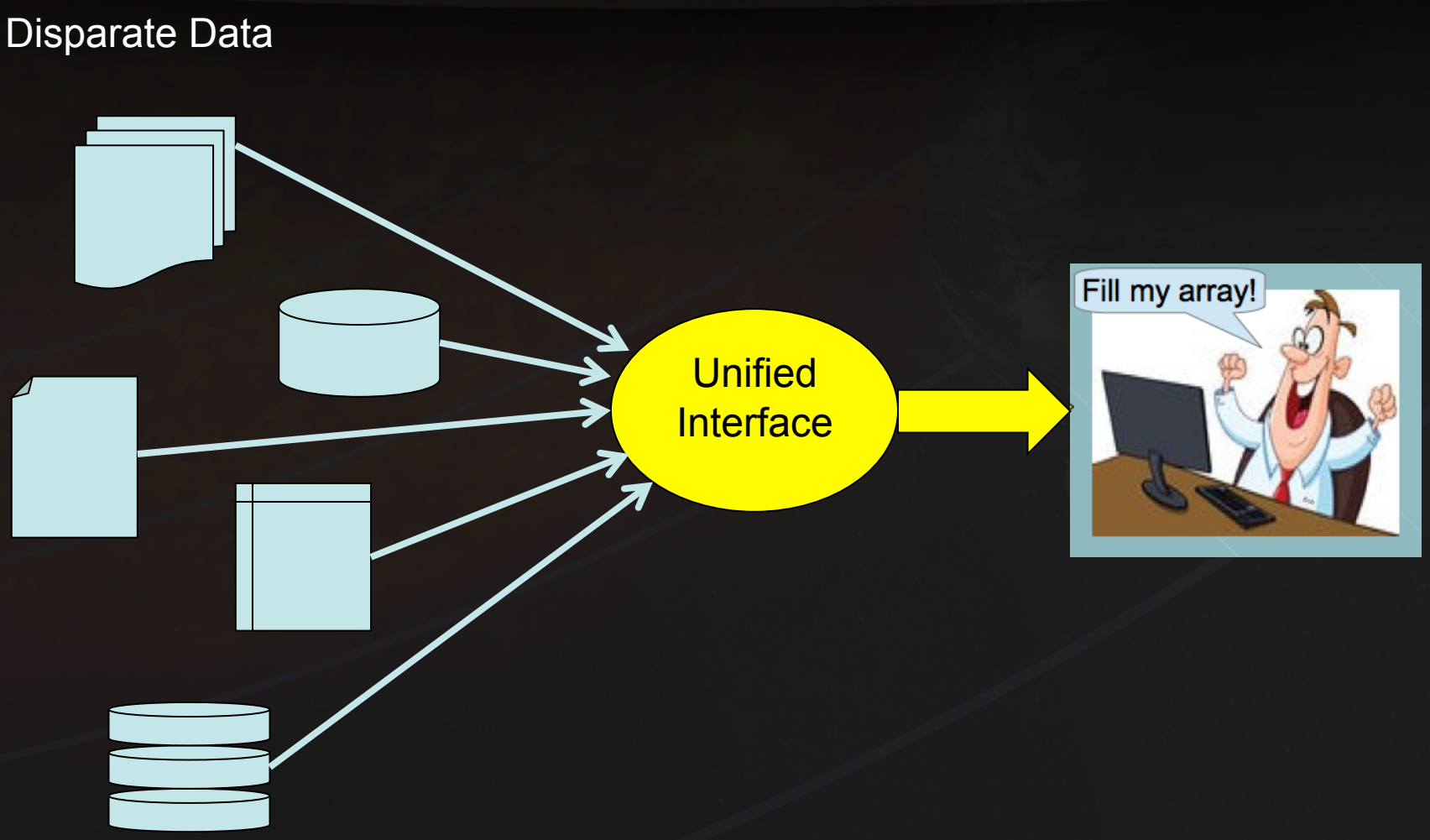
LaTiS
Data Access Service Architecture

Doug Lindholm
Laboratory for Atmospheric and Space Physics
University of Colorado Boulder
UCAR SEA – April 2013

Outline

- Motivation
- LaTiS Data Model
- Scala API
- Service Architecture
- Examples
- Conclusions

Motivation



LaTiS Unified Data Model

- Represent the Functional Relationships inherent in scientific data.
- Common underlying mathematical model for all data models.

Independent Variable (domain) → Dependent Variables (range)

Example: time series of gridded surface winds

```
Time -> ((Lon, Lat) -> (U,V))
```

LaTiS Data Model

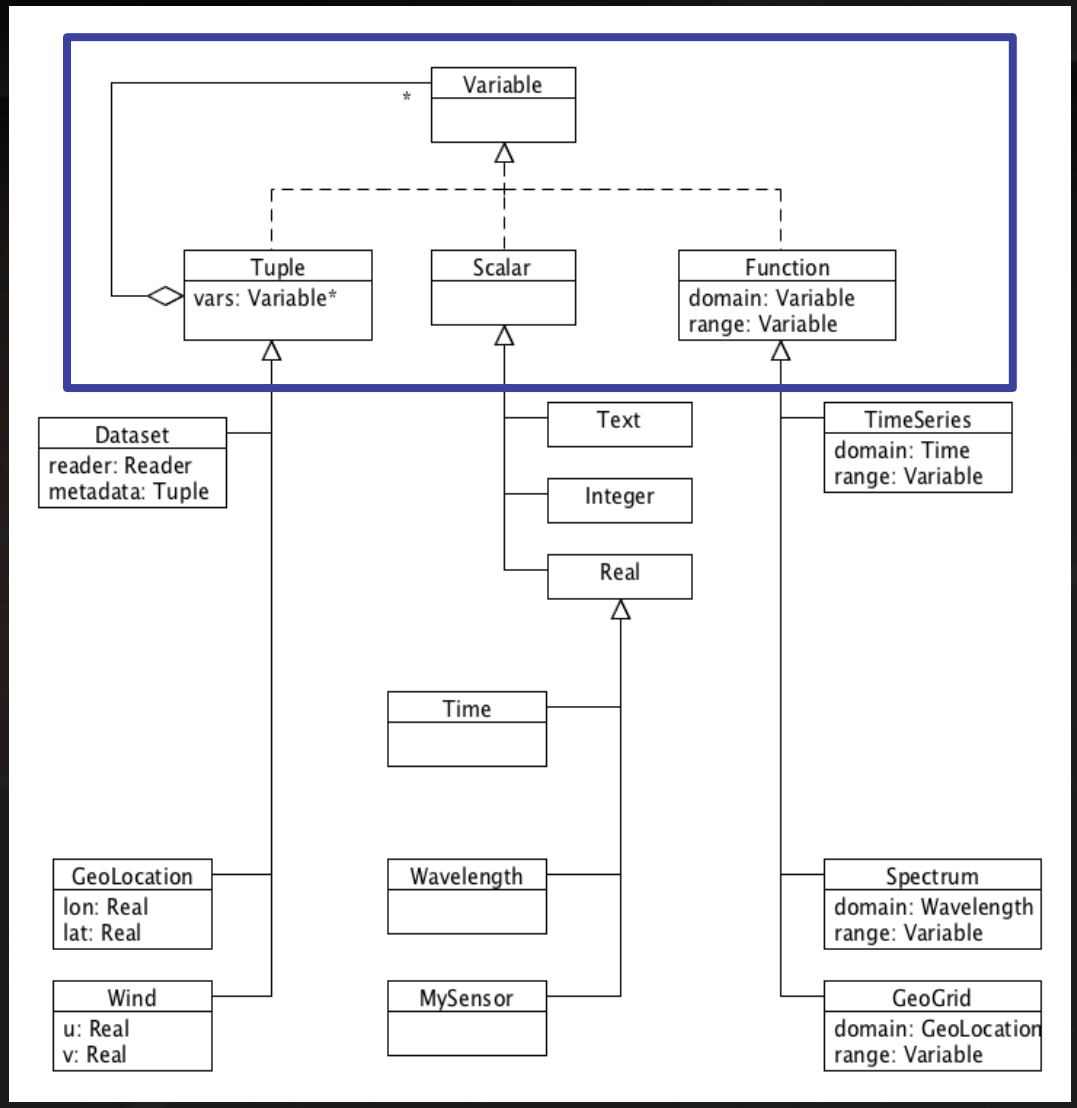
Only Three Variable Types:

Scalar: single Variable

Tuple: group of Variables

Function: mapping from one Variable to another

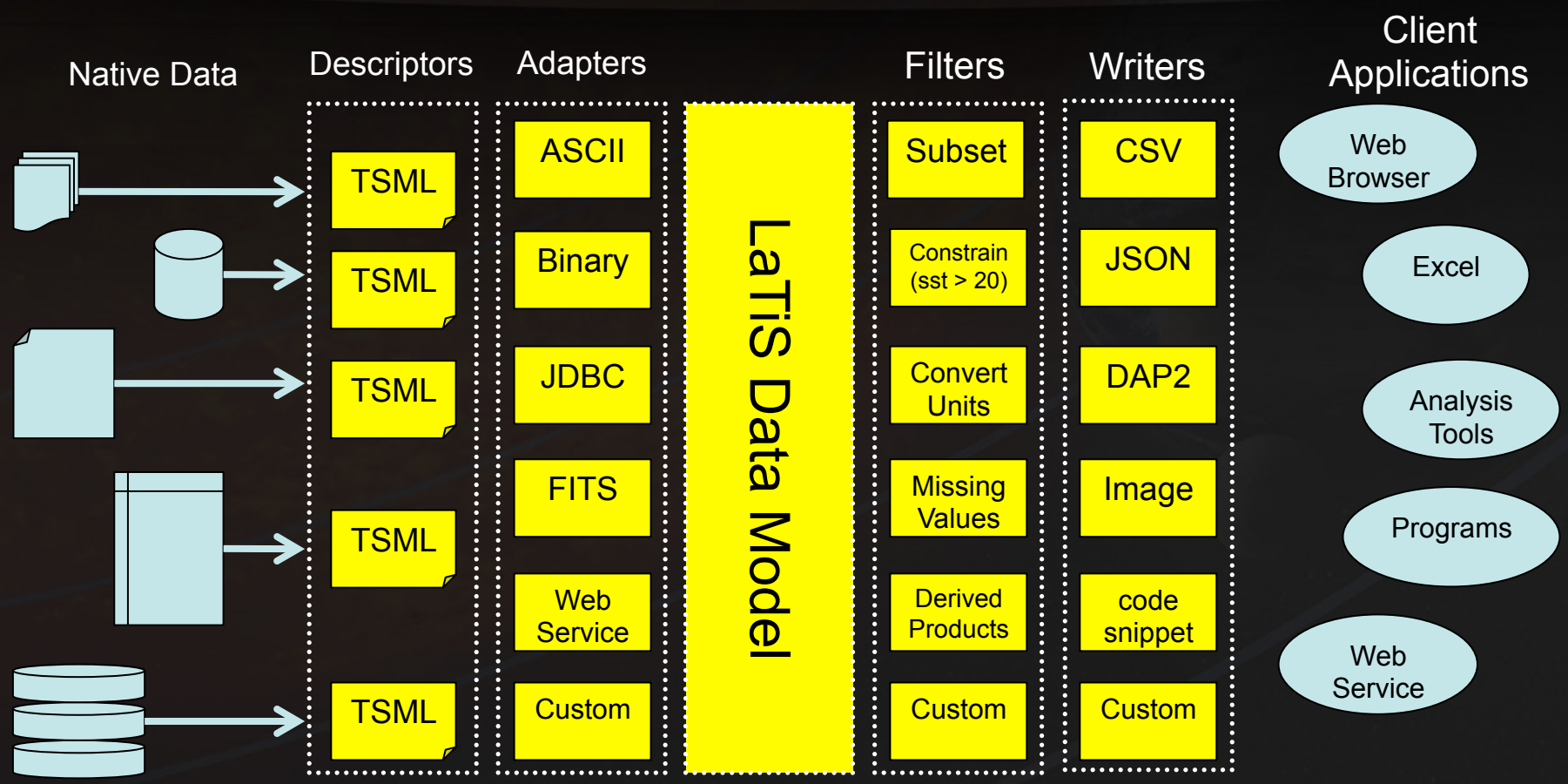
Extend to capture higher level, domain specific abstractions



Scala Implementation

- Dataset as a Scala collection
- Functional Programming Paradigms:
  - Function composition over object manipulation
  - Functions as first class citizens
    - a LaTiS Function can be used like a programming function
  - Immutable data structures
  - No side-effects: parallelizable, provable
  - Lazy evaluation: scalable
- Math and resampling mixed in
  - e.g. `dataset3 = (dataset1 + dataset2) / 2`
- Metadata encapsulated
  - enforce data consistency: unit conversions ...
  - track provenance

LaTiS Server Architecture



LaTiS Server Implementation

- RESTful web service API (OPeNDAP +)
- Java Servlet, build and deploy war file
- XML dataset descriptor (TSML) for each dataset
  - Specify Adapter to use
  - Map native data source to LaTiS data model
  - Define transformations as Processing Instructions
- Catalog to map dataset names to TSML
- Plugins: implement the Adapter, Filter or Writer interfaces or extend existing ones
- Properties file to map filter and writer names to implementing classes

Examples

- Mapping an ASCII file with TSML
- Add a new Writer
  - Include provenance metadata in output
- Add a new Filter
  - Math with Datasets
  - Aggregation
  - Provenance tracking

Example – Serving an ASCII File

Sunspot data
for October 2003

```
2003 10 01   75
2003 10 02   72
2003 10 03   59
2003 10 04   60
2003 10 05   53
2003 10 06   51
2003 10 07   50
2003 10 08   56
2003 10 09   58
2003 10 10   50
2003 10 11   44
2003 10 12   22
2003 10 13   12
2003 10 14    4
2003 10 15   17
2003 10 16   24
2003 10 17   37
2003 10 18   43
2003 10 19   43
2003 10 20   64
2003 10 21   66
2003 10 22   72
2003 10 23   68
2003 10 24   81
2003 10 25   89
2003 10 26  102
2003 10 27  141
2003 10 28  161
2003 10 29  167
2003 10 30  171
2003 10 31  156
```

TSML Dataset descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<tsml>

  <dataset name="Sunspot_Number"
           history="Read by LaTiS">

    <adapter class="latis.reader.tsml.DelimitedAsciiAdapter"
             url="file:/data/latis/ssn.txt" />

    <time units="yyyy MM dd"/>

    <scalar name="ssn" type="Integer"/>

  </dataset>

</tsml>
```

Example – Serving an ASCII File

http://localhost:8080/latis/ssn.csv

http://localhost:8080/latis/ssn.csv

```
1.0649664E12,75
1.0650528E12,72
1.0651392E12,59
1.0652256E12,60
1.065312E12,53
1.0653984E12,51
1.0654848E12,50
1.0655712E12,56
1.0656576E12,58
1.065744E12,50
1.0658304E12,44
1.0659168E12,22
1.0660032E12,12
1.0660896E12,4
1.066176E12,17
1.0662624E12,24
1.0663488E12,37
1.0664352E12,43
1.0665216E12,43
1.066608E12,64
1.0666944E12,66
1.0667808E12,72
1.0668672E12,68
1.0669536E12,81
1.06704E12,89
1.0671264E12,102
1.0672128E12,141
1.0672992E12,161
1.0673856E12,167
1.067472E12,171
1.0675584E12,156
```

Example – Custom Writer

Extend the CsvWriter to add header with provenance information

```
class MyWriter(out: OutputStream) extends CsvWriter(out) {

  override def write(dataset: Dataset) = {
    writeHeader(dataset)
    super.write(dataset)
  }

  def writeHeader(dataset: Dataset) {
    val writer = new PrintWriter(out)

    writer.println("#--- Provenance ---")
    writer.println(dataset.metadata("history"))
    writer.println("#------------------")
    writer.flush()
  }
}
```

Add mapping from the “mine” suffix to this Writer (latis.properties)

```
# Define Writers
csv.writer.class = latis.writer.CsvWriter
mine.writer.class = latis.writer.MyWriter
```

Example – Custom Writer

http://localhost:8080/latis/ssn.mine

http://localhost:8080/latis/ssn.mine

```
#--- Provenance ---
Read by LaTiS
#------------------
1.0649664E12,75
1.0650528E12,72
1.0651392E12,59
1.0652256E12,60
1.065312E12,53
1.0653984E12,51
1.0654848E12,50
1.0655712E12,56
1.0656576E12,58
1.065744E12,50
1.0658304E12,44
1.0659168E12,22
1.0660032E12,12
1.0660896E12,4
1.066176E12,17
1.0662624E12,24
1.0663488E12,37
1.0664352E12,43
1.0665216E12,43
1.066608E12,64
1.0666944E12,66
1.0667808E12,72
1.0668672E12,68
1.0669536E12,81
1.06704E12,89
1.0671264E12,102
1.0672128E12,141
1.0672992E12,161
1.0673856E12,167
1.067472E12,171
1.0675584E12,156
```

Example – Custom Filter

Extend the Filter interface to derive a new Dataset

```
class MyFilter extends Filter {

  override def filter(dataset: Dataset): Dataset = {
    //Add the dataset to itself
    val dataset2 = (dataset + dataset).asInstanceOf[Dataset]

    //Join the original and new dataset into a single dataset.
    //Note, this will add provenance to the Dataset's metadata.
    JoinAggregator.aggregate(dataset, dataset2)
  }
}
```

Add mapping from the "myfilter()" call to this Filter (latis.properties)

```
# Define Filters
myfilter.filter.class = latis.filter.MyFilter
```

Example – Custom Filter

http://localhost:8080/latis/ssn.mine?&myfilter()

http://localhost:8080/latis/ssn.mine?&myfilter()

```
#--- Provenance ---
Read by LaTiS
Wed Apr 03 11:49:14 MDT 2013: Joined datasets with JoinAggregator version 1.0.2
#------------------
1.0649664E12,75,150
1.0650528E12,72,144
1.0651392E12,59,118
1.0652256E12,60,120
1.065312E12,53,106
1.0653984E12,51,102
1.0654848E12,50,100
1.0655712E12,56,112
1.0656576E12,58,116
1.065744E12,50,100
1.0658304E12,44,88
1.0659168E12,22,44
1.0660032E12,12,24
1.0660896E12,4,8
1.066176E12,17,34
1.0662624E12,24,48
1.0663488E12,37,74
1.0664352E12,43,86
1.0665216E12,43,86
1.066608E12,64,128
1.0666944E12,66,132
1.0667808E12,72,144
1.0668672E12,68,136
1.0669536E12,81,162
1.06704E12,89,178
1.0671264E12,102,204
1.0672128E12,141,282
1.0672992E12,161,322
1.0673856E12,167,334
1.067472E12,171,342
1.0675584E12,156,312
```

Conclusions

A functional data model implemented in a Functional Programming language can be quite ... functional.

Open Source project on GitHub:

https://github.com/dlindhol/LaTiS