SMEDL Manual

University of Pennsylvania

July 20, 2020

Part I of this manual describes how to write a monitor. It discusses the features and syntax of the SMEDL specification language. Part II is about integrating monitors into your existing code. It describes the API for the generated code and the various transport mechanisms available for asynchronous connections.

Part I. The SMEDL and A4SMEDL Monitoring Language

A SMEDL monitoring system is designed at two levels: the monitor level and the architecture level. Monitor specifications describe a single monitor, that is, a list of state variables, a list of events that the monitor consumes and emits (known as “imported” and “exported” events), and a set of states, transitions, and actions describing the state machine. Architecture specifications describe how multiple monitors come together to form a monitoring system, including which monitor specifications are involved, how instances of the monitors are parameterized, and how imported and exported events are directed.

Monitor specifications are written in `.smedl` files, and there may be multiple in a monitoring system. Architecture specifications are written in `.a4smedl` files and there is exactly one per monitoring system.

1. SMEDL Specifications

A SMEDL specification describes a single monitor. This is where the actual monitoring logic is defined. At its core, a SMEDL monitor is a set of state machines. Transitions are triggered by events, with optional conditions and associated actions. Monitors may also contain state variables, which the conditions and actions can make use of.

1.1. A Basic Monitor

Listing 1 gives an example of a simple monitor. This guide will break it down piece-by-piece to familiarize you with the basic syntax of SMEDL.

```
object Adder;
```

Listing 1: A monitor that keeps a running total

```
object Adder;

state:
  float accumulator = 0;

events:
  imported measurement(float);
  exported sum(float);

scenarios:
  main:
    idle
      -> measurement(val) {
        accumulator = accumulator + val;
        raise sum(accumulator);
      }
      -> idle;
```

This line declares the monitor specification name. It must be present at the top of every monitor specification. The name can be any valid SMEDL identifier. To be specific, an upper or lowercase letter, followed by zero or more additional letters, numbers, or underscores.

This is the only line in the file that stands alone. The rest of the specification is split into three main sections, the start of each marked by a heading keyword.

```
state:
  float accumulator = 0;
```

This is the state variables section. State variables can be thought of as global variables for the monitor. They will be useful for conditions and actions (each of which are discussed later). There are seven types (and one alias) available, listed in Table 1 along with the most similar C type for each. This is the only optional section: If there is no need for state variables, it may be omitted entirely.

```
events:
  imported measurement(float);
  exported sum(float);
```

This is the event declarations section. Events are the input and output of the monitor. Imported events come from outside the monitor and may trigger transitions and actions. Exported events are generated within the monitor and sent out. (There are also internal events, introduced in subsection 1.2.)

Events can carry parameters with them. TODO

```
scenarios:
  main:
    idle
```

Table 1: The types available in SMEDL

SMEDL	C
int	int
float	double
double[a]	double
char	char
string	char * *or* char[]
pointer	void *
thread	pthread_t
opaque	SMEDLOpaque[b]

[a] double is an alias of float for convenience, not a distinct type.
[b] The SMEDLOpaque type is dicussed in

```
        -> measurement(val) {
          accumulator = accumulator + val;
          raise sum(accumulator);
        }
        -> idle;
```

1.2. A More Advanced Monitor

1.3. SMEDL Language Reference

2. A4SMEDL Specifications

2.1. A Basic Architecture

2.2. A More Advanced Architecture

2.3. A4SMEDL Language Reference

Part II.
The SMEDL Programming Interface

The generated monitoring code is separated into multiple layers, ordered here from lowest to highest:

Monitor contains the actual monitor state machines.

Local wrapper manages instances of a single monitor specification and dynamic instantiation.

Global wrapper manages all local wrappers in a single synchronous set and routes events between them.

Transport handles asynchronous communication between global wrappers and the target system. This layer is only generated when the `-t` option is used with `mgen`.

The monitor (SMEDL) specification corresponds with the monitor level and the architecture (A4SMEDL) specification corresponds with the three layers above it.

3. Common API Elements

Header: `smedl_types.h`

3.1. The `SMEDLValue` Type

labelsubsec:smedlvalue

The `SMEDLValue` type is used throughout the API for event parameters, monitor identities, etc. It is defined in Listing 2. Each `SMEDLValue` contains a type `t` and a value `v`. The type is a member of the `SMEDLType` enum and the value is a union. See Table 2 for the correspondence between the SMEDL types and union members.

Listing 2: The `SMEDLValue` and `SMEDLType` definitons

```c
typedef enum {SMEDL_INT, SMEDL_FLOAT, SMEDL_CHAR,
    SMEDL_STRING, SMEDL_POINTER, SMEDL_THREAD,
    SMEDL_OPAQUE, SMEDL_NULL} SMEDLType;

typedef struct {
    SMEDLType t;
    union {
        int i;
        double d;
        char c;
        char *s;
        void *p;
        pthread_t th;
        SMEDLOpaque o;
    } v;
} SMEDLValue;
```

The `SMEDL_NULL` type is special—it does not correspond with an actual SMEDL type. It is used internally in a couple places, but as far as the API is concerned, there is only one significant use: wildcard parameters. In a list of monitor identities, a `SMEDLValue` with `t` set to `SMEDL_NULL` is understood to be a wildcard. The `v` member is ignored in such cases.

3.2. The `SMEDLOpaque` Type

`SMEDLOpaque SMEDLOpaque`

Table 2: Correspondence between `SMEDLType` and union in `SMEDLValue`

`SMEDLType`	Union member
`SMEDL_INT`	`i`
`SMEDL_FLOAT`	`d`
`SMEDL_CHAR`	`c`
`SMEDL_STRING`	`s`
`SMEDL_POINTER`	`p`
`SMEDL_THREAD`	`th`
`SMEDL_OPAQUE`	`o`
`SMEDL_NULL`	–

4. Monitor API

5. Local Wrapper API

6. Global Wrapper API

7. Transport Adapters

7.1. RabbitMQ

7.2. File