ARA: Automatic Instance-Level Analysis in Real-Time Systems

Gerion Entrup, Benedikt Steinmeier, Christian Dietrich

Leibniz Universität Hannover

July 9, 2019
Getting a FreeRTOS project from Github:

```bash
% git clone https://github.com/grafalex82/GPSLogger
Cloning into 'GPSLogger'...
remote: Enumerating objects: 1245, done.
remote: Counting objects: 100% (1245/1245), done.
remote: Compressing objects: 100% (666/666), done.
remote: Total 9544 (delta 683), reused 992 (delta 567), pack-reused 8299
Receiving objects: 100% (9544/9544), 52.33 MiB | 9.47 MiB/s, done.
Resolving deltas: 100% (6615/6615), done.
```

- Repository size: 65 MiB
- 134 000 lines of code
Getting a FreeRTOS project from Github:

```bash
% git clone https://github.com/grafalex82/GPSLogger
Cloning into 'GPSLogger'...
remote: Enumerating objects: 1245, done.
remote: Counting objects: 100% (1245/1245), done.
remote: Compressing objects: 100% (666/666), done.
remote: Total 9544 (delta 683), reused 992 (delta 567), pack-reused 8299
Receiving objects: 100% (9544/9544), 52.33 MiB | 9.47 MiB/s, done.
Resolving deltas: 100% (6615/6615), done.
```

- Repository size: 65 MiB
- 134,000 lines of code

What is the systems architecture?
The Instance Graph

OSPERT’18:
Levels of Specialization in Real-Time Operating Systems

- Get instances of OS abstractions.
- Get interactions between them.
We have extracted the graph manually!
Not possible for larger code bases. We need automation!
Automatic Real-Time Systems Analyzer (ARA)

- Automatic instance graph extraction
- Static source code analysis
  - Application as input
- Supports multiple RTOS interfaces.
  (currently FreeRTOS and OSEK/AUTOSAR)
- Fields of use:
  - System overview
  - Knowledge extraction for specialization
  - OS-API usage validation
Agenda

- Motivation
- Technique
- Experiments
- Conclusion
# ARA in a Nutshell

## RTOS-API

![Source](image.png)

## Control flow analysis

## Value analysis

## Instance graph

```c
TaskHandle_t h = NULL;

int main() {
    xTaskCreate(vTask1, "Task1", NULL);
    xTaskCreate(vTask2, "Task2", h);
    // Should never reach this while(1);
    return 0;
}

void vTask2(void *param) {
    do_long_operation();
    xTaskDelete(h);
}
}
```

### RTOS-API

- GPIO
- Serial
- SPI
- UART
- Timer

### Control flow analysis

### Value analysis

### Instance graph

**ARA**

**RTOS mapping**

**Control flow analysis**

**Value analysis**

**Instance graph**
ARA in a Nutshell

Source

RTOS-API

programmed against

Instance graph

ORA

RTOS mapping

Control flow analysis

Value analysis

LUH ARA: Automatic Instance-Level Analysis in Real-Time Systems – Technique 6 – 21
OSEK/AUTOSAR vs. FreeRTOS

OSEK/AUTOSAR

```oil
TASK t1 {
    PRIORITY = 1;
    SCHEDULE = FULL;
    AUTOSTART = TRUE;
}

TASK t2 {
    PRIORITY = 2;
    SCHEDULE = FULL;
}
```

FreeRTOS

```cpp
TaskHandle_t t1, t2;

int main() {
    t1 = xTaskCreate(task_1, 2);
    t2 = xTaskCreate(task_2, 1);
    vTaskStartScheduler();
}

task_1 { // priority: 2
    vTaskNotifyGive(t1);
}

task_2 { // priority: 1
    while (true) {
        ulTaskNotifyTake();
        vTaskDelete(NULL);
    }
}
```
RTOS Mapping

- Detect all **system calls**
- Create unified model

# OSEK
"ActivateTask": (os_type.activate, ...)
"TerminateTask": (os_type.destroy, ...)
"GetResource": (os_type.take, ...)
"ReleaseResource": (os_type.commit, ...)

# FreeRTOS
"xTaskCreate": (os_type.create, ...)
"vTaskNotifyGive": (os_type.commit, ...)
"ulTaskNotifyTake": (os_type.take, ...)
"xQueueTakeMutexRecursive": (os_type.take, ...)
"xQueueGiveMutexRecursive": (os_type.commit, ...)

- Create parser for extra data (like OIL file).
ARA in a Nutshell

Source

RTOS-API

programmed against

Control flow analysis

Value analysis

Instance graph

ARA

RTOS mapping
ARA in a Nutshell

RTOS-API

programmed against

Source

Control flow analysis

Value analysis

Instance graph
System-Call Aware ICFG

1. Extract interprocedural control flow graph (with LLVM).

```c
void recv();
void send();

void create(int p2) {
    int foo = 0;
    xTaskCreate(recv, 3);
    if (foo == 0)
        foo ++;
    foo += 4;
    xTaskCreate(send, p2);
    return;
}

int main() {
    create(5);
    return;
}
```
System-Call Aware ICFG

1. Extract interprocedural control flow graph (with LLVM).
2. Split calls in separate blocks.
System-Call Aware ICFG

1. Extract interprocedural control flow graph (with LLVM).
2. Split calls in separate blocks.
3. Label block types. system call, call, computation
System-Call Aware ICFG

1. Extract interprocedural control flow graph (with LLVM).
2. Split calls in separate blocks.
3. Label block types.
   system call, call, computation
4. Merge appropriate computation blocks.
ARA in a Nutshell

```
int main() {
    xTaskCreate(vTask1, "Task1", NULL);
    xTaskCreate(vTask2, "Task2", &h);
    xTaskStartScheduler();
    while(1) {
        vTaskDelay(100);
    }
}
```

**RTOS-API**

- Control flow analysis
- RTOS mapping
- Value analysis
- Instance graph

**Source**

- LED
- SPI DMA
- I2C DMA
- Button
- Events
- SD Writer
- Thread
- Lock
- Semaphore
- Display
- Logging
- Queue
- GPS
- Thread
- ISR
- Serial DMA

**Programmed against**

- LUH ARA: Automatic Instance-Level Analysis in Real-Time Systems – Technique
ARA in a Nutshell

Source

RTOS-API

programmed against

Control flow analysis

Value analysis

Instance graph

ARA

RTOS mapping
Value Analysis

- Get arguments for system calls.
- Backward search from the call site.
- Follow def-use chain.
- Follow callee-caller relationship.
- Take unambiguous values.

```c
void recv();
void send();

void create(int p2) {
    int foo = 0;
    xTaskCreate(recv, 3);
    if (foo == 0)
        foo++;
    foo += 4;
    xTaskCreate(send, p2);
    return;
}

int main() {
    create(5);
    return;
}
```
Value Analysis

- Get arguments for system calls.
- Backward search from the call site.
- Follow def-use chain.
- Follow callee-caller relationship.
- Take unambiguous values.

```c
void recv();
void send();

void create(int p2) {
    int foo = 0;
    xTaskCreate(recv, 3);
    if (foo == 0)
        foo ++;
    foo += 4;
    xTaskCreate(send, p2);
    return;
}

int main() {
    create(5);
    return;
}
```
Value Analysis

- Get arguments for system calls.
- Backward search from the call site.
- Follow def-use chain.
- Follow callee-caller relationship.
- Take unambiguous values.

```c
void recv();
void send();

void create(int p2) {
    int foo = 0;
    xTaskCreate(recv, 3);
    if (foo == 0)
        foo ++;
    foo += 4;
    xTaskCreate(send, p2);
    return;
}

int main() {
    create(5);
    return;
}
```
ARA in a Nutshell
ARA in a Nutshell

```
int main() {
    xTaskCreate(vTask1, "Task1", &h);
    xTaskCreate(vTask2, "Task2", &h);
    vTaskDelay(100);
}

void vTask2(void* param) {
    do_long_operation();
    xTaskDelete(h);
}
```

**RTOS-API**

- Control flow analysis
- RTOS mapping
- Value analysis
- Source

**Instance graph**

- Programmed against
- LED
- Thread
- Lock
- Semaphore
- Display
- SPI DMA
- ISR
- Button
- Events
- Queue
- SD Writer
- Thread
- GPS
- Logging
- Queue
- Serial DMA

**LUH ARA:** Automatic Instance-Level Analysis in Real-Time Systems – Technique
Instance Graph Creation

- Instance creation in branch or loop?
  - ARA marks them with “?”.
- Instance creation before or after scheduler start?
  - Before: Only runs once.
  - After: Unknown number of runs.
  - ARA sets “late” attribute.

```
main
xTaskCreate
T1
late: False
xTaskCreate?
T2
late: True
```
Experiments

- Show viability of approach.
- Tested with 4 real-world systems:
  - GPSLogger (FreeRTOS)
  - SmartPlug\(^1\) (FreeRTOS)
  - I4Copter with events (OSEK)
  - I4Copter without events (OSEK)
- Implemented three validation tests:
  - FreeRTOS: Only ISR-capable system calls used in ISRs?
  - OSEK: Does OIL-file match the source code?
  - FreeRTOS/OSEK: Enter and exit of critical region always pairwise?

\(^1\)https://github.com/KKoopvalsky/Smartplug
GPSLogger

RTOS

SDThread

late: False

vTaskDelay

sdQueue

late: True

xQueueGenericCreate

xQueueReceive

LEDThread

late: False

vTaskDelay

buttonsQueue

late: False

xQueueGenericSend

xQueueSemaphoreTake

GPSTask

late: False

xQueueSemaphoreTake

xQueueCreateMutex?

xQueueSemaphoreTake

xQueueCreateMutex?

xQueueSemaphoreTake

xQueueCreateMutex?

xQueueSemaphoreTake

xQueueCreateMutex?

DisplayTask

late: False

vTaskDelay

xQueueSemaphoreTake

xQueueCreateMutex?

xQueueSemaphoreTake

xQueueCreateMutex?

DisplayTask

late: False

vTaskDelay

xQueueSemaphoreTake

xQueueCreateMutex?

xQueueSemaphoreTake

xQueueCreateMutex?

DisplayTask

late: False

vTaskDelay

xQueueSemaphoreTake

xQueueCreateMutex?

xQueueSemaphoreTake

xQueueCreateMutex?

DisplayTask

late: False

vTaskDelay

HAL_I2C_MemTxCpltCallback

late: False

vTaskNotifyGiveFromISR

USART1_IRQHandler

late: False

vTaskNotifyGiveFromISR

main

vTaskStartScheduler

xTaskCreate

xTaskCreate

xTaskCreate

xTaskCreate

xQueueGenericCreate

xTaskCreate

xTaskCreate
Future Work

- Build a global control flow graph (GCFG) [DHL17].
  - Include scheduler decisions.
- Improve value analysis.
  - Alias analysis.
  - Model ambiguous values.
- Interactive graph browsing.
  - Link source code and instance graph.
Conclusion

- **ARA\(^2\)**
  - Automatic extraction of an instance graph.
  - Supports multiple RTOS interfaces.
  - Show viability with 4 real-world applications.

- **Fields of use:**
  - Application architecture overview.
  - Knowledge extraction for specialization.
  - OS-API usage validation.

\(^2\)[https://github.com/luhsra/ara]
Conclusion

- **ARA$^2$**
  - Automatic extraction of an instance graph.
  - Supports multiple RTOS interfaces.
  - Show viability with 4 real-world applications.

- **Fields of use:**
  - Application architecture overview.
  - Knowledge extraction for specialization.
  - OS-API usage validation.

---

Thank you! Questions?

---

$^2$https://github.com/luhsra/ara