D5.3

CEA's use case intermediate report

Project number:	731453
Project acronym:	VESSEDIA
Project title:	Verification engineering of the safety and security of critical dynamic industrial applications
Start date of the project:	1st January, 2017
Duration:	36 months
Programme:	H2020-DS-2016-2017

Deliverable type:	Report
Deliverable reference number:	DS-01-731453 / D5.3 / 1.0
Work package contributing to the deliverable:	WP 5
Due date:	Jun 2018– M18
Actual submission date:	29th June, 2018

Responsible organisation:	CEA
Editor:	Mounir KELLIL
Dissemination level:	PU
Revision:	1.0

Abstract:	The objective of this document is to discuss the status of the analysis of source code associated to a number of critical functionalities of the CEA use case. The status of code analysis is also discussed, and future work on this subject is highlighted.
Keywords:	Firmware update, 6LowPAN, data communication

The project VESSEDIA has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731453.

Editor

Mounir Kellil (CEA)

Contributors (ordered according to beneficiary numbers)

Mounir Kellil (CEA)

Pierre Roux (CEA)

Boutheina BANNOUR (CEA)

Jens GERLACH (FOKUS)

Disclaimer

Executive Summary

Wireless low power and lossy channel networks (LLNs) are being extensively deployed in emerging IoT ecosystems thanks to the development of a variety of radio standards and technologies for low power devices like the 802.15.4 family, Z-WAVE, BL/BLE, etc. Because of the varying network conditions in the LLN context and in order to ensure LLN network sustainability, the management of LLN nodes (software updates, information collection, and configuration) is an important task. Indeed, this management enables the reconfiguration of the network to achieve better performance (end-to-end delays, energy saving, fault-tolerance, etc.), correct software bugs, upgrade the OS/software, enforce security services, and so forth. However, LLN networks are usually deployed in hard-to-reach environments where physical access may be difficult, dangerous, and/or expensive. For example, there could be too many deployed nodes, or the nodes could be deployed inside pipelines, hazardous zones, and so forth." The CEA use case exposes a remote management platform for 6LowPAN LLN networks[1]. The platform integrates over-the-air firmware update operations on the 6LowPAN nodes, where the firmware update may be partial/modular or full.

The objective of this document is to discuss the status of the analysis of the source code associated to a number of critical functionalities of the CEA use case, i.e. the 6LowPAN management platform. In particular, different source code parts of the CEA use case have been identified for static analysis. The identified parts of the C source code are the different operations on the Flash memory of the LLN node as well as the firmware data transmission in the 6LowPAN mesh network. For the Java source code, all the Activity templates have been targeted with a focus on detecting race conditions.

After an initial training phase on formal verification and static analysis tools, FRAMA-C WP and Verifast are now being applied respectively for the C code and Java code of the CEA use case.

Static analysis of the code with WP plugin and Verifast will be completed in the next step. In addition, the use of EVA plugin to detect run-time errors will be undertaken for critical phases of read/erase operations on the Flash memory (with a particular attention to interruption disabling during the Flash read/write processes) as well as firmware data transfer in the 6LowPAN mesh network. Finally, automatic inference of complex ACSL properties will be exploited based on inputs from WP3 (cf. D3.1) to complement the currently hand-written ACSL contracts in the source code of the CEA use case.

[1] 6LowPAN [1] is a standard protocol that enables transmission of IPv6 Packets over IEEE 802.15.4 Networks (RFC 6282).

Contents

Chapter 1 Introduction 1
1.1 Why code analysis for the 6LowPAN management platform 1
1.2 Structure of the Document 1
1.3 Related deliverables 1
Chapter 2 6LowPAN management platform 2
2.1 Overview 2
2.2 Target source code for static analysis 2
Chapter 3 Code Analysis 1
3.1 C Code 1
3.1.1 Use of WP plugin 1
3.2 Java Code 4
Chapter 4 Summary and Conclusion 5
Chapter 5 List of Abbreviations 6
Chapter 6 Bibliography 7

List of Figures

Figure 1: 6LowPAN Platform Overview 2
Figure 2: 6LowPAN Platform – key functionalities per functional component 3
Figure 3: Activity templates of the network manager's java source code 1
Figure 4: Example #1 of ACSL annotation for MPL routing source code (init() function) 3
Figure 5: Example #2 of ACSL annotation for MPL routing source code (buffer_allocate () function) 3
Figure 6: Example #3 of ACSL annotation for MPL routing source code (window_allocate () function) 4

List of Tables

Table 1: Critical functionalities and associated critical assets from D1.2 3
Table 2: Annotated sections for operations on LLN node's Flash memory 2
Table 3: Annotated sections for firmware data transmission in the 6LowPAN mesh network 2

Chapter 1 Introduction

1.1 Why code analysis for the 6LowPAN management platform

The management of low power networks like 6LowPAN networks is particularly important for network operators and network service providers because it aims at ensuring a good network performance, while maximizing network lifetime by correcting software bugs, enforcing security services, and so forth. However, LLN networks are usually deployed in hard-to-reach environments (e.g., inside pipelines, and hazardous zones) or could be massively deployed over large areas like industrial plants, smart cities, etc. This makes manual maintenance particularly difficult. In such cases, remotely managing nodes is an obvious alternative to the management of nodes via physical access. The CEA use case is represented by a remote management platform for 6LowPAN mesh networks (6LowPAN management platform). The platform integrates over-the-air firmware update operations on the 6LowPAN nodes, where the firmware update may be partial/modular or full.

The set of firmware update functions modify the behaviour of the 6LowPAN network and associated services and applications from one setting to another, without interrupting the current setting. This critical procedure should be safe in order to avoid service interruption or abnormal behaviour of the 6LowPAN network (e.g., unexpected node reboot, connection interruption, buffer overflow, etc). Code analysis of the 6LowPAN platform enables the verification of the correct behaviour of the said platform and identifies potential run-time errors during the firmware update phase.

1.2 Structure of the Document

The present document is structured as follows. Chapter 1 stresses the need for analyzing the source code of the CEA use case. Chapter 2 briefly reviews the CEA use case and points out the critical functionalities that need to be analysed. Chapter 3 presents the status of source code analysis for the CEA use case. Chapter 4 concludes this document.

1.3 Related deliverables

This deliverable is closely related to the D1.2, which describes the requirements of the different use cases of WP5 and points out the assets to protect in each use case.

This deliverable is also closely related to D3.1, which discusses the automation of the inference of properties on safety-critical scenarios, including the firmware update scenario.

Chapter 2 6LowPAN management platform

2.1 Overview

The 6LowPAN platform aims at providing firmware updates for 6LowPAN mesh networks. It comprises three functional components: 1) LLN Network Manager, 2) LLN node, and 3) LLN gateway (cf. Figure 1).

- **Management server:** this component runs on Android OS. It is in charge of transmitting firmware updates and reboot requests to the Low power & Lossy Network (LLN) node.
- **Gateway (GW):** it runs on Embedded Linux OS. It is in charge of interconnecting the LLN network with a WAN to enable communication exchange between the management server and the LLN node.
- **LLN node:** it is an embedded hardware platform with a microcontroller in addition to one or more sensors and/or actuators. It runs on Contiki OS. The managed node runs some specific application layer program (e.g., transmitting environmental/physical information like temperature, position, etc.) to the management server. In addition, the LLN node is in charge of forwarding/routing data packets in the LLN network. Also, the LLN node dynamically loads the new (piece of) firmware after it completes the reception of firmware update data from the management server.

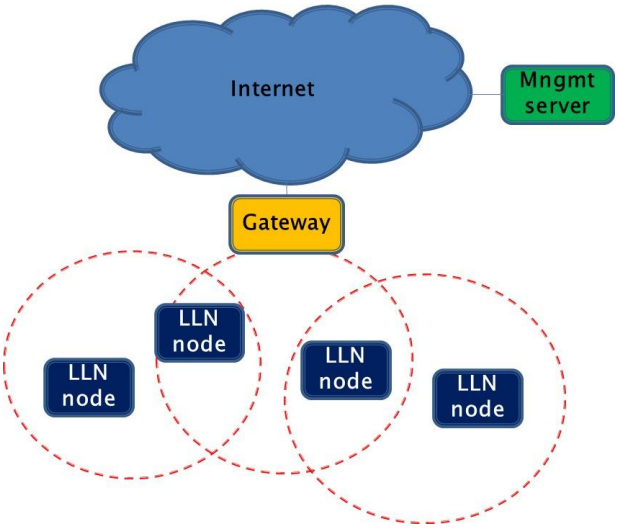

Figure 1: 6LowPAN Platform Overview

2.2 Target source code for static analysis

The gateway and LLN node functional components of the 6LowPAN management platform have been implemented using C in a Contiki OS environment (although the gateway runs on a Linux/embedded Linux environment). The network manager has been implemented with Java/Android. Figure 2 illustrates this and highlights the main functionalities per functional component.

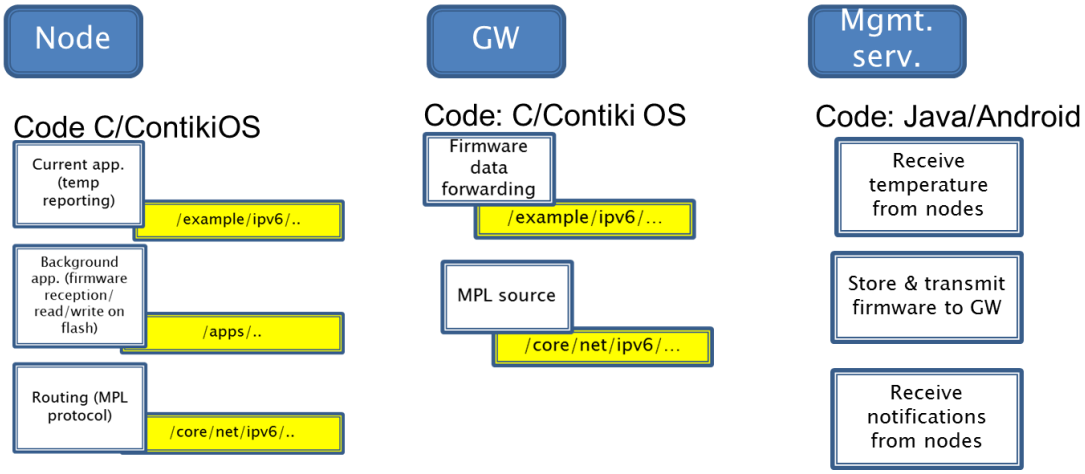

Figure 2: 6LowPAN Platform – key functionalities per functional component

Choice of C source code to be analyzed:

The D2.1 document exposes a set of critical assets related to the 6LowPAN management platform. The C source part represents a subset of the critical assets identified for the 6LowPAN management platform. This subset can be summarized in two critical functionalities:

- *Operations on the Flash memory of the LLN node:* it includes the set of operations related to Flash memory initialization before the LLN node starts writing the first data of the new firmware on the Flash memory, as well as read/write operations of firmware data on a pre-allocated Flash memory area. These operations are particularly critical because any Flash memory read/write failure will impact the LLN node (e.g., node failure, permanent reboot, etc.) and, consequently, impact its neighbouring nodes as well.
- *Firmware data transmission in the 6LowPAN mesh network:* it includes the operations related to the transmission of the firmware data hop-by-hop, starting from the gateway until the last-hop nodes. The transmission mechanism is based on MPL protocol [2], which operates, in a distributed fashion, following three parallel phases: broadcast packets to next-hop neighbours, announce missing packets, and re-broadcast announced missing packets. This three-phased protocol should run safely by ensuring that the full firmware has been delivered to all the network nodes with a minimum communication overhead.

Critical function	Critical asset (D2.1)				
	Firmware image transmission	Flash memory partition initialization	Notification of end of firmware transmission	Loading of the new firmware	Reboot command
Operations on the Flash memory of the LLN node.		X	X	X	X
Firmware data transmission in the 6LowPAN mesh network.	X		X		X

Table 1: Critical functionalities and associated critical assets from D1.2

Choice of java source code to be analyzed:

The Java source of the network manager comprises various parts (or Activity templates) illustrated in the following figure.

To perform code analysis on the network manger's Java source code, all the Activity templates will be analysed with the focus on detecting race conditions (e.g., variable sharing in a multi-threading scenario).

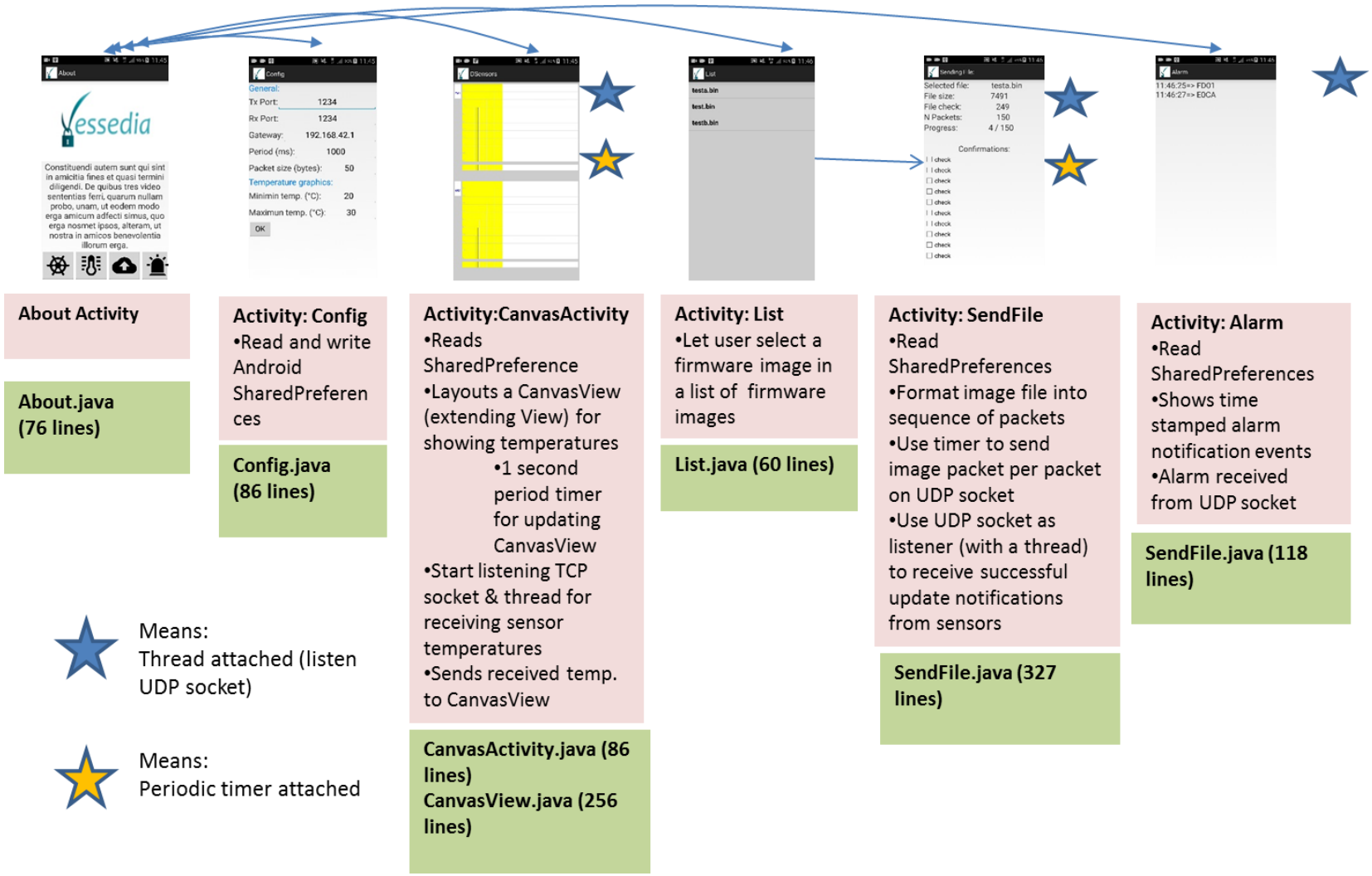

Figure 3: Activity templates of the network manager's java source code

Chapter 3 Code Analysis

This section presents the status of code analysis for the CEA use case based on the selected parts of source code glimpsed in section 2.2.

3.1 C Code

To analyse the C source code associated to the LLN node and gateway functional components of the 6LowPAN platform, the FRAMA-C WP [3] and EVA [4] plugins have been studied. The WP plugin has been considered to perform static analysis of the C code.

3.1.1 Use of WP plugin

After the WP plugin has been installed, it was necessary to write ACSL contracts in the target C source code. Also, because FRAMA-C WP plugin needs to recognize the Contiki libraries used by the 6LwPAN LLN node, a specific script written by FOKUS (script 'fr') has been adapted to enable FRAMA-C WP analysis for both the operations on Flash memory and firmware data transfer in the 6LowPAN mesh network.

Case of operations on memory Flash of the LLN node:

This part of the code utilizes Contiki OS general function calls (e.g., socket functions, timer functions, etc., cf. D5.1 for further details on the analysis of such functions), except for specific read/write/erase operations on the Flash memory that are worth analysing. CEA has written a copy of the source code of those operations, because only the associated API (like function prototype definition) is exposed by Contiki OS[2].

Table 2 summarizes the annotated functions to be analysed with WP.

Annotated section of the source code	Description	Status of analysis with WP
long rom_util_page_erase (unsigned long ulStartAddress, unsigned long ulEraseSize)[3]	Erase a flash memory bloc of size: ulEraseSize	Modification of Contiki OS is in progress to use CEA code instead of Contiki's Flash API.
long rom_util_program_flash (unsigned long *pulData, unsigned long ulAddress, unsigned long ulCount)[4]	Write on a Flash memory space starting from address: ulAddress	Modification of Contiki OS is in progress to use CEA code instead of Contiki's Flash API.

[2] The hardware platform used with Contki OS is opemote, which has a Cortex M3 CC2538 microcontroller, with a 512 KB Flash memory.

[3] Preliminary analysis of this function with FRAMA-C ACSL/RPP plugin has been presented in section 6.2 of D3.1.

[4] Same as for function 'rom_util_page_erase'.

Annotated section of the source code	Description	Status of analysis with WP
uint32 FlashGet(uint32_t ui32Addr)	Read 4 bytes at address: ui32Addr	Modification of Contiki OS is in progress to use CEA code instead of Contiki's Flash API.

Table 2: Annotated sections for operations on LLN node's Flash memory

Case of firmware data transmission in the 6LowPAN mesh network:

Analysing firmware transmission of the CEA use case leads to the analysis of the source code of the MPL routing protocol of Contiki OS[5]. This code is particularly critical in that if a bug occurs in the MPL routing, this may result in the interruption of the firmware transfer procedure. Therefore, it is particularly important to analyse the MPL source code with a particular focus on loops and buffers. Different ACSL annotations have been added to the MPL source code. The analysis of the full MPL source code with WP plugin is in progress. The table below summarizes the different sections of the MPL source code. Figure 4 to Figure 6 provide some examples of annotated sections of the MPL source code.

Annotated section of the source code	Description	Status of analysis with WP
static void init()	Initialization of MPL routing timers and buffers	ACSL annotations added for code analysis.
static struct sliding_window * window_allocate()	Allocation of the sliding window for packet sequence numbers	ACSL annotations added to perform code analysis.
static void window_update_bounds(void)	Update upper and lower bounds of the sliding window	ACSL annotations added to perform code analysis.
static struct mcast_packet * buffer_reclaim(void)	Release a data buffer entry	ACSL annotations added to perform code analysis.
static struct mcast_packet * buffer_allocate(void)	Allocate additional memory space for a data buffer	ACSL annotations added to perform code analysis.
static void icmp_output(void)	Transmit ICMP control messages	ACSL annotations added to perform code analysis.
static void icmp_input(void)	Receive ICMP control messages	ACSL annotations added to perform code analysis.

Table 3: Annotated sections for firmware data transmission in the 6LowPAN mesh network

[5] MPL implementation of Conitki OS has been a bit modified by CEA to enable interfacing with the firmware-specific operation on the Flash memory of the LLN node.

```
/*@ requires \valid iterswptr;
*/

static void
init()
{
  PRINTF("ROLL TM: ROLL Multicast - Draft #%u\n", ROLL_TM_VER);

  memset(windows, 0, sizeof(windows));
  memset(buffered_msgs, 0, sizeof(buffered_msgs));
  memset(t, 0, sizeof(t));

  ROLL_TM_STATS_INIT();
  UIP_MCAST6_STATS_INIT(&stats);

  /* Register the ICMPv6 input handler */
  uip_icmp6_register_input_handler(&roll_tm_icmp_handler);

/*@ loop invariant windows<=iterswptr<=&windows[ROLL_TM_WINS - 1];
    loop invariant \forall integer k: iterswptr<=k<=&windows[ROLL_TM_WINS - 1] ==> k->lower_bound = -1; k->upper_bound = -1; k->min_listed = -1;
     loop assigns iterswptr;
     loop variant iterswptr-windows;
     */
  for(iterswptr = &windows[ROLL_TM_WINS - 1]; iterswptr >= windows;
      iterswptr--) {
    iterswptr->lower_bound = -1;
    iterswptr->upper_bound = -1;
    iterswptr->min_listed = -1;
  }

  TIMER_CONFIGURE(0);
  reset_trickle_timer(0);
  TIMER_CONFIGURE(1);
  reset_trickle_timer(1);
  return;
}
/*---------------------------------------------------------------------------*/
```

Figure 4: Example #1 of ACSL annotation for MPL routing source code (init() function)

```
/*@ requires /valid locmpptr; */

static struct mcast_packet *
buffer_allocate()
{
/*@ loop invariant buffered_msgs<=locmpptr<=&buffered_msgs[ROLL_TM_BUFF_NUM - 1];
    loop invariant \forall integer k: locmpptr<=k<=&buffered_msgs[ROLL_TM_BUFF_NUM - 1] ==> k->lower_bound = -1; k->upper_bound = -1; k->min_listed = -1;
     loop assigns locmpptr;
     loop variant locmpptr- buffered_msgs;
     */

  for(locmpptr = &buffered_msgs[ROLL_TM_BUFF_NUM - 1];
      locmpptr >= buffered_msgs; locmpptr--) {
    if(!MCAST_PACKET_IS_USED(locmpptr)) {
      return locmpptr;
    }
  }
  return NULL;
}
```

Figure 5: Example #2 of ACSL annotation for MPL routing source code (buffer_allocate () function)

```
/* @ ensures \result==NULL || \reslut==iterswptr ; */

static struct sliding_window *
window_allocate()
{
    /*@ loop invariant windows<=iterswptr<=&windows[ROLL_TM_WINS - 1];
    loop invariant \forall integer k: iterswptr<=k<=&windows[ROLL_TM_WINS - 1] ==> k->lower_bound = -1; k->upper_bound = -1; k->min_listed = -1;
     loop assigns iterswptr;
     loop variant iterswptr-windows;
     */
   for(iterswptr = &windows[ROLL_TM_WINS - 1]; iterswptr >= windows;
       iterswptr--) {
    if(!SLIDING_WINDOW_IS_USED(iterswptr)) {
      iterswptr->count = 0;
      iterswptr->lower_bound = -1;
      iterswptr->upper_bound = -1;
      iterswptr->min_listed = -1;
      return iterswptr;
    }
   }
   return NULL;
}
/*----------------------------------------------------------------------------*/
```

Figure 6: Example #3 of ACSL annotation for MPL routing source code (window_allocate () function)

3.2 Java Code

To analyse Java source code associated to the group manager, the Verifast analysis tool [5] has been studied. Then, the Java source code has been reviewed, with the objective being to focus on detecting race conditions in the code. The question of race conditions may potentially arise from a number of Java class sources in the group manager application. This is due to the multithreading nature of the application. To check/confirm this risk, a collaborative work between CEA and KUL has been undertaken. In particular, some specific parts of the Java code has been reviewed, annotated, and ultimately analysed by Verifast.

For instance, a race condition has been detected in a Boolean variable, which may be jointly accessed from both an Android GUI thread and a UDP listening thread launched by a CEA-defined java class instance ("Alarm.java"). This class extends the Android "Activity" (android.app.Activity) and is in charge of receiving alarms from sensors, once they have been upgraded with movement detection capabilities.

Concretely, a Boolean variable is set in the "onDestroy" method of the Android Activity class. This variable is monitored in the UDP listening thread. And, once the Boolean variable is set, the thread terminates itself. The race condition detection was on the management of this Boolean variable.

To get rid of this race condition, it was possible either to declare the Boolean variable as "volatile" or to replace this simple Boolean variable by an object instantiated from the "AtomicBoolean" class. The second fix have been selected and implemented.

Then Verifast has been run again on the code with the appropriate annotation in order to confirm that the fixed up java code was free of race condition regarding this mechanism.

Another example of potential race is being investigated in the Android OS itself, by assessing the way Android OS uses a Handler object instantiated from the Handler java class (android.os.Handler). Indeed, the Android API forbids any networking operation to be performed inside graphical operations (because of non-responsive graphical applications in the case of networking lags). Therefore, a well-known practice consists in using a messaging mechanism provided with the Handler java class whenever graphical methods needs to interact with networking. However, this practice may potentially introduce a race condition.

Chapter 4 Summary and Conclusion

This document discussed the status of the analysis of the source code associated to a number of critical functionalities of the CEA use case, i.e. the 6LowPAN management platform. In particular, different parts of the CEA use case's source code have been identified for static analysis. After an initial training phase on formal verification and static analysis tools, FRAMA-C WP and Verifast are now being applied respectively for the C code and Java code of the CEA use case.

In the next step, static analysis of the code with WP plugin and Verifast will be completed. In addition, the use of EVA plugin to detect run-time errors will be undertaken for critical phases of read/erase operations on the Flash memory (with a particular attention to the interruption disabling during the Flash read/write processes) as well as firmware data transfer in the 6LowPAN mesh network.

Finally, automatic inference of complex ACSL properties will be exploited based on inputs from WP3 (cf. D3.1) to complement the currently hand-written ACSL contracts in the source code of the CEA use case.

Chapter 5 List of Abbreviations

Abbreviation	Translation
6LowPAN	IPv6 over Low-Power Wireless Personal Area Networks
GW	Gateway
IoT	Internet of Things
LLN	Low power and Lossy channel Network
MPL	Multicast routing Protocol for Low power and lossy channel networks

Chapter 6 Bibliography

[1] J. Hui et al., "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", IETF standard, RFC 6282, September 2011.

[2] J. Hui et al., "Multicast Protocol for Low-Power and Lossy Networks (MPL)", Internet Standard, RFC 7731, February 2016.

[3] FRAMA-C WP, https://frama-c.com/wp.html

[4] FRAMA-C EVA, https://frama-c.com/value.html

[5] Verifast, https://github.com/verifast/verifast