How to Build Emergent Software Systems (Tutorial)

Roberto Rodrigues Filho and Barry Porter
School of Computing and Communications
Lancaster University
Lancaster, UK
Email: {r.rodriguesfilho, b.f.porter}@lancaster.ac.uk

I. Introduction

Emergent software systems take a reward signal, an environment signal, and a collection of possible behavioural compositions implementing the system logic in a variety of ways, to learn in real-time how to best assemble a system to maximise reward. This reduces the burden of complexity in systems building by making human programmers responsible only for developing potential building blocks while the system determines how best to use them in its deployment conditions – with no architectural models or training regimes. Instead of adaptation being a special capability, emergent systems treat adaptation as continuous self-assembly, where a system is constantly reviewing its own behavioural composition to find alternative building blocks which better suit the currently perceived environment.

II. Scope

This is a hands-on tutorial in which participants will be encouraged to build their own emergent system, with autonomously learned behaviour that is a consequence of the current deployment environment. The tutorial is structured in two major parts: (i) the Dana programming language, and (ii) our emergent systems framework.

A. Dana Programming

In the first part we will introduce our hyper-adaptive *Dana* programming language [1]. Dana makes runtime adaptation safe, lightning fast, and pervasive to every element of a software system from TCP socket implementations up to graphical user widgets. Runtime adaptation is sufficiently cheap in Dana (taking a few microseconds) that a system can be in almost constant flux without impacting the user experience, allowing high-speed real-time learning to freely experiment with a wide range of different behaviours in a production environment.

The fundamental goal of runtime adaptation which is to move from one system version A to another system version B; and to do this with (i) no perceivable downtime, and (ii) in such a way that the running system in version B is indistinguishable from one which had always been in version B from the start of system's the execution.

While aspect (ii) of this is a very difficult to uphold for all elements of a system's global state machine [2], Dana uses a special theory of structural mechanics to uphold this condition for the structure of a running system. This allows the programmer to create a diverse range of design patterns, with complex and dynamic object reference graphs, while supporting seamless runtime adaptation for any component.

This tutorial will begin by outlining the theory of runtime adaptation, and exploring Dana's model of structural mechanics which implements this theory for system structure. The core of the Dana language is designed around this theory, offering generalised adaptation support, with its syntax and operators layered on top and designed to work within the constraints of its structural model.

Following the theoretical elements we will then run through a range of increasingly complex example systems which can be built with Dana and examine how those systems are pervasively adaptable at runtime.

The first half of the tutorial will conclude with hands-on experience of building an adaptive system in Dana, using its APIs to automatically search for components to integrate into a system, and gaining familiarity with the set of tools available to create simple abstractions for entire system composition and runtime recomposition.

B. Emergent Systems

In the second part of the tutorial we will introduce our emergent systems framework, itself built using Dana, which uses the language's capability to abstract entire compositions of behaviour into simple action/reward tables. We will cover the three major parts of our framework in detail, including assembly, perception, and learning.

Our assembly module discovers compositions of behaviour which implement a desired system, and presents those compositions as actions. When a new composition is selected to use, the assembly module computes a difference between its current composition and the new one, and triggers a series of runtime adaptations throughout the system to move between the two compositions. This provides the abstraction of a set of simple actions (compositions) which can be selected by a learning agent, but which each entail potentially complex behavioural changes to the running system. Some of the most recent research on emergent systems is using this idea to present both local composition options, and distributed compositions where selected components are relocated to remote hosts, all as a simple set of actions.

Our perception module injects probes into the running emergent system to transparently monitor metrics such as response time or energy usage. This builds on the general adaptation capability of Dana, in which we can swap one

implementation of an interface for a different one. In the case of injecting perception probes, we adapt a selected interface N to a proxy implementation P of that interface with associated metric capture logic. P itself has a required interface of type N, and we connect this required interface to the component originally implementing N before the probe was injected. Using this concept, we can selectively inject and remove probes into different parts of a system at runtime depending on the intensity of monitoring needed.

Finally, our learning module takes a list of actions from the assembly module, and the available perception data from the perception module, to learn at runtime how different environment conditions impact the observed behaviour, performance or health characteristics of the running system – and how to select the most ideal behaviour for each encountered operating environment range.

This section of the tutorial will give participants the opportunity to use our emergent systems framework and in particular to explore how the machine learning element works – gaining insights into how complex behavioural decision-making can be simplified to a set of real-time learning actions. Following on from this we will also cover some of the major theoretical and practical challenges of building an effective machine learning solution for this context, which generalises across any emergent system in any application domain. As part of our discussion of the learning element we will also introduce the concept of multi-armed bandits as one of the most prominent reinforcement learning techniques, examining the theory of UCB1 (*Upper Confidence Bound*) as a simple implementation of this approach.

III. Outcomes

By the end of this tutorial, participants will have had the opportunity to experiment with adaptation, compositional search, and real-time machine learning to drive system assembly and re-assembly as a product of a perceived environment. We hope that this will provoke new ways of thinking about autonomous and adaptive systems, where complex behavioural decisions can be driven by apparently simple actions and rewards in a reinforcement learning algorithm. You can find out more about ongoing work in emergent systems in recent publications [3], [4], [5].

Acknowledgements

This work was partly supported by the Leverhulme Trust Research Grant *The Emergent Data Centre*, RPG-2017-166.

References

[1] B. Porter, "Runtime modularity in complex structures: A component model for fine grained runtime adaptation," in *Component-Based Software Engineering*. ACM, June 2014, pp. 26–32.

[2] A. R. Gregersen and B. N. Jørgensen, "Dynamic update of java applications—balancing change flexibility vs programming transparency," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 21, no. 2, pp. 81–112, 2009.

[3] B. Porter, M. Grieves, R. Rodrigues Filho, and D. Leslie, "RE^{X}: A development platform and online learning approach for runtime emergent software systems," in *Symposium on Operating Systems Design and Implementation*. USENIX, November 2016, pp. 333–348.

[4] R. Rodrigues Filho and B. Porter, "Defining emergent software using continuous self-assembly, perception, and learning," *Transactions on Autonomous and Adaptive Systems*, vol. 12, no. 3, pp. 1–25, September 2017.

[5] R. Rodrigues Filho, B. Porter, F. Costa, and M. Pereira de Sa, "Towards emergent microservices for client-tailored design," in *International Workshop on Adaptive and Reflective Middleware*. ACM, December 2018, pp. 1–6.