IST-510255
EmBounded
Automatic Prediction of Resource Bounds for Embedded Systems

Specific Targeted Research Project (STReP)
FET Open

D34 (WP7e): Hume Problem Solving Environment

Due date of deliverable: 1st September 2008
Actual submission date: 1st September 2008

Start date of project: 1st March 2005 Duration: 48 months

Lead contractor: Heriot-Watt University Revision: 1.6

Purpose: The purpose of this deliverable is to provide an interactive *problem solving environment* for the Hume language. This should include the ability to debug, visualize and modify/recompile Hume code from within a unified environment.

Results: The main results of this deliverable is the Hume IDE, a java-based windowing environment which is able to control and interpret information from the Hume to C compiler (deliverable D23 WP7c). This tool allows Hume programs to be edited, compiled and debugged using graphical windows for displaying information conveniently during the software development process.

Conclusion: The main conclusions are that the Hume IDE is useful for the development of Hume programs and provides many convenience functions. However, the environment lacks the ability to graphically display Hume boxes in the manner of tools such as Stateflow or Statecharts.

Project co-funded by the European Commission within the 6^{th} Framework Programme (2002-06)			
Dissemination Level			
PU	Public		⁎
PP	Restricted to other programme participants	(including the Commission Services)	
RE	Restricted to a group specified by the consortium	(including the Commission Services)	
CO	Confidential only for members of the consortium	(including the Commission Services)	

Hume Problem Solving Environment

Robert F. Pointon <rpointon@macs.hw.ac.uk>
School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, Scotland

Abstract

We describe an interactive *problem solving environment* for the Hume language. This Hume IDE is a java-based windowing environment, allowing Hume programs to be edited, compiled and debugged. We demonstrate that in its current state the Hume IDE is useful for the development of Hume programs. Desirable features for future versions are the ability to graphically display Hume boxes in the manner of tools such as Stateflow or Statecharts, and, in the longer term, to provide a visual programming interface for Hume boxes.

Major Revisions		
Revision	Date	Changes
1.5	14 Aug. 2009	table-of-contents, abstract (addressing Review Report Year 4)
1.3	1 Sep. 2008	initial version

Contents

1 Introduction

Modern embedded systems designers and implementers have a range of tools at their disposal to facilitate and support their work. These include integrated environments such as the Renesas High-performance Embedded Workshop[1] or the IAR compiler suite[2] (both of which support the Renesas M32C used by the EmBounded project).

These are highly prefessional-quality tools with features such as; drivers and description files for a wide range of microprocessors and associated debugging modules, automatic generation of test suites, visualization of memory maps and debug traces, automatic update from company websites, etc. Hume currently cannot compete with tools of this standard since it remains a research tool and as such has minimal support for the development process.

In this deliverable we present an initial attempt at providing such an integrated development environment (IDE) for the Hume language. We chose Java as the development environment for portability and we initially support just the Hume to C compiler (`humec`) and the instrumentation analysis provided by the compiler. Full resource analysis will be integrated into the tool at a later date. We also, initially, only support a text-based view of the Hume sources. Future enhancements may include graphical representations of Hume boxes integrated with the editor/compiler/debugger tools.

2 Objectives

To support development and debugging of Hume programs we have constructed a simple problem-solving environment (PSE), revealing resource usage information and identifying hotspots. To this end we:

1. define a standard for Hume resource use and trace information;
2. modify the HAM to emit such resource use and trace information;
3. create a *text-based* visualization tool to display the current state of a running Hume program;
4. additionally annotate the visualization tool to display resource usage and trace information.

3 The Hume IDE

To achieve these objectives we have developed a Java platform implementation of a simple debug tool called `hide`, Figure 1. This implementation incorporates the `humec` compiler and a reimplementation of an earlier text-based debugger for Hume into a single environment.

This deliverable incorporates the installation media for the Java implementation and we reproduce the manual here as a means of describing the installation, functionality and use of the tool.

4 Installing the Hume IDE

There are several ways of installing the Hume IDE. If you have access to the sources you can compile and run from the sources directory with:

```
% javac hide/Main.java
% java hide.Main ‘which humec‘
```

[1] http://www.renesas.com/fmwk.jsp?cnt=ide_hew_tools_product_landing.jsp&fp=/products/tools/ide/ide_hew/
[2] http://www.iar.com/website1/1.0.1.0/50/1/index.php

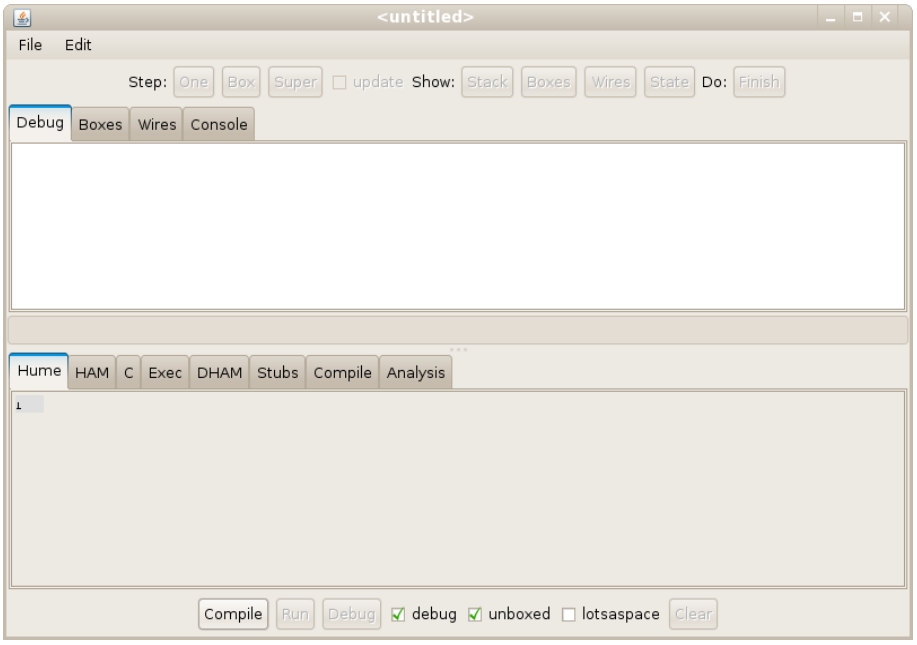

Figure 1: hide: Initial screen

Alternately, there is an `ant` build definition in the `build.xml` file and also a `Makefile` which allows building in the absence of `ant`. The `Makefile` should be able to detect the presence of `ant` and use it automatically. Note that you may have to edit the Makefile to point to the implementations of your `java`, `javac` and `ant` executables. In addition, you should set the `INSTALL_DIR` variable if you intend to install the compiled program somewhere.

From a distribution directory you should use the `hide` script provided. This script should be portable provided it is in the same directory as the `hide.jar` file. Additional support includes a `Gnome` desktop file with an associated icon.

Note that the current version of `hide` requires you to specify where the `humec` program exists on the command line. The `hide` script simply uses `‘which humec‘` to find it so `humec` must be in your `PATH` variable.

5 Using the Hume IDE

When the `hide` program is run, the user is presented with the basic screen shown in Figure 1. This has several components but is basically divided into two areas, the debug area in the upper half screen and the source view in the lower half. The components are:

- A menubar along the top, currently just `File` and `Edit`.
- Debug toolbar containing buttons for advancing the debug (`One`, `Box` and `Super`) and for altering the debug view (`Stack`, `Boxes`, `Wires` and `State`). There is also a `Finish` button which kills any running or debugged programs.
- A tabbed display area for showing the current debug view; `Debug`, `Boxes`, `Wires` or `Console`.
- A text input area. This is used to insert low-level commands into the data stream between the debugger and the running program.

- A tabbed source view (`Hume` to `Analysis`) which shows all of the current source information from Hume source to the compiled C code and also some additional status displays such as information on the executable or the resource usage.
- Finally, along the bottom there is a toolbar for overall control allowing setting of compiler options and running the compiler or running or debugging the program.

5.1 Loading and Saving files

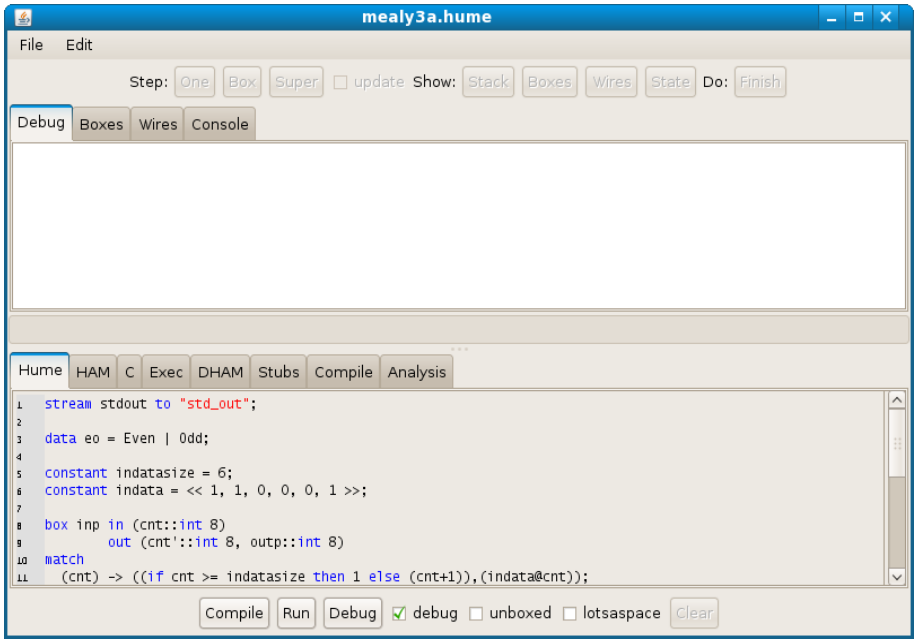

Figure 2: hide: Hume source view

To load a Hume file call `File` → `Open File...`, the file can be selected using a standard file dialogue box. Alternatively, there is a recent file selection under `File` → `Open Recent`.

Saving a file can be done with `File` → `Save (.hume)` which is bound to `Ctrl-S`. Saving under a different name can be done with `Save As...`.

Note that when a Hume file is loaded, any associated files are scanned and loaded in. These include the equivalent HAM file, DHAM file, C source file and the `humestubs.c` file which contains bindings to external code. The `Exec` and `Analysis` tabs are also populated. Once loaded the source window looks as shown in Figure 2.

5.2 Source view information

Figure 3 summarizes the information provided on the executable, including size and other OS-specific data.

Figure 4 shows the information extracted from the compilation process. This includes heap and stack requirements for both boxes and wires and also the Hume class of functions and boxes.

5.3 Editing facilities

All of the source files (Hume, HAM, C and `humestubs.c`) can be editted. Basic editing facilities are provided. Text can be input at the current cursor position within the window and the `Edit` menu provides the following features:

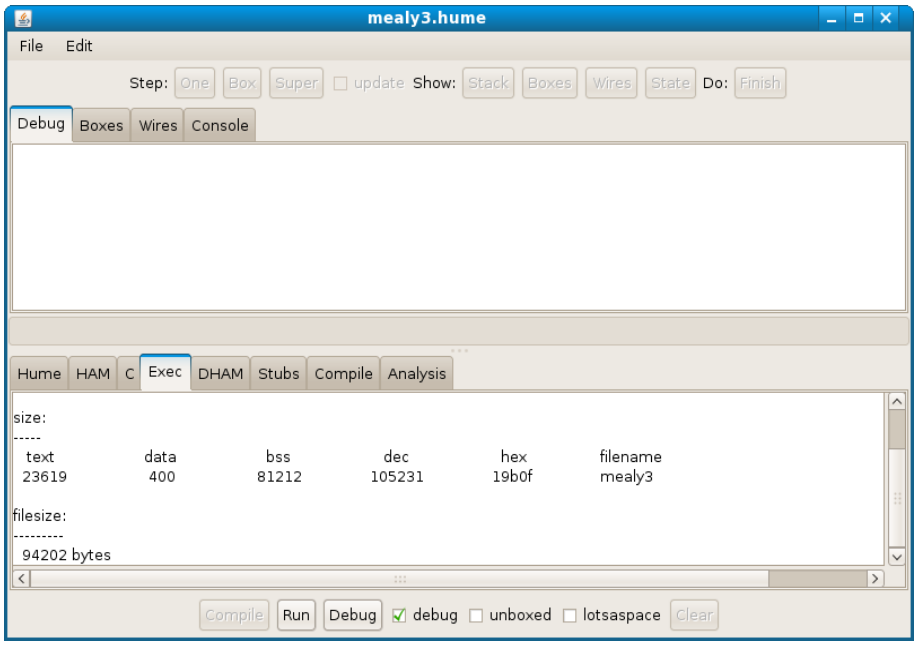

Figure 3: hide: Hume exec view

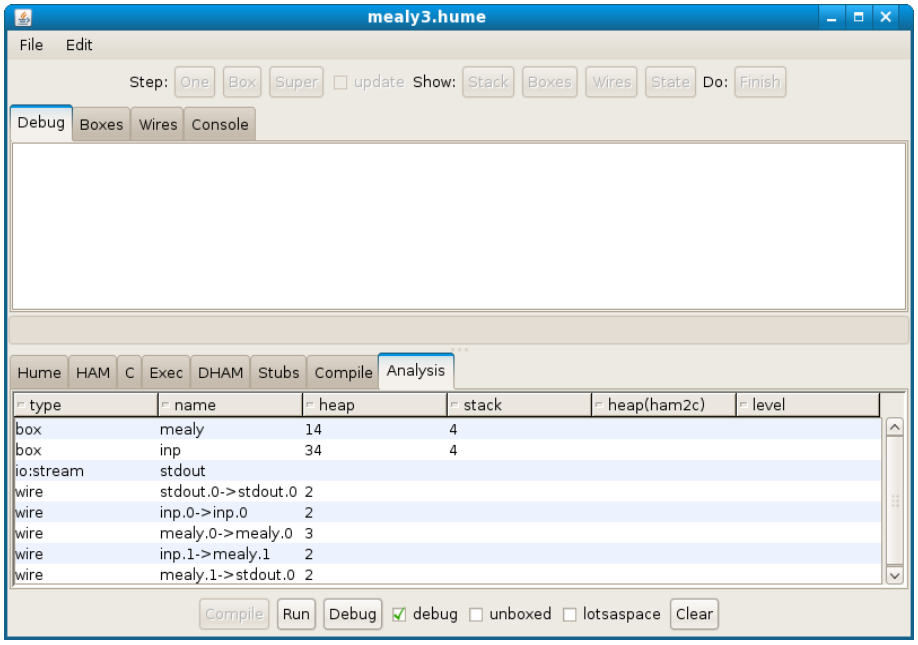

Figure 4: hide: Hume analysis view

- **Cut, Copy, Paste, Delete** are the usual operations on selected text.
- **Select All** selects the entire displayed buffer.
- **Goto Line...** moves the cursor to the numbered line.
- **Find...** searches the current window for the entered text.
- **Find Next/Prev**, once the text to be found has been entered these move to the next/previous instances of that text, wrapping round the buffer at the end/start of the text.

Currently, there is no **Undo** or **Redo** facility.

5.4 Compile, Run and Debug

5.4.1 Compiling

Once a Hume file has been loaded, it can be compiled. Note that attempting to compile before a Hume file has been loaded will result in a prompt for a Hume program to compile. Three of the principal `humec` options can be set from the toolbar along the bottom of the window:

- `debug`: this should be checked if you intend to debug the program but can be unchecked if the program is only to be run.
- `unboxed`: currently, this should match the state of the `humec` compiler since boxed/unboxed behaviour was built into the compiler when it was installed.
- `lotsaspace`: is essential for any Hume program which uses recursion of any kind.

When the `Compile` button is activated, the compilation is started from the *currently displayed source*. That is, if the Hume is displayed then the compilation is from the `.hume` file down to executable, if the HAM is displayed then the compilation is just from the `.ham` file to the executable, and likewise for the `.c` source. You can edit any of the source files but beware that editing the HAM or C source is for *low-level* debugging purposes only and requires a deep knowledge of the semantics and implementation of the compiler to be useful.

Currently, there is no notification of the success of the compilation on the main window so it is advisable to inspect the compiler output, viewable by selecting the `Compile` tab on the source view to verify correct compilation.

5.4.2 Running

If the file has been successfully compiled it can be run with the `Run` button. If your code outputs to standard output, the output can be viewed by selecting the `Console` tab in the debug view area. The `Clear` button empties the previous output.

Note that no output is flushed during the run so you may have to terminate the run to view output not terminated by a linefeed. The run is ended using the `Finish` button which simply kills the running executable.

5.4.3 Debugging

Debugging requires that the `debug` check-box is activated during compilation. Clicking the `Debug` button activates a debug. As per a program run, the `Console` window displays standard output from the debugged program, however, more information is available during a debug run.

When initiated, the debug window does not contain any information. You need to select a debug view and then *advance* the program to get the selected view. The views available are as follows:

- **Debug**, this view is used both for program state and for stack information, Figure 5. Click on the `Debug` tab to get this view.
- **Box** view displays information on a line-by-line basis for each of the boxes in the program, Figure 6. It can be displayed either by clicking the `Boxes` tab or the `Boxes` button on the debug toolbar.
- **Wire** view is similar to box view but gives information for wires, Figure 7.

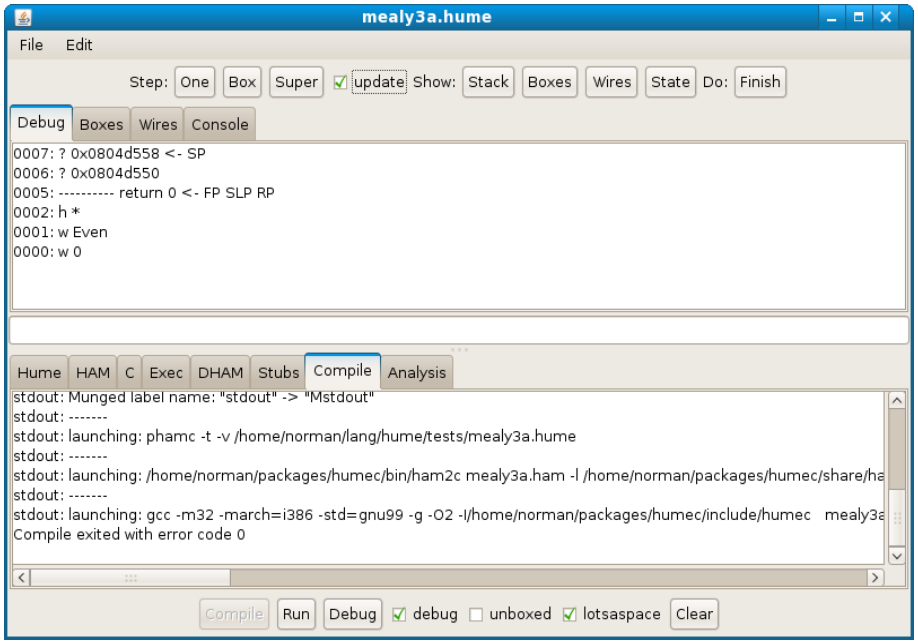

Figure 5: hide: Hume stack debug view

To advance the debug, there are three possibilities:

- **One** advances the execution by one HAM instruction. This only makes sense if one of the HAM source windows is displayed. The `Stack` debug view is also useful for this mode.
- **Box** advances the execution by one complete box execution from pattern matching through to box output. The `Box` and `Wire` displays are suitable for this mode.
- **Super** advances the by one complete Hume superstep.

Notice that the displays are not *automatically* updated after each advance unless the `update` box is checked.

Finally, the debug can be killed in the same way as a program run by clicking the `Finish` button.

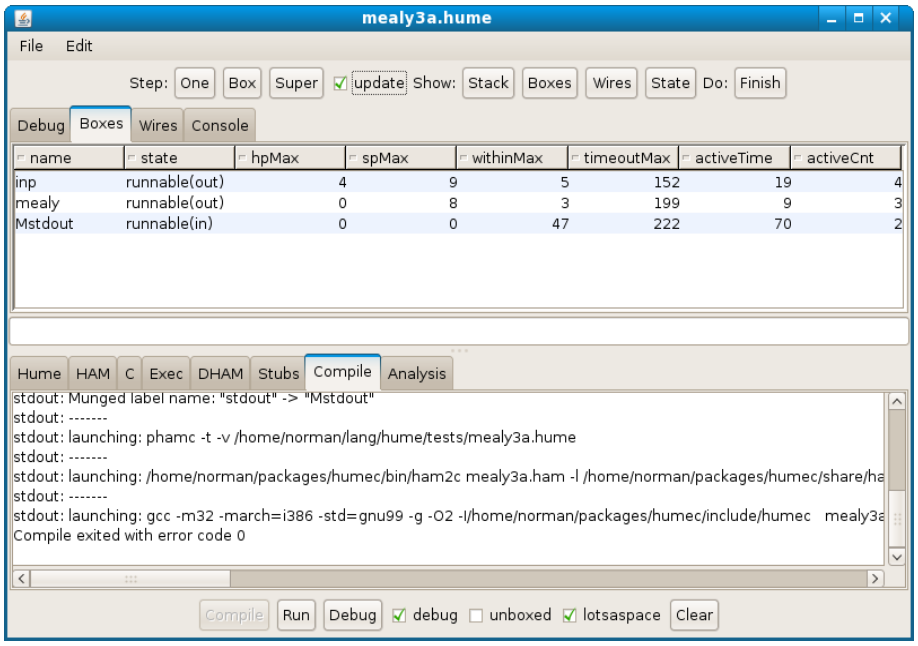

Figure 6: hide: Hume box debug view

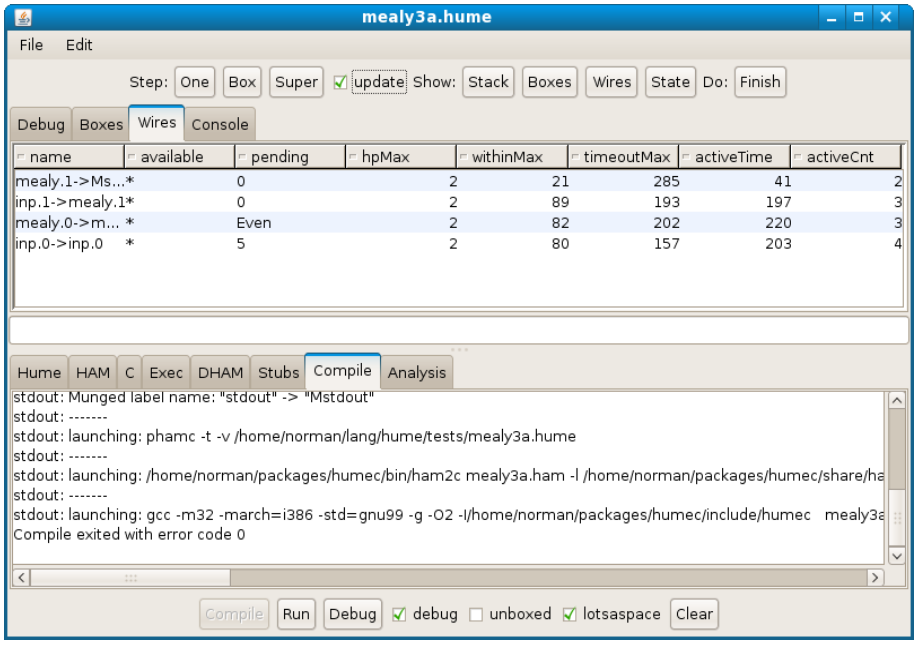

Figure 7: hide: Hume wire debug view

Note that, currently, there is no way of passing standard input to the running program.

The text area between the two display areas can be used to send low-level commands to the running program while debugging. The currently implemented commands are:

```
  step
| boxstep
| superstep
| quit
| run
| help
| show (state|stack|box)
| show box <number>
| show wire <number>
| show <address> [<size>]
```

The `help` command will print this list out while debugging.

6 Summary

We have presented an implementation for a graphical debugger for Hume programs. This tool is still under development but is already able to help with debugging simple Hume programs. We have met our objectives, as follows:

1. The Hume runtime system has been extensively modified to allow emission of resource and trace information in a standard, text-based format.

2. The interface between the running program and the debugger has been designed to allow culling of resource use and trace information as the program executes. A secondary port is opened up during debugging initialization and the debugger connects to this allowing external control of the execution process and reading of status information from the program to the debugger.

3. A Java-based graphical interface has been designed. This connects to the program under debug and can control the debug process and parse the status messages returned by the program. This interface also has all the features required to load, save and edit Hume source code.

4. The messages from the program being debugged are parsed and presented in a more readable format.

In summary, we have demonstrated a debug utility for Hume programs. Future enhancements will include the ability to link to the full resource analysis tools under development and the display and edit of Hume boxes in a graphical tool.

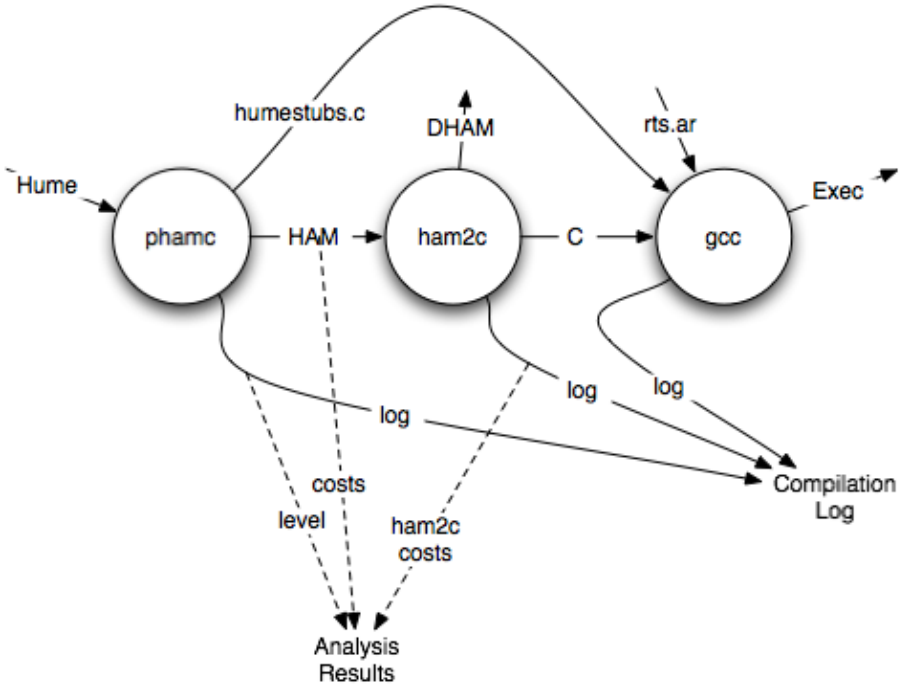

Figure 8: hide IDE Data Sources

A Data sources

The IDE culls information from the humec compilation process as shown in Figure 8.