Emergent Software Systems: Theory and Practice

Roberto Rodrigues Filho - INF/UFG
Barry Porter - Lancaster University, UK
Fábio M. Costa - INF/UFG
Iwens Sene Júnior - INF/UFG

SBRC'21 - Short course

robertovito@ufg.br
Course Overview

➔ Theory

➔ Motivation: Contemporary systems complexity;
➔ Background: Autonomic Computing and Self-adaptive Systems;
➔ Emergent Software Systems
  ◆ Introducing the concept
  ◆ Implementation (PAL framework)
Course Overview

➔ **Practice**

➔ **Case studies (practical application of the concept):**
  ◆ Emergent Web Server
  ◆ Emergent Microservice

➔ **Tool:** Dana language basic syntax and examples;

➔ **Demo:**
  ◆ Interactive Emergent System tool;
  ◆ Emergent Web Server;

➔ **Practical:** Hands-on experience with the Emergent Web Server;

robertovito@ufg.br
Theory
Motivation

➔ Contemporary Systems Complexity
Motivation

➔ Contemporary Systems Complexity

➔ Volatility: Constant changes in the operating environment;

➔ Heterogeneity: Plethora of different devices, OSes, protocols, programming models, and so on;

Motivation

➔ Contemporary Systems Complexity

➔ Volatility: Constant changes in the operating environment;

➔ Heterogeneity: Plethora of different devices, OSes, protocols, programming models, and so on;

Motivation

→ Contemporary Systems Complexity

➔ Volatility: Constant changes in the operating environment;

➔ Heterogeneity: Plethora of different devices, OSes, protocols, programming models, and so on;


Spoiler Alert: Emergent Software Systems is an attempt at a fresh perspective.
Contemporary Systems Complexity

Push more responsibility for managing the system to the software itself.
Background

➔ **Autonomic and Self-adaptive Systems**

➔ **Autonomic Computing (IBM vision):** Self-managing systems that take the necessary actions to achieve high-level systems goals;

Background

➔ Autonomic and Self-adaptive Systems

➔ Self-* properties:
  ◆ Self-optimising;
  ◆ Self-healing;
  ◆ Self-protecting;
  ◆ Self-configuring;
Background

→ Autonomic and Self-adaptive Systems

→ Self-adaptive Systems: Systems that are able to change their structure or behaviour as a response to changes or to maintain systems goals.

Autonomic and Self-adaptive Systems: MAPE-K
Background

Autonomic and Self-adaptive Systems

Developing Autonomic and Self-adaptive Systems is HARD:

- Develop the system;
- Identify potential autonomic elements;
- Create adaptation points (e.g., feature-models);
- Create adaptation mechanism;
- Create adaptation logic;
Background

➔ Autonomic and Self-adaptive Systems

➔ Developing Autonomic and Self-adaptive Systems is HARD:
  ◆ Create adaptation logic:
    ● Rule-based approach;
    ● Logical expressions;
    ● Training machine learning and other mathematical models;
    ● Reinforcement learning applied to SAS;
Emergent Software Systems

➔ Introducing the concept

➔ Emergent software system as an approach to facilitate SAS development.

Software systems able to build their own understanding of their structure and operating environment and build their adaptation logic accordingly.

robertovito@ufg.br
Emergent Software Systems

➔ Introducing the concept

➔ Emergent software system as an approach to facilitate SAS development.

Component-based models + Reinforcement Learning
Emergent Software Systems

➔ Introducing the concept

➔ Component-based model:

Component model separates **program logic** from **program structure** (i.e., how logic is wired);
Emergent Software Systems

➔ Introducing the concept

➔ Component-based model:

◆ Comparison with an object-oriented programming

```java
import java.util.ArrayList;

public class Example {

    public static void main(String[] args) {
        String str = "First item on the list";
        ArrayList<String> myList = new ArrayList<String>();
        myList.add(str);
    }
}
```
Introducing the concept

Component-based model:

Comparison with an object-oriented programming

```java
import java.util.ArrayList;

public class Example {
    public static void main(String[] args) {
        String str = "First item on the list";
        ArrayList<String> myList = new ArrayList<String>();
        myList.add(str);
    }
}
```
Emergent Software Systems

➔ Introducing the concept

➔ Component-based model:

◆ Comparison with an object-oriented programming

```java
import java.util.ArrayList;

public class Example {
    public static void main(String[] args) {
        String str = "First item on the list";
        ArrayList<String> myList = new ArrayList<String>();
        myList.add(str);
    }
}
```

➔ Classes are statically bound;
➔ Logic is entangled with structure;
➔ Generates a single binary (monolith);
Emergent Software Systems

➔ Introducing the concept

➔ Component-based model:

- Provided interface
- Required interfaces
Introducing the concept

Component-based model:

- **Interface** is a collection of function prototypes;
- **Component** is strongly encapsulated;
- **Composition** happens through a composer separate from the logic of the system;
Emergent Software Systems

➔ Introducing the concept

➔ Component-based model:
Emergent Software Systems

➔ Introducing the concept

➔ Component-based model:

Runtime software adaptation / evolution.
Emergent Software Systems

➔ Introducing the concept

➔ Reinforcement learning:

![Diagram showing an agent interacting with an environment through actions and receiving rewards and states.](image)
Emergent Software Systems

➔ Introducing the concept

➔ Reinforcement learning:

![Diagram showing reinforcement learning concept](image)
Emergent Software Systems

➔ Introducing the concept

➔ Reinforcement learning:

![Diagram](image)
Emergent Software Systems

➔ Introducing the concept

➔ Reinforcement learning:

![Diagram showing the environment process: action, reward, state]
Emergent Software Systems

➔ Introducing the concept

➔ Emergent Software Systems: Reinforcement Learning

Environment

action

reward, state

Component-based Models

Emergent Software Systems concept

robertovito@ufg.br
Emergent Software Systems

Introducing the concept

Emergent Software Systems requires:

- A set of component variants (multiple components implementing the same interface);
- A set of metrics are extracted from the live system to monitor systems health and classify its operating environment;
- Zero-training reinforcement learning algorithm to find the most suitable architectural composition for the identified operating environment;
Emergent Software Systems

➔ Introducing the concept

➔ Emergent Software Systems:

![Diagram of Emergent Software Systems concept]

- Reinforcement Learning
- Component-based Models

Emergent Software Systems concept
Emergent Software Systems

➔ Introducing the concept

➔ Emergent Software Systems:

Reinforcement Learning

Component-based Models

Action

Reward
Emergent Software Systems

➔ Introducing the concept

➔ Emergent Software Systems:

Reinforcement Learning

Component-based Models

Runtime adaptation
Emergent Software Systems

→ Introducing the concept

Emergent Software Systems:

→ Reinforcement Learning

→ Component-based Models
Emergent Software Systems

➔ Introducing the concept Reinforcement Learning

➔ Emergent Software Systems:
Emergent Software Systems

Implementation (PAL framework)

Overview:

- PAL stands for Perception, Assembly, Learning;
- Dana programming language (component-based model);
- Learning algorithm: Baseline and Multi-armed Bandits;
Emergent Software Systems

→ Implementation (PAL framework)
Emergent Software Systems

➔ Implementation (PAL framework)
Emergent Software Systems

Implementation (PAL framework)
Emergent Software Systems

→ Implementation (PAL framework)

Dana runtime

Seamless runtime adaptation

robertovito@ufg.br
Emergent Software Systems

Implementation (PAL framework)

Learning

Perception

Assembly

Dana runtime

Seamless runtime adaptation

robertovito@ufg.br
Emergent Software Systems

Implementation (PAL framework)

Learning

Perception

Assembly

Dana runtime

Seamless runtime adaptation

robertovito@ufg.br
Emergent Software Systems

Implementation (PAL framework)


Dana runtime

Seamless runtime adaptation

robertovito@ufg.br
Emergent Software Systems

➔ Implementation (PAL framework)

Reinforcement Learning

Dana runtime

robertovito@ufg.br
Emergent Software Systems

Implementation (PAL framework)

- Learning
- Perception
- Assembly

Dana runtime

Reinforcement Learning
- Baseline
- UCB1
- Thompson Sampling

robertovito@ufg.br
Emergent Software Systems

Implementation (PAL framework)

- Learning
- Perception
- Assembly

Dana runtime

Reinforcement Learning

Baseline

UCB1

Thompson Sampling

robertovito@ufg.br
Emergent Software Systems

Implementation (PAL framework)

- Learning
  - REST
- Perception
  - LOCAL FUNCTION CALLS
- Assembly

Reinforcement Learning
- Baseline
- Thompson Sampling
- UCB1

Porter, B., Rodrigues Filho, R. 2016 "Losing control: The case for emergent software systems using autonomous assembly, perception, and learning." In: 2016 IEEE SASO


Porter, B., Rodrigues Filho, R. "Distributed emergent software: Assembling, perceiving and learning systems at scale." In: 2019 IEEE SASO
Emergent Software Systems

Implementation (PAL framework)
Emergent Software Systems

Implementation (PAL framework)

Assembly

Assembly API
- void setMain(char mainComp[])
- bool setConfig(char config[])
- char[] getConfig()
Emergent Software Systems

Implementation (PAL framework)

Assembly API

- void setMain(char mainComp[])
- bool setConfig(char config[])
- char[] getConfig()

Data structure
Emergent Software Systems

Implementation (PAL framework)

Assembly

Assembly API
- void setMain(char mainComp[])
- bool setConfig(char config[])
- char[] getConfig()

App

Main

Data structure
Emergent Software Systems

→ Implementation (PAL framework)

Assembly

Assembly API

- `void setMain(char mainComp[])`
- `bool setConfig(char config[])`
- `char[] getConfig()`

Data structure
Emergent Software Systems

Implementation (PAL framework)

Assembly

Assembly API
- void setMain(char mainComp[])
- bool setConfig(char config[])
- char[] getConfig()

Main

Component A

Data structure
Emergent Software Systems

Implementation (PAL framework)

Assembly API

- `void setMain(char mainComp[])`
- `bool setConfig(char config[])`
- `char[] getConfig()`
Emergent Software Systems

Implementation (PAL framework)

Assembly

Assembly API

- void setMain(char mainComp[])
- bool setConfig(char config[])
- char[] getConfig()

Data structure

Component A

Component B

Main

App

robertovito@ufg.br
Emergent Software Systems

→ Implementation (PAL framework)

Assembly API
- void setMain(char mainComp[])
- bool setConfig(char config[])
- char[] getConfig()
Emergent Software Systems

Implementation (PAL framework)

Assembly API

- void setMain(char mainComp[])
- bool setConfig(char config[])
- char[]getConfig()
Emergent Software Systems

Implementation (PAL framework)

- void setMain(char mainComp[])
- bool setConfig(char config[])
- char[]getConfig()

Assembly API

Assembly

Executing system

ComponentA

ComponentB

ComponentC

Main

App

robertovito@ufg.br
Emergent Software Systems

→ Implementation (PAL framework)

Assembly API

- void setMain(char mainComp[])
- bool setConfig(char config[])
- char[] getConfig()

Assembly

Main

ComponentA

ComponentB

ComponentC

Executing system

robertovito@ufg.br
Emergent Software Systems

→ Implementation (PAL framework)

Assembly API

- void setMain(char mainComp[])
- bool setConfig(char config[])
- char[] getConfig()
Emergent Software Systems

→ Implementation (PAL framework)

Assembly

Assembly API

- void setMain(char mainComp[])
- bool setConfig(char config[])
- char[] getConfig()

Executing system

Main

ComponentA

ComponentB

ComponentC

App

Load

robertovito@ufg.br
Emergent Software Systems

Implementation (PAL framework)

Assembly API

- void setMain(char mainComp[])
- bool setConfig(char config[])
- char[] getConfig()
Emergent Software Systems

Implementation (PAL framework)

Assembly API

- void setMain(char mainComp[])
- bool setConfig(char config[])
- char[] getConfig()

Main

App

Rewire

Executing system
Emergent Software Systems

Implementation (PAL framework)

Assembly API

- `void setMain(char mainComp[])`
- `bool setConfig(char config[])`
- `char[] getConfig()`
Emergent Software Systems

Implementation (PAL framework)

Assembly API

- void setMain(char mainComp[])
- bool setConfig(char config[])
- char[] getConfig()
Emergent Software Systems

Implementation (PAL framework)

Assembly API

- void setMain(char mainComp[])
- bool setConfig(char config[])
- char[] getConfig()

Architectural Description

|Main, ComponentA, ComponentB|0:Int1:1,0:Int2:2|
Emergent Software Systems

Implementation (PAL framework)
Emergent Software Systems

Implementation (PAL framework)

Perception

Perception API
- void addProxy(char exp[])
- char[] getPerceptionData()
Emergent Software Systems

→ Implementation (PAL framework)

**Perception API**

- `void addProxy(char exp[])`
- `char[] getPerceptionData()`
Emergent Software Systems

Implementation (PAL framework)

Perception API
- void addProxy(char exp[])
- char[] getPerceptionData()
Emergent Software Systems

Implementation (PAL framework)

Perception API

- void addProxy(char exp[])
- char[] getPerceptionData()
Emergent Software Systems

→ Implementation (PAL framework)

Perception API

- void addProxy(char exp[])
- char[] getPerceptionData()
Emergent Software Systems

Implementation (PAL framework)

Perception API

- void addProxy(char exp[])
- char[] getPerceptionData()

{"metric":
  { "name": "response_time", "value": 300, "count": 10 }
}

{"event":
  { "name": "mime_type", "value": "image", "count": 10 }
}
Emergent Software Systems

➔ Implementation (PAL framework)
Emergent Software Systems

Implementation (PAL framework)

Baseline algo:

- Setup
- Exploration
- Exploitation
Emergent Software Systems

→ Implementation (PAL framework)

Baseline algo:
- Setup
- Exploration
- Exploitation

Consists of executing:
- `setMain();`
- `addProxy("exp");`
- Set up the learning algorithm;

robertovito@ufg.br
Emergent Software Systems

→ Implementation (PAL framework)

Baseline algo:
- Setup
- Exploration
- Exploitation

Consists of executing:
- setMain();
- addProxy("exp");
- Set up the learning algorithm;

Learning

Main

ComponentA

ComponentB

App

Int2

Executing system
Emergent Software Systems

➔ Implementation (PAL framework)

Baseline algo:

- Setup
- Exploration
- Exploitation

Consists of executing:

- setMain();
- addProxy("exp");
- Set up the learning algorithm;

robertovito@ufg.br
Emergent Software Systems

Implementation (PAL framework)

Baseline algo:
- Setup
- Exploration
- Exploitation

Consists of executing:
- setMain();
- addProxy("exp");
- Set up the learning algorithm;
  - configs[] = getAllConfigs();

robertovito@ufg.br
Emergent Software Systems

Implementation (PAL framework)

Baseline algo:
- Setup
- Exploration
- Exploitation

Consists of:
- For all compositions
  - Observe;
  - Collect perception data;
- Compare all metrics to find the best;
Emergent Software Systems

General software system

Status: Observing

Baseline algo: Setup, Exploration, Exploitation

Data:

App

Main

Proxy

Component A

Component B

Executing system

robertovito@ufg.br
Emergent Software Systems

→ Implementation (PAL framework)

Baseline algo:
- Setup
- Exploration
- Exploitation

Status: Get perception data  Data: 83 / 125

robertovito@ufg.br
Emergent Software Systems

→ Implementation (PAL framework)

Baseline algo:
- Setup
- Exploration
- Exploitation

Status: Get perception data  Data: (arch1: resp_time 20ms, text)
Emergent Software Systems

→ Implementation (PAL framework)

Baseline algo:
- Setup
- Exploration
- Exploitation

Status: New action  Data: (arch1: resp_time 20ms, text)
Emergent Software Systems

Implementation (PAL framework)

Baseline algo:
- Setup
- Exploration
- Exploitation

Status: New action  Data: (arch1: resp_time 20ms, text)

robertovito@ufg.br
Emergent Software Systems

Implementation (PAL framework)

Baseline algo:
- Setup
- Exploration
- Exploitation

Status: Observing
Data: (arch1: resp_time 20ms, text)
Emergent Software Systems

Implementation (PAL framework)

Baseline algo:
- Setup
- Exploration
- Exploitation

Status: Get perception data
Data: (arch1: resp_time 20ms, text)
Emergent Software Systems

→ Implementation (PAL framework)

Baseline algo:

- Setup
- Exploration
- Exploitation

Status: Get perception data

Data: (arch1: resp_time 20ms, text)
      (arch2: resp_time 30ms, text)
Emergent Software Systems

→ Implementation (PAL framework)

Baseline algo:
- Setup
- Exploration
- Exploitation

Status: Select best Data: (arch1: resp_time 20ms, text) (arch2: resp_time 30ms, text)

robertovito@ufg.br
Emergent Software Systems

→ Implementation (PAL framework)

Baseline algo:

- Setup
- Exploration
- Exploitation

Status: Select best  
Data: (arch1: resp_time 20ms, text)  
(arch2: resp_time 30ms, text)

robertovito@ufg.br
Emergent Software Systems

→ Implementation (PAL framework)

Baseline algo:
- Setup
- Exploration
- Exploitation

Consists of:
- For the best found composition:
  - Observe;
  - Collect data perception;
- If something changes: trigger exploration, otherwise continue;
Emergent Software Systems

→ Implementation (PAL framework)

Baseline algo:

- Setup
- Exploration
- Exploitation

Status: Observing  Data: (arch1: resp_time 20ms, text)  (arch2: resp_time 30ms, text)
Emergent Software Systems

→ Implementation (PAL framework)

Baseline algo:
- Setup
- Exploration
- Exploitation

Status: Get perception data
Data: (arch1: resp_time 20ms, text) (arch2: resp_time 30ms, text)
Emergent Software Systems

→ Implementation (PAL framework)

Baseline algo:
- Setup
- Exploration
- Exploitation

Status: Get perception data
Data: (arch1: resp_time 20ms, text)
(arch2: resp_time 30ms, text)
(arch1: resp_time 25ms, text)

robertovito@ufg.br
Practice
Case studies

➔ Application of the Emergent Software System Concept

➔ Emergent Web Server

◆ Application of Emergent Software Systems concept to build a web server.

➔ Emergent Microservice

◆ Application of Emergent Software Systems concept to build microservices.
Case studies

➔ Emergent Web Server

➔ Motivation:

◆ Web-based systems make up a large sum of systems we daily interact with;
◆ Manually optimising web-based systems is challenging;

➔ Goal:

◆ To find the best web server composition, at runtime, with no predefined domain-specific knowledge;
Case studies

➔ Emergent Web Server
Case studies

➔ Emergent Web Server
Case studies

Emergent Web Server

Handling TCP connections:
- Creating a thread per client;
- Assigning a client to a thread from a pool;
Case studies

Emergent Web Server

Handling HTTP request:
- Default;
- Cache;
- Compression;
- Cache-compression;
Case studies

➔ Emergent Web Server

Cache replacement algorithm
Case studies

➔ Emergent Web Server

Compression algorithms
Case studies

Emergent Web Server

- Ground truth: Optimality divergence
Case studies

➔ Emergent Web Server

◆ Online Learning

![Graph showing response time over time for Workload A and Workload B]
Case studies

➔ **Emergent Microservice**

Volume

<table>
<thead>
<tr>
<th>Time</th>
<th>Text</th>
</tr>
</thead>
</table>

Changes in Workload Volume

Microservices

Relies on (at runtime)

Volume

<table>
<thead>
<tr>
<th>Time</th>
<th>Text</th>
<th>Image</th>
</tr>
</thead>
</table>

Changes in Workload Pattern

Microservices

Relies on (at runtime)

kubernetes

Docker

CLOUD

NOTHING

robertovito@ufg.br
Case studies

➔ Emergent Microservice

Monolith ➔ Microservices

Emergent Microservice

RL Algorithm
Case studies

Emergent Microservice

Reinforcement Learning

Component-based Models

Emergent Software Systems concept

action

reward, state

Microservice
Case studies

 ➔ Emergent Microservice
Case studies

➔ Emergent Microservice

robertovito@ufg.br
Case studies

➔ Emergent Microservice
Case studies

Emergent Microservice

- WS.core
- Dispatcher
- DataCollector
- MySQLDriver
Case studies

Emergent Microservice

Default

- WS.core
- Dispatcher
- DataCollector
- MySQLDriver

Components that generate composition variants for DC

- Cache
- Compression
- Cache + Compression
Case studies

➔ Emergent Microservice

- WS.core
- Dispatcher
- DataCollector
- MySQLDriver
- Cache + Compression
- Compression
- Cache
Case studies

➔ Emergent Microservice

1. WS.core
2. Dispatcher
3. DataCollector
4. MySQLDriver

Cache + Compression
Cache
Compression
Case studies

⇒ Emergent Microservice

- WS.core
- Dispatcher
- DataCollector
- MySQLDriver
- Compression
- Cache
- Cache + Compression
Case studies

![Diagram showing execution time and workload phases](image-url)

- **WORKLOAD A**
- **WORKLOAD B**
- **WORKLOAD A**

Execution Time (ms):
- 0.00
- 25.00
- 50.00
- 75.00
- 100.00

Time (s):

- Learning / exploration
- Convergence / exploitation

Robertovito@ufg.br
Case studies

➔ Emergent Microservice

- Horizontal Pod Autoscaler (elasticity):
  - Set a metric (e.g., CPU utilisation);
  - Set a metric threshold;
  - Monitor the system;
  - Replicate POD in case the metric surpasses the threshold;
Case studies

- Triggers replication
- EM learning

Elasticity threshold

Execution Time (ms)

- Elastic
- Cache-Compression
- Elastic Learner

# of instances

1 instance each

Observation cycles (every 30s)

3 instances for default, 1 instance for both EM and Cache-compression

robertovito@ufg.br
Tool

Dana programming language

Introducing the language
Showing its syntax, compiling and running a program

Please visit: https://github.com/robertovrf/sbrc21_minicurso
Demo

Interactive Emergent System

Running Interactive Emergent System tool and showing how to interact with PAL through a command prompt

Please visit: https://github.com/robertovrf/sbrc21_minicurso
Emergent Web Server

Executing the Emergent Web Server

Please visit: https://github.com/robertovrf/sbrc21_minicurso
Practical

➔ Hands-on experience with the Emergent Web Server

Access our Github repository and follow the ShortCourse Practical

Please visit: https://github.com/robertovrf/sbrc21_minicurso
robertovito@ufg.br