Demo Abstract: The Lorien dynamic component based OS

Barry Porter   Utz Roedig   François Taïani   Geoff Coulson
barry.porter@comp.lancs.ac.uk
School of Computing and Communications
Lancaster University, UK

1 Lorien

In this demo we show how the Lorien operating system [5] supports lightweight, efficient and safe online changes to any aspect of the software running on sensor nodes – and how this promotes reuse of deployed sensor networks through run-time software evolution.

Lorien is based on three principles:

i. *pure dynamic component design* enabling **lightweight** software evolution to all parts of node software

ii. *abstract architecture description* promoting **independence** and **persistence** of software evolutions

iii. *system integrity rules* promoting **safety** of online changes to any part of node software

We believe that Lorien is unique in its support for safe, incremental online software evolution, and in the scope of this support which ranges from the lowest level drivers through protocols and application components. This goes significantly beyond contemporary WSN operating systems such as TinyOS [4], Contiki [2] and SOS [3] which either offer only offline image-based software updates or else provide only limited-scope online software evolution that is restricted to application-level code and lacks strong integrity support.

1.1 Pure Dynamic Component Design

Lorien's uniformly applied component-based programming model is designed to support online changes to any aspect of node software – the ability to add/remove applications is therefore simply one particular use of this model; adding new protocols, drivers, filesystems, schedulers and so on works in exactly the same way.

This capability is based around a dynamic component model written in plain C and supported by a component runtime through which components can be individually loaded, instantiated, destroyed and unloaded at will. Even Lorien's dynamic loader/linker and component runtime are components that can be replaced or architecturally reconfigured at runtime. Components use formal *required* and *provided* interfaces to interact (where an interface is a typed collection of functions); required interfaces are connected at runtime to compatible provided interfaces to satisfy dependencies.

Copyright is held by the author/owner(s).
SenSys'10, November 3–5, 2010, Zurich, Switzerland.
ACM 978-1-4503-0344-6

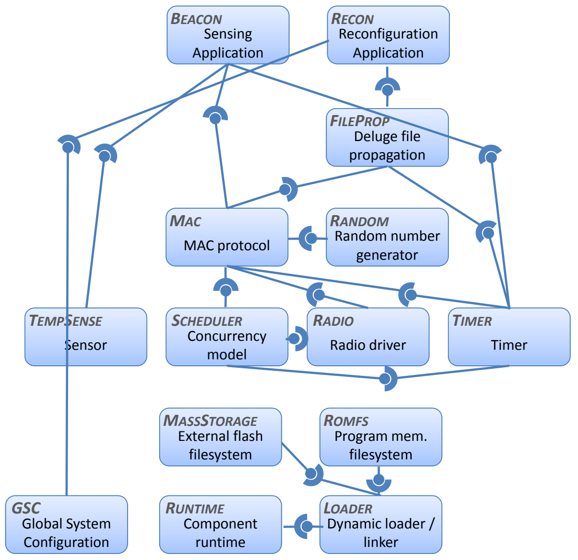


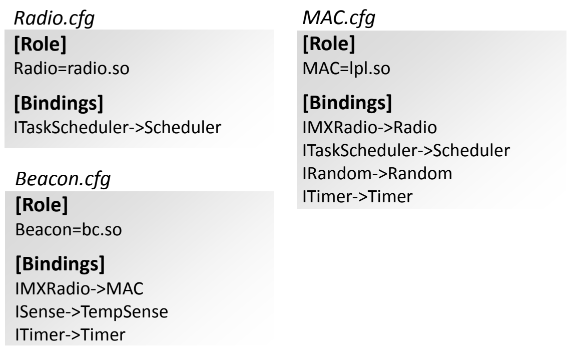
Figure 1. Architecture of a typical pure dynamic Lorien system. Each shaded box is an instance of an independently loadable/unloadable & instantiable component.

1.2 Abstract Architecture

Rather than hard-coding a system's configuration, Lorien uses an abstract architecture description that facilitates the *independence* and *persistence* of software evolution steps. Each element in a Lorien system is described by a 'configuration fragment', consisting of an abstract *role name*, the specific component currently filling that role, and a list of that component's immediate dependencies; the component filling a role can of course change using a different fragment.

A list of configuration fragments describing the entire system is then maintained in a 'manifest' stored in persistent memory. As components come and go Lorien opportunistically satisfies dependencies whenever possible. Components can therefore be added to or removed from the system in any order, each configuration fragment being an independent piece of sub-architecture with respect to the rest of the system, able to be composed into a range of system architectures without needing an understanding of the big picture.

A complete example Lorien system in shown in Figure 1, with abstract role names in capitals. Corresponding example configuration fragments are given in Figure 2.



```
Radio.cfg
[Role]
Radio=radio.so

[Bindings]
ITaskScheduler->Scheduler
```

```
MAC.cfg
[Role]
MAC=lpl.so

[Bindings]
IMXRadio->Radio
ITaskScheduler->Scheduler
IRandom->Random
ITimer->Timer
```

```
Beacon.cfg
[Role]
Beacon=bc.so

[Bindings]
IMXRadio->MAC
ISense->TempSense
ITimer->Timer
```

Figure 2. Sample configuration fragments for the Radio, MAC and Beacon roles of Figure 1.

1.3 Maintenance of System Integrity

Allowing a software image to be modified in any way while online promotes very lightweight – and therefore energy-efficient – system evolution. Without constraints however it can present significant dangers to the integrity of a node's software and can also make development very difficult. Lorien's solution is a set of simple *integrity rules* which make online software evolution safe and dramatically simplify development.

In essence these integrity rules guarantee that no component X will ever have a required interface connected to a provided interface of a component Y such that Y does not have all of its own dependencies satisfied.

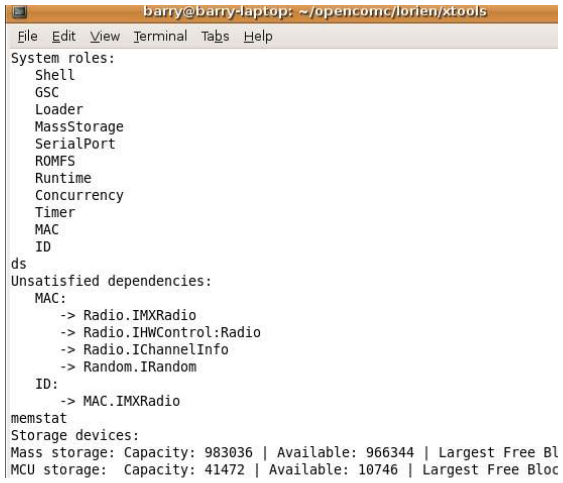
This is useful both to X, since it knows an inter-component call will never fail due to a 'configuration error', and to Y, since it knows that when a function is called on it there is no need to check whether it is itself sufficiently connected to its dependencies to service that call.

2 Demonstration

The steps of the demo will be as follows:

1. We use a simple interactive shell application (see Figure 3) running on a TelosB node to show how Lorien works in terms of its abstract architectural roles, dependency management and configuration persistency. We show incrementally adding low level driver components, protocols and applications, all without system restarts, and demonstrate how dependencies between components are tracked and safely and opportunistically satisfied. Powering off the node and restarting it demonstrates Lorien's configuration persistency, where a node always boots back into the configuration it was last in, including starting all applications that were running.

2. Having demonstrated the principles of lightweight online software evolution we now show over-the-air changes to a network of battery-powered TelosB nodes, with one node connected to a host PC as a base station. Each node runs a system like that in Figure 1 and can receive, install or replace components as they arrive. We deploy a simple temperature data gathering application by sending only an SHT11 driver and application components to the network. The drivers and application are persistently installed on nodes as they arrive and start up their functionality as soon as all supporting components are present on the system manifest, sending data back to the base station when this happens.


barry@barry-laptop: ~/opencomc/lorien/xtools
File Edit View Terminal Tabs Help

```
System roles:
   Shell
   GSC
   Loader
   MassStorage
   SerialPort
   ROMFS
   Runtime
   Concurrency
   Timer
   MAC
   ID
ds
Unsatisfied dependencies:
   MAC:
      -> Radio.IMXRadio
      -> Radio.IHWControl:Radio
      -> Radio.IChannelInfo
      -> Random.IRandom
   ID:
      -> MAC.IMXRadio
memstat
Storage devices:
Mass storage: Capacity: 983036 | Available: 966344 | Largest Free Bl
MCU storage:  Capacity: 41472 | Available: 10746 | Largest Free Bloc
```

Figure 3. Sample output.

3. Finally we deploy alongside the temperature application two further applications to run concurrently on our network: (i) an application that uses LEDs to indicate in which direction the highest amount of radio traffic is emanating; and (ii) a traffic aggregation component that runs on selected nodes to reduce overall network traffic (and therefore energy drain). This further demonstrates the safe incremental online software evolution that our platform promotes.

Besides the guided demo attendees are encouraged to explore just how much our system can evolve by suggesting actions to take via the interactive shell. Lorien can be downloaded, including all parts of this demo, from [1].

3 Acknowledgments

This work was partially supported by the ICT Programme of the European Union under contract number ICT-2008-224460 (WISEBED).

4 References

[1] Download: http://opencomc.sourceforge.net/lorien/.

[2] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *LCN '04: Proc. of the 29th Annual IEEE International Conference on Local Computer Networks*, pages 455–462. IEEE Computer Society, 2004.

[3] C.-C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava. A dynamic operating system for sensor nodes. In *MobiSys '05: Proc. of the 3rd international conference on Mobile systems, applications, and services*, pages 163–176, Seattle, Washington, USA, June 2005.

[4] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. *SIGOPS Operating Systems Review*, 34(5):93–104, 2000.

[5] B. Porter and G. Coulson. Lorien: A pure dynamic component-based operating system for wireless sensor networks. In *MidSens '09: Proc. of the 4th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks*, pages 7–12, New York, NY, USA, 2009. ACM.