Macroprogramming with the Abstract Task Graph (ATaG)

Key Concepts

- Data-driven control flow over a distributed data store
  - An event can carry information about a phenomenon (target location, temperature reading) or merely indicate its occurrence to the next phase of processing.
- Task firing rules
  - periodic, any-data, and all-data - allow the specification of a range of sophisticated execution patterns
- Scheduling and communication is managed by a runtime
  - Programmer does not write networking code
  - System-level optimizations can be performed without requiring rewrite of application-level code
- Proc: Modularity, reusability, extensibility, low program complexity
- Cons: Asynchronous interaction, unpredictable delays, limited control over execution ordering

Mixed imperative-declarative program specification

- Separation of concerns: "what" of processing vs. "when and where"
- Imperative portion is a traditional (C/Java) sequential program interacting with the data pool
- Declarative portion is interpreted – at compile time and at run time – in the context of a particular network architecture
- Imperative and declarative parts can be modified independently

Why data-driven computing?

Programming abstractions

- Tasks can use data items at the desired level of abstraction without worrying about who produces it and how
- Tasks are unaware of each other, leading to highly extensible, reusable programs
- Communications and coordination is performed in the underlying system and hidden from the programmer

Software and hardware

- Event-driven processing leads to efficient resource utilization
- Parameterized runtime system template is amenable for automatic software synthesis
- Runtime system can be ported to an entirely different platform while providing the same application-level interface

Programming Idioms

- Controlled Collection
- Neighbor-to-Neighbor
- Tree-based Aggregation

DART: Data Driven ATaG Runtime

- Abstracts 'system-level' functionality and the related optimizations
- Application-independent control and coordination mechanism is separated from application-specific configuration
- Modular structure with well-defined inter-module interfaces: (a) simplifies software synthesis, and (b) enables replacement or enhancement of inter-module protocols with minimal system redesign
- Requires support for multi-threaded execution and fixed-priority preemptive scheduling — available in most RTOSes (e.g., μC/OS-II, RT Java)

Visual ATaG Programming and Software Synthesis

Auto-generated code skeleton

- Populated by programmer
- Config file for each node

Customization of the DART template

- Create notification of acquisition or loss of target

Functional simulation and visualization

- Simulator: Simulate, trace, profile
- Runtime: Real-time execution
- GUI: Visualization, debugging, monitoring

DART: Data Driven ATaG Runtime

- Sensor(s)
- Actuator(s)
- UserTask 1
- UserTask n
- NetworkArchitecture
  - Maintain neighborhood information, virtual topologies, logical namespaces
- DataPool
  - Store task code and task/channel annotations, spawn ready tasks
- ATaGManager
  - Store task code and task/channel annotations, spawn ready tasks
- Dispatcher
  - Translate annotations, dispatch data to other nodes
- NetworkStack
  - Routing, medium access, physical layer protocols
- Transceiver


http://indus.usc.edu/atag/