Component and System-Wide Self-* Properties in Decentralised Distributed Systems

Jim Dowling, Raymond Cunningham, Eoin Curran and Vinny Cahill
Distributed Systems Group, Trinity College Dublin
{jpdowlin,rcnnnghm,vjcahill@cs.tcd.ie, currane@maths.tcd.ie}

Abstract

This paper introduces a two-layered approach to building autonomic distributed systems, with self- properties supported at both the component and system-wide levels. K-Components is presented as a self-adaptive component model that supports the separate specification of a component's self-* behaviour using declarative contracts. Distributed Reinforcement Learning (DRL) is introduced as a system-wide optimisation technique that supports the establishment and maintenance of system-wide properties in distributed systems where there is no support for global state.*

1 Introduction

In the future, many distributed systems will consist of interacting, autonomous components that organise, regulate and optimise themselves without human intervention. However, with increasing system size and complexity our ability to build self-managed distributed systems using existing programming languages, top-down design techniques and management infrastructures is reaching its limits [1], as solutions they produce require too much global knowledge.

Massive autonomic distributed computer systems, on a scale comparable with biological autonomic systems, require a decentralised, bottom-up approach to their construction. Such systems can be modelled as a collection of components or agents where decisions are totally or partially taken by these components, interaction between components is based only on local information and system-wide self-* properties emerge from the interactions between components [2]. The benefits of such an approach include improved robustness and scalability, the possibility of self-regulation, self-configuration and self-organisation, the lack of centralised points of failure or attack, as well as possible evolution of the system by evolving the local interaction rules of the components.

The construction of decentralised distributed systems with system-wide properties presents a number of challenges. These include designing a component model that can support self-* behaviours, designing decentralised models of interaction between components that establish and maintain emergent self-* properties over collections of components and determining the functionality required at the component-level to produce emergent properties at the system-level.

2 Self-* Behaviour in K-Components

Many self-* behaviours, such as self-healing and self-optimisation, can be engineered using self-adaptive components. Self-adaptive components have the ability to change their behaviour or structure at run-time in order to accomplish specified goals [3], such as adapt to discovered faults or sub-optimal performance. Self-* behaviour for components requires the active monitoring of component states and external dependencies for events that cause adaptation actions, such as reconfiguring dependencies on faulty components.

K-Components is a component framework for building self-adaptive components [3]. A K-Component contains modularised self-adaptive behaviour. Programmers specify a component's self-adaptive behaviour using a declarative programming language called the Contract Description Language (CDL). The CDL allows programmers to declaratively associate feedback events (regarding component or connector states) with adaptation actions, e.g. self-healing or self-optimising actions, using the event-condition-action (ECA) paradigm.

Self-adaptive behaviour specified in the CDL is encapsulated at run-time in a set of reflective agent programs that can monitor feedback events and execute adaptation actions, see Figure 1. The agent programs

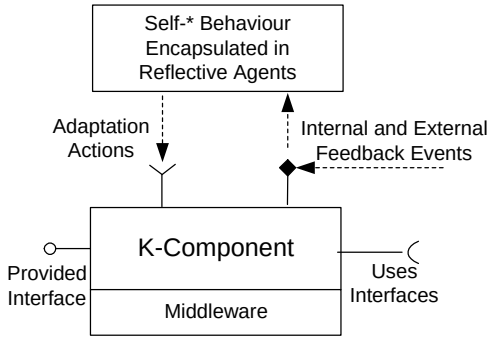


Figure 1. Self-* Behaviour in K-Components

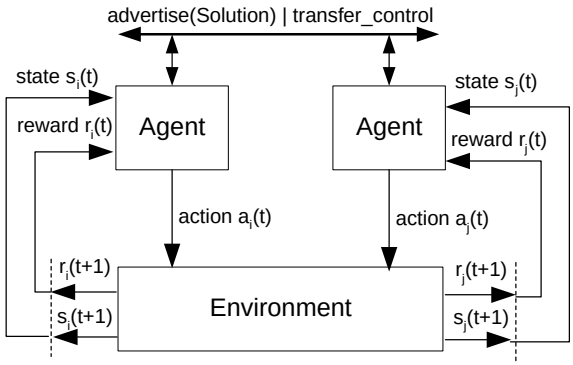


Figure 2. DRL Agent Model

are threads that are interleaved with component execution and operate on both feedback events and an architecture meta-model of the component.

One problem with the ECA approach to specifying self-adaptive behaviour is that it becomes infeasible as the space of possible feedback events and adaptation actions increases. For complex, instrumented distributed systems, programmers cannot be expected to know about and handle all possible internal component and external environmental states or be able to accurately predict the outcome of executing some adaptation action in a dynamic environment.

For this reason, in K-Components self-adaptive behaviour can also be learnt by components using an unsupervised technique called distributed reinforcement learning (DRL). DRL also enables the decentralised coordination of groups of connected components for the purpose of establishing system-wide properties. The next section introduces system-wide properties for distributed systems and shows how system-wide self-* properties of a distributed system can emerge from the interaction of its components (or agents as they are called in the next section).

3 Self-* Distributed Systems using Distributed Reinforcement Learning

Self-* distributed systems establish and maintain system-wide properties, e.g. properties such as being deadlock-free, fault tolerant or load-balanced. Existing techniques that can engineer system-wide properties into distributed systems, such as dynamic software architectures, communicating sequential processes and group communication protocols, do so in a top-down manner, decomposing system behaviour and making it amenable to formal analysis [1]. These approaches are not suitable for dynamic environments or environments that have no support for global state, such as wireless ad-hoc networks or peer-to-peer systems.

DRL is a decentralised approach to establishing and maintaining system-wide properties in distributed systems. DRL is an extension to Reinforcement Learning [4] (RL) for decentralised multi-agent systems. DRL does not make use of system-wide knowledge and individual agents (or components) only know about and interact with their neighbours.

In RL, an agent associates actions with system states in a trial-and-error manner and the outcome of an action is observed as a reinforcement that, in turn, causes an update to the agent's *action-value policy* using a reinforcement learning strategy [4]. The goal of reinforcement learning is to maximise the total *reward* (reinforcements) an agent receives over a time horizon by selecting optimal *actions*. Agents may take actions that give a poor payoff in the short-term in the anticipation of higher payoff in the longer term. In general, actions may be any decisions that an agent wants to learn how to make, while states can be anything that may be useful in making those decisions. As action selection is probabilistic, there is some trial-and-error in the selection of actions and RL is not a suitable technique for learning self-* behaviour for the classes of distributed system that are intolerant to suboptimal action selection, such as real-time systems.

DRL is based on a variant of the coordination model found technique in swarm intelligence algorithms where agents learn from the successes of their neighbours. DRL solves system-wide optimisation problems by specifying how individual agents solve discrete optimisation problems (DOP) using RL, share their results with neighbours and transfer control to neighbours by initiating the start of a new DOP on a neighbouring agent, see Figure 2. In terms of RL, DRL is a model-based learning strategy for problems that can be distributed across a partially connected set of hosts and

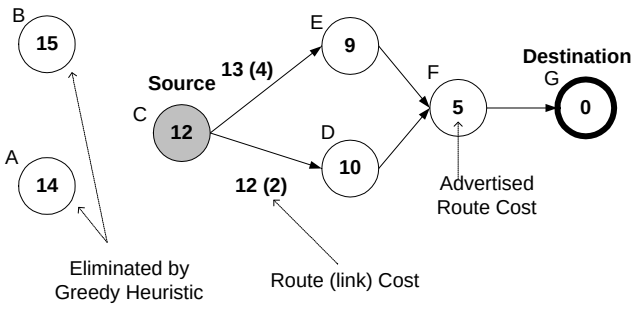


Figure 3. Routing Decision in SAMPLE

each DOP can be modelled as an absorbing Markov Decision Process. In DRL, system-wide self-* properties emerge from the interaction of the solution of discrete optimisation problems.

3.1 Specifying Optimisation Problems

In DRL, system-wide optimisation problems are decomposed into a set of DOPs, the solution of which should be performed by the set of agents that result in a near-optimal system-wide use of resources. System-wide optimisation problems are initiated at some agent that can either solve the DOP itself or delegate its solution to a neighbouring agent that in turn may handle the DOP or delegate it further. As a result the solution to the problem is parallelised amongst the agents in the system. It is our belief that many self-* properties in decentralised distributed systems can be represented as optimisation problems and can be solved using DRL.

3.2 SAMPLE: Ad-hoc Routing using DRL

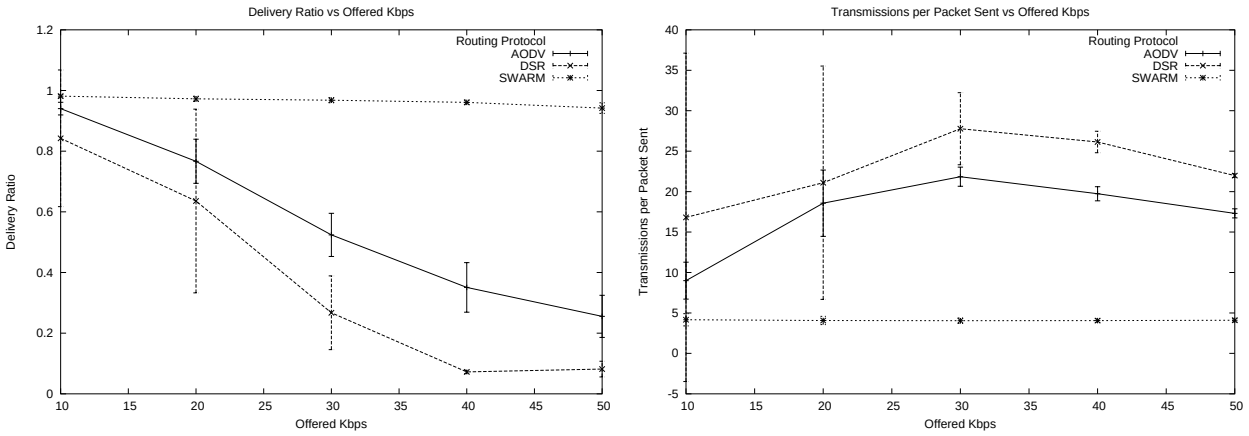
SAMPLE is a probabilistic on-demand ad hoc routing protocol based on DRL [5] that contains system-wide self-optimising properties, such as the adaptation of network traffic patterns around areas of congestion and wireless interference and the exploitaton of stable routes. Standard ad hoc routing protocols such as (AODV) and Dynamic Source Routing (DSR) use discrete models of links in the network. In SAMPLE, we use a statistical model of links based on RL and routing agents share their link information with neighbours using DRL. Hence, routing decisions are based both on locally acquired experience and information acquired from neighbours using DRL, see Figure 3.

We have implemented the SAMPLE routing protocol in the NS-2 network simulator and performance results show better performance in the face of adverse network conditions than AODV and DSR [5]. In our experimental setup and simulations, there are 33 fixed nodes, 50 mobile nodes and 3 server nodes are the fixed nodes at the centre of the simulation arena. The fixed nodes in the simulation provide stable links in the network that the routing protocols could exploit. Figure 4 shows the variation in performance of SAMPLE, AODV and DSR as the number of clients in the network is increased. For these figures the packet size sent by clients was kept fixed at 64 bytes, sent 3 times a second. Figure 5 shows the same experiment, this time with 512 byte packets. Offered Throughput is used in both figures to enable comparison of the results. As the number of clients in the network is increased, the offered throughput to the routing protocols is increased. This in turn increases the level of network congestion and the amount of contention that the MAC protocol must deal with. This increased congestion increases the number of failed MAC unicasts in the network.

SAMPLE performs better than existing ad-hoc protocols in the presence of failed unicasts due to the ability of routing agents to learn that a link failed to some transient factor (and hence retrying the link may succeed) or some more serious factor such as link failure, and retrying the link is unlikely to succeed. Existing protocols assume the link has failed and perform poorly when packet error rates increase. As routing agents share their experience about links it collectively improves their rate of learning and convergence on more optimal system-wide routing behaviour. SAMPLE displays the system-wide property of routing traffic around areas of congestion or wireless interference. This property optimises throughput available in the network and emerges from the solution to and interaction of the individual routing DOPs at nodes. An important lesson from SAMPLE is the need for experimentation, as the emergence of more optimal routing properties is sensitive to tuneable parameters in DRL and the RL system model in SAMPLE.

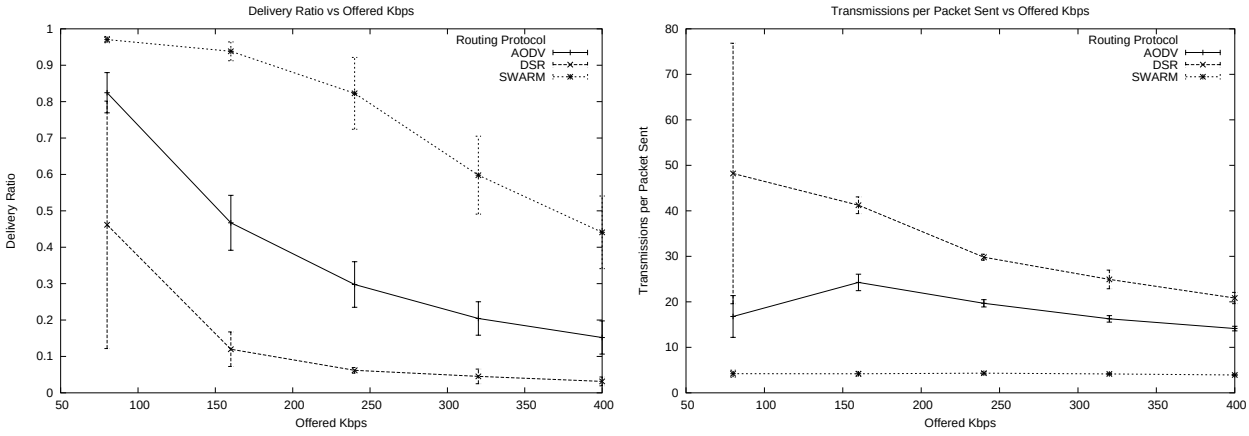
4 Conclusions and Future Work

In this paper, we describe self-* properties of distributed systems at the component level with K-Components and at the system-wide level using DRL. We believe that many self-* properties can be engineered at the component level using a self-adaptive component model and that self-* properties at the system level can be established and maintained by solving system-wide optimisation problems using interacting agents that solve discrete optimisation problems, such as in DRL. Future work will investigate the construction of self-* systems using K-Components that co-ordinate their self-adaptive behaviour using DRL.



(a) Delivery Ratio (b) Throughput

Figure 4. Performance with Varying Load. 64 byte packets



(a) Delivery Ratio (b) Throughput

Figure 5. Performance with Varying Load. 512 byte packets

References

[1] A. Montresor, H. Meling, and O. Babaoglu, "Towards self-organizing, self-repairing and resilient distributed systems," *Future Directions in Distributed Computing, LNCS 2584*, 2003.

[2] J. Dowling, "Coordinating self-adaptive components for emergent distributed system behaviour and structure," *8th Cabernet Radicals Workshop*, 2003.

[3] J. Dowling and V. Cahill, "The k-component architecture meta-model for self-adaptive software," *3rd Conference on Reflection, LNCS 2192*, 2001.

[4] R. Sutton and A. Barto, *Reinforcement Learning.* MIT Press, 1998.

[5] E. Curran and J. Dowling, "Sample: An on-demand probabilistic routing protocol for ad-hoc networks," *Technical Report: Department of Computer Science, Trinity College Dublin*, 2004.