The Waveform Description Language
Moving from Implementation to Specification in Ptolemy II

E.D. Willink, Thales Research Limited,
Ed.Willink@rrl.co.uk,

Ptolemy Mini-conference,
22 March 2001

Overview
The system specification problem
better implementations
Waveform Description Language solutions
Comparison of WDL and Ptolemy characteristics
WDL as an abstract Ptolemy domain
Summary

UK Programmable Digital Radio (PDR) Phase 1,
Waveform Description Language (WDL) programme
(DERA contract CU009-0000002745)

Raytheon, Communication Systems Division, Fort Wayne, Indiana,
Racal, Racal Research Limited, England
The system specification problem

Complex systems are
- costly
- late
- mis-functional
- inflexible

Complex system specifications are
- large
- ambiguous
- contradictory

Complete specification (High Level)

System specifications - very challenging
- large, unreadable

Informal specification
- terse - omissions lead to ambiguities
- verbose - duplications lead to contradictions

Formal specification
- good in principle
- impractical for real applications
- unapproachable for most practitioners

WDL
- pragmatic compromise
- formalisable, modular, familiar, practical, acceptable
The system implementation problem

Implementation practices do not support re-use
- porting - old code on new platform
- copying - old code in new context
- sharing - pre-written generic code

Re-use
- code already written
- bugs already removed

requires a well defined functionality
inhibited by implementation details

‘Lego’ Bricks

Software Defined Radio - JTRS
- assemble arbitrary waveform from pre-defined library blocks
  - waveform == radio protocol
- new waveforms incur minimal costs and delays

UK Programmable Digital Radio (phase 1)
- script for assembling ‘Lego’ bricks
- identify the ‘Lego’ bricks and their parameters

System defined by composition of sub-systems
- composition rules must be defined
- subsystem behaviour must be defined

Need a meaningful specification - WDL
Specification not Implementation

Implementation - how it can be done
Full of non-portable constraints

Specification - what must be done
Ignore all the inconvenient details

Modern tools support an increasingly generalised implementation
Ptolemy better than most
Ptolemy also an implementation
WDL abstracts Ptolemy to a specification

WDL Antecedents

Implementation practices re-applied in the specification domain

message flow diagrams
state machines
Comparison of WDL and Ptolemy

**Ptolemy**
- simulates a generalised implementation
- cross code-generates the same implementation

**WDL more abstract than Ptolemy**
- defines an abstract specification
- refines to simulate a reference model
- refines further to generate a production implementation
- more abstract scheduling
- more abstract type system
- more abstract leaf specification

Scheduling Abstraction

**Ptolemy scheduling - one domain per diagram**
- uniform policy is convenient
- uniform policy is restrictive
- chosen policy is restrictive
- each domain is a scheduling implementation

**Ptolemy does not support**
- input as Continuous Time
- overflow as Discrete Event
- samples as Synchronous Data Flow
WDL Flow Types

Ptolemy - one domain per diagram
BDF, CT, DDF, DE, DT, HDF, PN, SDF, SR, ...

WDL - one flow type per message path
- event - (Synchronous Reactive semantics - one at a time - OR)
- token - (Data Flow semantics - all at once - AND)
- value - (asynchronous - breaks sender/receiver coherence)
- signal - ('continuous flow' - CT or SDF)

WDL does support
- input as a signal flow
- overflow as an event flow
- samples as a token flow

WDL Flow Deductions

Ptolemy
- domain restricts an implementation

WDL
- flows specify a behaviour

WDL is an abstract specification domain
- translator to specific implementation domains
  - regions of consistent flow
  - smart policies for region boundaries
  - automatically exploits the EvenBetterDataFlow domain
WDL ‘extension’ to UML:
state behaviour may be a
message flow

Entity1 AND Entity2

WDL Message Flow Diagram
each arc has defined data and flow type, connecting at ports
each entity is self-scheduling - rendezvous of relevant ports
external ports to define hierarchy
**UML Comparison**

**UML Collaboration Diagram**
- no hierarchical ports
- no arc semantics
- no multi-input handling
- external scheduling

**UML Concurrent State Machine**
- ‘solid’ AND semantics
- no hierarchical ports
- arcs connect by name
- no multi-input handling

**CORBA Components**

**Extension of Object Oriented Concepts**

- Objects can provide data encapsulation
- Objects provide no synchronisation

**Components add synchronisation**

**Components remove hierarchy**
- just two scales
  - outside component - disciplined
  - inside component - anarchic

**WDL Entities add synchronisation**

**WDL Entities support hierarchy**
Example State Machine

The Waveform Description Language: Moving from Implementation to Specification in Ptolemy II

constructed type abstraction

simplistic systems
double and may be int too
array is a sequential accident

useful systems
record and array type constructors

ptolemy deduces type from values
WDL deduces or declares types
WDL has discriminated union as well
record expresses AND of fields within a message
union expresses OR of alternate messages
discriminated union ensures type safety
Bit-true Type Abstraction

Simplistic systems
just int

Implementation perspective
octet or char
fix<13,5>

Specification perspective
any type with at least 40 dB dynamic range

WDL defines minimum requirements of types
translator selects/synthesises implementation type

Bit-true Type Overlay

Implementation approach
embed bit-true declarations

But: a re-use may require a different bit-truth

Specification approach
A bit-true layout is overlaid onto abstract type
A pattern matching layin construct supports type discovery
Leaf Behaviour Abstraction

Ptolemy Classic
one template per actor per domain per target
template per domain led to domain inconsistencies
SDF without corresponding CGC support

Ptolemy II
one (Pt)Java template per actor (simulation)
one template per language per AST node (code generation)

(Pt)Java is not a specification language

Precision is implemented not specified
ok for 32 bit RISC
bad for 24 bit DSP
really bad for FPGA

Statement scheduling is sequential not parallel
ok for single CPU
bad for FPGA

Code is over-specified
loop counters must be analysed away

Overloading is limited
C++ better but not a solution
WDL leaf specification

entity Subtractor
{
in minuend;
in subtrahend;
out difference;
response minuend subtrahend // Whenever a rendezvous of
  { // minuend and subtrahend exists
    specification
    { // receive minuend and subtrahend
      difference(minuend - subtrahend); // subtract values
    };
  }
};

Polymorphic

type, shape, flow, language

Specification in WDL

Progressive decomposition
systems - subsystems - components - building blocks

Single hierarchical perspective
clear readable specification
removes ambiguities, avoids contradictions

Implementation in WDL

Progressive refinement of specification
further decomposition
practical constraints
recomposition

Minimal refinement
eexecutable reference model
The Waveform Description Language: Moving from Implementation to Specification in Ptolemy II

WDL Refinement

Progressive decomposition

Progressive refinement

Specification preserved

3 April, 2001
P6957-25-019

WDL Refinement

Sponsor

Abstract Specification
- layers
- coders
- modulators

Actual Specification
- algorithms
- parameters
- bit-truth

Reference Model
- bit precisions
- acquisition algorithms

WDL Specification

Implementer

System Design
- filter algorithms
- acquisition algorithms
- minimum precisions

System Configuration
- filter coefficients
- decimation ratios

Hardware Mapping
- partitioning
- library matching

Component Configuration
- parameterisation
- actual precisions
- bit-truth

Product

3 April, 2001
P6957-25-019
Realising WDL

**Reuse/adapt COTS**
- Ptolemy II most appropriate

**Original perception**
- WDL-specific Vergil extensions
  - WDL to Ptolemy translator embedded in Vergil

**Simpler perception**
- WDL is just a new abstract domain
  - translates itself to other domains
- WDL-enabling Vergil extensions
  - Port/Parameter configuration form
  - UML statechart
  - Nested flow chart

**WDL Compilation**

- **WDL Program**
- **WDL Compiler**
- **WDL Libraries**
- **Standard Interfaces**
  - bit truth
- **Standard Libraries**
  - pre-existing code

**WDL Libraries**
- **WDL Translator**
- **Standard Compilers**
  - C, VHDL, ... source code

**Standard Libraries**
- **Standard Libraries**
- **Standard Compilers**
- **Executables**
  - Interfaces, Configurations
  - "Waveform Bundle"
Ptolemy II Translations

WDL Translations
The Waveform Description Language: Moving from Implementation to Specification in Ptolemy II

Ptolemy extension for WDL

WDL Products

One specification

Many alternate implementations

Simulation Environments

- HLA,
- Ptolemy,
- ...

Execution Environments

- Application,
- Process,
- CORBA Component,
- FPGA,
- Analogue hardware,
- ...

3 April, 2001
P6957-25-019
The Waveform Description Language: Moving from Implementation to Specification in Ptolemy II

New programming paradigm

Only specification
- eliminates the implementation problems
- refinable to any target

Behavioural parameterisation
- inner behaviour passed in from outside
- dead specification elimination

Parallel computing
- entity/actor parallelism
- array parallelism

Re-use at last
Real or bogus panacea?

Status

Phase 1 (4 months: December 1999 to April 2000)
- Initial consideration of language concepts
- Example decomposition of FM3TR (1 month)
  - clearer, many anomalies reported back

Phase 2 (2 years: April 2001 to April 2003)
- Preliminary Editor - October 2001
- Preliminary Simulator - April 2002
- Preliminary Code Generator - October 2002
- Open source, IPR free on web.
Summary of Language

WDL specifies a determinate behaviour
Decomposition with single perspective

Refinement to a reference model
Refinement to product implementations

Polymorphism to exploit generic libraries
Realistic scheduling models
Type-oriented code generation for flexibility

Free

Summary of Comparison

Ptolemy Classic - most powerful/versatile (fast)
Ptolemy II - most powerful/versatile (flexible)
Ptolemy encourages implementation perspective
domains, exact types, (Pt)Java leaves

WDL (domain) abstracts to a specification
flows, constrained types, constraint language

WDL refines specification to an implementation
refinement to reference model part of a specification
Summary of Goals

WDL is an open program
  collaboration welcome and needed
  ? industry standard via SDR Forum and OMG ?

Better quality specifications
  sponsor provides reference model

Semi-automated code generation
  months rather than years