

High-Level Application Protocols

Nikolay Malitsky

Eight International Workshop on Personal Computers and Particle Accelerators

5-8 October, 2010
Saskatoon, Canada

BROOKHAVEN
NATIONAL LABORATORY

a passion for discovery



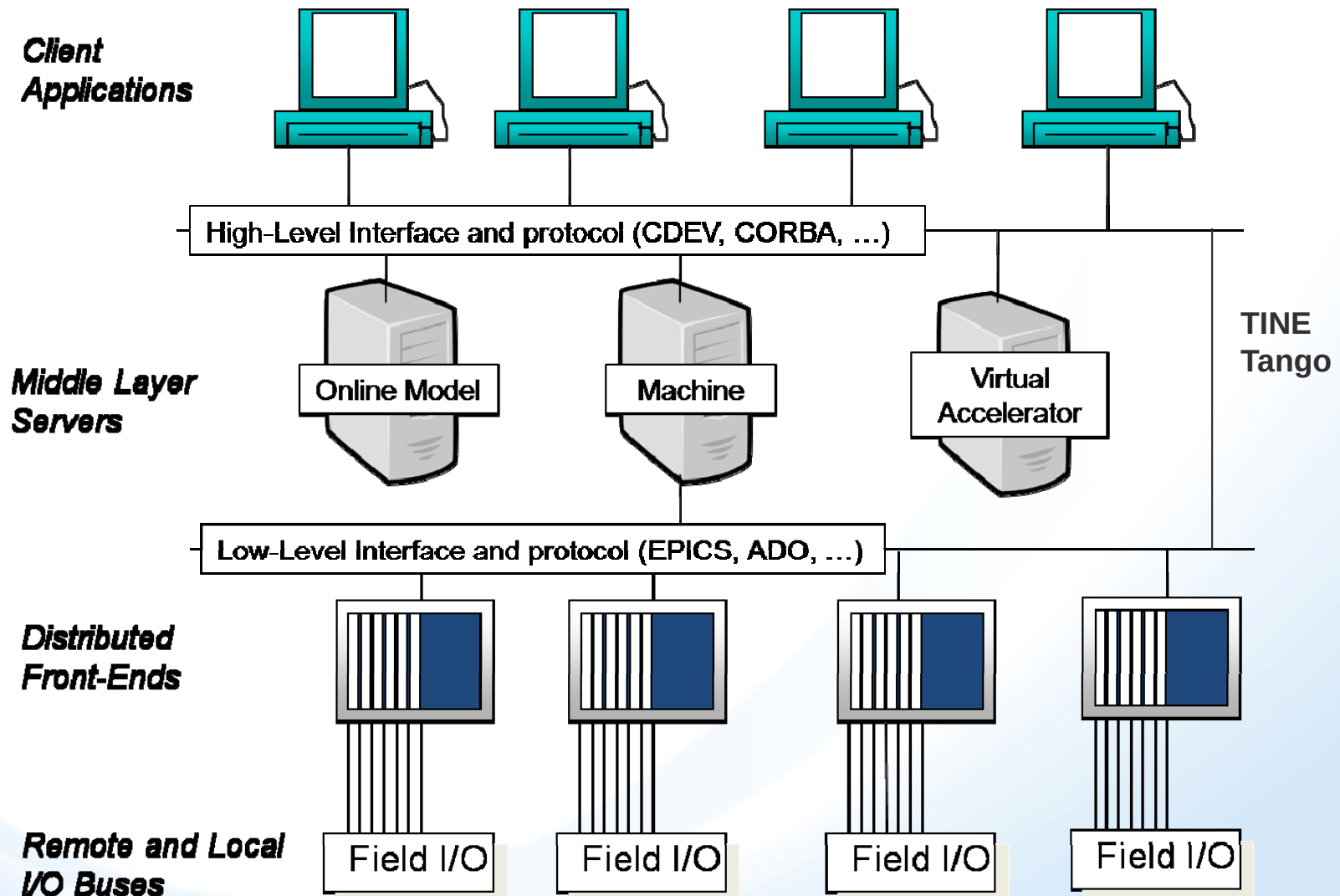
U.S. DEPARTMENT OF
ENERGY

Office of
Science

Outline

- Rationale
- Type-related concerns
- (Some of) the previous approaches
 - CDEV
 - TINE
 - CORBA 2.x
- (Some of) the new directions
 - EPICS-PVData
 - Google Protocol Buffers
 - DDS Dynamic Data
- Next step

Typical three-tier high level application environment



Type-related concerns

Extensible and Dynamic Topic Types for DDS, 2010

- Type System
- Type Representation
- Data Representation
- Language Binding
 - **Plain Language Binding (type-specific)**: based on compile-time knowledge of the type
 - **Dynamic Language Binding (generic)**: allows dynamic type definition and manipulation of data without compile-time knowledge

CDEV: Common DEVice

C. Watson, J.Chen, D.Wu, W. Akers (1996)



- **Type system:** primitive types, multi-dimensional arrays, maps
- **External type representation:** none
- **Payload:** binary stream that represents the contents of the *cdevData* objects encoded with SUN's External Data Representation (XDR)
- **Language binding:**
 - **Plain language binding:** none
 - **Dynamic language binding:** all data types are represented with a *cdevData* object, an associative collection of tagged values.

```
class cdevDevice:
{
public:
    int send (char* msg, cdevData& out, cdevData& result);
    int sendNoBlock (char* msg, cdevData& out, cdevData& result);
    int sendCallback (char* msg, cdevData& out, cdevCallback cb);
};
```

CDEV Linear Internet Protocol

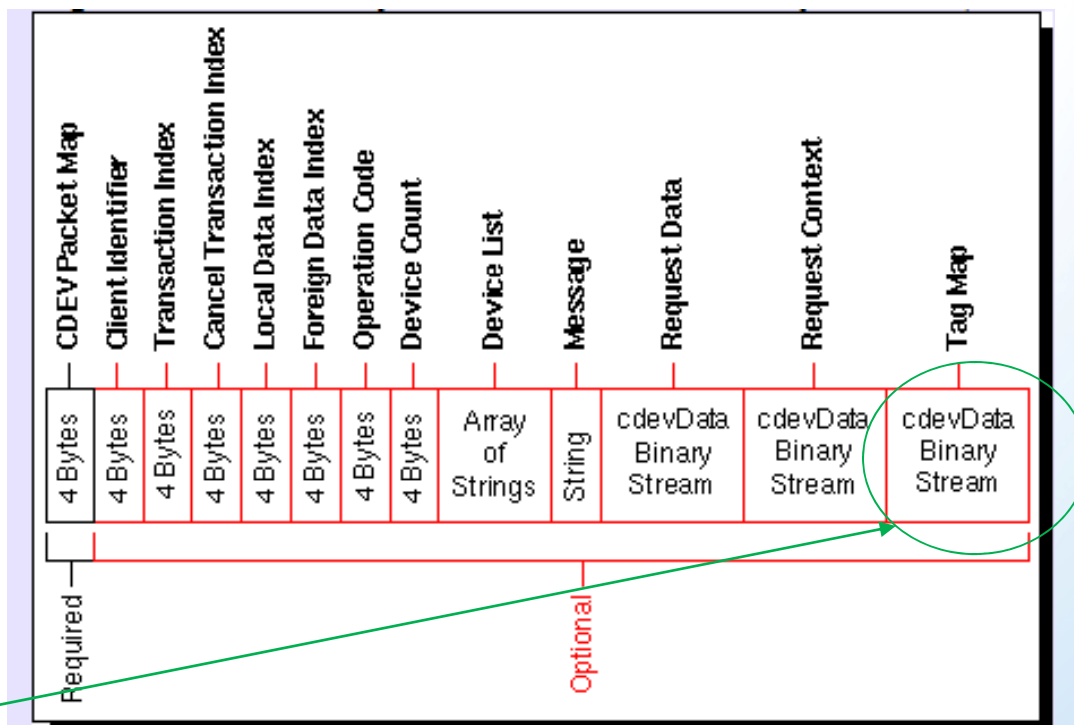
Message: a string including command and attribute.

Request/Reply Data: a binary stream that represents the contents of a cdevData object .

Request Context: a binary stream that represents the contents of a cdevData object that contains the context of the device/message combination at the time the message was transmitted

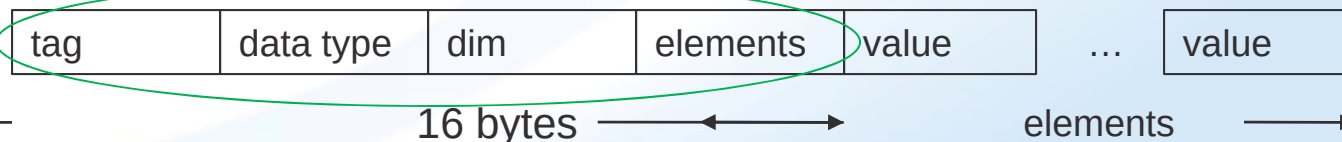
Tag Map: a binary stream that represents the contents of a cdevData object. This object is populated with current tag table integers and tag table strings that are in use in the client process.

Structure of the CDEV Packet Binary Stream



where cdevData is a sequence of tagged values:

Embedded type definition



In the case of the multi-dimensional arrays, each dimension additionally contains offset and length.

TINE, 2.x

P.Duval, S. Herb, J. Speth, 1994



- **Type system**: primitive types, arrays, structures (nested)
- **External type representation**: none (defined on the server side)
- **Payload**: binary stream encoded with the TINE representation
- **Language binding**:
 - **Plain language binding**: yes (the C structure + structStruct/structFormat)
 - **Dynamic language binding**: is represented by the **structStruct** and **structFormat** reflection interfaces and the navigation interfaces for accessing data.

```
struct structFormat{
    int siz; /**< number of elements. */
    int fmt; /**< TINE CF_ format. */
    int off; /**< offset in the byte stream */
    int addr; /**< Byte offset within the structure. */
    char field[FIELD_NAME_SIZE]; /**< structure field name */
    char stag[TAG_NAME_SIZE]; /**< sub tag if fmt == CF_STRUCT */
    struct structStruct *owner; /**< reference to struct owner of this structFormat */
    struct structFormat *nxt; /**< Pointer to next element. */
};
```

CORBA 2.x: Common Object Request Broker Architecture



OMG, 1996 - 2001



- **Type system:** primitive types, ..., structures, objects (values and remote references)
- **External type representation:** IDL
- **Payload:** binary stream encoded with Common Data Representation (CDR)
- **Language binding:**
 - **Plain language binding:** is generated from IDL as client stubs and server skeletons
 - **Dynamic language binding:** is represented by the DynAny interface for dynamically composing and decomposing any values.
- **Communication models:**
 - Blocked synchronous request
 - Deferred synchronous request
 - One-way (best-effort) request
 - Publish/Subscribe with the Event or Notification services

Common Data Representation (CDR)

Primitive Types:

- specified for both big-endian and little-endian orderings
- aligned on their natural boundaries. Any primitive of size n octets must start at an octet stream index that is a multiple of n.
- representation of floating point numbers follow the ANSI/IEEE Standard 754-1985

Sequences: encoded as an unsigned long value (number of elements), followed by the elements of the sequence.

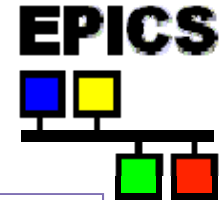
Structures: encoded in the order of their declaration in the structure

Any: encoded as a **TypeCode** followed by the encoded value.

TCKind	Integer Value	Type	Parameters
tk_null	0	empty	– none –
tk_void	1	empty	– none –
tk_short	2	empty	– none –
tk_long	3	empty	– none –
tk_ushort	4	empty	– none –
tk_struct		15	complex
			string (repository ID), string (name), ulong (count) {string (member name), TypeCode (member type)}

EPICS-PVData – 1 of 2

M. Kraimer, 2008



<http://epics-pvdata.sourceforge.net/>

Features

PVData

PVData (Process Variable Data) defines and implements an efficient way to store, access, and create memory resident structured data. Although defined with Java syntax the interfaces also document what is required for implementation in other languages such as C++.

channel access - network access

caV4 implements a version of channel access that provides network support for transferring PVData.

interface definitions

The Java package org.epics.pvData.pv has Java interface definitions that define PVData. Package org.epics.ca.client has java interface definitions that define channelAccess. Although defined with Java syntax the interfaces also document what is required for implementation in other languages such as C++.

implementation

Project pvData provides a complete Java implementation of PVData. It is used by other EPICS PVData projects. Project CAjV4 provides network support for pvData. Project javaIOC implements a smart memory resident soft real time database. Each field of each record can optionally have attached support code which is called when a record is processed.

efficient

Small memory footprint, low cpu overhead, and concise code base.

data storage

PVData defines separate introspection and data interfaces. The introspection interfaces provide access to immutable objects, which allows introspection instances to be freely shared. The introspection interface for a process variable can be accessed without requiring access to the data.

data access

Client code can access PVData via the introspection and data interfaces. For "well known" data, e.g. timeStamp, specialized interfaces can be provided without requiring any changes to the core software.

data transfer

The separation of introspection and data interfaces allows for efficient network data transfer. At connection time introspection information can be passed from server to client. Each side can create a data instance. The data is transferred between these instances. The data in the network buffers does not have to be self describing since each side has the introspection information.

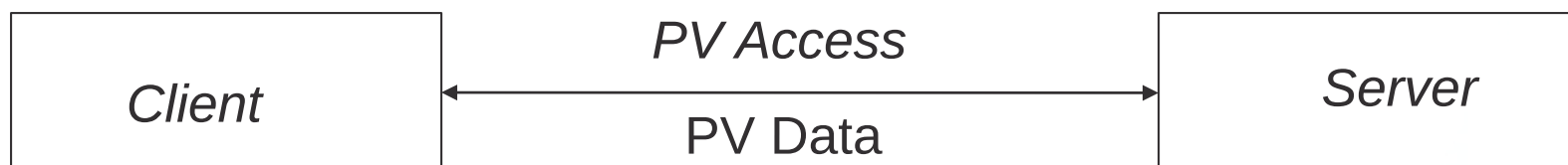
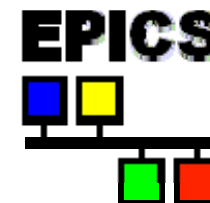
memory resident

PVData only defines memory resident data.

structured data

PVData has three types: scalar, array, and structure. A scalar can be one of the following: boolean, byte, short, int, long, float, double, string. An array is a one dimensional array with the element type being one of the scalar types. A structure is an ordered set of fields where each field has a name and type. Since a field can have type structure complex structures are supported. No other types are needed since structures can be defined that simulate types.

EPICS-PVData - 2 of 2



- **Type system:** primitive types, arrays, structures
- **External type representation:** XML
- **Payload:** binary stream encoded with the PVAccess representation
- **Language binding:**
 - **Plain language binding:** none
 - **Dynamic language binding:** is represented by the **Field** reflection interfaces and the **PVData** interfaces for accessing data.

In EPICSv4 CA, the API is primarily asynchronous, and the underlying design is optimized for asynchronous operation. Synchronous API will be created atop of the asynchronous one.

```
requestResponse(responseHandler, errorHandler);  
...  
void responseHandler(char *response) { ... }  
void errorHandler(char *error) { ... }
```

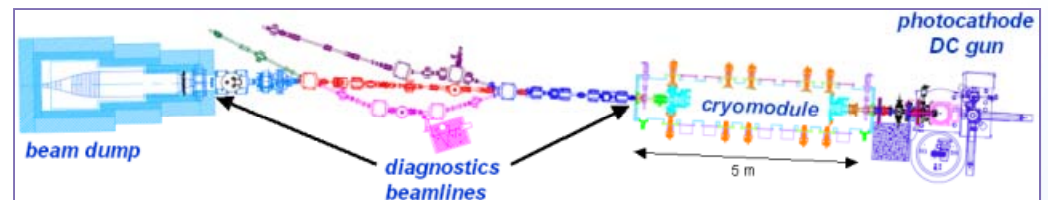
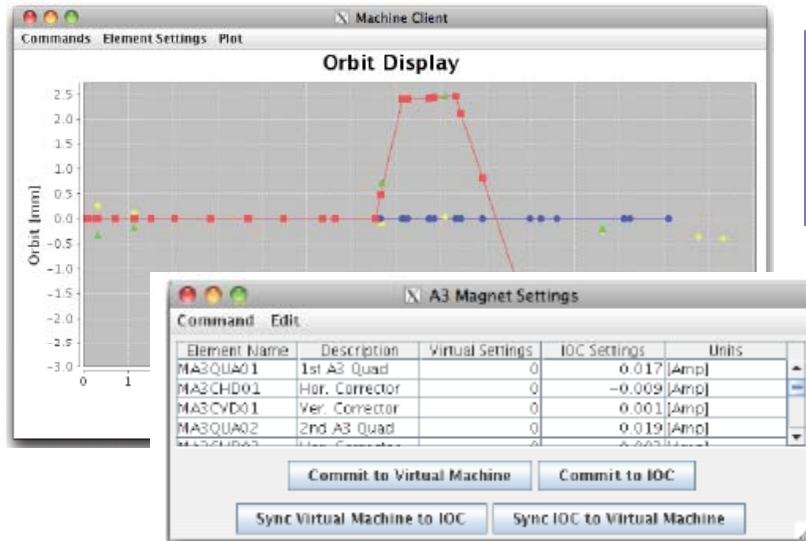
K. Zagar, et al. Evolution of the EPICS Channel Access Protocol, ICALEPCS '09

In this case, responseHandler is called when the response is ready, and requestResponse call completes immediately (no blocking). Also, unexpected conditions can be handled in the errorHandler.

Applications – 1 of 2

IPAC'10: THE NTMAT EPICS-DDS VIRTUAL ACCELERATOR FOR THE CORNELL ERL INJECTOR

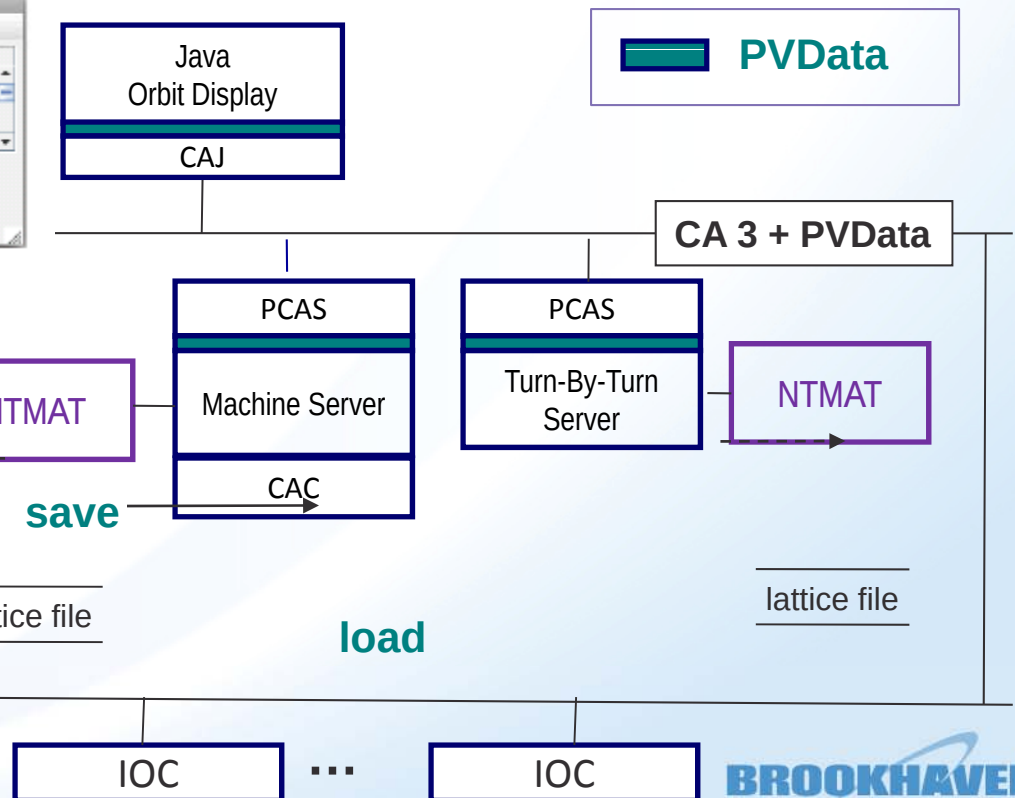
C. Gulliford, I. Bazarov, J. Dobbins, R. Talman (Cornell University), N. Malitsky (BNL)



PVData-based structures:

- Machine Server Request
- Turn-by-turn data

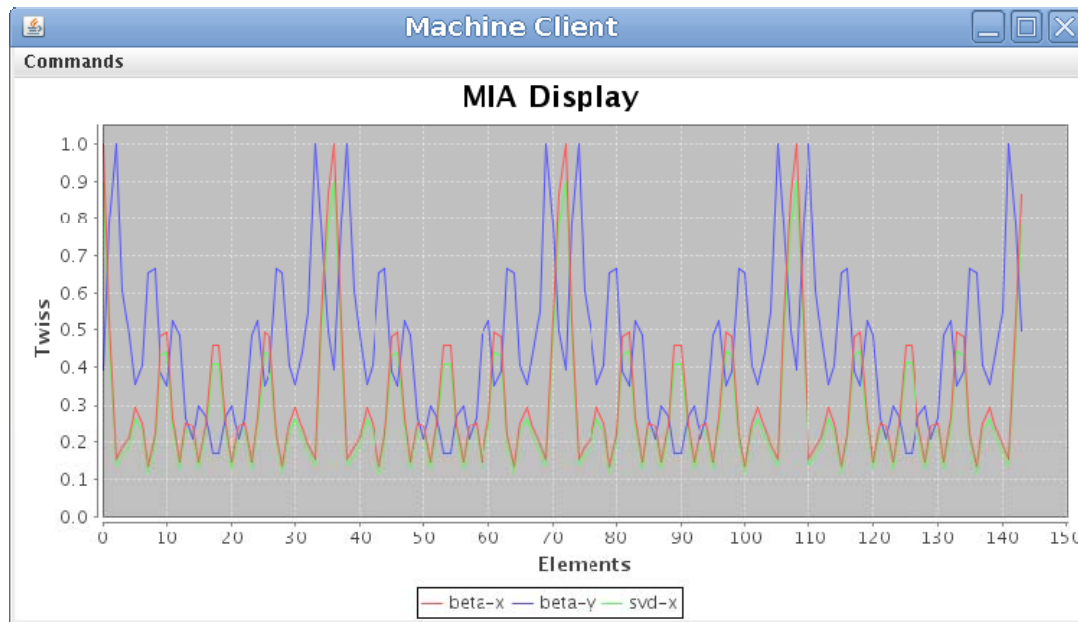
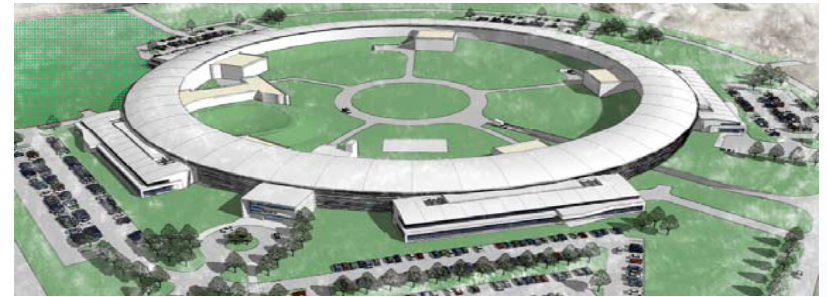
```
struct Position{
    double x;
    double y;
};
struct TBTDData {
    string name;
    vector<Position> tbt;
};
typedef vector<TBTDData> TBTDArray;
```



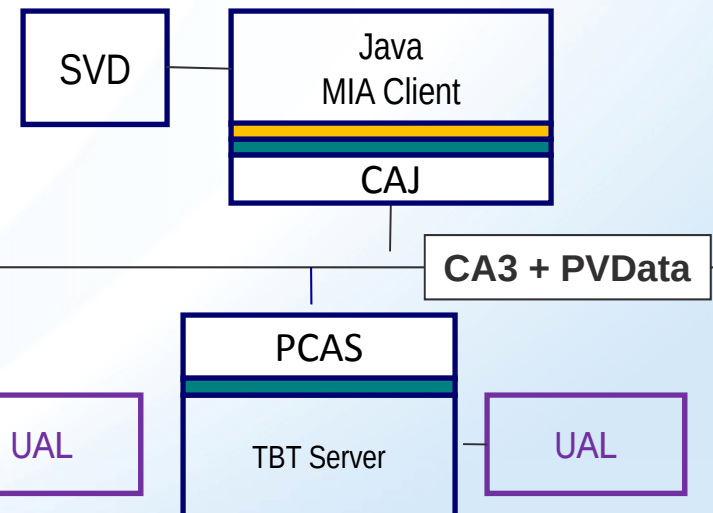
Applications – 2 of 2

IPAC 10: APPLICATION OF MODEL INDEPENDENT ANALYSIS WITH EPICS-DDS

N. Malitsky, I. Pinayev (BNL), R. Talman (Cornell U),
C. Xiaomeng (Stony Brook University)



DDS Dynamic Data
PV Data



Google's Protocol Buffers -1 of 2



<http://code.google.com/p/protobuf/>

Message Structure

As you know, a protocol buffer message is a series of key-value pairs. The binary version of a message just uses the field's number as the key – the name and declared type for each field can only be determined on the decoding end by referencing the message type's definition (i.e. the `.proto` file).

When a message is encoded, the keys and values are concatenated into a byte stream. When the message is being decoded, the parser needs to be able to skip fields that it doesn't recognize. This way, new fields can be added to a message without breaking old programs that do not know about them. To this end, the "key" for each pair in a wire-format message is actually two values – the field number from your `.proto` file, plus a *wire type* that provides just enough information to find the length of the following value.

The available wire types are as follows:

```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phone = 4;
}
```

Type	Meaning	Used For
0	Varint	int32, int64, uint32, uint64, sint32, sint64, bool, enum
1	64-bit	fixed64, sfixed64, double
2	Length-delimited	string, bytes, embedded messages, packed repeated fields
3	Start group	groups (deprecated)
4	End group	groups (deprecated)
5	32-bit	fixed32, sfixed32, float

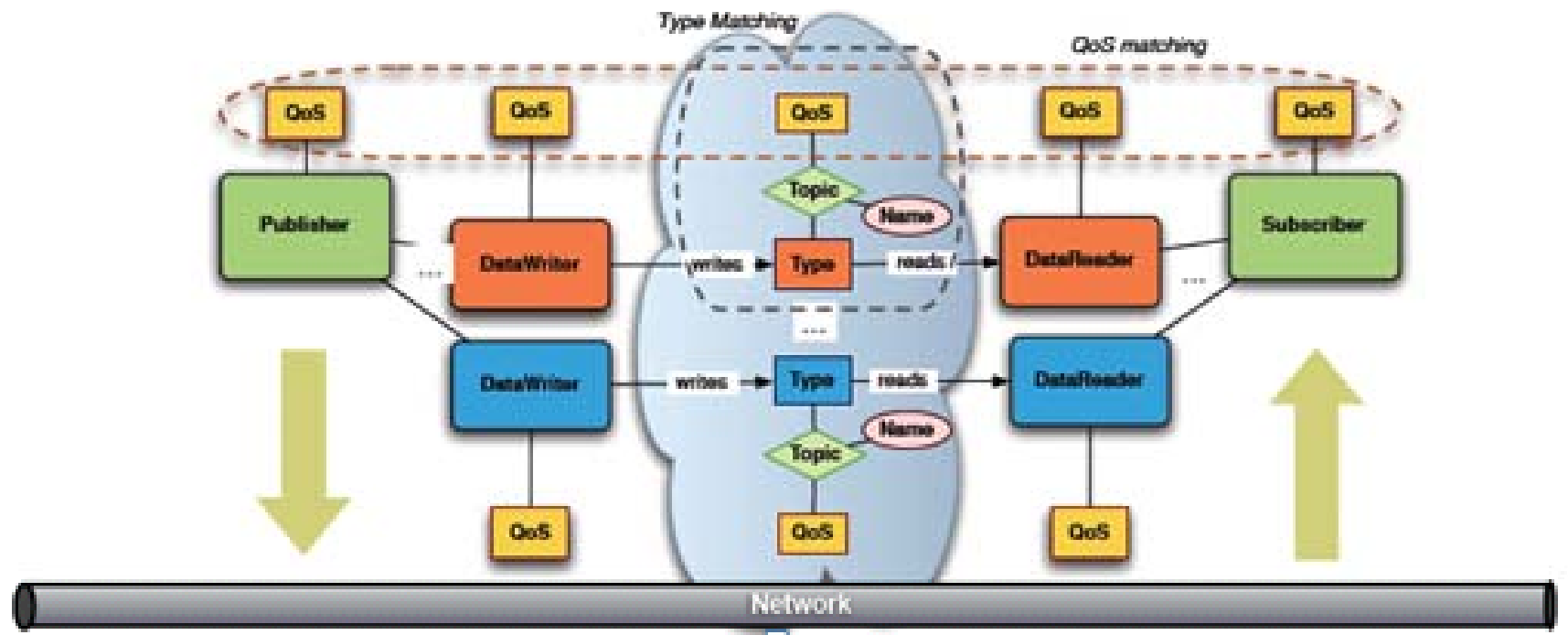
Google's Protocol Buffers -2 of 2



“Protocol Buffers are a way of encoding structured data in an efficient yet extensible format. Google uses Protocol Buffers for almost all of its internal RPC protocols and file formats”.

- **Type system:**
 - primitive types, variable-size sequences, structures
 - type versioning and evolution
 - sparse types
- **External type representation:**
 - .proto files
- **Payload:**
 - Coded Input and Output streams
- **Language binding:**
 - **Plain language binding:** is generated from .proto files
 - **Dynamic language binding:** is represented by the **Descriptor(s)** and **DynamicMessage** interfaces.

QoS: reliability, predictability, availability, timeliness, etc.



BROOKHAVEN
NATIONAL LABORATORY

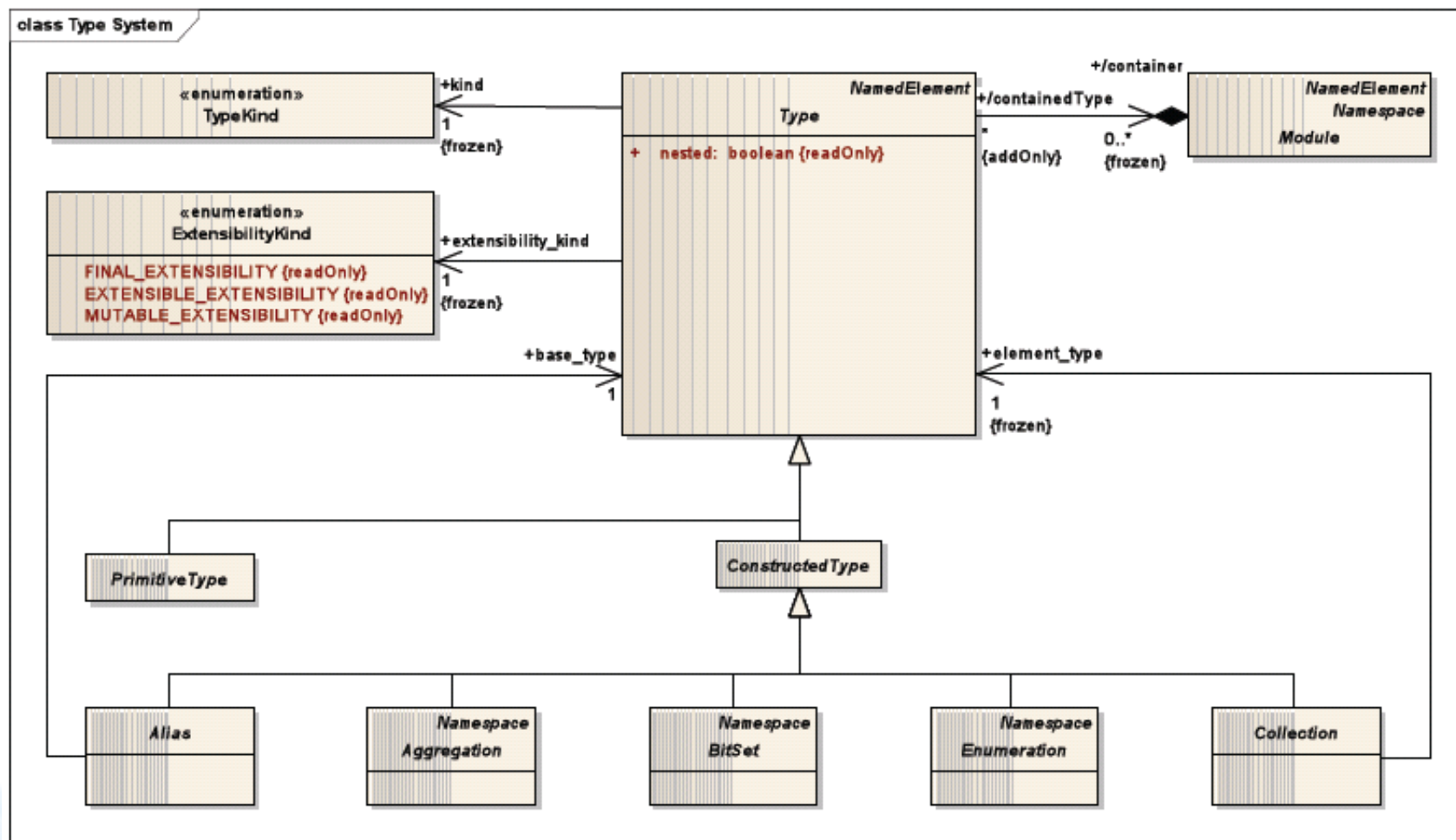
DDS specifications

- Data Distribution Service for Real-Time Systems, Version 1.2
formal 07-01-01, January 2007
- **Extensible Dynamic Topic Types for DDS**
RFP: June 27, 2008
Revised Submission: February 22, 2010
Adopted Beta 1: May 21, 2010
Deadline for comments: November 29, 2010
FTF (Finalization Task Force) Recommendation and Report : April 4, 2011
Beta 2 (finalized specification): TBD

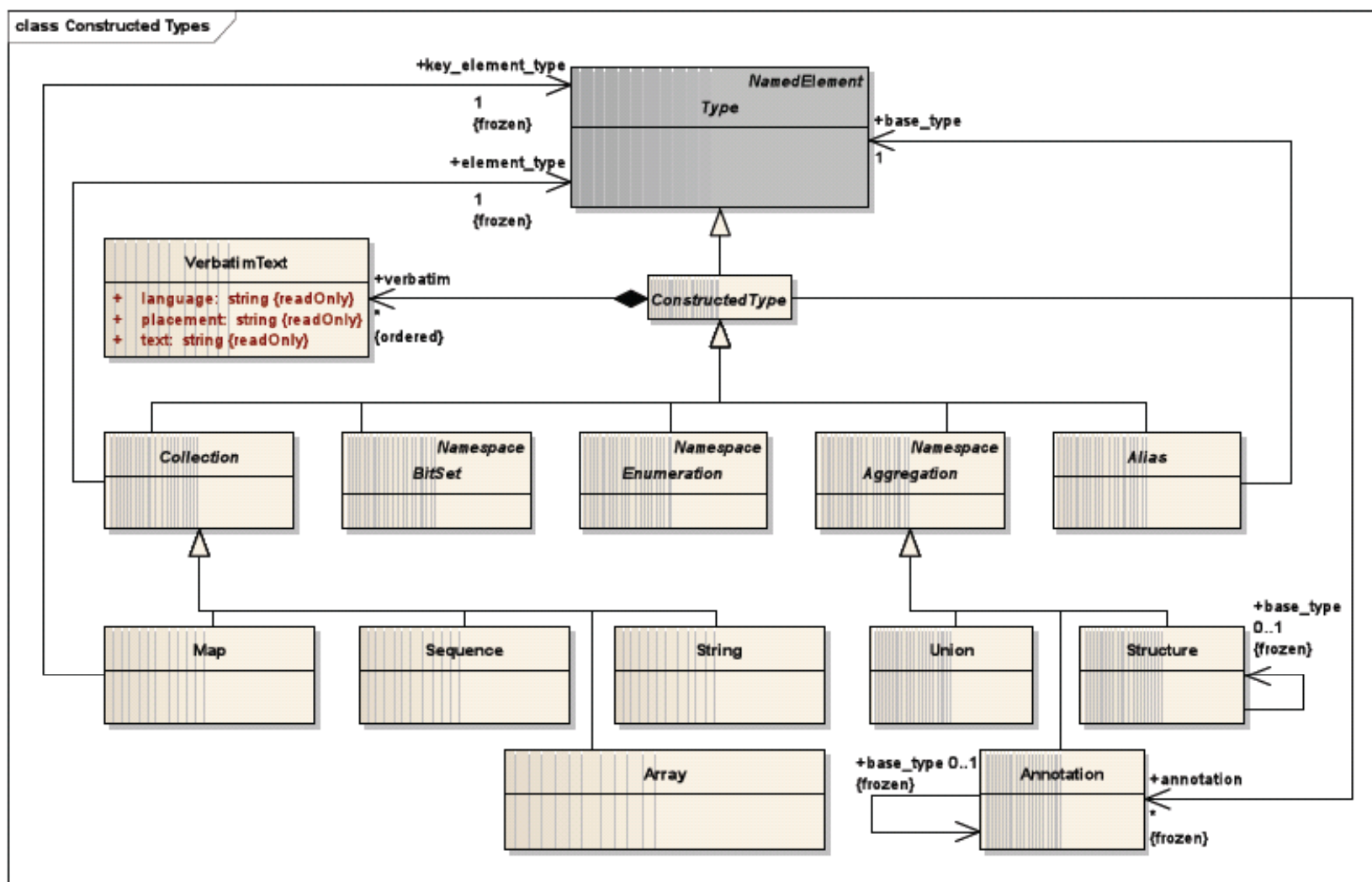
DDS Extensible Dynamic Topic Types

- **Type system:**
 - primitive types, fixed-sized arrays, variable-size sequences, structures
 - type versioning and evolution
 - sparse types
- **External type representation:**
 - IDL
 - XSD, XML
 - TypeObject (binary)
- **Payload:**
 - Common Data Representation (CDR)
 - Parameterized CDR that supports evolution
- **Language binding:**
 - **Plain language binding:** Equivalent to the type definitions generated by existing standard IDL-to-programming language mappings.
 - **Dynamic language binding:** is represented by the **DynamicType** and **DynamicData** interfaces.

Type System Model – 1 of 2



Type System Model – 2 of 2



DynamicData interface

The current version:

Pros: Single universal and efficient interface for all data types

Cons: 270+ methods

Our compact interface presented at the OMG technical meeting, 20-23 Sept. 2010:

- 20 setters and getters for primitive data types
- 2 methods for collections
- 2 methods for complex types
- 4-5 methods for introspection, etc.

Next Step

Open-source implementation of the Extensible Dynamic Topic Types co-owned by EPICS, RTI, and PrismTech



The Global Leader in DDS

