=================================================================

EMSOFT 2016 Reviews for Submission #20

=================================================================

Title: Bluetooth Loader for mruby Bytecode in Multiple Virtual Machine Environment

Authors: Takuro Yamamoto, Hiroshi Oyama and Takuya Azumi

=================================================================

REVIEWER #1

=================================================================

---------------------------------------------------------------------------

Comments

---------------------------------------------------------------------------

The mbury on TECS (TOPPERS Embedded Component System) framework can be used for the development of component-based embedded systems. The paper proposes a bluetooth loader to simplify the loading process of mruby bytecode and a RiteVM Scheduler to enable the execution of multiple VMs.

The work lacks novelty and seems to be pure engineering without gained insight. The proposed technology is not new and already exists in several commercial and open source products. Being an implementation-based paper, I was expecting some fundamental questions to be answered. For instance, (1) what is the key innovation of the paper, or (2) what are the real challenges in implementing a Bluetooth loader inside mruby on TECS.

See these examples for a Bluetooth-based loader:

Open source Code e.g.,: <https://github.com/luciodj/BlueBoot>

Article:

<http://electronicdesign.com/communications/bootload-wirelessly-using-bluetooth>-

serial-port-profile

LightBlue Bean: <http://embeddedcomputing.weebly.com/lightblue-bean.html>

Component-based design was introduced to reduce the complexity of building large systems. It allows reusability of components. TECS is a framework that offers CBD, and as such, may offer all the benefits of CBD. The proposed approach is only a component of TECS and therefore, the paper cannot claim the benefits of CBD as contribution.

The examples in Figures 4, 5, 6 are somehow difficult to follow. It would be better to provide some detail explanations about the example used in the figures. Moreover, the author could shorten this background section as it has been elaborated in previous papers on mruby on TECS.

The description of the algorithms in Figures 12 and 13 are too low level and difficult to follow. Many functions are used without any explanation, which does not help the reader. The same issue appears later with the presentation of the scheduler. Again, what are the challenges in implementing the loader? It seems you do not perform any sanity check on the code you received. Lines 23-28 show that you exit if the execution fails. Is there a mechanism to inform the host that the upload failed?

How to start a new RiteVM and what is the cost of doing so? What is the key challenge behind the scheduler? From the description in Section 3, the proposed RiteVM scheduler only rotates the list of tasks and is therefore a wrap-around to the existing rotateReadyQueue system call, which does the actual rotation.

The paper lacks an empirical comparison between the overhead caused by using the RiteVM scheduler on mruby on TECS vs mruby on TECS.

The process to load a new byte-code is unclear. Is it required by the developer to wait until the previous program has finished execution?

Bluetooth loader reduces setup time such as pairing is unclear. Once two devices are paired, it is unnecessary to pair them again unless they do not store this information. And for most of the cases, BT modules store this information. (Pages 2 & 4)

The evaluation is not robust enough and lacks detailed explanation and objectives. For instance the data in Table 1 are not clearly explained. It therefore becomes difficult to understand the results and raises many questions. What is the structure of the program? How many lines of code? What is the meaning of each column? The same questions can be asked for the evaluation of the scheduler. How did you decide the size of the loops? Does it make any difference if the loops have equal size? Concerning the overhead of the task switching, how does this affect a large-size queue? "It was confirmed that the time was within (periodic time)×(number of RiteVM tasks - 1)". How did you confirm this? Empirically or analytically?

The paper provides no insight on the data and the implementation itself. Related work section is incomplete, as it does not look at existing Bluetooth loaders and schedulers used.

Minor items

Section 3.1.1 on Page 4: Received byte-code is used to generate mruby "library" byte-codes. Library or application? Is it a typo?

Section 3.1 on Page 4: It's mentioned that RiteVMs and library are compiled and copied to a storage device. But in the footnote, it's mentioned that the complete software should be compiled and linked on the storage device. So what is the correct procedure to start the initial development?

=================================================================

REVIEWER #2

=================================================================

---------------------------------------------------------------------------

Comments

---------------------------------------------------------------------------

This paper aims at the speeding up of the mruby application development process on a TECS (TOPPERS Embedded Component System) platform. In the past, application developers need to compile a whole program, re-flash the SD card on the target device and reboot the system. It conflicts with the concept of mruby which reduces application development time. The authors proposed a Bluetooth loader for mruby applications. In this framework, mruby applications are split into two parts, frequently modified part and infrequently modified part, which correspond to application codes and mruby libraries respectively. Libraries are flashed to the target device while application codes are transferred via Bluetooth. With the help of the proposed mechanism, there’s no need to re-flash and reboot the target device for future updates. The speed of application development is drastically improved.

Providing an online application update mechanism is definitely useful as well as interesting. I think targeted users of this mechanism are not limited to application developers. Vendors and users may benefit from it, too. Ordinary users can get an updated version without a cumbersome flashing process. Vendors can push updates at a rapid pace. Above all, this work eliminates the boundaries of various application development approaches. It would be great if similar ideas can be applied to different programming languages and different operating systems.

Comments:

1. As full source code is available on the Internet, it’s better to decrease code snippets. Explaining source code line by line should also be reduced to improve the overall readability.

2. In Section of RELATED WORK, there should be little description for other application development processes on various embedded systems. Readers need to have a broad view of the current status on this topic, and the necessity of the Bluetooth loader looks more salient.

3. The paper title contains only the Bluetooth loader part in this work. It might be better to include the scheduler part as well.

=================================================================

REVIEWER #3

=================================================================

---------------------------------------------------------------------------

Comments

---------------------------------------------------------------------------

This paper presents an extension of the mruby on TECS platform with support for a Bluetooth loader. The main motivation for this approach is to improve the development cycle, that is, to shorten the time for uploading new applications. The key idea is to separate between application code and library code, and to just upload the application code that is changed more frequently. The paper also introduced a scheduler to enable multitasking on the platform.

The paper presents relevant engineering work, but it is unclear if there are any scientific contributions in the work that makes it relevant to the EMSOFT community. On page 2, four contributions are listed. However, all these contributions seem to be about pure engineering and implementation work and not scientific contributions. What was the scientific interesting problem with implementing the Bluetooth loader?

In the introduction, some claims are questionable and not really motivated. For instance, what do you mean by the statement that “development in C language results in large code size”. Do you mean that the assembly code of generated C code takes more space than Ruby code, and that this is a major problem? Is this really true?

The implementation is described to some detail, but the description does not really relate to any scientific problem. Some figures are clear, for instance Figure 11, but the actual interesting challenge with this solution is unclear. In a nutshell, this figure says that the mruby application is transferred using Bluetooth from the host to the target device, whereas the library code is not transferred.

The experimental evaluation is performing execution time tests, but it seems like there is only one large synthetic benchmark. Also, table 2 states that the number of code lines in CDL is compared with the number of code lines written C. It is quite questionable if this is a good way of evaluating the strength of “component-based development” (as the headline says).

The paper describes related work by showing different embedded scripting language environment, together with a table (Table 3) that shows which features that are supported for the different frameworks.

=================================================================

REVIEWER #4

=================================================================

---------------------------------------------------------------------------

Comments

---------------------------------------------------------------------------

This paper appears to make two contributions: a mechanism for updating a running Ruby application on a remote embedded system through a Bluetooth link (i.e., for rapid software development), and a fair scheduler for running Ruby bytecode on such little systems.

One thing that really isn't clear is the range of applications for which this approach is applicable. The paper repeatedly observes the far lower level of efficiency of Ruby and other scripting languages compared to C, but doesn't really address the circumstances under which such a slow-down is acceptable.

>From the paper, it appears that they only experimented on synthetic benchmarks (i.e., trivial time-wasting loops) to evaluate the performance of their approach, which doesn't answer anything for me.

I'm left wondering in what situations such a framework would be applicable. Providing concurrency in a scripting language setting for embedded applications is interesting, but I'm not sure how difficult it would be to program in practice, and whether its performance would be high enough to be useful.

The other big issue here is whether either of these are substantial enough contributions to merit publication. The Bluetooth loader seems straightforward, depending on the setting n which it is to run. The scheduler seems similarly straightforward. I don't really see what the contribution is.