunta del árbol.

A.1.5 GraphizView

Clase que muestra en su ventana un diagrama que se construye a través de Graphiz.

Métodos:

- **void updateGraphContent(IFile iFile)**: Muestra en la vista el diagrama que se encuentra en el fichero que se pasa como parámetro de entrada.

Atributos:

- **IFile selectedFile**: Archivo que contiene el diagrama actual que se muestra en la vista.

A.1.6 StartDebugServer

Clase encargada de realizar la acción de iniciar el proceso de depuración secuencial.

A.1.7 StartConcurrentDebugServer

Clase encargada de realizar la acción de iniciar el proceso de depuración concurrente.

A.1.8 StopDebugServer

Clase encargada de realizar la acción de terminar el proceso de depuración secuencial.

A.2 Comunicación Java/Erlang

En este apartado se describirán las principales clases que se utilizan en la comunicación entre el *plugin* de Eclipse y el servidor EDD, y las cuales se muestran en el diagrama de clases de la *Ilustración A-2*.

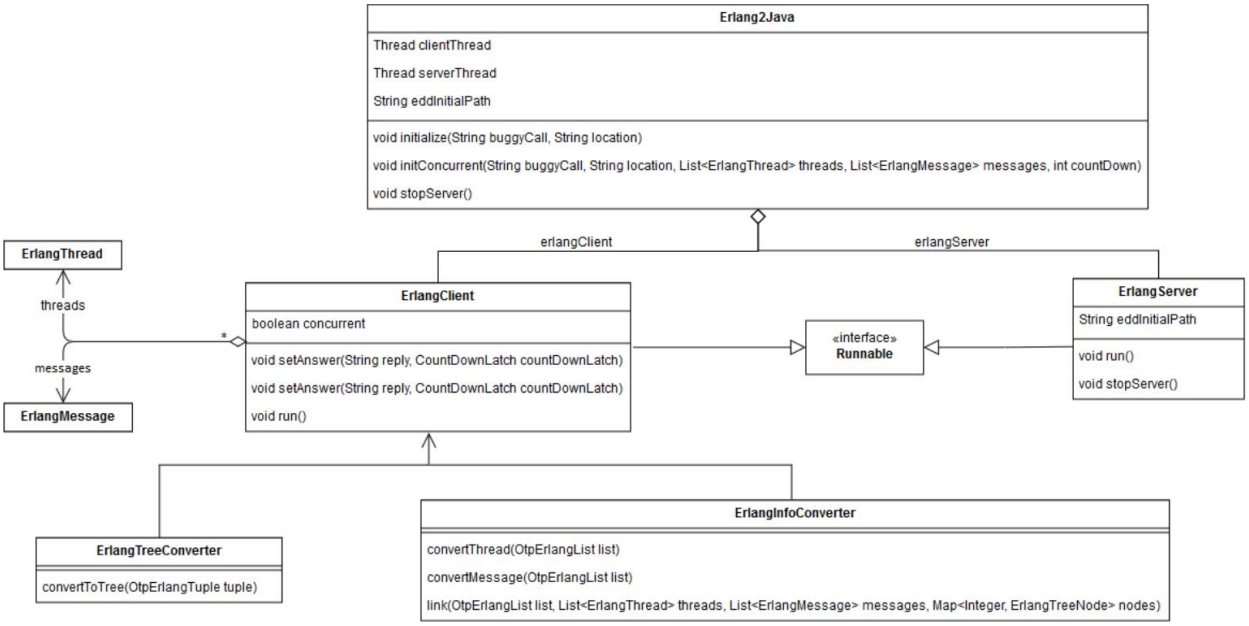


Ilustración A-2 Diagrama de clases del proceso de comunicación Eclipse/EDD.

A.2.1 Erlang2Java

Se encarga de gestionar los objetos que se comunican directamente con la aplicación EDD.

Métodos:

- **void initialize(String buggyCall, String location)**: Inicia los objetos que se encargan de gestionar EDD e inicializa la depuración secuencial con los parámetros de entrada que se reciben.
- **void initConcurrent(String buggyCall, String location, List<ErlangThread> threads, List<ErlangMessage> messages, int countDown)**: Inicializa los objetos que se encargan de gestionar EDD e inicializa la depuración concurrente con los parámetros de entrada que se reciben.
- **void stopServer()**: Termina la ejecución del proceso de depuración y del servidor EDD.
- Atributos:
- **Thread clientThread**: Hilo donde se ejecuta el objeto **ErlangClient**.
- **Thread serverThread**: Hilo donde se ejecuta el objeto **ErlangServer**.
- **ErlangClient erlangClient**: Objeto con el que se realiza el paso de mensajes con EDD.

- **ErlangServer erlangServer**: Objeto que se encarga de inicializar y apagar la aplicación EDD.
- **String eddInitialPath**: Ruta donde se encuentra el proyecto EDD.

A.2.2 ErlangClient

Clase es la encargada de realizar la comunicación con EDD a través del paso de mensajes.

Métodos:

- **void setAnswer(String reply, CountDownLatch countDownLatch)**: Envía un mensaje con la réplica que se pasa como parámetro de entrada a la aplicación EDD.
- **void run()**: Proceso que se ocupa de recibir y traducir los mensajes recibidos por la aplicación EDD.

Atributos:

- **EddModel eddModel**: Modelo que contiene la información de las preguntas a realizar en el proceso de depuración.
- **boolean concurrent**: Indicador de si se está realizando un proceso de depuración concurrente o secuencial.
- **private List<ErlangThread> threads**: Información relativa de los procesos creados durante la depuración concurrente.
- **private List<ErlangMessage> messages**: Lista con la información de los mensajes que se han intercambiado entre los procesos durante la depuración concurrente.
- **private ErlangTreeInfo tree**: Árbol que contiene la información de las preguntas que se realizan en la depuración concurrente.

A.2.3 ErlangServer

Clase encargada gestionar la inicialización y el apagado del servidor EDD con el que se realiza la depuración del código.

Métodos:

- **void run()**: Inicia la aplicación Erlang para empezar la depuración.
- **void stopServer()**: Finaliza la ejecución de la aplicación EDD.

Atributos:

- **String eddInitialPath**: Ruta donde se encuentra el proyecto EDD.

A.2.4 ErlangInfoConverter

Clase encargada de traducir la información de la depuración concurrente obtenida de EDD, transformándola en objetos EDDInfo.

Métodos:

- **convertThread(OtpErlangList list):** extrae la información de los procesos que se han generado en la depuración de la lista pasada por Erlang, convirtiéndolos en una lista de objetos **ErlangThread**.
- **convertMessage(OtpErlangList list):** extrae la información de los mensajes que se han generado en la depuración de la lista pasada por Erlang, convirtiéndolos en una lista de objetos **ErlangMessage**.
- **link(OtpErlangList list, List<ErlangThread> threads, List<ErlangMessage> messages, Map<Integer, ErlangTreeNode> nodes):** Utilizando la información pasada por Erlang en el objeto list, relaciona entre sí los procesos, mensajes, nodos que ha comunicado con anterioridad EDD.

A.2.5 ErlangTreeConverter

Clase encargada de traducir la información del árbol de preguntas de la depuración concurrente obtenida de EDD, obteniendo su correspondiente objeto **ErlangTreeInfo**.

Métodos:

- **convertToTree(OtpErlangTuple tuple):** Extrae del objeto que se pasa como parámetro de entrada la información del árbol preguntas en el proceso de depuración de EDD, guardando la información correspondiente en los objetos pertinentes, y devolviéndolo todo junto en un objeto de la clase **ErlangTreeInfo**.

A.3 Diagrama de secuencia

En este apartado se describirán las principales clases que se utilizan en el proceso relativo al diagrama de secuencia de la depuración concurrente, y las cuales se muestran en el diagrama de clases de la *Ilustración A-3*.

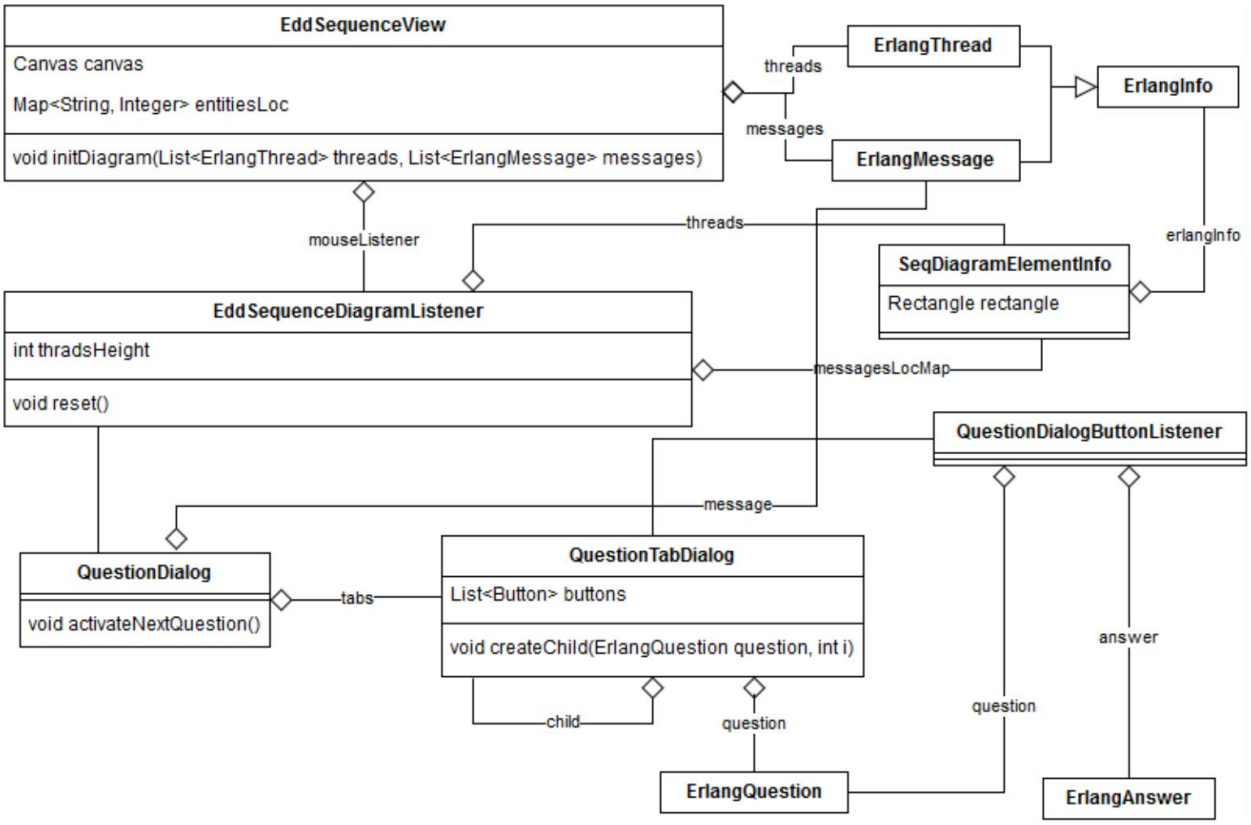


Ilustración A-3 Diagrama de clases del diagrama de secuencia.

A.3.1 EddSequenceView

Clase se encarga de generar y mostrar el diagrama de secuencia resultante de la depuración concurrente.

Métodos:

- **void initDiagram(List<ErlangThread> threads, List<ErlangMessage> messages)**: Dibuja el diagrama de secuencia definido por los procesos y diagramas que se pasan por parámetro.

Atributos:

- **List<ErlangThread> threads**: Lista de procesos que se muestran en el diagrama.
- **List<ErlangMessage> messages**: Lista de mensajes que se muestran en el diagrama.
- **Canvas canvas**: Panel donde se dibuja y se interactúa con el diagrama de secuencia.
- **EddSequenceDiagramListener mouseListener**: Gestiona la interacción con el diagrama de secuencia.

- **Map<String, Integer> entitiesLoc**: Mapa con la coordenada X de la línea de secuencia de cada proceso.

A.3.2 EddSequenceDiagramListener

Esta clase se encarga de monitorizar el ratón en el diagrama de secuencia cambiando el icono del ratón o mostrando las preguntas correspondientes cuando se pasa por encima o se hace click sobre las zonas interactivas.

Métodos:

- **void reset()**: Vacía los atributos y cierra la ventana que abre la clase.

Atributos:

- **Map<Integer, SeqDiagramElementInfo<ErlangMessage>> messagesLocMap**: Lista de mensajes que hay en el diagrama de secuencia guardados según su altura.
- **List<SeqDiagramElementInfo<ErlangThread>> threads**: Lista de procesos que hay en el diagrama de secuencia.
- **int thradsHeight**: Altura que separa los procesos de los mensajes.

A.3.3 SeqDiagramElementInfo

Clase que enlaza un recuadro de texto que se encuentra en el diagrama con su información.

Atributos:

- **Rectangle rectangle**: Área del diagrama de secuencia que se enlaza con la información que contiene la clase.
- **ErlangInfo erlangInfo**: Información de la depuración correspondiente a un mensaje o un proceso, que se puede observar en el diagrama.

A.3.4 QuestionDialog

Esta clase se encarga de mostrar una ventana de dialogo con las preguntas relativas a un mensaje del diagrama de secuencia.

Métodos:

- **void activateNextQuestion()**: Habilita la siguiente pregunta a contestar por el diálogo.

Atributos:

- **ErlangMessage message**: Mensaje del diagrama de secuencia por el cual se ha abierto el diálogo.

- **List<QuestionTabDialog> tabs**: Lista de pestañas con las diferentes preguntas que contiene el diálogo.

A.3.5 QuestionTabDialog

Pestaña que muestra el contenido de una pregunta junto a las diferentes respuestas que se pueden dar para contestarla.

Métodos:

- **void createChild(ErlangQuestion question, int i):** Crea una nueva pestaña con la pregunta asociada a la respuesta de la pregunta de la pestaña.

Atributos:

- **QuestionDialog dialog**: Diálogo en el que se encuentra la pestaña.
- **List<Button> buttons**: Lista de botones que correspondes a las distintas respuestas.
- **QuestionTabDialog child:** Pestaña asociada a la respuesta dada a la pregunta.
- **ErlangQuestion question**: Pregunta que se quiere responder en esta pestaña.

A.3.6 QuestionDialogButtonListener

Acción que se realiza cuando se ejecuta cuando se pulsa sobre una respuesta en la pestaña de QuestionTabDialog.

Atributos:

- **QuestionDialog dialog**: Diálogo en el que se encuentra el botón.
- **ErlangAnswer answer**: Respuesta enlazada al botón que escucha el objeto.
- **ErlangQuestion question**: Pregunta a la que se responde pulsando el botón.

A.4 Información

En este apartado se describirán las principales clases que contienen la información que se utiliza, y las cuales se muestran en el diagrama de clases de la *Ilustración A-4*.

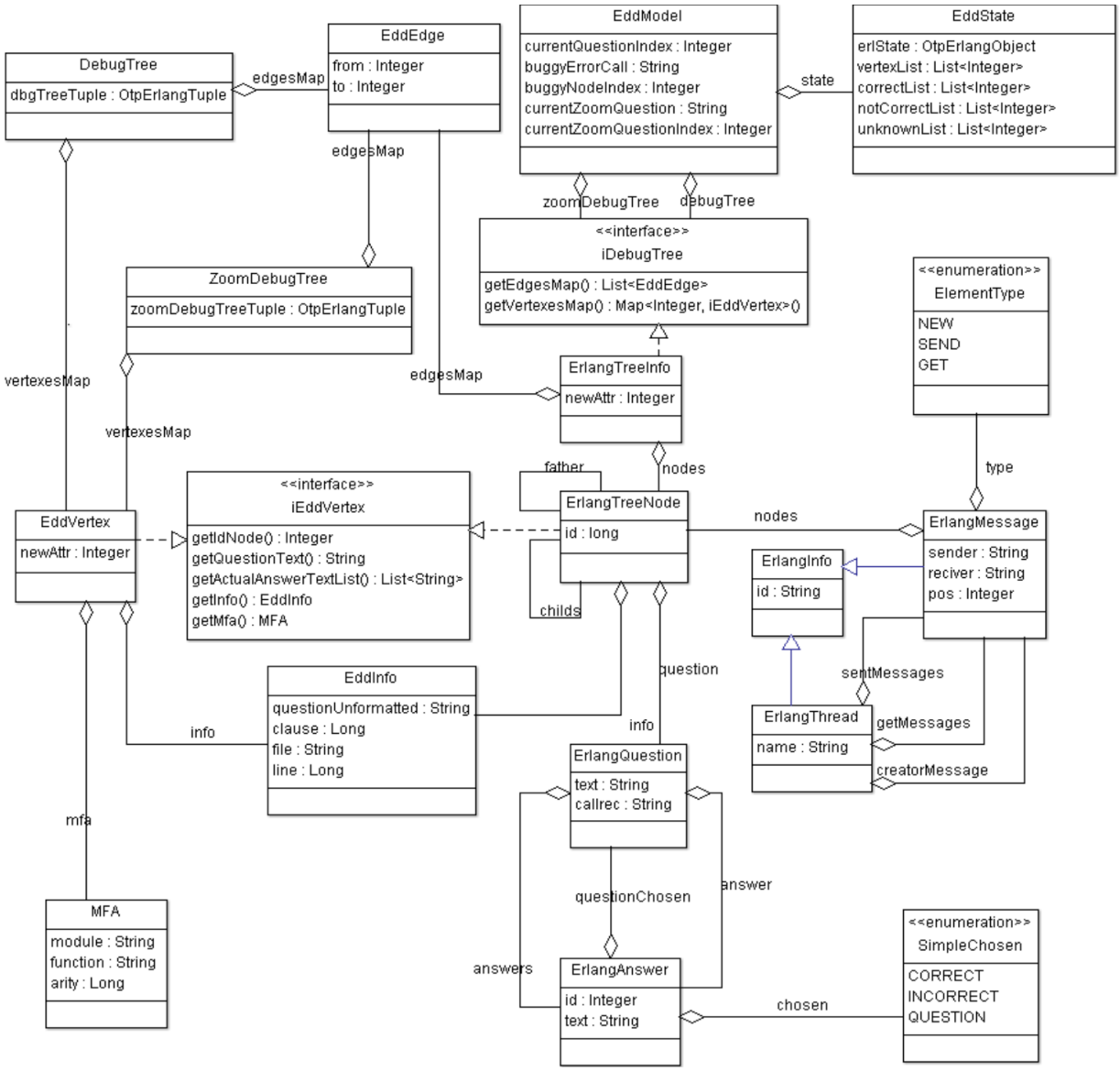


Ilustración A-4 Diagrama de clases de la información del plugin.

A.4.1 EddInfo

Clase que contiene la información general de la información dada por la depuración concurrente.

Atributos:

- **String id:** Identificador del elemento.

A.4.2 ErlangThread

Clase que contiene la información de un proceso generado durante la depuración concurrente.

Atributos:

- **String name**: Nombre del proceso.
- **List<ErlangMessage> sentMessages**: Lista de mensajes que ha enviado el proceso.
- **List<ErlangMessage> getMessages**: Lista de mensajes que ha recibido el proceso.
- E**rlangMessage creatorMeessage:** Mensaje que creó el proceso.

A.4.3 ErlangMessage

Posee la información relativa a un mensaje, así como las preguntas a realizar cuando se selecciona el mensaje en el diagrama de secuencia.

Atributos:

- **String sender**: Identificador del proceso que emite el mensaje.
- **String reciver**: Identificador del proceso al que se envía el mensaje.
- **int pos**: Posición relativa del mensaje en el servidor Erlang.
- **List<ErlangTreeNode> nodes**: nodos del árbol que contienen las preguntas relacionadas con este mensaje.
- **ElementType type**: Tipo del mensaje, descrito en el enumerado ElementType:
  - **NEW**: crea un nuevo proceso.
  - **SENT**: envío de un mensaje.
  - **GET**: recepción de un mensaje.

A.4.4 EddModel

Modelo que guarda la información actual de la depuración.

Atributos:

- **int currentQuestionIndex:** Identificador de la pregunta actual.
- **String buggyErrorCall:** Texto que se recibe al terminar la operación de depuración.
- **int buggyNodeIndex:** Identificador de la pregunta que ocasiona el fallo.
- **int currentZoomQuestion:** Identificador de la pregunta actual del árbol de depuración zoom.

- **int currentZoomQuestionIndex:** Identificador de la pregunta actual del árbol de depuración.
- **idebugTree debugTree:** Árbol que contiene la información de la depuración secuencial o concurrente.
- **idebugTree zoomDebugTree:** Árbol que contiene la información de la depuración con zoom.

A.4.5 iDebugTree

Interfaz creado para reutilizar la gestión de las preguntas, de tal manera que se tenga que hacer las mínimas modificaciones en el código para añadir la funcionalidad de la depuración concurrente.

Métodos:

- **Map<Integer, iEddVertex> getEdgesMap():** devuelve un mapa que contiene la información de las conexiones que hay entre las preguntas.
- **List<EddEdge> getVertexMap():** devuelve un mapa con información de las preguntas a realizar, guardadas por su identificador.

A.4.5.1 DebugTree

Árbol que contiene la información de las preguntas a hacer en la depuración secuencial.

A.4.5.2 ZoomDebugTree

Árbol que contiene la información de las preguntas a hacer en la depuración secuencial con zoom.

A.4.5.3 ErlangTreeInfo

Contiene la información del árbol que contiene las preguntas de la depuración concurrente.

A.4.6 EddEdge

Clase que contiene la relación entre dos nodos:

Atributos:

- **int from**: Identificador del nodo padre.
- **int to**: Identificador del nodo hijo.

A.4.7 iEddVertex

Interfaz creado para reutilizar la gestión de las preguntas, de tal manera que se tenga que hacer las mínimas modificaciones en el código para añadir la funcionalidad de la depuración concurrente.

Métodos:

- **int getIdNode():** Obtiene el identificador del nodo.
- **String getQuestionText():** Obtiene el texto de la pregunta actual
- **List<String> getActualAnswerTextList():** Obtiene una lista con las posibles respuestas a la pregunta. Este parámetro sólo está relleno en la depuración concurrente pues sus respuestas varían entre pregunta y pregunta.
- **GetInfo():** Obtiene la información que enlaza la pregunta con el código

A.4.7.1 EddVertex

Información relativa a una pregunta de la depuración secuencial.

A.4.7.2 ErlangTreeNode

Contiene la información de un nodo del árbol de preguntas obtenido durante la depuración concurrente.

Atributos:

- **ErlangTreeNode father:** Nodo padre.
- **List<ErlangTreeNode> childs:** Lista de nodos hijos.
- **ErlangQuestion question:** Pregunta asociada al nodo.

A.4.8 EddInfo

Información que enlaza el código con una pregunta.

Atributos:

- **String file:** Archivo que contiene el código.
- **long line**: Número de la línea en la que se encuentra el código relativo a una pregunta.

A.4.9 ErlangQuestion

Contiene la información de una pregunta que se realiza en la depuración concurrente

Atributos:

- **String Text:** Texto de la pregunta.
- **List<ErlangAnswer> Answers**: Lista de posibles repuestas a la pregunta.
- **ErlangAnswer Answer**: Respuesta que ha dado el usuario a la pregunta.

A.4.10 ErlangAnswer

Contiene el texto y la operativa a ejecutar para la respuesta de una pregunta de la depuración concurrente.

Atributos:

- **String Text:** Texto de la respuesta.
- **SimpleChosen chosen:** Acción a realizar cuando se selecciona esta respuesta, descrito en el enumerado SimpleChosen.
  - **CORRECT**: Se ha de responder al servidor con correct.
  - **INCORRECT**: Se ha de responder al servidor con incorrect.
  - **QUESTION**: Se ha de realizar la pregunta enlazada con esta respuesta.
- **ErlangQuestion QuestionChosen**: Pregunta ha realizar si se responde esta pregunta.

Anexo B - Descripción técnica de los procesos

En este apartado se describirán los procesos más importantes que se realizan en el *plugin*, a través de diagramas de secuencia y su explicación de ellos.

B.1 Depuración secuencial

A continuación, se describirán los distintos procesos que se llevan a cabo durante la depuración secuencial.

B.1.1 Inicialización

En la *Ilustración B-1* se muestra el diagrama de secuencia del proceso que se realiza cuando se pulsa en la inicialización del proceso de depuración secuencial.

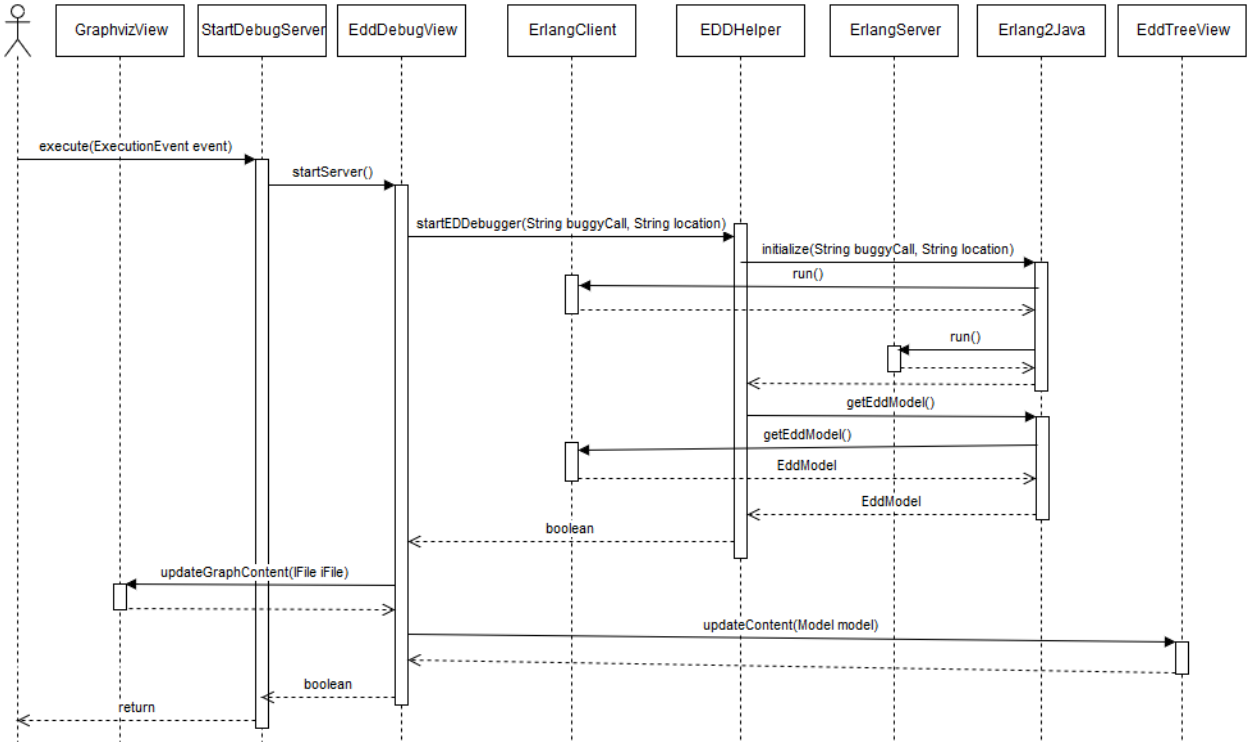


Ilustración B-1 Diagrama de secuencia de la inicialización de la depuración secuencial.

- Cuando el usuario inicializa el proceso de depuración secuencial, se ejecuta la clase **StartDebugServer**, el cual comprobará que los parámetros están rellenos, mostrando un mensaje de advertencia en caso de que no estén, e inicializando el proceso en caso afirmativo.

- Se llamará al método startServer de **EddDebugView** para inicializar el servidor, la vista y obtener la información correspondiente.
- **EddDebugView** pedirá que se inicialice el servidor a **EDDHelper**, el cual llamará al constructor de la clase **Erlang2Java** y le comunicará que ha de iniciarse el servidor Erlang para una depuración secuencial.
- **Erlang2Java** inicializará y ejecutará el cliente y el servidor Erlang, que se ocuparán de la ejecución, finalización y comunicación con el servidor Erlang, a través del envío y recepción de los mensajes explicados en el apartado [C.1 Depuración secuencial].
- Una vez ejecutado el servidor, **EddHelper** obtendrá de **ErlangClient** la información del proceso de depuración contenido en la clase **EddModel**.
- Se devolverá el control a **EDDebugView**, y este generará los archivos que describen los árboles de preguntas, y lo mostrará en la ventana **GraphvizView** y en la ventana de **EDDTreeView**.

B.1.2 Respuesta a la pregunta actual

En la *Ilustración B-2* se muestra el diagrama de secuencia del proceso de respuesta de la pregunta actual de la depuración secuencial.

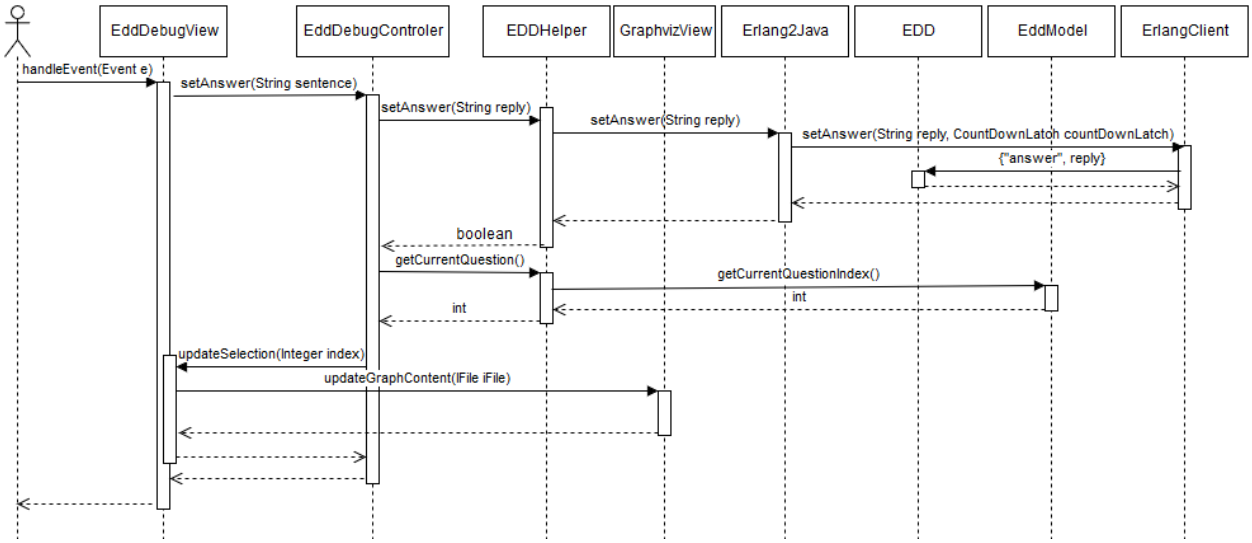


Ilustración B-2 Diagrama de Secuencia del proceso de respuesta a la pregunta actual en la depuración secuencial.

- Cuando el usuario selecciona una respuesta la vista envía un mensaje con la respuesta seleccionada a **EddDebugControler**.
  - Este reenvía la información a **EddHelper**.
  - Este reenvía la información a **Erlang2Java**
  - Este reenvía la información a **ErlangClient**
  - **ErlangClient** prepara el mensaje a enviar al servidor Erlang con la respuesta y los envía.
  - **EddHelper** comprobará si ya se ha encontrado el error y se lo devolverá.
  - Si se ha encontrado la causa del error **EddDebugControler** lo informará y se empezará el proceso de depuración exhaustivo, si no se estaba en él y el usuario así lo desea.
- Por último, **EddDebugControler** informará a la vista que debe de actualizar su contenido con el de una nueva pregunta y actualizará el árbol mostrado en la ventana **Graphviz View**.

B.1.3 Respuesta a una pregunta distinta a la actual

En la *Ilustración B-3* se muestra el diagrama de secuencia del proceso de respuesta de una pregunta distinta a la actual de la depuración secuencial, que se observa en la ventana principal.

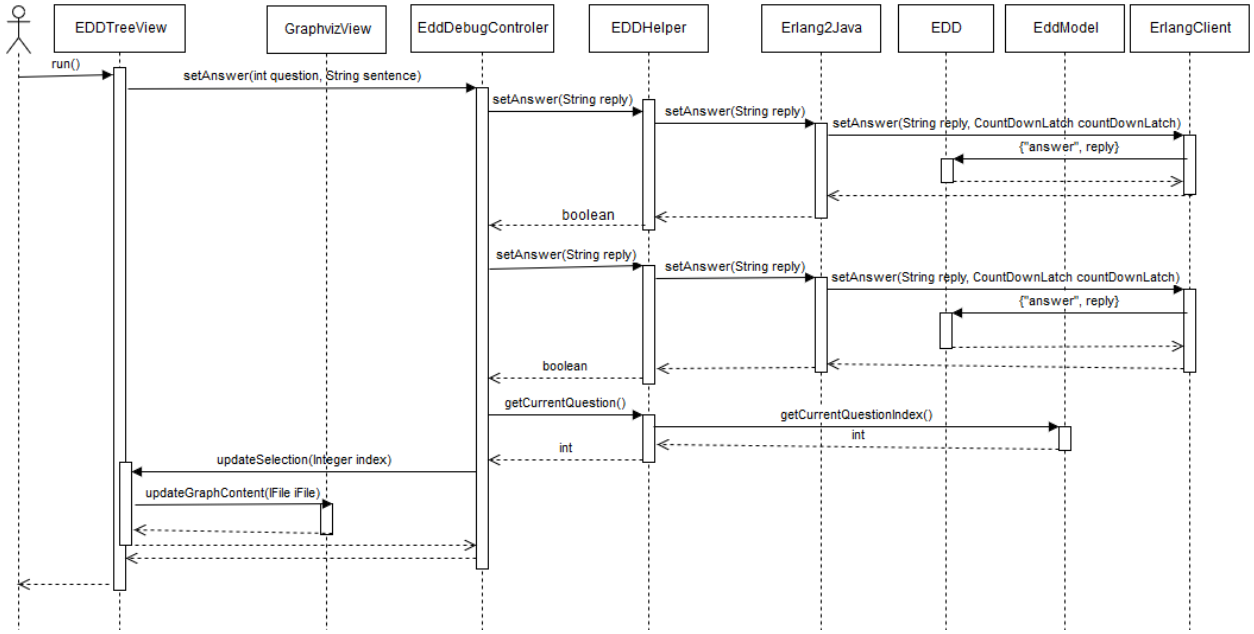


Ilustración B-3 Diagrama de Secuencia del proceso de respuesta a una pregunta distinta a la actual en la depuración secuencial.

- Cuando el usuario responde a una pregunta de **EDDTreeView**, se envía el número del nodo y la respuesta a **EddHelper**.
- **EddDebugControler** comprobará si la pregunta corresponde a la pregunta actual, y en caso negativo enviará el identificador a **EddHelper**.
  - Este reenvía la información a **Erlang2Java.**
  - Este reenvía la información a **ErlangClient**.
  - **ErlangClient** prepara el mensaje ha enviar al servidor Erlang con la respuesta y los envía, esperando el tiempo indicado para recibir la respuesta.
- **EddDebugControler** enviará la respuesta de la pregunta a **EddHelper**.
  - Este reenvía la información a **Erlang2Java**.
  - Este reenvía la información a **ErlangClient**.
  - **ErlangClient** prepara el mensaje a enviar al servidor Erlang con la respuesta y los envía, esperando el tiempo indicado para recibir la respuesta.
  - **EddHelper** comprobará si ya se ha encontrado el error y se lo devolverá.
  - Si se ha encontrado la causa del error **EddDebugControler** lo informará y se empezará el proceso de depuración exhaustivo, si no se estaba en él y el usuario así lo desea.
- Por último, **EddDebugControler** informará a la vista que debe de actualizar su contenido con el de una nueva pregunta.

B.2 Depuración concurrente

A continuación, se describirán los distintos procesos que se llevan a cabo durante la depuración concurrente.

B.2.1 Inicialización

En la *Ilustración B-4* se muestra el diagrama de secuencia del proceso que se realiza cuando se pulsa en la inicialización del proceso de depuración concurrente.

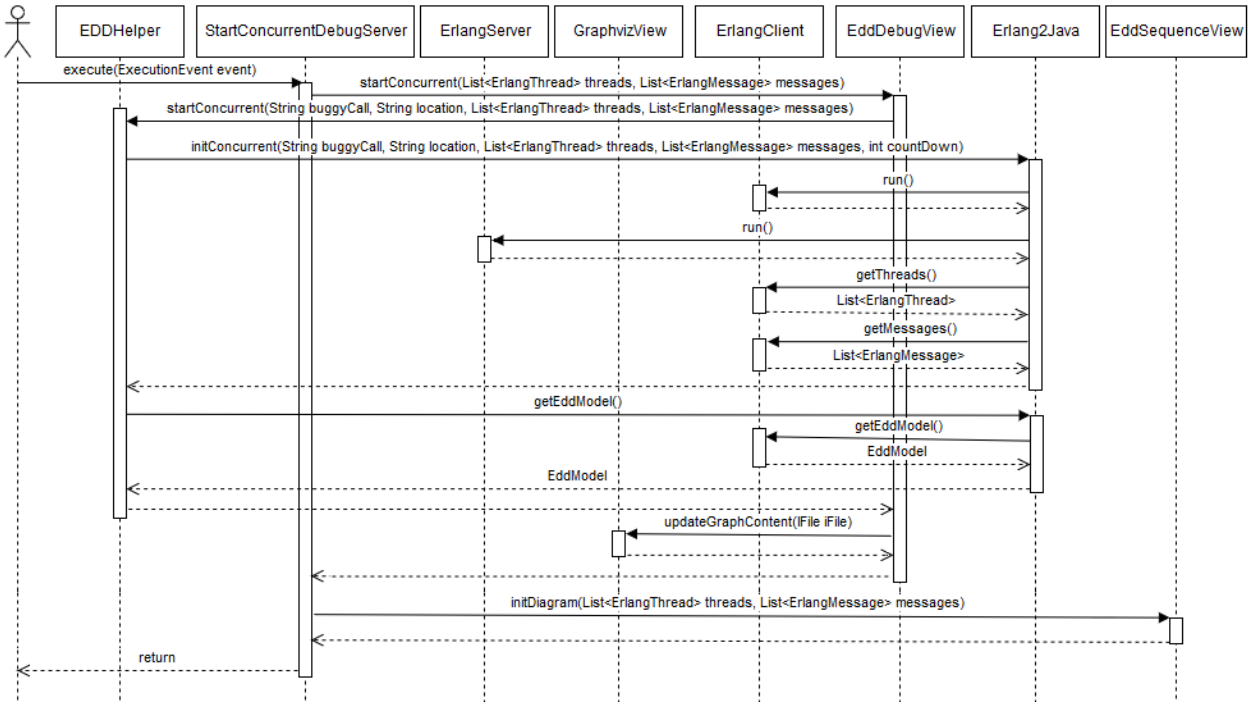


Ilustración B-4 Diagrama de secuencia de la inicialización de la depuración concurrente.

- Cuando el usuario inicializa el proceso de depuración concurrente, se ejecuta la clase **StartConcurrentDebugServer**, el cual comprobará que los parámetros están rellenos, mostrando un mensaje de advertencia en caso de que no estén, e inicializando el proceso en caso afirmativo.
- Se llamará al método **startConcurrent** de **EddDebugView** para inicializar el servidor, la vista y obtener la información correspondiente.
- **EDDHelper** llamará al constructor de la clase **Erlang2Java** y le comunicará que ha de iniciarse el servidor Erlang para una depuración concurrente.
- **Erlang2Java** inicializará y ejecutará el cliente y el servidor Erlang, que se ocuparán de la ejecución, finalización y comunicación con el servidor Erlang, a través del envío y recepción de los mensajes explicados en el apartado [C.2 Depuración concurrente].
- Una vez ejecutado el servidor se esperará para obtener los datos correspondientes a los procesos y los mensajes enviados en la depuración, los cuales serán obtenidos a través de mensajes enviados por EDD, de los que se extraerán su información con las clases **ErlangTreeConverter** y **ErlangInfoConverter** para convertirlos en los objetos que utilizaremos en la parte Java.

- Una vez ejecutado el servidor, **EddHelper** obtendrá de **ErlangClient** la información del proceso de depuración contenido en la clase **EddModel**.
- Se devolverá el control a **EDDebugView**, y este generará el archivo que describe el árbol de generación de procesos, y lo mostrará en la ventana **EDDThreadsTreeView**.
- Una vez terminada la generación de los diagramas se entregará el control a **StartConcurrentDebugServer**, el cual transmitirá la información obtenida a **EddSequenceView** para que genere el diagrama de secuencia correspondiente.

B.2.2 Diagrama de Secuencia

Para dibujar el diagrama de secuencia la clase **EddSequenceView**, que se realizará cuando se llame al método **initDiagram** o cuando se necesite refrescar la vista, utiliza un objeto **Canvas** sobre el que dibuja los procesos y mensajes que tiene guardados, siguiendo una serie de reglas de estilo para conseguir que se visualice como un diagrama de secuencia, tal y como se explica a continuación:

Primero empieza a dibujar las cabeceras de los procesos. Para ello, para cada elemento **ErlangThread** se dibuja un recuadro con el texto separados a una distancia proporcional al tamaño de la letra del último proceso. Este recuadro se envía junto a su **ErlangThread** al listener, para que sepa que esa zona es interactiva y que habrá que mostrar la información respectiva al proceso cuando se pulse sobre este.

Por último, se dibujan los mensajes, para cada mensaje se obtiene la línea de los procesos emisor y receptor y se conectan con una flecha con una separación proporcional al tamaño de la letra debajo del último mensaje, y se coloca sobre este el texto del mensaje. La ubicación del texto del mensaje junto con su información se envía a la clase **EddSequenceDiagramListener**, para que sepa que esa zona es interactiva y que habrá que mostrar las preguntas correspondientes cuando se haga click sobre esta.

Cuando se realiza una acción con el ratón el listener que pertenece a la clase **EddSequenceDiagramListener** comprueba si este se encuentra en una zona de interacción, con lo que modificará el icono que muestra el ratón, y si se ha hecho click sobre el proceso se mostrará su información y en caso de que sea sobre un mensaje se levantará si no hay otro diálogo de su tipo levantado, el diálogo de preguntas **QuestionDialog**, al que se pasará el mensaje seleccionado para que muestre las preguntas relativas a este, las cuales cada una estarán en una pestaña generada

por la clase **QuestionTabDialog** y serán controladas por los diferentes listener, **QuestionDialogButtonListener**, enlazados a cada botón de las respuestas, lo cual ejecutará el proceso que se describe en el apartado [B.2.4 Respuesta a una pregunta distinta a la actual].

B.2.3 Respuesta a la pregunta actual

En la *Ilustración B-5* se muestra el diagrama de secuencia del proceso de respuesta de la pregunta actual de la depuración concurrente.

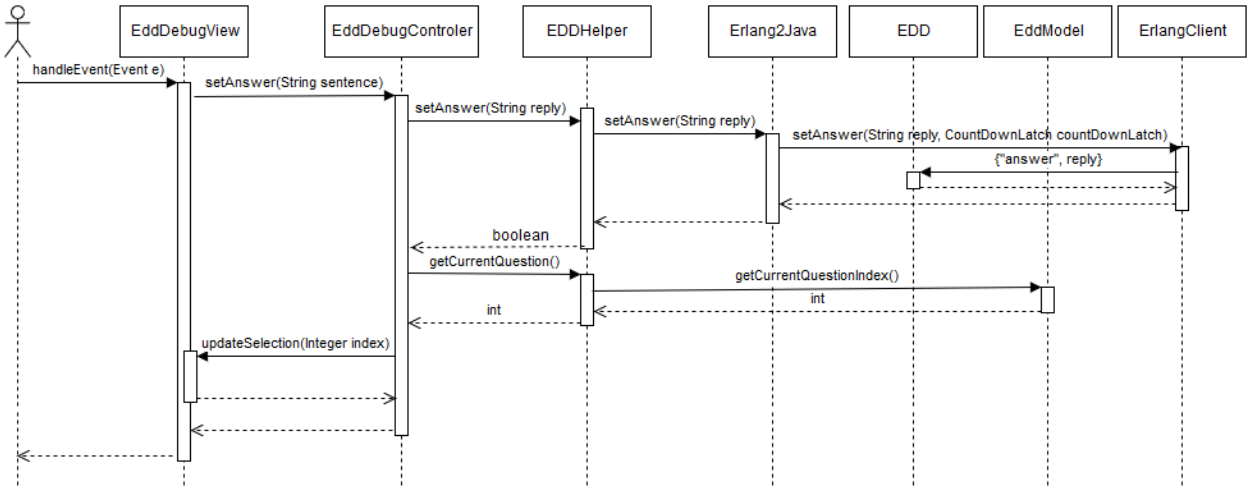


Ilustración B-5 Diagrama de Secuencia del proceso de respuesta a la pregunta actual en la depuración concurrente.

- Cuando el usuario selecciona una respuesta la vista envía un mensaje con la respuesta seleccionada a **EddDebugControler**.
- Este reenvía la información a **EddHelper**.
- Este reenvía la información a **Erlang2Java**
- Este reenvía la información a **ErlangClient**
- **ErlangClient** prepara el mensaje a enviar al servidor Erlang con la respuesta y los envía.
- **EddHelper** comprobará si ya se ha encontrado el error y se lo devolverá.
- Si se ha encontrado la causa del error **EddDebugControler** lo informará y se terminará el proceso.
- Por último, **EddDebugControler** informará a la vista que debe de actualizar su contenido con el de una nueva pregunta.

B.2.4 Respuesta a una pregunta distinta a la actual

En la *Ilustración B-6* se muestra el diagrama de secuencia del proceso de respuesta de una pregunta distinta a la actual en la depuración concurrente.

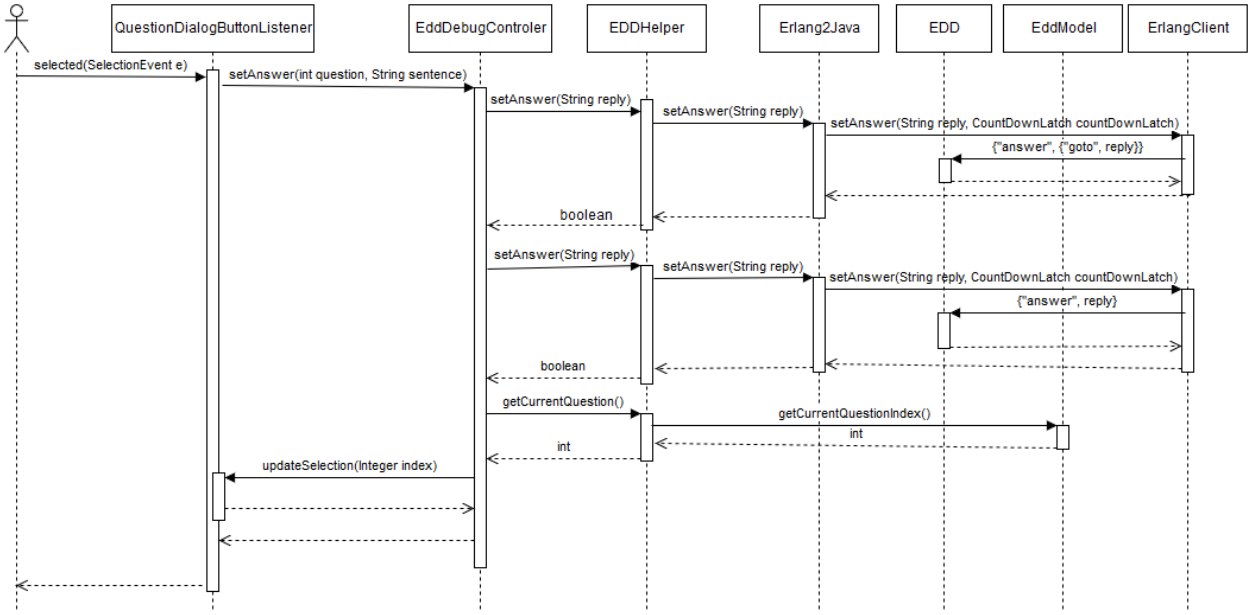


Ilustración B-6 Diagrama de Secuencia del proceso de respuesta a la pregunta por el diagrama de secuencia en la depuración concurrente.

- Cuando el usuario responde a una pregunta de **EDDTreeView**, se envía el número del nodo y la respuesta a **EddHelper**.
- **EddDebugControler** enviara el identificador a **EddHelper**.
  - Este reenvía la información a **Erlang2Java**.
  - Este reenvía la información a **ErlangClient**.
  - **ErlangClient** prepara el mensaje a enviar al servidor Erlang con la respuesta y los envía, esperando el tiempo indicado para recibir la respuesta.
- **EddDebugControler** enviará la respuesta de la pregunta a **EddHelper**.
  - Este reenvía la información a **Erlang2Java**.
  - Este reenvía la información a **ErlangClient**.
  - **ErlangClient** prepara el mensaje a enviar al servidor Erlang con la respuesta y los envía, esperando el tiempo indicado para recibir la respuesta.
  - **EddHelper** comprobará si ya se ha encontrado el error y se lo devolverá.

    - Si se ha encontrado la causa del error **EddDebugControler** lo informará y se terminará el proceso.
- Por último, **EddDebugControler** informará a la vista que debe actualizar su contenido con el de una nueva pregunta.

Anexo C - Comunicación Java-Erlang

Para la comunicación entre la aplicación EDD, realizado en Erlang, y su interfaz gráfico E-EDD, hecho en Java, se realiza un intercambio de mensajes a través del API de Jinterface [9], la cual nos ofrece un medio de comunicación entre ambos códigos, a través de un buzón (**OtpMbox**) y de una serie de clases en las que se pasará la información, las cuales se pueden observar en la *Ilustración C-1*.

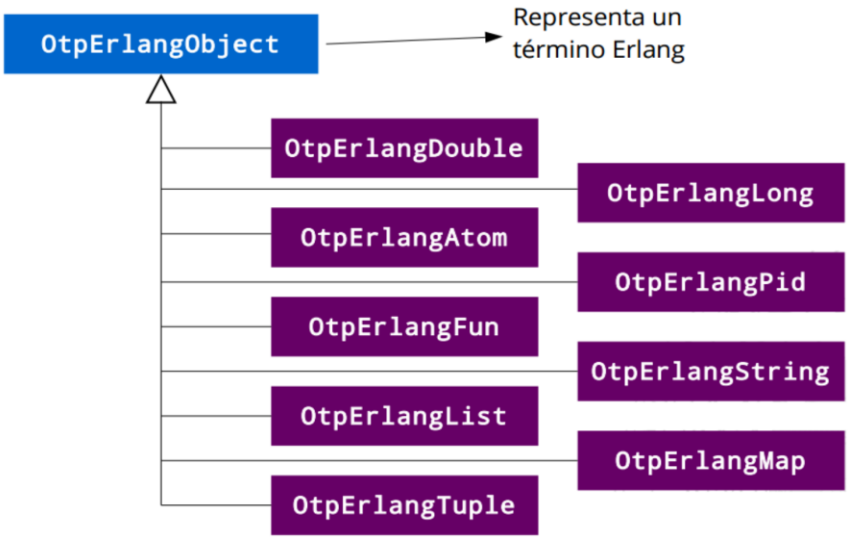


Ilustración C-1 Objetos de JInterface para el paso de información Java/Erlang

Los mensajes que se intercambian entre el *plugin* y el servidor EDD se verán en los siguientes apartados, separándolos según para el tipo de depuración para los que se utilizan y volviéndolos a separar según si los mensajes son de entrada o de salida de EDD.

C.1 Depuración secuencial

Durante el ciclo de vida del proceso de depuración declarativa secuencial entre Eclipse y EDD se intercambian los siguientes mensajes mostrados en el diagrama de secuencia de la *Ilustración C-2*.

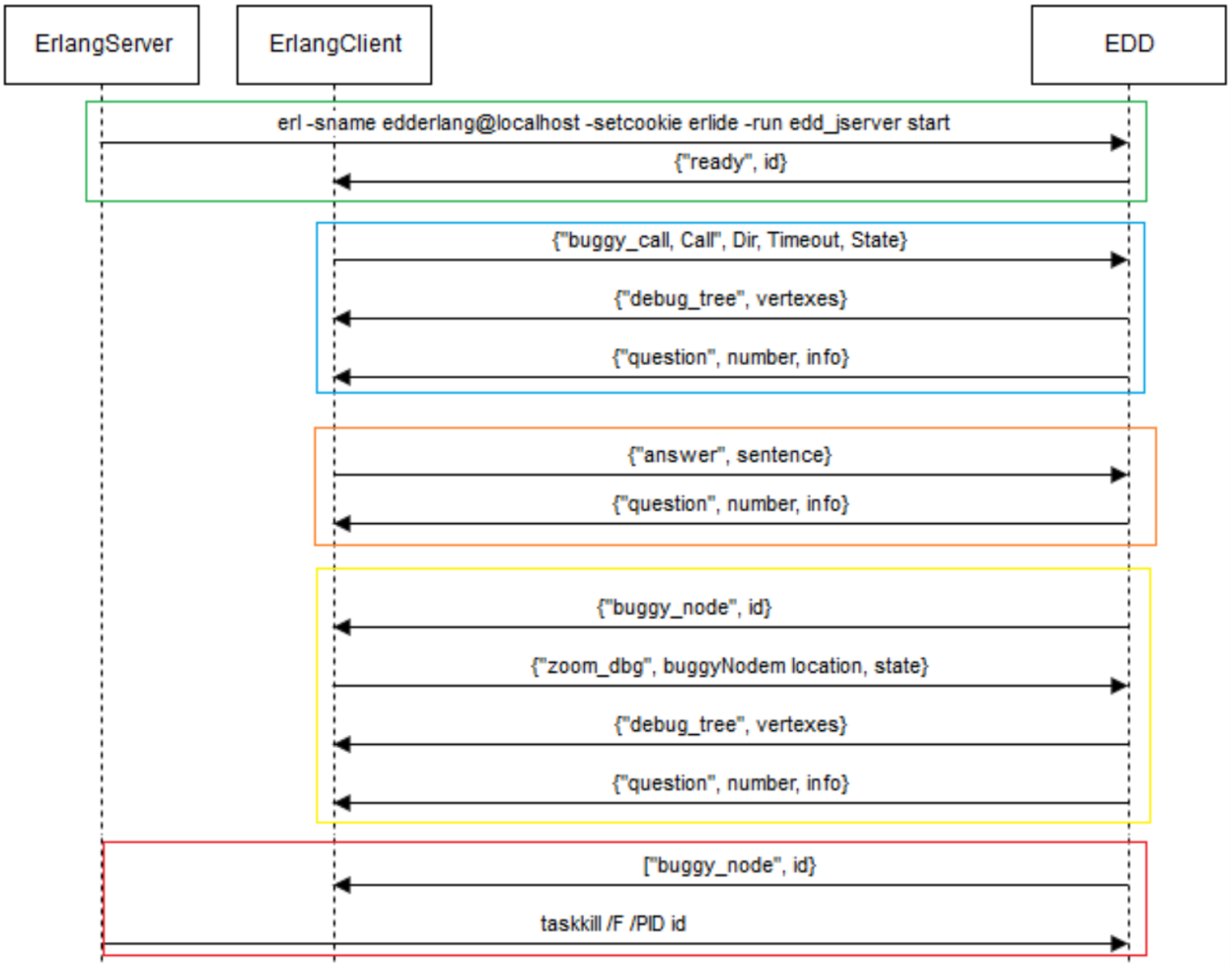


Ilustración C-2 Comunicación Eclipse/EDD durante la depuración secuencial.

Los mensajes que se muestran en la *Ilustración 28* se encuentran agrupados según su funcionalidad de la siguiente manera:

- **Verde**: Mensajes de inicialización del servidor.
- **Azul**: Mensajes de comienzo de la depuración y obtención de datos.
- **Naranja**: Repuesta/cambio de pregunta.
- **Amarillo**: Comienzo del proceso de depuración exhaustivo.
- **Rojo**: Finalización de la depuración.

C.1.1 Entrada

- **"erl -sname edderlang@localhost -setcookie erlide -run edd_jserver start":** Mensaje con el cual se inicia el servidor Erlang.

- **{“buggy_call, Call”, Dir, Timeout, State}**: Directiva con la que se inicia la depuración secuencial.
  - **Call:** Llamada Erlang a evaluar por el depurador.
  - **Dir:** Dirección donde se encuentra el código Erlang a depurar.
- **{“answer”, sentence}**: Este mensaje se envía al servidor para responder a una pregunta. Los posibles valores del campo “sentence” son:
  - “y”: Respuesta a la pregunta Erlang actual, con el valor “yes”.
  - “n”: Respuesta a la pregunta Erlang actual, con el valor “no”.
  - “t”: Respuesta a la pregunta Erlang actual, con el valor “thrusted”.
  - “d”: Respuesta a la pregunta Erlang actual, con el valor “don't know”.
  - “i”: Respuesta a la pregunta Erlang actual, con el valor “Inadmissible”.
  - “u”: Deshace la última ejecución.
  - “a”: Aborta la depuración
  - id: Identificador de la pregunta a la que se quiere cambiar.
- **{“zoom_dbg”, buggyNode, location, state}:** Mensaje que se utiliza para empezar el proceso de depuración con zoom.
  - buggyNode: Instrucción errónea.
  - location: Localización del fichero fuente.
  - state: Estado del servidor
- **“TaskKill /F /PID” id:** Mensaje con el cual se para la ejecución del servidor.

C.1.2 Salida

- **{“ready”, Id}:** Indica que el servidor se ha levantado, el valor que devuelve se corresponde con su identificador.
- **{“debug_tree”, vertexes}:** Mensaje que nos envía la información relativa a las preguntas que va a realizar EDD durante la depuración.
- **{question, number, info}:** Mensaje que indica la pregunta que se está procesando actualmente en el servidor EDD.
  - **number**: Identificador de la pregunta.
  - **info**: Información sobre la pregunta.
- **{“buggy_node”, id}**: Mensaje que indica que se ha encontrado el error.

- **id:** Identificador con el que se identifica la instrucción incorrecta del código a depurar.

C.2 Depuración concurrente

Durante el ciclo de vida del proceso de depuración declarativa concurrente se intercambian entre E-EDD y EDD los siguientes mensajes, mostrados en el diagrama de secuencia de la *Ilustración C-3*.

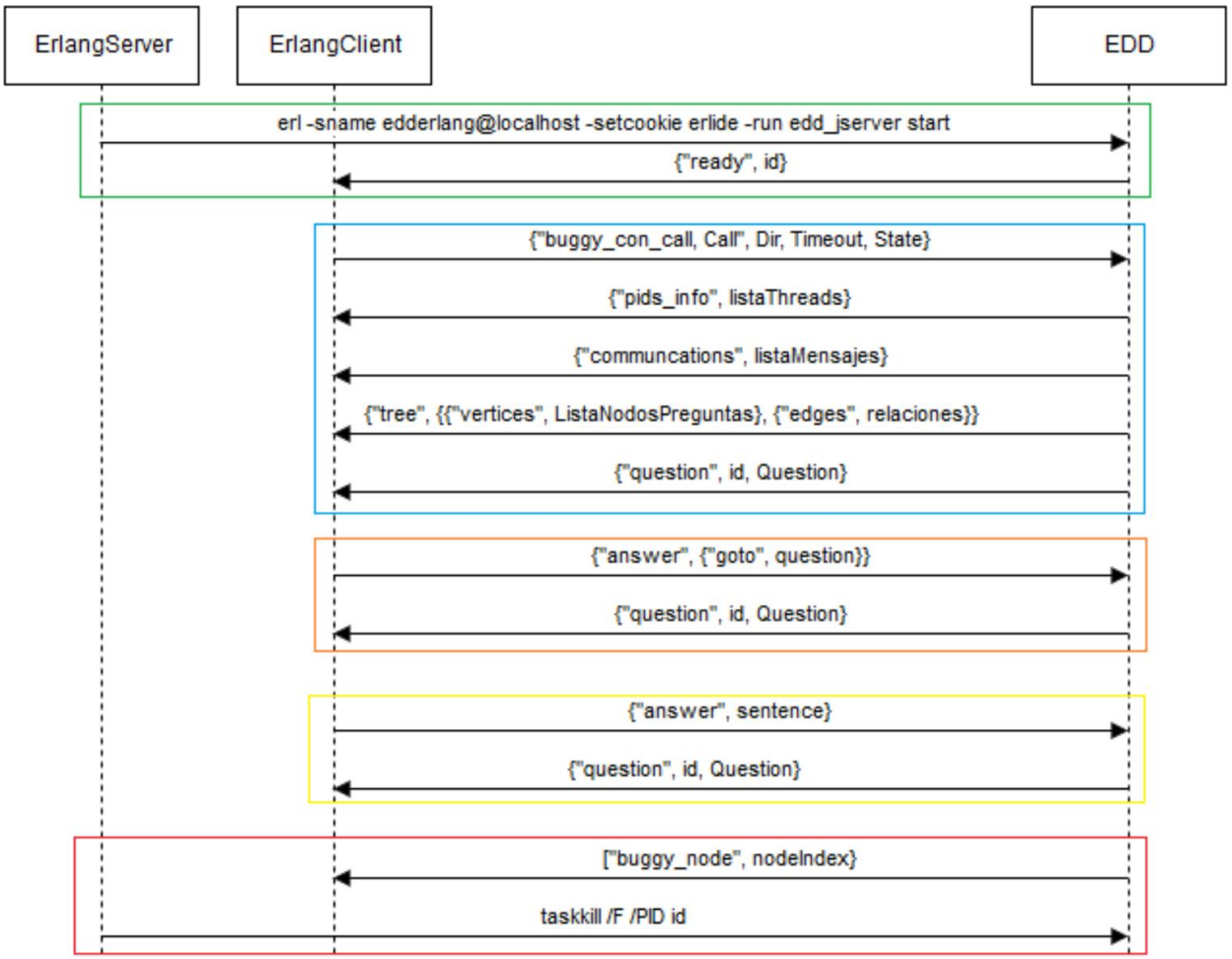


Ilustración C-3 Comunicación Eclipse/EDD durante la depuración concurrente.

Los mensajes que se muestran en la *Ilustración 29* se encuentran agrupados según su funcionalidad de la siguiente manera:

- **Verde**: Mensajes de inicialización del servidor.
- **Azul**: Mensajes de comienzo de la depuración y obtención de datos.
- **Naranja**: Cambio de pregunta.

- **Amarillo**: Repuesta a una pregunta
- **Rojo**: Finalización de la depuración.

C.2.1 Entrada

- **“erl -sname edderlang@localhost -setcookie erlide -run edd_jserver start”:** Mensaje con el cual se inicia el servidor Erlang.
- **{“buggy_con_call, Call”, Dir, Timeout, State}**: Directiva con la que se inicia la depuración concurrente.
  - **Call:** Llamada Erlang a evaluar por el depurador.
  - **Dir:** Dirección donde se encuentra el código Erlang a depurar.
- **{“answer”, sentence}**: Este mensaje se envía al servidor para responder a una pregunta. Los posibles valores del campo “sentence” son:
  - **“CORRECT”**: valor adjunto a la respuesta dada por el usuario.
  - **“INCORRECT”**: valor adjunto a la respuesta dada por el usuario.
  - **“undo”** Deshace la última contestación.
  - **“abort”** Para la depuración.
  - **{“goto”, question}** Pide que se cambie la pregunta actual por la pregunta con el identificador que se pasa en question.
- **“TaskKill /F /PID” id:** Mensaje con el cual se para la ejecución del servidor.

C.2.2 Salida

- **{“pids_info”, listaThreads}**: Este mensaje comunica la información relativa a los diferentes procesos que se han manejado en la depuración. Cada proceso se compone de: {“pid_info”, pid, first_call, spawned, sent, result, is_first, callrec_stack, snapshots}
  - **pid**: Identificador del proceso tal cual lo representa Erlang,
  - **first_call**: Contiene la llamada con la que se arrancó el proceso.
  - **spawned**: Lista con los ids de los procesos que ha creado este.
- **{“ready”, Id}:** Indica que el servidor se ha levantado, el valor que devuelve se corresponde con su identificador.

- **{“communcations”, listaMensajes}**: Este mensaje comunica la información de los diferentes mensajes que se han enviado, en el orden en que están guardados en Erlang. Los mensajes que se describen en los elementos de listaMensajes pueden ser de tres tipos:
  - **{“sent”, message_info}:** Representa un mensaje que ha sido emitido.
  - **{“received”, message_info}:** Representa un mensaje que ha sido recibido.
  - **{“spawned”, spawn_info}:** Representa la creación de un nuevo proceso.
  - La información de estos mensajes puede ser de dos tipos:
    - **{“message_info”, { from, to, msg}}:** Información de un mensaje, donde from es el identificador del emisor del mensaje, to es el identificador del receptor del mensaje y msg es el texto del mensaje.
    - **{“spawn_info”, {spawner, spawned}}:** Información de la creación de un nuevo proceso donde spawner es el identificador del proceso padre y spawned es el identificador del proceso que se ha creado.
- **{“tree”,{{“vertices”, ListaNodosPreguntas}, {“edges”, relaciones}}}**: Árbol de preguntas a realizar. Cada elemento de ListaNodosPreguntas corresponde con la información de un nodo del árbol que posee el siguiente formato {{“id”, Id}, {“question”, Question}, {“info”, Info}}, donde:
  - **Id**: Identificador del nodo.
  - **Question**: Información de la pregunta, posee el siguiente formato {“question”, text, answers, str_callrec}
    - **text**: Texto de la pregunta.
    - **answers**: Lista de respuestas de la pregunta. Cada pregunta tiene el siguiente formato {“answer”, text, when_chosen}
      - **text**: Texto de la respuesta.
      - **when_chosen**: Acción a realizar cuando se selecciona esta pregunta, el valor de este campo puede ser: “correct”, “incorrect” o una nueva pregunta.
  - **Info**: Información adicional del nodo.
- **{“question”, id, Question}**: Indica la pregunta actual del depurador:
  - **id**: Identificador de la pregunta actual.
  - **Question**: Pregunta actual del depurador.
- **{“buggy_node”, nodeIndex}:** Mensaje que indica que se ha encontrado un error.

Anexo D – Guía de Instalación

En este apartado explicaremos como instalar nuestro plugin en Eclipse.

D.1 Programas requeridos

Antes de proceder a la instalación del plugin necesitamos tener una serie de herramientas necesarias para el perfecto funcionamiento de este, estas serán las mismas que para la versión anterior del plugin [14, D1].

D.1.1 EDD (Erlang Declarative debugger)

EDD es el depurador declarativo al que este plugin aporta una interfaz gráfica, y del cual ya hemos visto su funcionabilidad en la sección [3.4 EDD (Erlang Declarative debugger)]. Para instalarlo se deberá ir a la web donde se encuentra el código del Proyecto [14] y seleccionar la opción de descarga, como vemos en la *Ilustración D-1*.

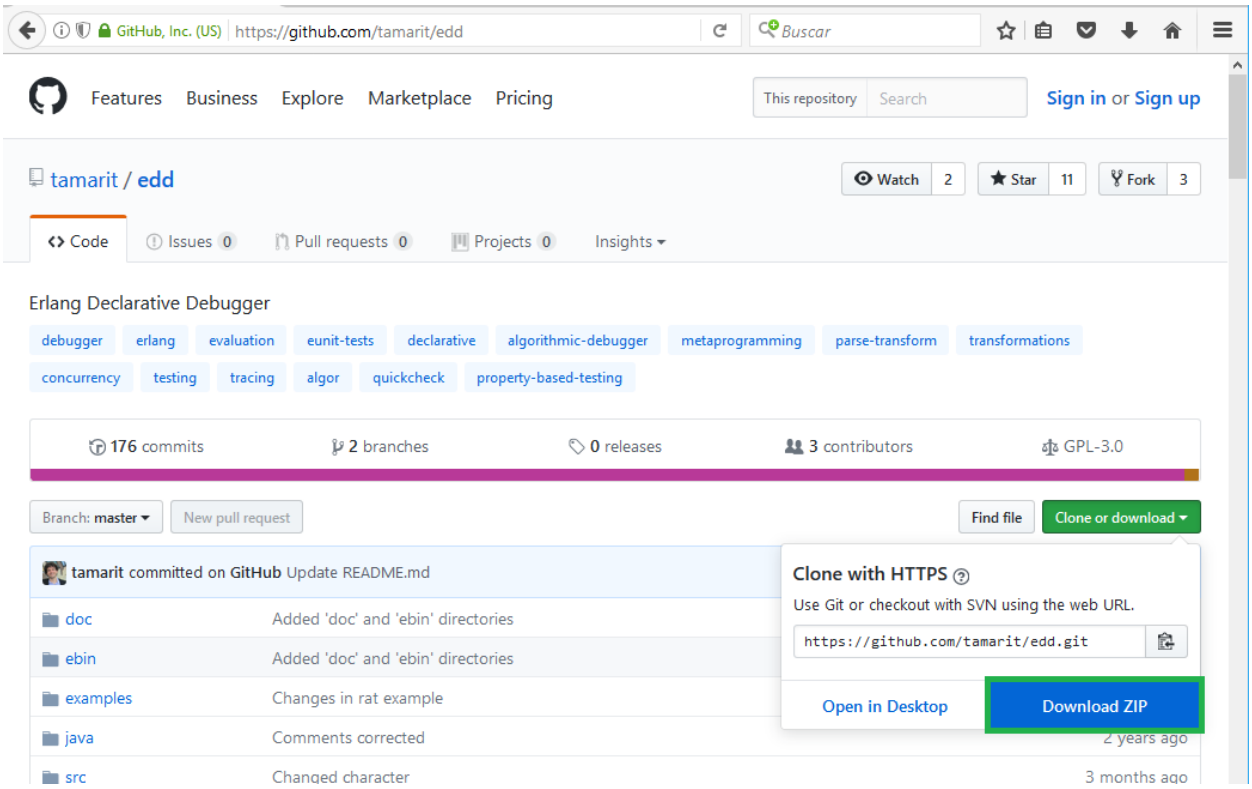


Ilustración D-1 Descarga del proyecto EDD

Una vez descargado el proyecto lo deberemos descomprimir y compilar sus clases como vemos en la *Ilustración D-2*.

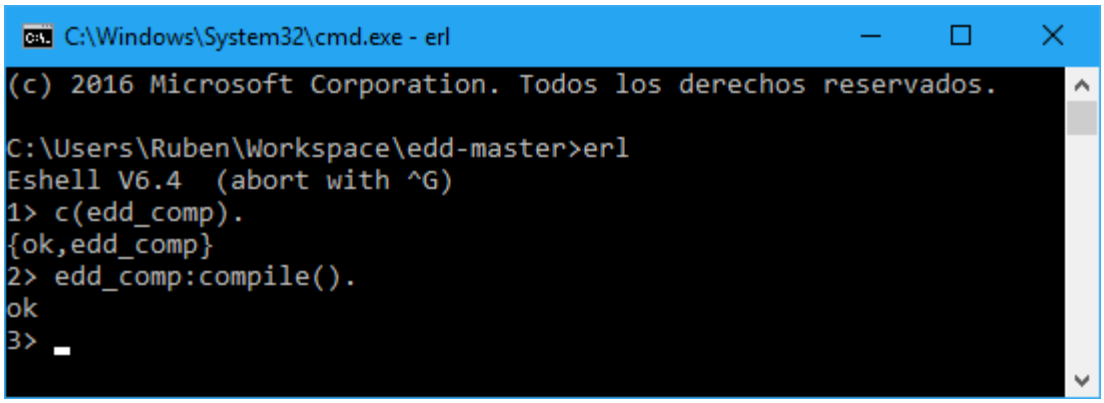


Ilustración D-2 Compilación del proyecto EDD

D.1.2 Graphviz

Graphviz es una herramienta gratuita que nos permite generar diagramas a partir de un esquema que lo describa. Este programa se usa para generar los árboles que aparecen en la ventana "Graphviz View".

Para instalar esta aplicación bastará con descargarla de su página oficial [16], descomprimir el archivo obtenido y añadir la ruta de este a una variable de entorno, como observamos en la *Ilustración D-3*.

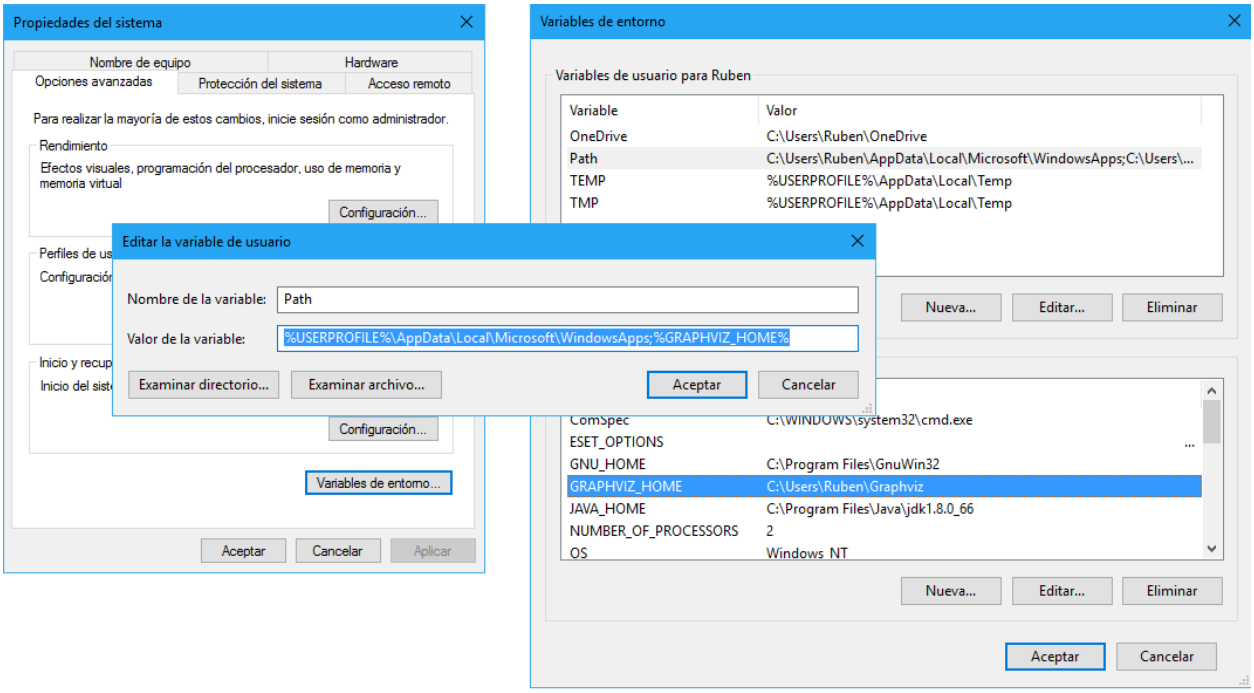


Ilustración D-3 Variable de entorno de Graphviz

Para verificar que está funcionado desde línea de comandos ejecutaremos la instrucción "dot -v" para verificar la versión utilizada en dicho comando, como vemos en la *Ilustración D-4*.

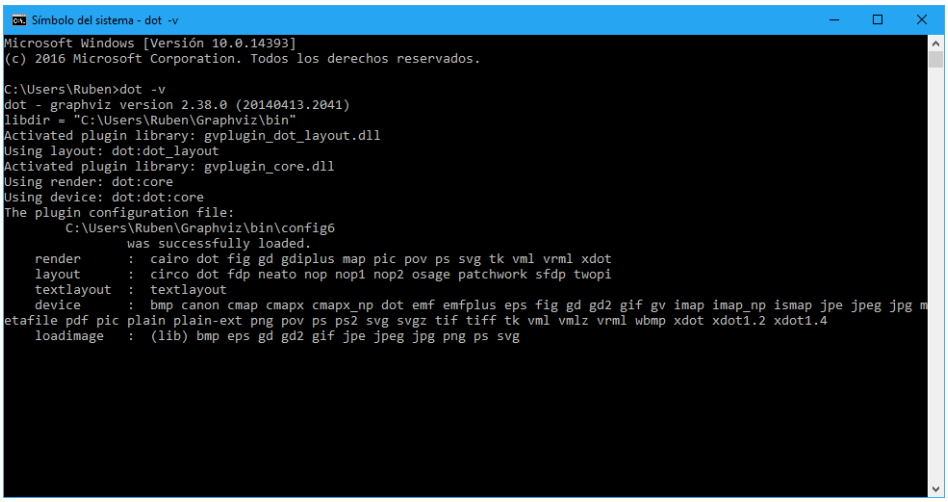


Ilustración D-4 Comprobación del funcionamiento de Graphviz

D.1.3 UMLGraph

UMLGraph [20] es una herramienta que nos permite a partir de una especificación declarativa, construir diagramas de clase y diagramas de secuencia bajo el estándar UML. Para su instalación bastará con descargar el archivo zip desde la página oficial del proyecto y proceder tras ello a la declaración de la variable de entorno, como observamos en la *Ilustración D-5*.

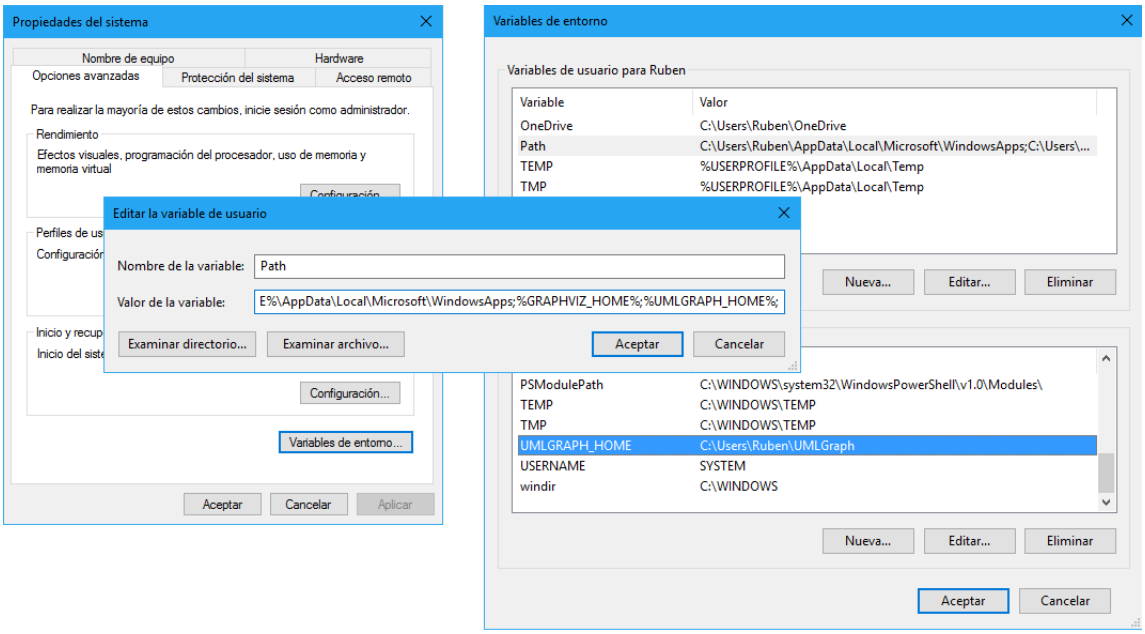


Ilustración D-5 Variable de entorno de UMLGraph

Para verificar que está funcionado desde línea de comandos podemos ejecutar el comando "pic2plot --v", como lo vemos en la *Ilustración D-6*.

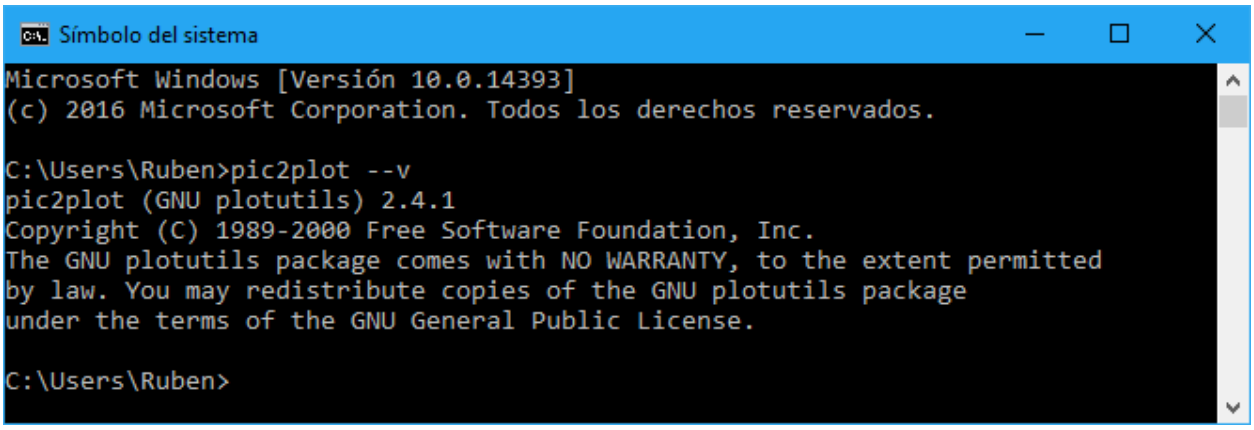


Ilustración D-6 Comprobación del funcionamiento de UMLGraph

D.1.4 Eclipse

Eclipse es el entorno de desarrollo sobre el cual está desarrollado E-EDD. Para instalarlo basta con ir a su página oficial [1], buscar y descargar la versión que se quiera utilizar, se recomienda la versión Eclipse Modeling Tools bajo la distribución Luna, dado que esta versión ya cuenta con los plugins para trabajar bajo EMF (Eclipse Modeling Framework) y GMF (Graphical Modeling Framework), que se utilizan en el apartado gráfico de E-EDD. En caso de descargar otra versión, se deberá instalar estos *plugins* por separado.

D.1.4.1 Plugins

Una vez instalado Eclipse se deberán instalar en él una serie de plugins necesarios para el funcionamiento de nuestra herramienta.

- **Erlide** [10, 11]: Este *plugin* le añade compatibilidad a Eclipse con el lenguaje Erlang. Este *plugin* es usado en nuestro proyecto para gestionar los módulos Erlang que se desean depurar.
- **EclipseGraphviz** [16, 17]: Este plugin permite a Eclipse interactuar con Graphviz de forma sencilla. Este *plugin* se usa para generar los árboles que aparecen en la ventana "Graphviz View".
- **EMF** [7]: Este plugin ofrece un interfaz gráfico para la generación de modelo de datos, el cual se ha utilizado para crear el modelo original del proyecto.

Para instalar un *plugin* debemos de ir al apartado de instalación de plugins de Eclipse, el cual se encuentra en el menú "Help/Install New Software", como podemos observar en la *Ilustración D-7*.

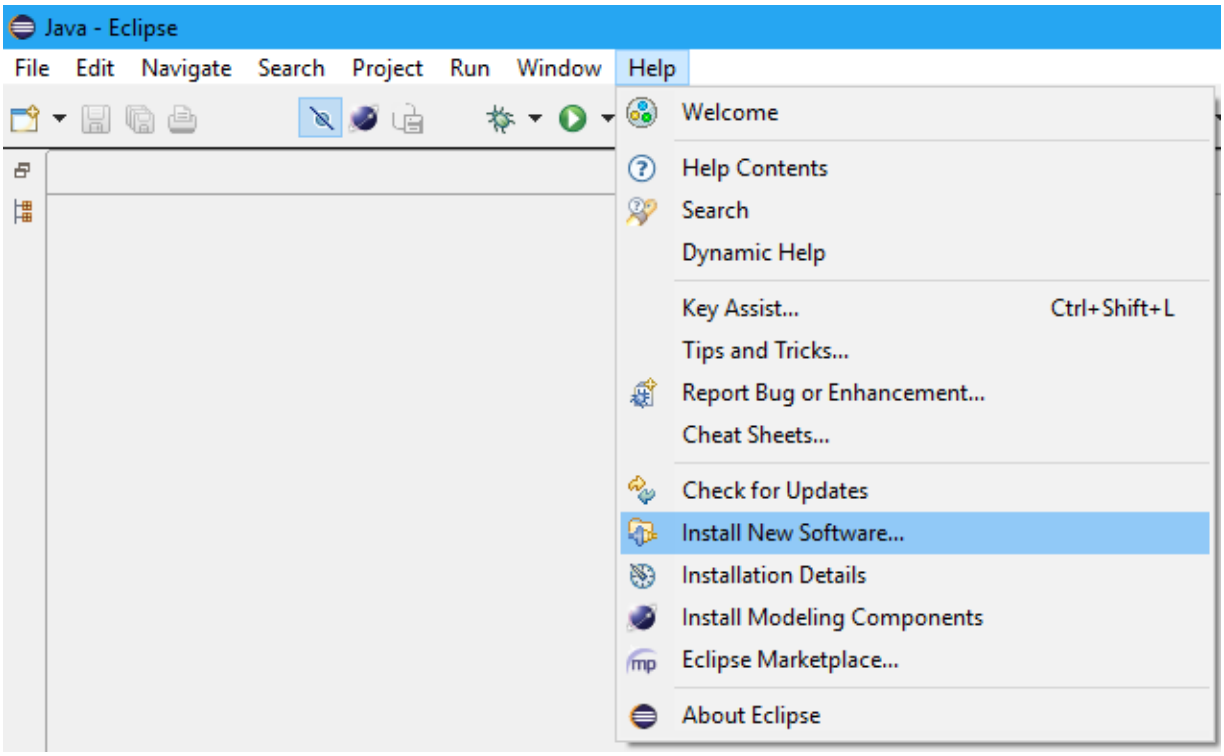


Ilustración D-7 Opción de instalación de Plugins en Eclipse

Una vez abierta la ventana de instalación de plugins tenemos que añadir en el campo de entrada la URL de donde se descargará el plugin, mostrándose con ello los plugin disponibles en esa URL, que habrá que seleccionarlos, como se ven para cada uno de los plugins que debemos instalar en la *Ilustración D-8*, la *Ilustración D-9* y la *Ilustración D-10*. Tras introducir la URL seleccionamos los plugins que deseamos instalar y pulsamos el botón siguiente, con lo que empezará el proceso de instalación, una vez terminado el proceso se deberá reiniciar Eclipse para que podamos utilizarlo.

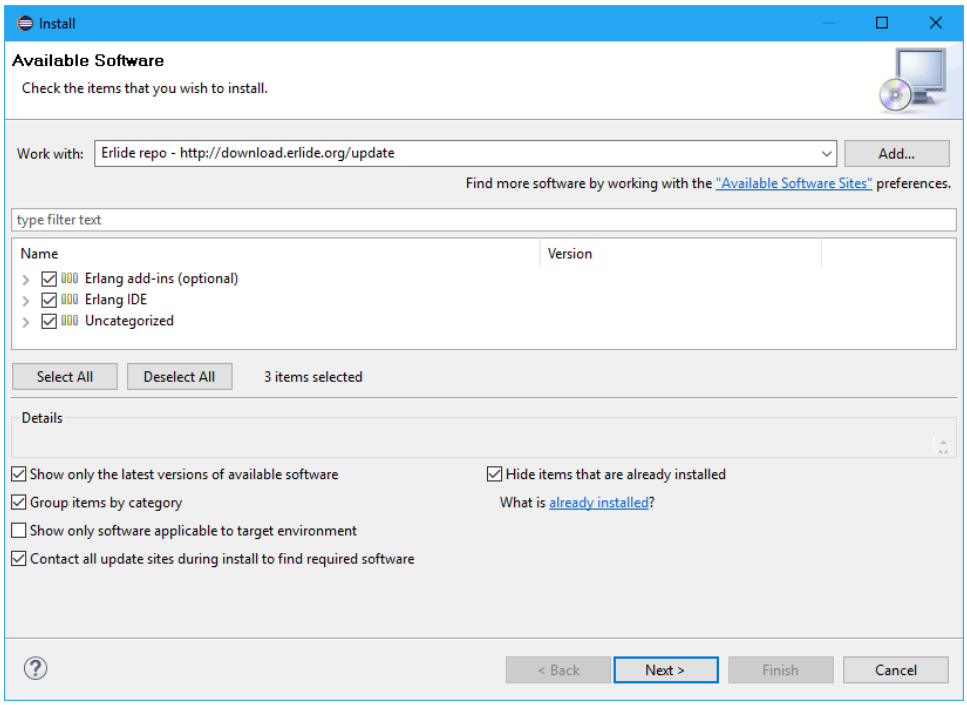


Ilustración D-8 Información de instalación del plugin Erlide

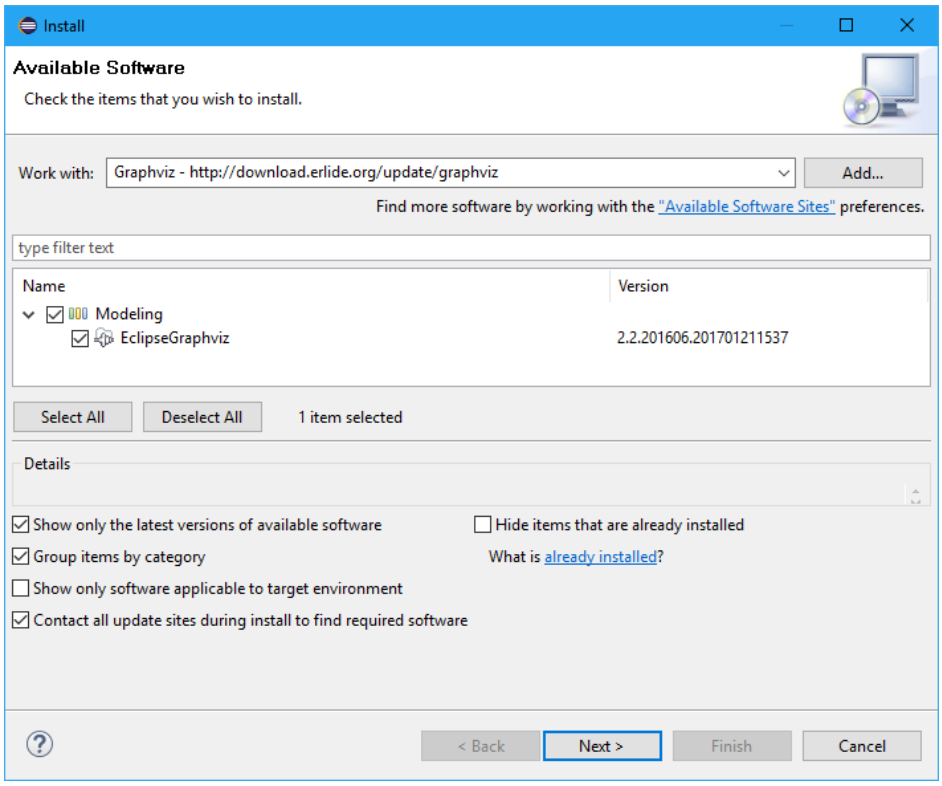


Ilustración D-9 Información de instalación del plugin Graphviz

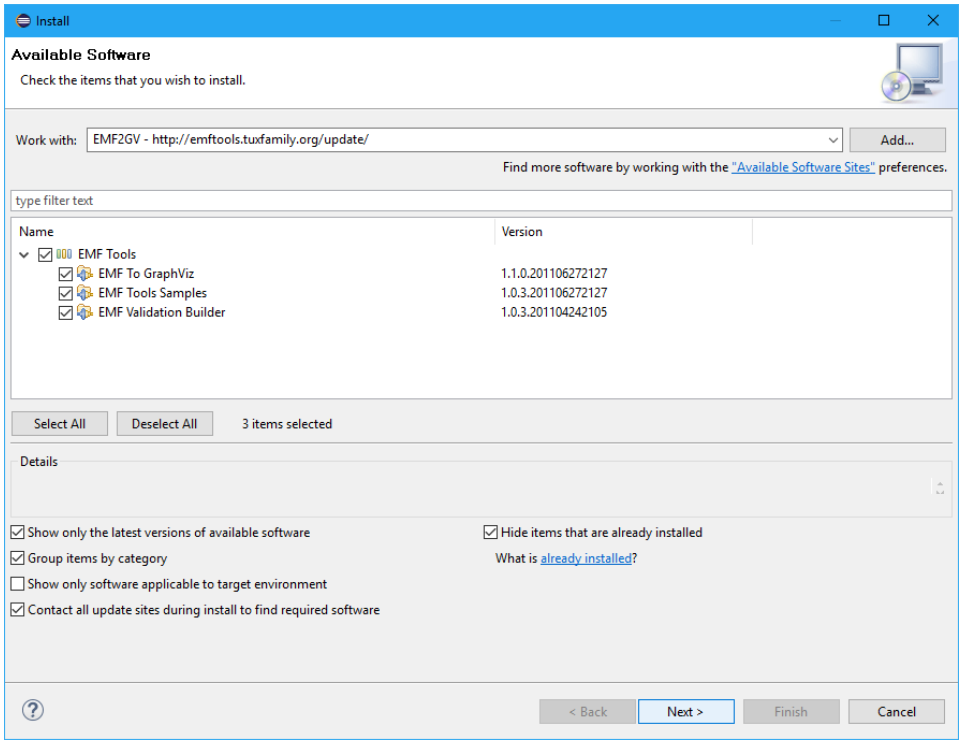


Ilustración D-10 Información de instalación del plugin EMF

D.2 Instalación de E-EDD 2.0

Una vez descargado Eclipse, e instalado todos los prerrequisitos, empezaremos con la instalación del proyecto, para ello primero deberemos ir a la web donde se encuentra el código del Proyecto [15] seleccionar la opción de descarga, como vemos en la *Ilustración D-11*. Una vez descargado el proyecto lo deberemos descomprimir para que Eclipse pueda instalarlo.

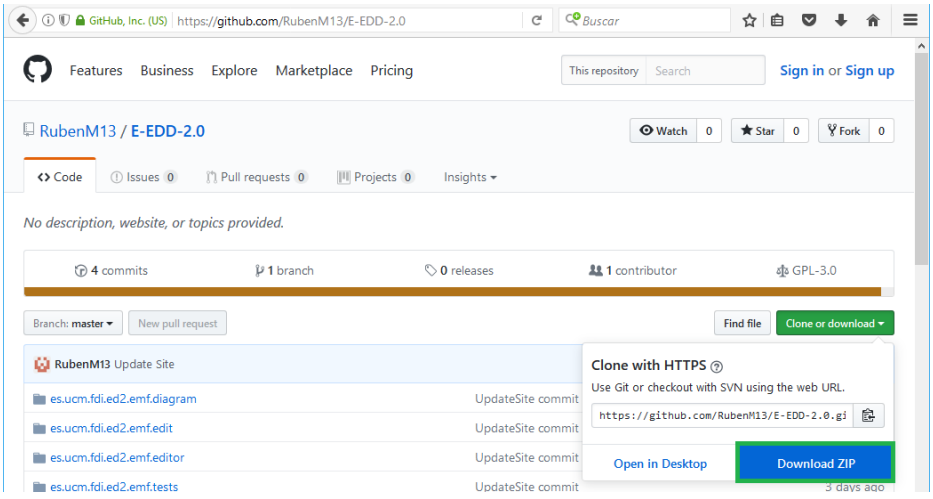


Ilustración D-11 Descarga del proyecto E-EDD 2.0.

Tras ello deberemos abrir Eclipse e ir al apartado de instalación de los plugins, que se encuentra en el menú “Help/Install New Software” de la barra de herramientas de Eclipse, como podemos observar en la *Ilustración D-12*.

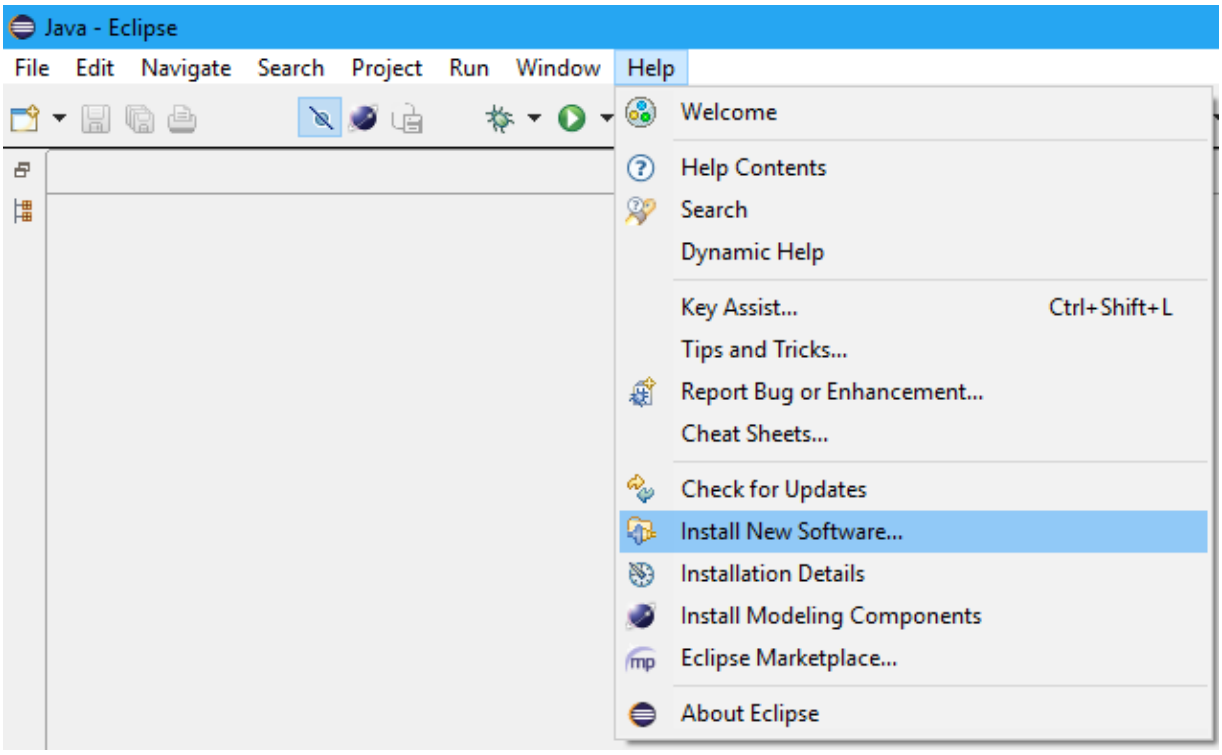


Ilustración D-12 Opción de instalación de Plugins en Eclipse.

Tras hacer click en este menú se nos abrirá una ventana de diálogo, donde tendremos que poner el path al proyecto “es.ucm.fdi.edd.updatesite” de E-EDD 2-0, tal y como podemos observar en la *Ilustración D-13*, con lo que aparecerá la opción para instalar el plugin el cual seleccionaremos y pulsaremos en el botón “Next” para empezar con ello el proceso de instalación de E-EDD 2.0.

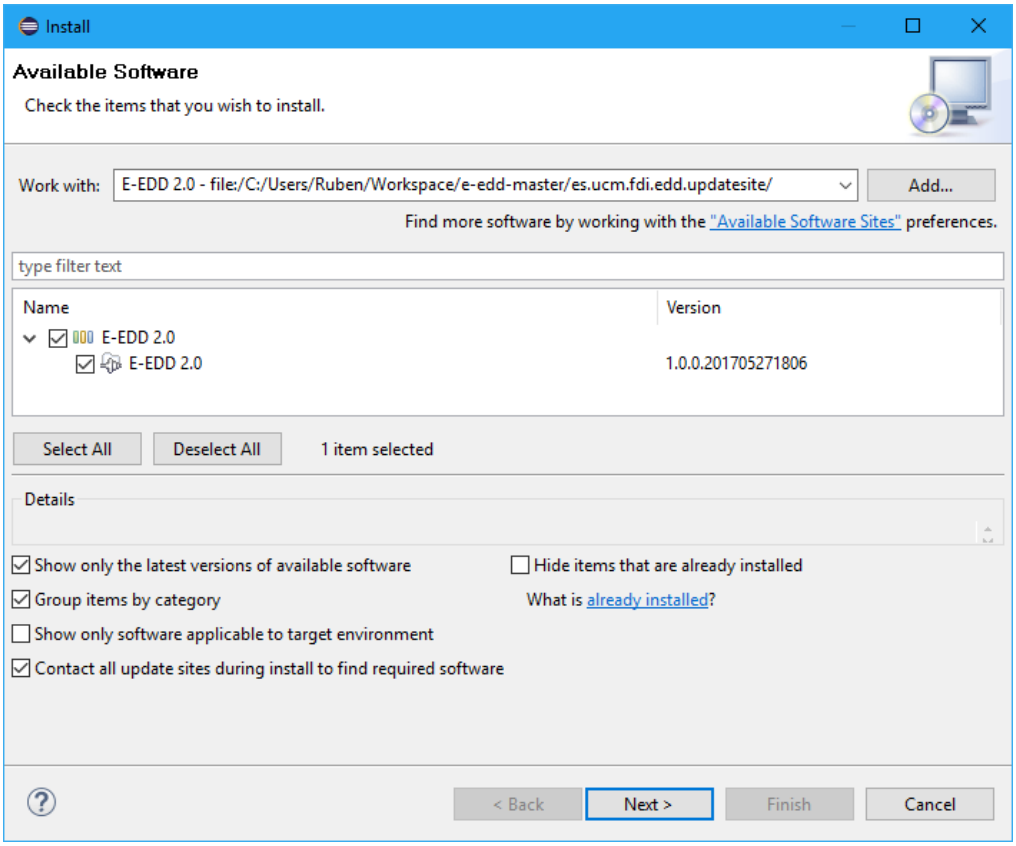


Ilustración D-131 Información de instalación del plugin E-EDD 2.0.

Una vez instalado se necesitará reiniciar Eclipse y configurar las opciones de E-EDD, para comunicarle la localización de UMLGraph, tal y como vemos en la *Ilustración D-14*.

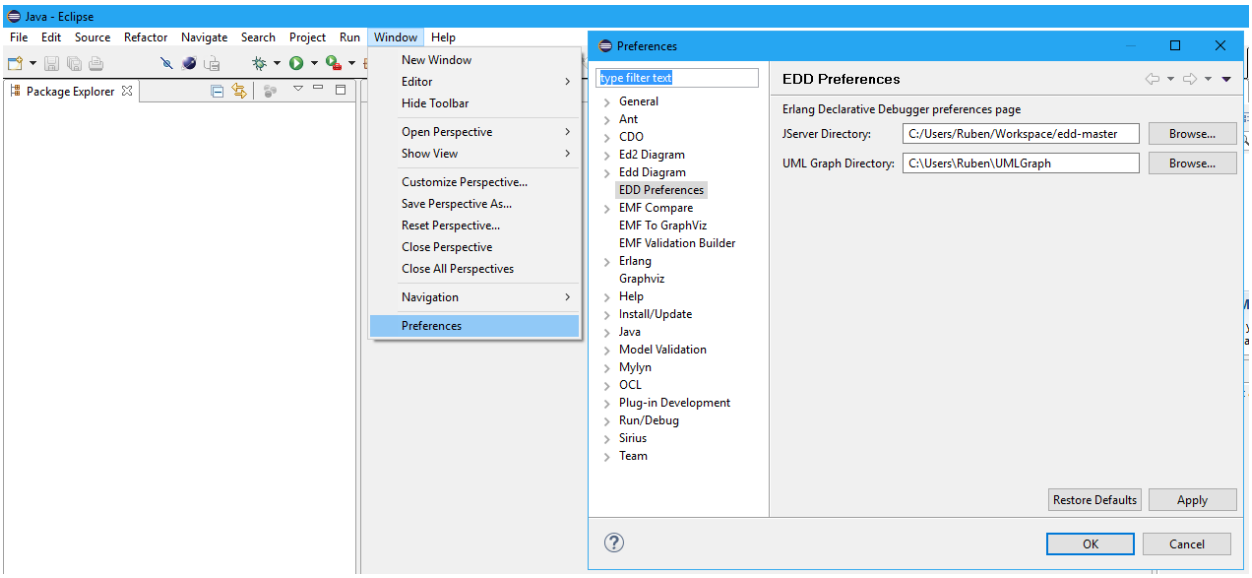


Ilustración D-14 Configuración de E-EDD

Una vez que lo hayamos configurado solo tendremos que abrir su perspectiva, como veos en la Ilustración D-15, para trabajar con él.

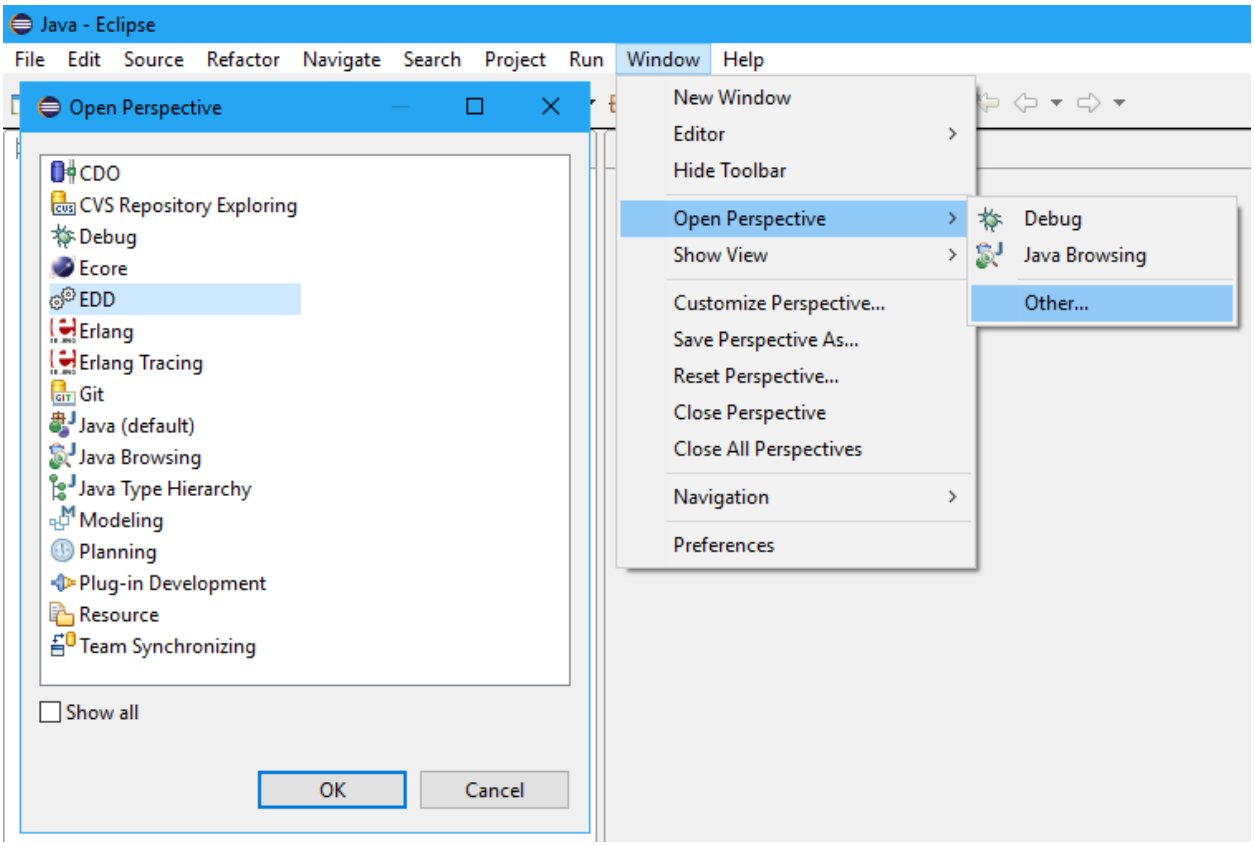


Ilustración D-15 Apertura de la perpectiva de E-EDD.