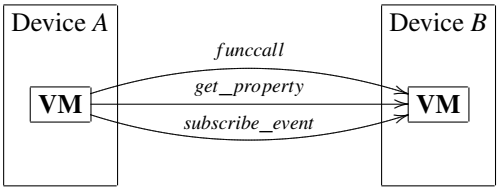
Meadow VM: Runtime for Meadow Devices

September 9, 2014

Contents

1 Introduction

The **Meadow VM** is a virtual machine which comprises the backbone of the **the Meadow system**. A Meadow device contains at least one Meadow VM running inside. Each VM is unique in the entire Meadow system, which means that each device has a globally-unqiue name.



1.1 Basic functionality

A Meadow VM in a Meadow device is responsible for maintenance of four types of values:

- **properties**, which are *pull-based data items*
- **events**, which are *push-based data items*
- **functions**, which are *pull-based computation*

- **reactors**, which are *push-based computation*

These values can be used locally by another value in the same VM or remotely from a different VM.

1.2 Meadow VM as Meadowview interpreter

A Meadow VM is capable of evaluating Meadowview definitions – i.e. Meadow code. Basically, a Mewdowview allows to *define* properties, events, functions, and reactors. A Meadow VM evaluates such definitions.

1.3 Meadow VM as special form handler

1.4 Lightweight vs Full-fledged VM

For some resource-limited devices, we could consider building a simplified device which only supports properties, etc.

1.5 Virtue of Meadow VM

Meadow VMs are useful in the sense that it hides all lower-level details in providing/using services from other devices.

2 Virtual Machines in the Meadow System

2.1 Meadow system as collection of VMs

The meadow system is a collection of VMs, which cooperate to perform tasks.

2.2 Hierarchy of VMs

VMs are organized in a hierarchy of VMs. VMs can join and leave the system and the hierarchy can change dynamically.

2.3 Name of VMs

The name of a VM is globally-unique in the entire Meadow System in any time. The VM namespace is persistent in the sense that the name of a VM will be always the same across the lifetime of a VM – creation and removal.

2.4 Namespaces of the Meadow system

There are two types of namespaces in the Meadow system: *transient* and *persistent*. A name inside a transient namespace disappears when its supervising VM is not active. A namespace of a Meadow system is implemented by one or more VMs.

2.5 Persistent namespace

A name inside a persistent namespace exists regardless of its supervising VM is active or not.

2.6 Transient namespace

3 Meadow VM Architecture

3.1 Task Execution Unit

Meadow VM contains a **task execution unit** which continuously fetches a task from the **task queue** and executes it, where a **task** is a named sequence of VM instructions.

4 VM Instruction Set

4.0.1 Instruction for events

4.0.2 ADD_EVENT

4.0.3 REMOVE_EVENT

4.0.4 SUBSCRIBE_EVENT

4.0.5 UNSUBSCRIBE_EVENT

4.0.6 Instruction for properties

4.0.7 ADD_PROPERTY

4.0.8 REMOVE_PROPERTY

4.1 Instructions for tasks

4.1.1 ADD_TASK

4.1.2 REMOVE_TASK

4.1.3 SCHEDULE_TASK

4.1.4 EVAL_TASK

4.2 Arithmetic and logic instructions

```
RATOR DEST RAND1 RAND2
```

where `DEST`, `RAND1`, and `RAND2` are symbols which can be found in the environment. If any of these three symbols are not in the environment, lookup failure exception is raised.

4.3 Branching

```
IF COND B1 B2
```

where `COND` is a symbol. `B1` and `B2` are names of blocks.

4.4 Jumping

```
GOTO B1
```

where `BLOCK` is a block name.

4.5 Block definition

```
BLOCK_BEGIN BLOCKNAME
BLOCK_END BLOCKNAME
```

Indicates the beginning and end of a block. A block is a named sequence of VM instructions which can be executed without any nonblocking events. Also, it is a basic block.

5 VM API

Each of the VM supports the APIs for handling properties, events, functions, and reactors.

5.1 API for VM

5.1.1 `get_vm_id()`

5.1.2 `get_vm_name()`

5.2 API for properties

5.2.1 `add_property(name)`

Adds the property with the given name. After its execution, the given property can be manipulated locally from within this VM or remotely from a different VM.

5.2.2 `remove_property(name)`

Remove the property with the given name from the VM.

5.2.3 `add_native_property(name, tag, ⟨tag_specific_uri⟩)`

Adds a native property service which is already provided through a distributed communication framework, such as AllJoyn. Each such communication framework should have some unique property name in its own framework.

```
add_native_property(batterylife, AllJoyn, "com.fitbit.sn002839.battlife");
```

After its execution, this AllJoyn property of the given URI can be accessed from the Meadow system.

5.2.4 `remove_native_property(name, tag, ⟨tag_specific_uri⟩)`

5.2.5 `get_property(name)`

This allows to obtain the value of the property with the given name.

5.2.6 `set_property(name, value)`

This allows to set the value of the property with the given name.

1

6.1.2 vm_uptime

6.2 Builtin per-VM events

6.2.1 vm_started_event

The event generated when VM starts execution. This event is an aggregate value, which consists of the unique ID of this VM, timestamp, etc.

6.3 Builtin per-VM functions

6.4 Builtin per-VM reactors

7 Virtual Machine Group

7.1 Rationale

A group of VMs can form a VM group. This can be either a proximity-based group of VMs or a ad-hoc, security-enforced group of VMs. A group of VMs provids additional capabilities which can be described as a "group capability."

7.2 Formation of a VM group

7.3 Dismissal of a VM group

A VM group is dismissed based on group context – dismissWhenOneLeft, neverDismiss, etc.

7.4 Lifetime of a VM group

7.5 VM vs VM group

A single VM itself is an interpreter. However, a group of VM could be dynamically combined into a larger interpreter. Since an interpreter is just a maintainer of name–value tables, this larger interpreter just additionally maintains larger-scale name–value tables of events, properties, functions, and reactors. This group table is maintained by a group of VMs – REAL CHALLENGE!!

7.6 Builtin per-VM-group properties

7.6.1 num_vms

7.7 Builtin per-VM-group events

7.7.1 vm_joined_event

7.7.2 vm_left_event

7.8 Builtin per-VM-group functions

7.8.1 list_of_vms

7.9 Builtin per-VM-group reactors

7.9.1 vm_joined_reactor

Executed when a new VM joins the given VM group. Updates num_vms property, etc. (customizable)

8 Implemenation of Virtual Machine

8.1 Protocol synthesis

Need protocols between VMs for:

- msg format
- msg types
- distributed environment (symtab)
- calling convention
- process deployment protocol
- device join/leave
- port usages

8.2 Name–value maps

Basically, a virtual machine is an interpreter which maintains maps between names and values. For VM group maps, we could use distributed hash table (DHT).

8.3 Overlay networks for VM groups

We could use overlay networks for implementing VM groups. For each VM group of which a VM is a member, the VM can be assigned a temporary name through which the members of the group can refer to the VM. This temporary name is effective only during the time while the VM group is alive.

9 Meadow Messages

All meadow communications takes place in the form of message passing. Messages can contain control information and/or payload.

9.1 Representing values

Values can be represented using multiple methods but string-encoded representation would be reasonable – e.g. AllJoyn method.

9.2 Default Message fields

- **source**
- **destination**
- **timeout**
- **timestamp**

9.3 Message types

9.3.1 Events

- EventOccurred:
  - device name
  - event name
  - field values
- EventSubscriptionRequest
  - subscriber device name
  - event name
- EventPublishRequest: actually, this is "external" request for generating internal events (e.g. PropGetRequest)
  - publisher device name
  - event name
  - field values

9.4 Functions

- AddFunction:
  - function name
  - parameters
  - function body type (raw code, compiled-IR, external-ref (alljoyn ref), etc.)
  - function body (optional?)
- RemoveFunction:
  - function name
  - function body type
- FuncCallRequest:
  - function name
  - zero or more argument values
  - token
- FuncCallResponse:
  - function name
  - zero or more argument values
  - token