Requirements for uQuery Engines

• Combine the problems of networking, signal processing, real-time and embedded computing

• Large-scale coordination

• Resource constraints (memory, bandwidth, energy)

• Enable new class of applications uQuery Engine

• Easy deployment, no infrastructure required

• Fine-grained sensing

Benefits:

• Perform the requested action

• Wait for a request message from the user

• Execute detectBeamEvent() primitive

• Keep checking the sensor until a change in status is detected

• Send the result of detection back to the user

The user (programmer) is responsible for choosing the right OS and network components, and assembling them along with the specific business logic programming elements

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

• Execute detectBeamEvent() primitive

• Keep checking the sensor until a change in status is detected

• Send the result of detection back to the user

The user (programmer) is responsible for choosing the right OS and network components, and assembling them along with the specific business logic programming elements.

/* Detect break beam event application (code excerpt) */

// configuration Example M

implementation {

    // list of application components
    components Main, ExampleM, LedsC, GenericComm, TimerC;

    // statically link all components
    Main.StdControl -> GenericComm;
    Main.StdControl -> TimerC;
    Main.StdControl -> ExampleM;
    ExampleM.PADC -> Photo;
    ExampleM.Leds -> LedsC;
    ExampleM.Timer -> TimerC.Timer[unique("Timer")];
    ExampleM.CC1000Control -> CC1000ControlM;
    ExampleM.ReceiveMsg -> GenericComm.ReceiveMsg[10];
    Main.StdControl -> Photo;
    Main.StdControl -> TimerC;
    Main.StdControl -> GenericComm;
}

module ExampleM {

    /* Detect break beam event application (code excerpt) */
    // configuration Example M

    implementation {

        // list of application components
        components Main, ExampleM, LedsC, GenericComm, TimerC;

        // statically link all components
        Main.StdControl -> GenericComm;
        Main.StdControl -> TimerC;
        Main.StdControl -> ExampleM;
        ExampleM.PADC -> Photo;
        ExampleM.Leds -> LedsC;
        ExampleM.Timer -> TimerC.Timer[unique("Timer")];
        ExampleM.CC1000Control -> CC1000ControlM;
        ExampleM.ReceiveMsg -> GenericComm.ReceiveMsg[10];
        Main.StdControl -> Photo;
        Main.StdControl -> TimerC;
        Main.StdControl -> GenericComm;
    }

    void sendPacket(uint8_t *buf, uint8_t n)
    {
        // Communication: receive requests for execution and send results
        void sendPacket(uint8_t *buf, uint8_t n)
        {
            return list(2, newWord(20), newWord(data));
        }

        // implementation {
        // list of application components
        components Main, ExampleM, LedsC, GenericComm, TimerC;
        // statically link all components
        Main.StdControl -> GenericComm;
        Main.StdControl -> TimerC;
        Main.StdControl -> ExampleM;
        ExampleM.PADC -> Photo;
        ExampleM.Leds -> LedsC;
        ExampleM.Timer -> TimerC.Timer[unique("Timer")];
        ExampleM.CC1000Control -> CC1000ControlM;
        ExampleM.ReceiveMsg -> GenericComm.ReceiveMsg[10];
        Main.StdControl -> Photo;
        Main.StdControl -> TimerC;
        Main.StdControl -> GenericComm;
    }

    void sendPacket(uint8_t *buf, uint8_t n)
    {
        // Communication: receive requests for execution and send results
        void sendPacket(uint8_t *buf, uint8_t n)
        {
            return list(2, newWord(20), newWord(data));
        }

        // implementation {
        // list of application components
        components Main, ExampleM, LedsC, GenericComm, TimerC;
        // statically link all components
        Main.StdControl -> GenericComm;
        Main.StdControl -> TimerC;
        Main.StdControl -> ExampleM;
        ExampleM.PADC -> Photo;
        ExampleM.Leds -> LedsC;
        ExampleM.Timer -> TimerC.Timer[unique("Timer")];
        ExampleM.CC1000Control -> CC1000ControlM;
        ExampleM.ReceiveMsg -> GenericComm.ReceiveMsg[10];
        Main.StdControl -> Photo;
        Main.StdControl -> TimerC;
        Main.StdControl -> GenericComm;
    }

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

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

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

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

Query Interpretation and Execution

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 Framework

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

Query Representation Example

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

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)

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:
• dynamism and concurrency of macroprogramming, while enabling
• load balancing and other optimizations required by the WSN environment

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

How does Ambiance satisfy the requirements of uQuery Engines?