Åmbiance Project
Autonomous Systems Group
University of Luxembourg

Reza Razavi (PI)
University of Luxembourg
reza.razavi@uni.lu

Laboratoire d'Informatique de Paris 6 – CNRS
Université Pierre et Marie Curie

Open Systems Lab
University of Illinois at
Urbana-Champaign

Kirill Mechitov, Sameer Sundresh and Gul Agha
University of Illinois at Urbana-Champaign
{mechitov, sundresh, agha}@cs.uiuc.edu

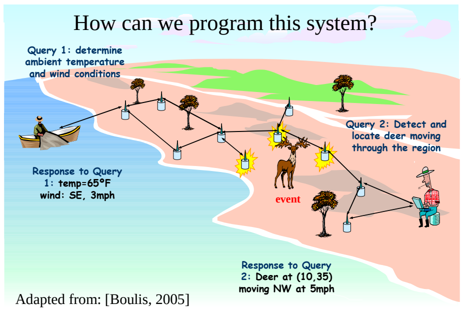
Jean-Francois Perrot
Université Pierre et Marie Curie
jean-francois.perrot@lip6.fr

Wireless Sensor Networks (WSNs)

Benefits:
• Fine-grained sensing
• Easy deployment, no infrastructure required
• Enable new class of applications uQuery Engine

Challenges:
• Resource constraints (memory, bandwidth, energy)
• Large-scale coordination
• Combine the problems of networking, signal processing, real-time and embedded computing

Requirements for uQuery Engines
• Targeted at end-users, not programmers
• Dynamic: deploy and change behavior at run-time
• Support concurrency inherent to ambient systems
• Multiplicity of end-users


How can we program this system?
Query 1: determine ambient temperature and wind conditions
Query 2: Detect and locate deer moving through the region
Response to Query 1: temp=65°F wind: SE, 3mph
event
Response to Query 2: Deer at (10,35) moving NW at 5mph
Adapted from: [Boulis, 2005]

WSN Programming

State of the Art
• Programming tools focused on *efficiency*
• Low-level, C-based programming language (nesC)
• Lightweight, component-based OS (TinyOS)
  • Supports real-time programming, sensing, concurrency
  • Applications are compiled together with the OS

Macroprogramming WSNs
• Regiment, Semantic Streams, spreadsheet programming
• Do not meet requirements for uQuery Engines

Connecting together a large number of small computers with sensing and actuating capabilities, to collectively and cost-effectively solve problems, based on real-time data.

Application Example: Break Beam Detector

We want to detect an object passing through a break beam sensor, on request:
• Wait for a request message from the user
• Perform the requested action
  • Execute detectBeamEvent() primitive
  • Keep checking the sensor until a change in status is detected
• Send the result of detection back to the user

Mica2-Dot and Telos motes
http://research.sun.com/

The user (programmer) is responsible for choosing the right OS and network components, and assembling them along with the specific application logic into an executable program.

Note the following issues with the code below.
• Static:
  • Specification and linking of components at compile time
  • All components are compiled into a single image deployed on the sensor
• Low-level:
  • Programmer is responsible for managing: timing, communication, memory management, error handling
• No separation of concerns:
  • OS and network programming elements are inextricably linked with business logic programming elements

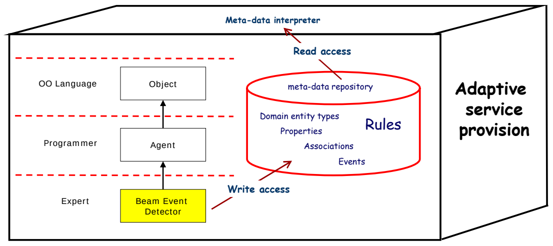
```
/* Detect break beam event application (code excerpt) */
configuration Example {}
implementation {
     // list of application components
     components Main, ExampleM, LedsC, GenericComm, TimerC,
          Photo, CC1000ControlM;
     // statically link all components
     Main.StdControl -> GenericComm;
     Main.StdControl -> TimerC;
     Main.StdControl -> Photo;
     Main.StdControl -> ExampleM;
     ExampleM.SendMsg -> GenericComm.SendMsg[10];
     ExampleM.ReceiveMsg -> GenericComm.ReceiveMsg[10];
     ExampleM.CC1000Control -> CC1000ControlM;
     ExampleM.Timer -> TimerC.Timer[unique("Timer")];
     ExampleM.Leds -> LedsC;
     ExampleM.PADC-> Photo;
}

module ExampleM {
     /* … */
}
implementation {
     TOS_Msg msgbuf;
     uint8_t msglen, sendPending;
     volatile uint8_t ioPending;
     uint16_t ioData;

     /* … */

     // primitive function #20: detect beam event (using photo sensor)
     uint16_t detectBeamEvent();

     // I/O: convert split phase non-blocking I/O to blocking I/O
     uint16_t IO(uint16_t a, uint16_t b) __attribute__((C,spontaneous)) {
          while (ioPending) yield();
          if (a == 20) { call PADC.getData(); ioPending=1; }
          while (ioPending) yield();
          return ioData;
     }
     async event result_t PADC.dataReady(uint16_t data) {
          ioPending=0; ioData=data;
          return SUCCESS;
     }

     // Communication: receive requests for execution and send results
     void sendPacket(uint8_t *buf, uint8_t n)
     __attribute__((C,spontaneous)) {
          memcpy(msgbuf.data, buf, n);
          msglen = n;
          if (call SendMsg.send(TOS_BCAST_ADDR, msglen, &msgbuf)
           == SUCCESS)
               sendPending = 1;
     }
     uint8_t isSendPending() __attribute__((C,spontaneous)) {
          return sendPending;
     }
     event result_t SendMsg.sendDone(TOS_MsgPtr mp, result_t success) {
          if (!success) call Timer.start(TIMER_ONE_SHOT, 200);
          else {
               call Leds.redToggle(); sendPending = 0;
          }
          return SUCCESS;
     }
     event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr mp) {
          TOS_Msg m;
          call Leds.greenToggle();
          if ((uint8_t)mp->data[0] == 20) {
               m.data = deref(detectShadow());
               sendPacket((uint8_t *)m.data, strlen(m.data));
          }
          return mp;
     }
     event result_t Timer.fired() {
          return call SendMsg.send(TOS_BCAST_ADDR, msglen, &msgbuf);
     }

     /* … */

     // Implementation of detectBeamEvent primitive
     uint16_t detectBeamEvent() {
          int i;
          uint16_t data, avg = 0;
          ledSet(0);
          for (i = 0; i < 10; i++)
               avg += IO(2, 0);
          avg /= 10;
          while ((data = IO(2, 0)) > avg - 15) yield();
          ledSet(7);
          return list(2, newWord(20), newWord(data));
     }
}
```

Ambiance: Adaptive Object Model-based Platform for Macroprogramming Sensor Networks

Standard AOM Architecture

AOM is a meta-data interpreter

Meta-data corresponds to data that specifies the Programs':
• Object-model (Structure and Behavior)
• Windows, Menus, Configuration Panel, …
• Saved as configuration data

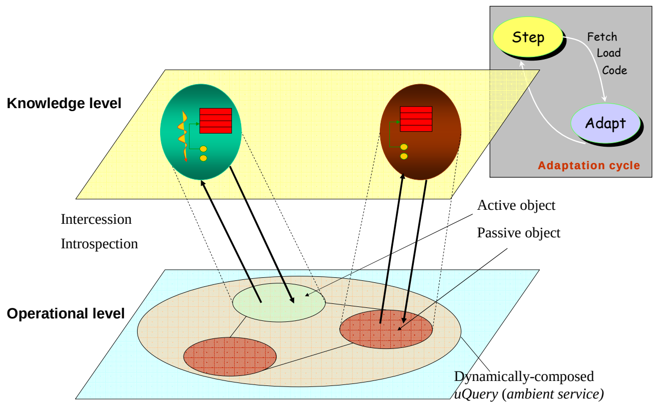


Open Issues with AOMs

• Have not been applied to WSNs
• Lack of standard techniques for
• WSN dynamic code generation
  • Supporting concurrency
  • Supporting separation of high-level control from the execution
  • Run-time optimization

Extended AOM Architecture for Macroprogramming WSNs

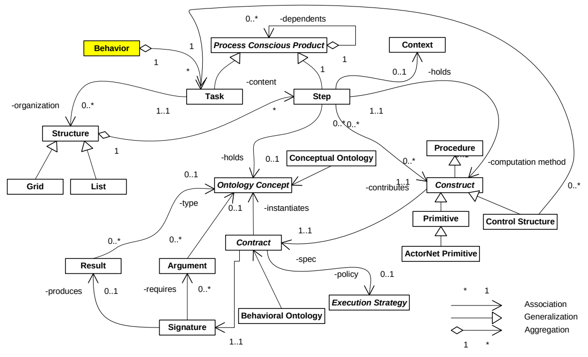
Knowledge level
• Comprises:
  • Conceptual ontology
  • Behavioral ontology
  • Framework for specifying queries as a composition of services through mediation of concepts
• Assumptions:
  • Completeness of the service ontology
  • Acknowledgeability of the users in the domain covered by the ontologies
  • Low-level data, such as the sensor id, may be provided by users (in the process of being relaxed)
• Keeps track of static and dynamic metadata.



Operational level
• Comprises a set of mobile agents
• The agents:
  • Are defined dynamically
  • Execute concurrently
    • within the WSN, and
    • on a single node
• Based on a formal model of computation
  • In order to be verifiable
  • Actors

Query Interpretation and Execution

Dart: Query Representation Framework

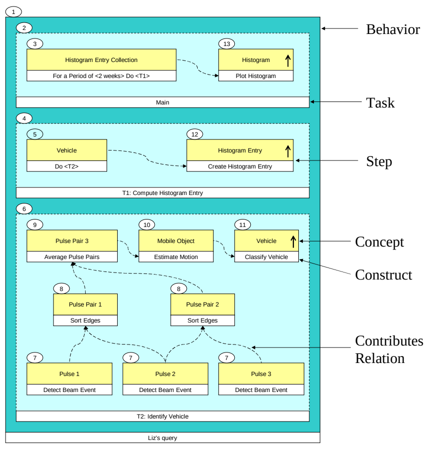


The Core Design of Dart: A Reusable and Extendible (Global) Behavior Representation Framework

Structure of queries
• Finite directed acyclic graph
• Recursive
  • Steps may hierarchically point to tasks
• Reflective
  • Same set of concepts reused to extend the system

Semantics of queries
• Parallel evaluation of contributions
• Limited to their dependencies
• Different execution semantics
  • Same set of concepts reused to extend the system
• Late
  • value binding
  • method binding

Query Representation Example



Liz's Query: compute histogram of vehicle arrival times for a period of two weeks
Source: [Whitehouse, Liu, Zhao 2006]

Collaborators:

Noury Bouraqadi
Computer Science Research Team.
Ecole des Mines de Douai

Christoph Dony
Université Montpellier-II
LIRMM, France

Ralph Johnson,
Software Architecture Group
University of Illinois at Urbana-Champaign

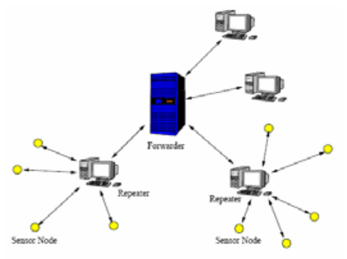
BioMedical Informatics
ERCIM Working Group

Alain Cardon
Université Pierre et Marie Curie

Vincent Ginot
Mobidyc Project
French National Institute for Agricultural Research

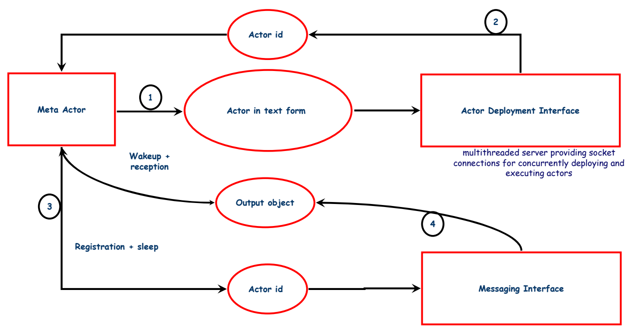
ActorNet: Implementation of the Operational Level

At the operational level, queries are executed by ActorNet
• A system of mobile, concurrently executing agents called *actors*
• Actor code is dynamically generated by the meta-level
• ActorNet language is extended with new keywords and services providing the means to *link* the meta-level and the operational level of the Ambiance platform



ActorNet platforms are deployed on sensor nodes or PCs
• Provide resource management, scheduling, communication, migration, sensing and actuation, etc., for actors.

Primitive Processing Algorithm


Actor id
2
Meta Actor
1
Actor in text form
Actor Deployment Interface
multithreaded server providing socket connections for concurrently deploying and executing actors
Wakeup + reception
3
Output object
4
Registration + sleep
Actor id
Messaging Interface

Break Beam Detector Example

For the break beam detector, the meta-level will generate the code for an actor of the Detect Beam step.
• An ActorNet agent template is provided by the execution strategy
• The Detect Beam meta-actor computes and fills in:
  • the destination sensor id (for migration)
  • meta-actor id (for communication)
  • the primitive to be executed (for application-specific functionality)
  • the arguments to the primitive (for control)

ActorNet Agent code for a call to the Detect Beam Primitive

```
( (lambda (migrate)
     ; actor behavior
     (seq
          ; migrate to destination (node 200, meta-actor id 111)
          (migrate 200 111)

          ; migrate to source (node 100) and report result
          (par (extmsg 111 (migrate 100
               ; perform application logic:
               ; detectBeamEvent() primitive (#20)
               ; which takes no arguments (nil)
               (prim 20 nil)
          )))
     ))

     ; migrate subroutine
     (lambda (adrs val)
          (callcc (lambda (cc)
               (send (list adrs cc (list quote val)))))
     )
)
```

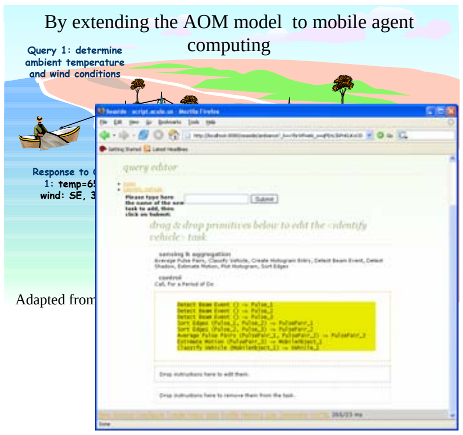
How does Ambiance satisfy the requirements of uQuery Engines?

The Ambiance platform supports:
• Using a WSN to serve concurrent users
• Dynamic, end-user-driven service specification
• Complex queries, comprising sensing and actuation

While meeting WSN constraints:
• Embedded, concurrent, distributed computing
• On highly resource-limited hardware components
• Work with a dynamic set of sensing resources

The two-level approach to architecting uQuery Engines allows separating:
• query representation and reasoning concerns, from
• those of their effective execution on divers runtime platforms
• through model-to-code transformation.

Using a mobile agent system as the query execution environment provides:
• dynamicity and concurrency of macroprogramming, while enabling
• load balancing and other optimizations required by the WSN environment


By extending the AOM model to mobile agent computing
Query 1: determine ambient temperature and wind conditions
Response to Q
1: temp=65
wind: SE, 3
Adapted from
query editor
Submit

Please type here the name of the new task to add, then click on Submit.

drag & drop primitives below to edit the <identify vehicle> task

sensing & aggregation
Average Pulse Pairs, Classify Vehicle, Create Histogram Entry, Detect Beam Event, Detect Shadow, Estimate Motion, Plot Histogram, Sort Edges

control
Call, For a Period of Do

```
Detect Beam Event () -> Pulse_1
Detect Beam Event () -> Pulse_2
Detect Beam Event () -> Pulse_3
Sort Edges (Pulse_1, Pulse_2) -> PulsePair_1
Sort Edges (Pulse_2, Pulse_3) -> PulsePair_2
Average Pulse Pairs (PulsePair_1, PulsePair_2) -> PulsePair_3
Estimate Motion (PulsePair_3) -> MobileObject_1
Classify Vehicle (MobileObject_1) -> Vehicle_1
```

Drop instructions here to edit them.

Drop instructions here to remove them from the task.

Web-based uQuery Engine User Interface

Uses: Seaside framework (http://www.seaside.st/) and Squeak (http://www.squeak.org/)

Separation of business logic primitives from the core of the mobile agent system, facilitates addition of new domain-specific primitives

Hooks are provided for quality attributes, such as:
• Security: automated supervision for security checks
• Auditability: who has been involved in what
• Non-repudiability: who has initiated which action

Reusability and extendibility of:
• The Ambiance Platform
• Its query representation framework