Software Engineering Challenges for Adaptive Robotic Ecologies

ICRA 2013
Eighth full-day Workshop on
Software Development and Integration in Robotics (SDIR VIII)
May 6, 2013

Mauro Dragone, David Swords, G.M.P. O'Hare
RUBICON
Robotic UBIquitous COgnitive Network

2.5M, 3 years STREP
Funded by FP7 (#269914)
Robotic Ecologies

Many specialized, pervasive, simple “robotic” devices
Robots, sensors, appliances, ...=> Complex services achieved through cooperation
Intrinsically modular and expandable (just add new devices...)
Extend the type of application that can be realistically envisaged
GOAL: Self-Sustaining Learning for Robotic Ecologies

Reduce need for
- Reprogramming
- Configuration
- Maintenance
- Supervision

Increase
- Adaptability
- Flexibility
- Robustness
- Open new application areas

Over time, the ecology adapts to changes to its environments and to user's needs and improves the way it carries out its services.
Self-Sustaining Learning

- Each node of the ecology contributes to a cooperative self-learning dynamic.

- RUBICON uses:
  - Statistical and computational (neuroscience) learning methods
  - Recurrent Neural Networks (RNN)
  - Reservoir Computing (RC) models
  - Novelty Detection as a Driver for Self-Organisation and Continuous Learning
Control & Software Requirements
(the focus of this presentation)

- Coordination: find, execute and monitor cooperative strategies
- Provide a base-line “structure” to the learning system
- Enable the use of existing functional components
- Support highly heterogeneous and distributed systems
- Deal with both action and configuration
- Deal with resource and temporal and spatial constraints and ...
- Work in computational constrained environments
- Work in open & dynamic environments
RUBICON Architecture

- Cognitive Layer
- Learning Layer
- Configuration Planner
- Peis Middleware
- Proxy 1
- Proxy n
- Agent 1
- Agent m

Goals → feedback → Coordination Level → Proxy/Agent Service Level

Events

IE-802-15.4 KNX
PEIS Middleware
Physically Embedded Intelligent Systems

- Dynamic addition and removal of PEIS
- Smooth integration of robots, sensors, tagged objects, human interfaces...
- Configuration, dynamic connection and communication between PEISs using a tuplespace abstraction

http://aass.oru.se/~peis/
Proxies

Proxy Peis components “represents” and provides access to less computational capable components, e.g. ZigBee Networks, RFID tags, KNX devices => E.g. used to handle multiple networks
WSANs Proxy

New sensors can join (and be represented in the tuplespace) or leave the RUBICON at running time.
Re-configuration of a Peis-component system

A Peis-component has meta-tuples that can be used (set by an external configurator) to define (at run-time) its collaborations.

Configuration Planner
Constraint-Based Representation

Sensors, actuators and human modeled as state variables
Configuration Planner

Architecture of the Configuration Solver: multiple reasoners act on the same temporal network. Each modification is checked through a temporal consistency check.

Integration with Functional Robotic Components (ROS)

If everything was a Peis component it would be easy!

Re-use existing, off-the-shelf functional components is essential to make our approach feasible.

Functional components come in different languages and with their own software frameworks (even not component-based, as ROS)
Integration with Functional Robotic Components (ROS): Options

Options enabled by our tools:

1) Create Peis wrappers to ROS components, possibly using generic Peis/ROS bridge
   → ROS component becomes Peis-enabled

2) Apply Proxy-design pattern with Self-OSGi agent system to control ROS sub-systems
   Wrapper/proxies to (existing) functional components generated at run-time
   (alternative to approaches such as Genome-3, BRIDE...)
Integration with Functional Robotic Components (ROS): 1- Wrappers & B

Hybrid PEIS/ROS Components
- Good for configuration
- but bad for closed-loop control
- difficult to apply if nodes are tied to ROS
- adds significant overheads to prepare wrappers

Configurator

ROS links (e.g. topics, services ...)
Peis (for configuration & to bridge across different nodes of the peer-to-peer network)
Integration with Functional Robotic Components (ROS): 2 - Self-OSGi

**Approach:** use an existing component framework to project a common component model across heterogeneous components. Reduce the gap between mainstream software frameworks used in our domain (e.g. universAAL)
Self-OSGi Proxy (GoalManager)

- Decouples the component plan requesting a service goal from the component plan providing it
- Monitors performances of components (e.g. to trigger component replacement and to give feedback to learning systems)
- Functional components can be external to OSGi (e.g. ROS, Peis,...)
Self-OSGi XML Declarative Model

- Dependencies of functional components (their required and provided interfaces) are described declaratively (in XML)
- Self-OSGi checks these dependencies without the need for code generation (monitors are instantiated at run-time, by reading the XML, and by re-using the links of the underlying framework, e.g. ROS topics)
Self-OSGi Monitor Example

- Self-OSGi takes care of monitoring the dependencies of the functional component it proxies, and offers an homogeneous API to the system configurator, which can more easily control component wiring, component parameters, and component life-cycle.
RUBICON Architecture

Cognitive Layer

Learning Layer

goals

feedback

events

Configuration Planner

Peis Middleware

Proxy 1

Proxy n

Agent 1

Agent m

Coordination Level

Proxy/Agent Service Level

IE-802-15.4

KNX

IE-802-15.4
Conclusions and Future Work

Our software integration approach addresses fast re-use of off-the-shelf heterogeneous components and the application of a consistent component model geared toward system adaptation.

To date, we had only evaluated / benchmarked our single layers

Future work will try to evaluate the fully integrated architecture (including the learning methods not discussed in this presentation)


