Hume: A Domain-Specific Language for Real-Time Embedded Systems*

Kevin Hammond[1] and Greg Michaelson[2]

[1] School of Computer Science
University of St Andrews, St Andrews, Scotland
kh@dcs.st-and.ac.uk
Tel: +44-1334-463241, Fax: +44-1334-463278
[2] Dept. of Mathematics and Computer Science
Heriot-Watt University, Edinburgh, Scotland
greg@macs.hw.ac.uk
Tel: +44-131-451-3422, Fax: +44-131-451-3327

Abstract. This paper describes Hume: a novel domain-specific language whose purpose is to explore the expressibility/costability spectrum in resource-constrained systems, such as real-time embedded or control systems. Hume provides a number of high level features including higher-order functions, polymorphic types, arbitrary but sized user-defined data structures, asynchronous processes, lightweight exception handling, automatic memory management and domain-specific metaprogramming features, whilst seeking to guarantee strong space/time behaviour and maintaining overall determinacy.

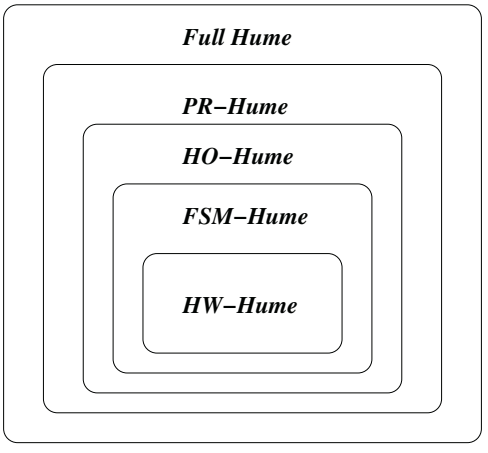
1 Essential Language Properties in the Real-Time Embedded System Domain

Embedded systems are becoming increasingly prevalent in everyday life. From a computer science perspective such systems represent a software/hardware blend at a much lower level of abstraction than usual. They also introduce strict cost-driven requirements on system capabilities. Adapting general purpose languages for such a domain often leads to a poor fit between the language features and the implementation requirements. Domain-specific language designs such as Hume allow low-level system requirements to guide the design of the high level language features we desire. We can identify a number of essential or desirable properties for a language that is aimed at real-time embedded systems [20].

– *determinacy* – the language should allow the construction of determinate systems, by which we mean that under identical environmental constraints, all executions of the system should be *observationally equivalent*;

* This work is generously supported by a grant from the UK's Engineering and Physical Sciences Research Council.

F. Pfenning and Y. Smaragdakis (Eds.): GPCE 2003, LNCS 2830, pp. 37–56, 2003.



Full Hume
Full recursion

PR–Hume
Primitive Recursive functions

HO–Hume
Non–recursive higher–order functions
Non–recursive data structures

FSM–Hume
Non–recursive first–order functions
Non–recursive data structures

HW–Hume
No functions
Non–recursive data structures

Fig. 1. Hume Design Space

- *bounded time/space* – the language must allow the construction of systems whose resource costs are statically bounded – so ensuring that *hard real-time* and *real-space* constraints can be met;
- *asynchronicity* – the language must allow the construction of systems that are capable of responding to inputs as they are received without imposing total ordering on environmental or internal interactions;
- *concurrency* – the language must allow the construction of systems as communicating units of independent computation;
- *correctness* – the language must allow a high degree of confidence that constructed systems meet their formal requirements [1].

Our goal with the Hume design is to allow as high a level of program abstraction as possible whilst still maintaining these properties. For example, we provide exception handling, automatic memory management, higher-order functions, polymorphism and recursion in different levels of the language. In each case, the requirement for transparent time and space costing has been paramount in formulating the language design.

1.1 The Hume Design Model

We have designed Hume as a three-layer language: an outer (static) declaration/metaprogramming layer, an intermediate coordination layer describing a static layout of dynamic processes, and an inner layer describing each process as a dynamic mapping from patterns that match inputs to expressions that produce outputs. Rather than attempting to apply cost modelling and correctness proving technology to an existing language framework either directly or by altering the language to a greater or lesser extent (as with e.g. RTSj [6]), our approach