Dynamic Reconfiguration in the RUNES Middleware

Geoff Coulson[#], Richard Gold[†], Manish Lad[†], Cecilia Mascolo[†], Luca Mottola[*], Gian Pietro Picco[*], and Stefanos Zachariadis[†]
[#]Lancaster University [†]University College London [*]Politecnico di Milano
geoff@comp.lancs.ac.uk c.mascolo@cs.ucl.ac.uk picco@elet.polimi.it

***Abstract*— Next generation embedded systems will be composed of large numbers of heterogeneous devices. These will typically be resource-constrained (such as sensor motes), will use different operating systems, and will be connected through different types of network interfaces. Additionally, they may be mobile and/or form ad-hoc networks with their peers, and will need to be adaptive to changing conditions based on context-awareness.**

As an example of these system we consider disaster recovery scenarios where large numbers of different devices need to interconnect in an ad-hoc manner. In this respect, our goal is the provisioning of a middleware framework for such system environments. Our approach is based on a small and efficient *middleware kernel* supporting highly modularised and customisable component-based middleware services. These services can be tailored for specific embedded environments, and are runtime-reconfigurable to support adaptivity.

This paper describes a demonstration that highlights some of the features available in our middleware. In particular, we focus on *heterogeneity handling* by showing our middleware running on resource-rich as well as resource-constrained devices, and on *adaptivity features* by demonstrating runtime reprogramming and on-the-fly component deployment.

I. Introduction

Miniature computing devices are being embedded in an increasing range of objects around us including home appliances, cars, transport infrastructures, buildings, and people. Furthermore, the *networking* of such embedded environments is enabling advanced scenarios in which devices leverage off each other and exhibit autonomous and coordinated behaviour to solve complex tasks.. Recent developments in wireless networking are pushing these trends even further by enabling new application scenarios, as witnessed by the recent surge of interest in wireless sensor networks.

However, research into such networked embedded environments has so far focused much on the development of miniaturised devices with increasingly powerful and general capabilities. As a result, the *software fabric* that ultimately makes innovative applications possible has tended to be overlooked. Instead, software is typically developed in an ad-hoc fashion, with little or no provision for reusable services and abstractions. Furthermore, even where attempts have been made to provide such features, the range of devices involved in networked embedded environments inevitably leads to significant complexity in appropriately configuring, deploying, and dynamically reconfiguring the software. There is therefore a need for dedicated *middleware platforms* for networked embedded systems, providing abstractions that can span the full range of heterogeneous systems, and offering consistent mechanisms with which to configure, deploy, and reconfigure dynamically both system and application level software.

The work discussed in this paper is addressing the need for such middleware platforms. The work is being carried out in the context of the EU-funded RUNES project (Reconfigurable, Ubiquitous, Networked, Embedded Systems) [1], which has the goal of developing an architecture for networked embedded systems that encompasses dedicated radio layers, networks, middleware, and specialised simulation and verification tools.

Our middleware platform, which is at the heart of the RUNES architecture, is component-based and encapsulates the functionality provided by its various components behind well-defined interfaces [2]. This decoupling not only enables one to deploy different variants of the same component (e.g., tailored to a specific device type), but also enables dynamic reconfiguration of component instances and their interconnections. This provides support for dynamic adaptation to changing conditions — a fundamental requirement in the context-aware scenarios typical of networked embedded systems.

Our work in the scope of the project is motivated by a realistic application, for which our middleware was developed: a tunnel fire scenario, where the tunnel is instrumented with sensor nodes that report to a central controller when possible, but that can dynamically adapt their behaviour to report to firefighters entering the

tunnel in groups to rescue the situation. The middleware needs to allow communication among different devices, and must allow adaptation of behaviour in a context-aware manner.

This paper describes a demonstration of a subset of the aforementioned features. In particular, we use commercially available "mote" nodes as the sensors in the tunnel, and laptops to represent the firefighters. A minimal software configuration on the sensor nodes allows to perform data reporting under normal operating conditions. Nevertheless, as soon as the firefighters enters the tunnel, they are able to dynamically reprogram the motes to change their behavior according to the needs arising in the emergency situation. In addition, we also show how the firefighters are able to share the data sensed by nearby sensor nodes to coordinate their actions. This is achieved thanks to a reconfigurable communication infrastructure, that is able to provide communication among the rescue team members despite their physical mobility.

In Section II we first present the tunnel fire scenario in detail. Then, in Section III we describe our solutions to the challenges inherent with the scenario, and how we actually demonstrate them. Finally, Section IV describes the details regarding the components we implemented for this demonstration.

II. Reference Scenario

The general scenarios we envisage are heterogeneous in terms of resources, networking interfaces and operating systems. Our design and development work is grounded in a number of networked embedded systems scenarios that we employ throughout the RUNES project. The most prominent of these is a *road transport infrastructure* based scenario in which a road tunnel is instrumented with sensors and actuators to detect and guard against potential disasters arising from events such as fire or chemical spillage. More specifically, the road tunnel is instrumented with a number of sensors (e.g., temperature sensors and sensors to detect toxic fumes) that feed data back to a tunnel control centre. In addition, there are various actuators present such as fire sprinklers and road traffic management signs. To maximise resilience in disaster situations, these sensors and actuators are interconnected using redundant network technologies — both wired and wireless.

In the event of a disaster occurring, the sensor and actuator networks may become partitioned and may thus need to reconfigure themselves to maintain their operational status. Furthermore, sensor and actuator devices may need to be brought under the direct control of emergency personnel such as firefighters, and this may require further ad-hoc networks to be established and may additionally require that sensors be dynamically reprogrammed — for example, firefighters may need to poll sensor devices rather than wait for periodic push-based reports, and this may require new software to be loaded onto the sensors.

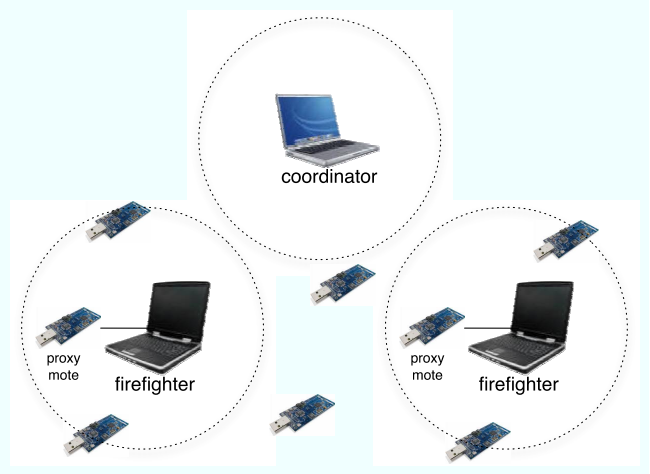

Fig. 1. Demo setup.

Scenarios such as this are clearly highly *heterogeneous*. They involve a range of sensor and actuator devices which may run different operating systems and programming languages. They also clearly involve heterogeneous networks: wired, infrastructure-based wireless, and ad-hoc wireless. Furthermore, such scenarios are highly *dynamic*—especially during emergency situations—and thus require to be highly *adaptive* and *reconfigurable*. Networks must be repaired, reconfigured and instantiated, new devices must be accommodated (e.g., devices on vehicles or attached to firefighters), and new software must be loaded onto devices. These are precisely the types of characteristics that our middleware platform is addressing.

III. Demo Overview

To highlight the aforementioned challenges in a real demonstration, we devised a setup in which the firefighters are represented using 2 laptop PCs, and the sensing devices in the tunnel are represented by a number of TMote Sky nodes, as illustrated in Figure 1. Link layer connectivity between the firefighters' laptops and the tunnel motes is achieved by relying on a "proxy mote" attached to the laptops via a USB cable. This mote does not perform any specific action, it simply acts as an antenna to allow communication with the motes in the tunnel. The remaining laptop represents the coordinator of the emergency team. As such, it does not have direct

access to the motes, but instead receives notifications from the other laptops.

Some of the most difficult technical issues in the aforementioned scenario lie in letting different devices co-operate seamlessly, in reconfiguring sensor behavior under emergency conditions, and in supporting information sharing among firefighters. In our demonstration, these challenges are faced by exploiting the RUNES middleware as follows:

Heterogeneity. Device heterogeneity is handled by exploiting different implementations of the RUNES middleware. In particular, the firefighter laptops run the Java version of the middleware, whereas the sensing devices run the Contiki version. Even if these implementations clearly differ in the underlying technology used, they nevertheless provide the same, simple, component-based programming environment to the application developers. This is centered around the notion of *Component* as an encapsulated unit of functionality and deployment. The functionality a component provides is made available to other components through a set of *Interfaces*. Dually, components can exploit functionality provided by other components through the use of *Receptacles*, i.e., required interfaces. When a component relies on the functionality provided by another, the two must be explicitly associated through a *Bind* operation the RUNES middleware provides[1].

Our component model is simple enough to be implemented on a wide range of devices, from powerful laptops up to resource constrained devices, while also being expressive enough to allow for a wide range of mechanisms to be implemented in terms of components. Thank to this, the programmers can face the heterogeneity of multiple operating systems and programming languages by relying on the same set of programming abstractions.

Software Reconfiguration. While a number of components can be defined to overcome the aforementioned challenges, it is not possible for all of them to be available simultaneously on the relatively resource-constrained sensor devices. However, the RUNES component model fosters the development of loosely coupled components, that are able to function independently.

In our demonstration, sensor nodes operating under normal conditions run with a basic set of components that include a component to take periodic measurements, and a component implementing a form of unicast communication to report data to the tunnel control centre. Under emergency situations, we assume each firefighter is interested in readings sensed by nearby motes. To this end, as soon as a firefighter's laptop enters the communication range of a mote, it dynamically deploys a component implementing a publish-subscribe [4] paradigm. This component either complements or replaces the existing reporting mechanism, and allows sensor data to be locally broadcast and thus made available to the firefighters.

The runtime supporting the RUNES middleware allows pairs of components to bind and re-bind dynamically to one another. So, initially, the measurement component is bound to the unicast notification component. However, when the publish-subscribe component is uploaded dynamically on a sensor node, the measurement component re-binds dynamically to the interface provided by the publish-subscribe component. This allows a transparent change to occur in the method of measurement data dissemination.

Topological Reconfiguration. The firefighters laptops form a mobile ad-hoc network. Information dissemination within this network is achieved across multiple hops through a content-based publish-subscribe infrastructure, implemented as a set of RUNES components. This allows for a different form of reconfiguration, which now does not affect the software running on a single node, but instead how different nodes communicate with each other in a mobile environment.

Once the publish-subscribe component is deployed on the motes, these start to broadcast locally their readings. When a firefighter's laptop receives these data from a mote, it disseminates the information by publishing messages. The publish-subscribe infrastructure then delivers the messages to only the interested firefighters, determined according to the subscriptions they expressed up to that moment. This is achieved using a form of subscription forwarding on a tree-shaped overlay.

In the demonstration, we show how this communication infrastructure can cope with the mobility of the firefighters. To this end, laptops are physically moved or turned off to show reconfiguration of the event dissemination routes. This is achieved using a self-reconfiguring tree overlay, whose mechanisms are explicitly devised for content-based routing in mobile environments [5], joint with an efficient scheme to reconfigure subscription routes established on top of the overlay [6].

IV. Components and Interactions

The RUNES component model allows us to overcome the aforementioned challenges in a light-weight, flexible and incremental manner. Here, we detail on a number of components used in our demonstration, and illustrate how they interact dynamically with one another.

[1]Further details on the RUNES component-model and programming abstractions can be found in [2].

A. Sensor Node Components

Measurement Data Collection. The measurement component encapsulates the functionality to obtain readings from the sensor hardware and to process them for dissemination. It is implemented in C/Contiki [3] as a RUNES component, and defines a process thread that is executed automatically on instantiation. This essentially runs an infinite loop that clears existing buffers, takes measurement readings, evaluates the current operational status based on these readings, formats readings for dissemination, and initiates dissemination through a generic `notify()` interface call.

Under normal operating conditions, measurement readings are taken and reported at a regular pre-defined interval. However, under emergency conditions, it is essential to gather as much accurate and up-to-date information as possible. So, the abnormal sensor readings result in a change in the evaluated operational status, that triggers a reduction in the interval that measurements are taken, allowing more frequent measurement readings to be disseminated. The granularity of operational status is controlled within the component code itself, thus several different operational levels could be defined, adjusted and used.

Measurement Data Dissemination. Measurement dissemination is performed differently depending on the type of dissemination required. Under normal operating conditions, a unicast notification component is present for the transmission of measurement readings to the tunnel control centre. It provides the generic interface `notify()` that takes in parameters indicating the type of notification and the actual data. After constructing a measurement data packet ready for transmission, a packet brokering component undertakes a unicast transmission of the measurement data packet.

Conversely, the publish-subscribe component the firefighters deploy onto sensor nodes enables the additional, local broadcast of measurement readings. This can be received by the firefighters whose proxy mote is in range of the sensor mote, so that the measurement readings can be disseminated also among firefighters according to their interests.

B. Laptop Components

Publish-Subscribe Infrastructure To implement subscription forwarding in mobile environments across multiple hops, we use a self-reconfiguring tree overlay and a dedicated scheme to reconfigure subscription routes. These are implemented in different components, hence enabling the dynamic reconfiguration of the two mechanisms separately.

Overlay maintenance is implemented in a `TreeOverlayManager` component, that takes care of reconfiguring the tree overlay in case a link fails because of mobility or node disappearing. Conversely, a `PSTransport` component embodies a strategy to reconfigure the routes laid on the overlay, and followed by events. The two components are loosely coupled, as the interface between the two is basically composed of a single `openLink` operation, used by the overlay manager to inform the `PSTransport` component that a new link on the overlay has been activated. Moreover, a further service component is implemented to take care of UDP transmissions. The whole solution poses a very little requirement on the underlying network layers, as only the availability of 1-hop, broadcast UDP transmission is required.

V. Conclusions

In this paper we have presented the core characteristics of the RUNES middleware and we have described some basic components developed in the scope of our target application (tunnel fire scenario). We have presented the key themes of our demo which are *heterogeneity*, *software reconfiguration* and *topology reconfiguration*. The paper has also presented the steps of the RUNES middleware demonstration, in which we will demonstrate practically how these components allow us to overcome each of the challenges faced within the tunnel fire scenario.

Acknowledgments: We would like to acknowledge the support of the European Union through project IST EU RUNES.

References

[1] IST EU RUNES. www.ist-runes.org.

[2] P. Costa, G. Coulson, C. Mascolo, G. P. Picco, and S. Zachariadis. The RUNES Middleware: A Reconfigurable Component-based Approach to Network Embedded Systems. In *Proc. of* 16^{th} *International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC05)*. IEEE Press, Sept. 1995.

[3] A. Dunkels, B. Groenvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors*, Tampa, Florida, USA, Nov. 2004.

[4] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 2(35):114–131, June 2003.

[5] L. Mottola, G.Cugola, and G. Picco. A Self-Repairing Tree Overlay Enabling Content-Based Routing in MANETs. Submitted for publication. Available at `www.elet.polimi.it/upload/mottola`, 2006.

[6] G. P. Picco, G. Cugola, and A. Murphy. Efficient content-based event dispatching in the presence of topological reconfigurations. In *Proc. of the* 23^{rd} *Int. Conf. on Distributed Computing Systems (ICDCS03)*, pages 234–243, 2003.