Towards Dynamically Composed Real-Time Systems

Leandro Batista Ribeiro
lbatistaribeiro@tugraz.at
16 November 2017

Institute of Technical Informatics
Embedded Automotive Systems Group
Graz University of Technology
Agenda

• Introduction

• Software Generation

• Update Mechanisms

• Real-Time Awareness
Agenda

- Introduction
- Software Generation
- Update Mechanisms
- Real-Time Awareness
Real-Time Systems

Real-time systems operate on time constraints. Reaction to events or inputs must occur within a defined time window – not too late, not too soon.

Failure on keeping the time requirements might result in:
- Heavy damages in hard real-time systems
- Undesirable, but tolerable problems in soft real-time systems
Introduction

Dynamically Composed Systems

- Modular systems
- Partial updates
- On-the-fly updates
Introduction

Dynamically Composed Real-Time Systems

- Modular systems
- Partial updates
- On-the-fly updates

- System remains real-time at any point in time: before, during and after any updates.
Dynamically Composed Real-Time Systems

- Modular systems
- Partial updates
- On-the-fly updates

- System remains real-time at any point in time: before, during and after any updates.
Introduction

Motivation

- Internet of Things (IoT)  $\rightarrow$ Billions of devices
- More software customization  $\rightarrow$ Software diversity
- More software providers  $\rightarrow$ Access restrictions
- More classes of systems  $\rightarrow$ Common processor and services
Agenda

• Introduction

• Software Generation

• Update Mechanisms

• Real-Time Awareness
Monolithic

Software Generation

Third party libs, drivers, etc.

Compiler

Relocatable
0011001101 110 111000 1000011 11111110 000111010010010 100100011111110

Relocatable
0011001101 110 111000 1000011 11111110 000111010010010 100100011111110

Executable
0011001101111110 0110111100111100 001101011000011 1111111100000001 0001111010010010 100100011111110 001100110111110 011011100111100 001101010000011 1111111100000001 0001111010010010 100100011111110 001100110111110 0110111100111110 011011001111100 001101011000011 1111111100000001 0001111010010010 100100011111110 001100110111110 0110111100111111 011010001111110 1111111100000001 0001111010010010 100100011111110 001100110111110 0110111100111111 011010001111110 1111111100000001 0001111010010010 100100011111110 001100110111110 0110111100111111 011010001111110 1111111100000001 0001111010010010 100100011111110

Linker
Software Generation

Modular

- **Plain Text**
  ```c
  int i = 0xF0;
  int main(void){
      .
      while (cond){
          .
  }
  ```

- **Compiler**

- **Relocatable**
  ```
  00110011001110
  1100011100
  001101010000111
  11111111
  000111010010010
  10010001111111
  00100000000011
  1001000111110
  ```

- **Linker**

- **Internal Processing**

- **Relocatable**
  ```
  00110011001110
  1100011100
  001101010000111
  11111111
  000111010010010
  10010001111111
  00100000000011
  1001000111110
  ```
Modular Executable

NETFLIX

amazon

AVL

MCsmart

Multi-Core Sustainable modular adept real-time Operating System

System Core
Agenda

• Introduction

• Software Generation

• Update Mechanisms

• Real-Time Awareness
Update Mechanisms

Why update?

- Bugfixes
- Security breaches
- New requirements/legislation
- Enhancements
- Reconfigurable hardware
Update Mechanisms

How to update?

- Physical access

Normal Operation Disrupted?

- Remote updates
  - User awareness
  - Background updates
Our Focus

- Physical access
- Remote updates
  - User awareness
  - Background updates

Normal Operation Disrupted?

No!
Update Mechanisms

General Steps

- Update Unit Choice
  - Monolithic
  - Native Module
  - VM Module

- Size Reduction
  - ELF Optimizations
  - Delta Files
  - Compression

- Data Transmission
  - Server-Client
  - Dissemination Protocols

- Installation
  - Authenticity
  - System Configuration
  - Linking/Relocation
  - Load and Execution
Update Unit – Monolithic

- Example: TinyOS
Update Unit – Native Modules

- **Relocatable Code Only**
  - Example: Contiki
  - 45%-55% metadata overhead [1]
  - ~13% faster than PIC [2]

- **Position Independent Code (PIC)**
  - Example: SOS
  - Less metadata overhead
  - Compiler and architecture dependent
Update Unit – Native Modules

Potential Problems

- Removal of a module needed by other modules
- Dependencies not present in current system
Update Mechanisms

Update Unit – VM Module

VM execution overhead
- processing overhead due to code interpretation at runtime mostly outweighs the costs saved in the transmission [3]
Size Reduction

ELF Optimization
- CELF [3]
  - Fields size reduction 32/64 bits $\rightarrow$ 8/16 bits
  - CELF $\rightarrow$ typically $\sim$ 50% of ELF

- SELF [4]
  - Fields size reduction
  - Tailoring of relocation, string and symbol tables
  - SELF $\rightarrow$ 15%-30% of ELF $|$ 38%-83% of CELF
  - Loading speed 40%-50% of standard mechanism
Size Reduction

Delta Files

Incremental Approach
- Target version built on server
- Delta file generated on server
- Delta file transmitted
- Target version rebuilt on client

Techniques
- Slop regions[5]
- Similarity[6]
Update Mechanisms

Size Reduction

Compression
- Decompression on client → More processing overhead
- Gzip on sensor nodes [7]
Update Mechanisms

Data Transmission

Client - Server
- Point-to-point connection between server and target system

Dissemination Protocols
- Direct connection with some nodes
- Data distributed among remaining nodes

Nodes able to communicate with server
Update Mechanisms

Installation

Authenticity Check
- Make sure updates are legit

System Configuration
- Check/resolve dependencies
- Set up control blocks (tasks, resources, etc)

Linking/Relocation
- Transform a relocatable file in executable

Load/Execution
- Make sure software ready to run
Our Approach

Update Unit Choice
- Monolithic
- Native Module
- VM Module  **No PIC!**

Size Reduction
- ELF Optimizations
- Delta Files
- Compression

Data Transmission
- Server-Client
- Dissemination Protocols

Installation
- Authenticity
- System Configuration
- Linking/Relocation
- Load and Execution
  + Real-Time Awareness
Agenda

• Introduction

• Software Generation

• Update Mechanisms

• Real-Time Awareness
Current Approaches

Simple scenarios
- Rate-monotonic scheduling
- No resource sharing
- No task synchronization

Examples:
- A model for updating real-time applications [8]
  - New WCET \( \leq \) Old WCET
  - New modules stored in the heap
- A method for dynamic software updating in real-time systems [9]
  - Schedulability analysis before accepting update
  - Update finishes within two hyper-periods
Our Goals

Offer partial on-the-fly updates and make sure the system remains real-time at any point in time: before, during and after any modification.

- Unintrusive updates
- Runtime schedulability analysis
- Minimize execution/memory overheads
- Loose coupling
- Portability / Use of standards
- Support wide range of devices
Trade-offs

Efficient analysis and low memory overhead
- Too little metadata \rightarrow More modules, slow or impossible analysis
- Too much metadata \rightarrow Less modules, faster or easier analysis

Low execution overhead and loose coupling
- Few Indirections \rightarrow Low execution overhead, modules strongly coupled
- Many indirections \rightarrow High execution overhead, modules loosely coupled

Generic updates and low client processing/memory overhead
- Server-only processing \rightarrow Tailored updates, client simply loads the update
- Client-only processing \rightarrow Generic updates, client analyzes and tailors the update
Our Approach

Metadata Analysis
- Memory layout and size
- Symbols
- Version information
- Tasks configuration
- Modules dependencies
- Synchronization points
- Worst case execution time
- Worst case response time
- Interference time
- Priority inversions

Plain Text
```
int i = 0xF0;
int main(void){
    while(cond){
    }

```

Relocatable
```
00110011001
110 111000
1000011
11111110
000111010010010
100100011111110
```

Exec. Model
Real-Time Awareness

Our Approach

Execution Model
- Describe software execution
  - Individual modules
    - Tasks WCET
    - Synchronization pairs
  - Whole System
    - Tasks WCRT
    - Potential deadlocks or starvations

<table>
<thead>
<tr>
<th>Plain Text</th>
<th>Relocatable</th>
</tr>
</thead>
<tbody>
<tr>
<td>int i = 0xF0; int main(void){ ... while(cond){ ... }</td>
<td>00110011001 110 111000 1000011 11111110 000111010010010 100100011111110</td>
</tr>
</tbody>
</table>

Before update
After update
Our Approach

Updates will only be accepted if they are compatible with the system.

Compatibility
- Pluggability: Dependencies
- Interoperability: Execution
Our Approach

Update Protocols
- Metadata exchange
- Find good trade-offs
  - Generic updates x Client processing

Metadata
- List of installed modules and respective versions
- Global symbol table
- Memory layout
- Execution models
Our Approach

Metadata Location

- Server

- Client
  - List of installed modules and respective versions
  - Global symbol table
  - Memory layout
  - Execution models

39
Our Approach

Metadata Location

- Server

- Client
  - List of installed modules and respective versions
  - Global symbol table
  - Memory layout
  - Execution models

40

Real-Time Awareness

Leandro Batista Ribeiro
Towards Dynamically Composed Real-Time Systems
Echtzeit 2017
Our Approach

Metadata Location

- Server

- Client
  - List of installed modules and respective versions
  - Global symbol table
  - Memory layout
  - Execution models

Real-Time Awareness
Our Approach

Metadata Location

- Server
- Client
  - List of installed modules and respective versions
  - Global symbol table
  - Memory layout
  - Execution models
Our Approach

Metadata Location

- Server
- Client
  - List of installed modules and respective versions
  - Global symbol table
  - Memory layout
  - Execution models
Our Approach

Metadata Location

- Server
  - List of installed modules and respective versions
  - Global symbol table
  - Memory layout
  - Execution models

- Client
Our Approach

- Investigate overhead with diverse update protocols.
- Define what performance classes of devices will support given protocols.
Thank you!

Leandro Batista Ribeiro
lbatistaribeiro@tugraz.at

16 November 2017

Institute of Technical Informatics
Embedded Automotive Systems Group
Graz University of Technology
References


References


