Design of an Object-Oriented Framework for Data Format Classification and Transformation

Dustin Graves
dgraves@gwu.edu

May 9, 2008

Core Concepts

- Management and Manipulation of live data streams
- Dynamic composition of data processing pipelines to transform stream data
- Efficient processing of high-rate streams
- Reduction of application code duplication

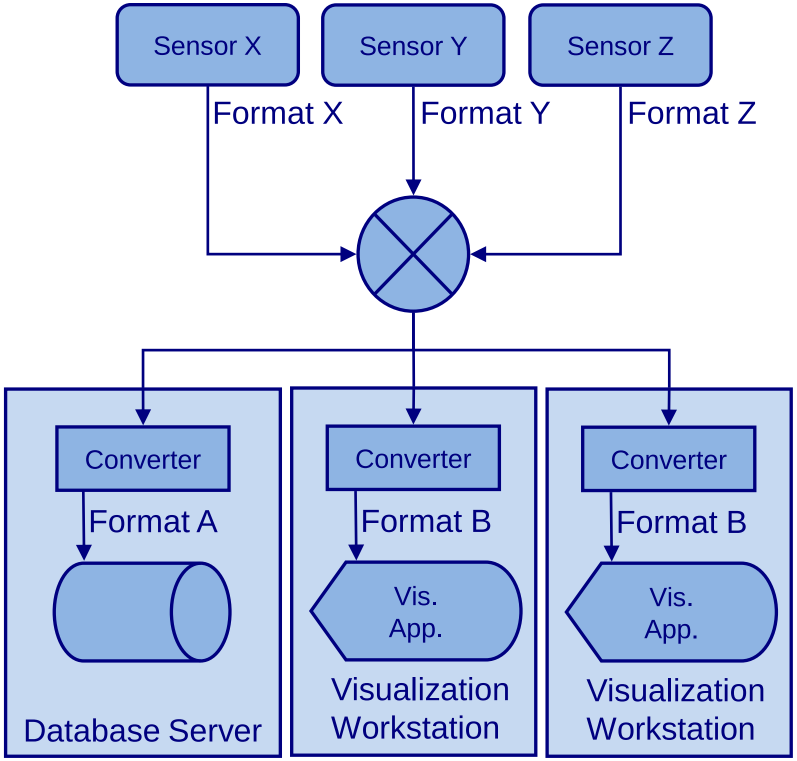
Framework Overview

- Facilitate comprehensive data exchange among software applications
- Encapsulate common elements of the data interchange process
- Assist with development of interoperable applications
  - Sharing data among software using different formats
    - Interoperation with legacy software
    - Conversion of non-standard/proprietary formats
- Exploit parallelism existing among independent streams and independent objects within streams
  - Concurrently process multiple high-rate data streams
  - Hide parallel programming details from framework user
  - Scale from small-scale to large-scale systems

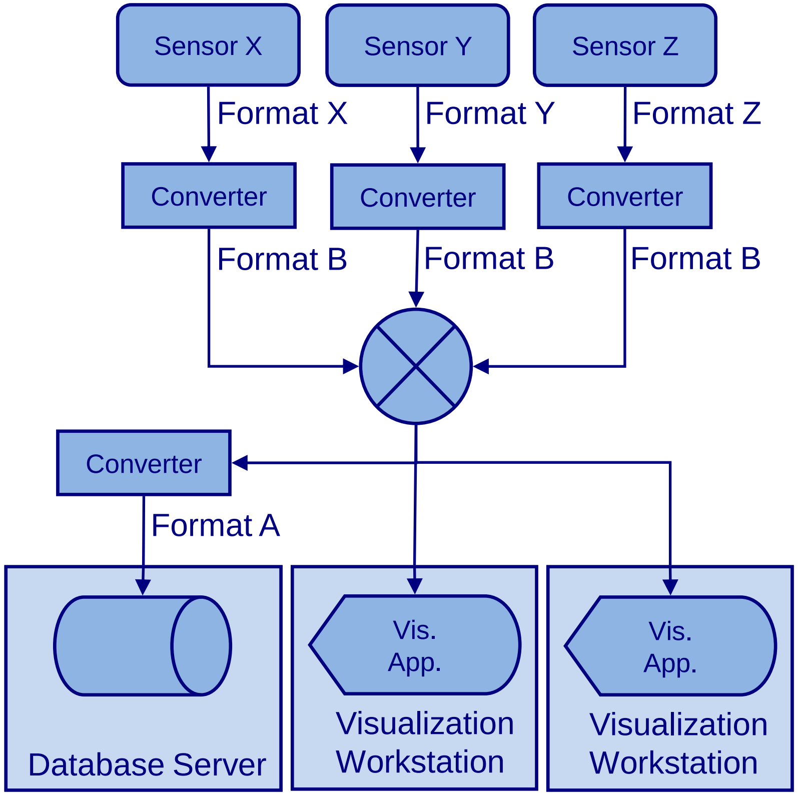
Motivation

- Integration of software with military test and training ranges
    - Sensor networks for tracking range activities composed of applications and devices from different vendors and eras
- Different approaches to interoperability with sensor networks
    - Consumer is responsible for data format conversion
        - Inefficient, duplication of effort, does not scale well
    - Producer is responsible for data format conversion
        - Each producer generates data represented with a “standard” format
        - Addition of new data producers to the system does not require modification to existing consumers
    - Gateways are responsible for data format conversions
        - Not required to directly modify consumers and producers for system integration
        - Connect multiple ranges, each with its own native data formats

Approaches to Interoperability



Sensor network with multiple data formats: Consumer must perform format conversion



Sensor network with a common data format: Producer or gateway must perform format conversion

Related Work

- Principles of object-oriented design
- Evolving frameworks
- Design patterns
- Linear types for packet processing
  - The PACLANG programming language

Properties of Data Formats

- Assign structure to data such that it may be processed by an application or understood by a human
- Important properties of data formats:
  - All data that may be safely processed by an application has structure
  - Structure may be separated from the associated data
  - Structure may be specified at run-time

Properties of Data Streams

- Sequences of digitally encoded signals representing information in transmission
- May consist of aggregations of data describing multiple objects
- Categories of data streams:
  - *Single object, single format*: Stream contains messages of a single type describing a single object
  - *Single object, multiple formats*: Stream contains messages of multiple types describing a single object
  - *Multiple objects, single format*: Stream contains messages of a single type describing multiple objects
  - *Multiple objects, multiple formats*: Stream contains messages of multiple types describing multiple objects

Our Work: Exploiting Parallelism

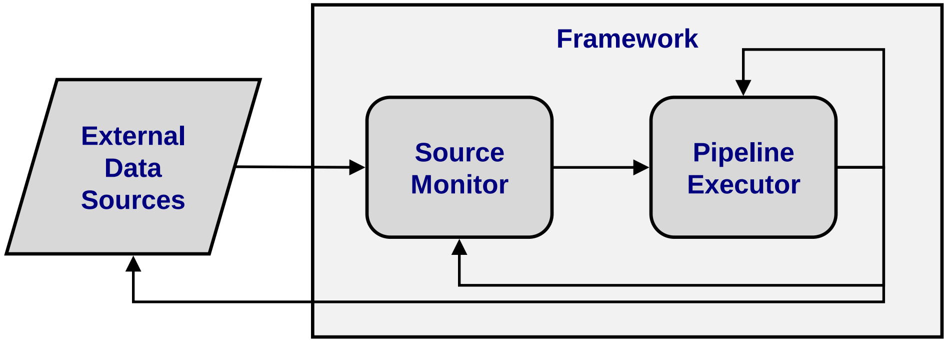
- Potential for exploiting data independence within streams transmitting multiple objects and formats
- Independence existing among individual items within aggregate streams may be exploited for concurrent processing
  - Partial independence may require serialization of some processing stages
  - Full independence allows unrestricted processing, providing greater scalability
- Data independence is not limited to streams containing multiple items
  - Blocks of data transmitted by single object, single format streams can potentially be processed concurrently
    - Need to ensure the preservation of block ordering

Framework Design

- A white-box framework to manage data communication and processing
- Data is received and transmitted through abstractions of communication resources
- Data received from a communication resource is submitted to a data processing pipeline
  - A linear type system ensures that each processing pipeline has unique ownership of data items
  - Pipeline stages performing write operations must contain exactly one reference to a data item
  - Stages performing read-only operations may contain multiple references to a data item.
- The application developer provides application-specific data formats and transformations for use with the framework

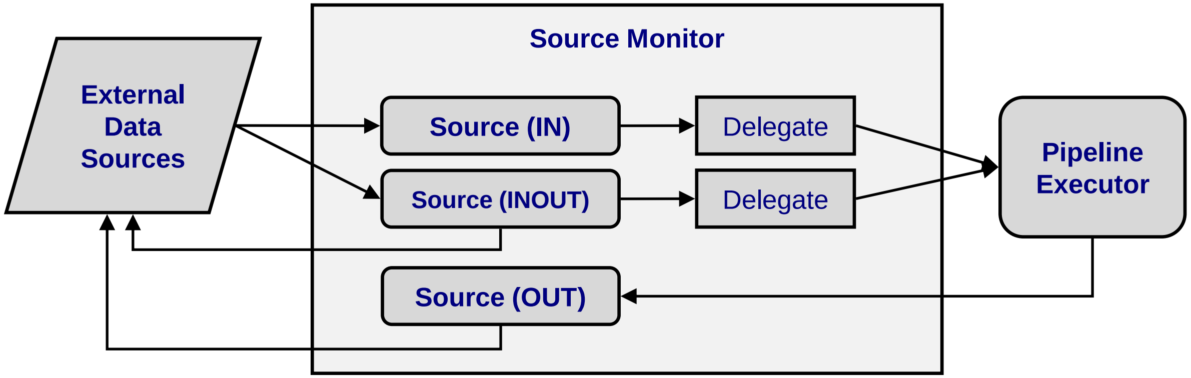
Framework Modules

- The framework consists of two main modules for monitoring communication resources and transforming data
  - The Source Monitor manages communication resources
  - The Pipeline Executor manages data processing pipelines



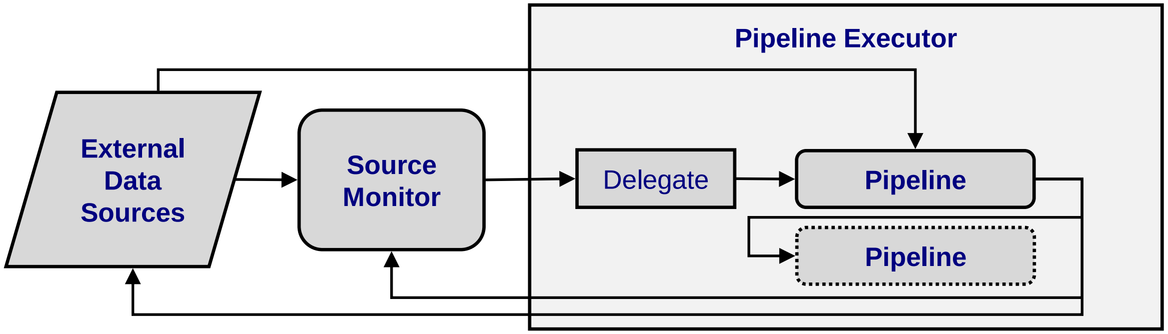
Source Monitor

- Responsible for detecting and executing pending communication events from communication resources
  - May operate within a dedicated thread
  - Communication resources are processed concurrently
  - Methods for registering, querying, and deregistering communication resources must be thread safe



Pipeline Executor

- Responsible for managing data processing pipelines to filter, transform, and translate data received from communication resources
  - May operate within a dedicated thread
  - Pipelines are processed concurrently
  - Methods for adding, modifying, and removing pipelines must be thread safe



Implementation Results

- Phylum is an implementation of the framework using C++ and the Intel Threading Building Blocks library
  - Template-based algorithms for parallel processing provided by TBB are used extensively
- Supports dynamic creation of data formats
  - Provides a generic class encapsulating a collection of fields of different types
  - Each field contains one or more elements
  - New data formats are created through composition of fields
- Addresses memory performance issues
  - Attempt to reduce impact of constant data format allocation and destruction by recycling memory with a free list
    - Provides a free list container for each format type
    - Requires locking within concurrent environments, limiting scalability
  - Alternate choice of a scalable memory allocator based on McRT-Malloc
  - Free list or scalable allocator selection is made at compile-time

Future Direction

- Continued framework development
  - Libraries of fine-grained objects
  - Black-box framework
- Development of a domain specific language
  - Run-time definition, creation, and manipulation of framework objects from within a language interpreter
  - Distributed language with each interpreter acting as a node within a network of interpreters
  - Remotely and securely manipulate other interpreters
- Creation of visualization tools for monitoring the flow of data through the framework

Backup Slides

Interchange Process

- The data interchange process consists of a number of common operations
  - Receive data from a communication resource
    - May need to detect availability of data
  - Decode or de-serialize data (if necessary)
  - Transform/prepare data for consumption
    - Discard invalid data
    - Transform individual fields of a data format
    - Reorganize fields of a data format relative to each other
  - Submit data for processing
    - Submit to application, submit to communication resource , re-submit to framework

Framework Requirements

- Support multiple communication resource types
- Observe communication resource state and process communication events without disrupting normal application operation
- Support definition of new data formats at the application level
- Identify and map unstructured data received from an I/O resource to a structured format
- Apply transformations to prepare data for consumption
- Support definition of application specific transformations and methods for data consumption
- Allow the end-user, with limited knowledge of software-engineering, to define new data formats and transformations

Related Work: OOD

- Characterizing the extent of component reuse with object domains
  - *Foundation-domain*: General purposes classes not specific to any environment
  - *Architectural-domain*: Classes particular to a specific architectural environment
  - *Business-domain*: Classes particular to a specific industry
  - *Application-domain*: Classes particular to a specific application

Related Work: Evolving Frameworks

- Frameworks capture the general design of components common to specific types of applications
  - *White-box Frameworks*: Provide abstract classes to be extended when adding application-specific code
  - *Black-box Frameworks*: Provide libraries of predefined subclasses containing application-specific code
- Start with a white-box framework
- Introduce new components with each framework application
- Identify hot-spots of application specific code and separate generic functionality from application specific functionality
- Library of fine-grained, concrete objects grows large enough to form a black-box framework
- Create tools for constructing applications through composition of framework components
- Create language tools for inspecting and debugging framework based applications

Related Work: Design Patterns

- Describe common elements of reusable object-oriented software
  - Three categories of design pattern: creational, structural and behavioral
- Two important communication specific patterns for demultiplexing and dispatching communication events
  - *Reactor pattern*: Behavioral pattern to synchronously monitor communication resources and dispatch communication events as they occur
  - *Proactor pattern*: Behavioral pattern to asynchronously monitor communication resources and process/dispatch communication events as they occur

Related Work: Linear Types

- Linear types may have exactly one reference and may be neither duplicated nor discarded
  - Explicit specification of type allocation and destruction
  - Safe modification within concurrent environments
- Relax requirements such that each value may have multiple read-only references
- Allow multiple references when providing a complete copy of a value to each reference
- PACLANG is a concurrent, linear-typed language for packet processing
  - Uses a linear type system providing unique ownership of packets to a single thread
  - Multiple references are allowed within a thread
  - Eliminates the need for locking and allows threads to safely transfer packet ownership to other threads