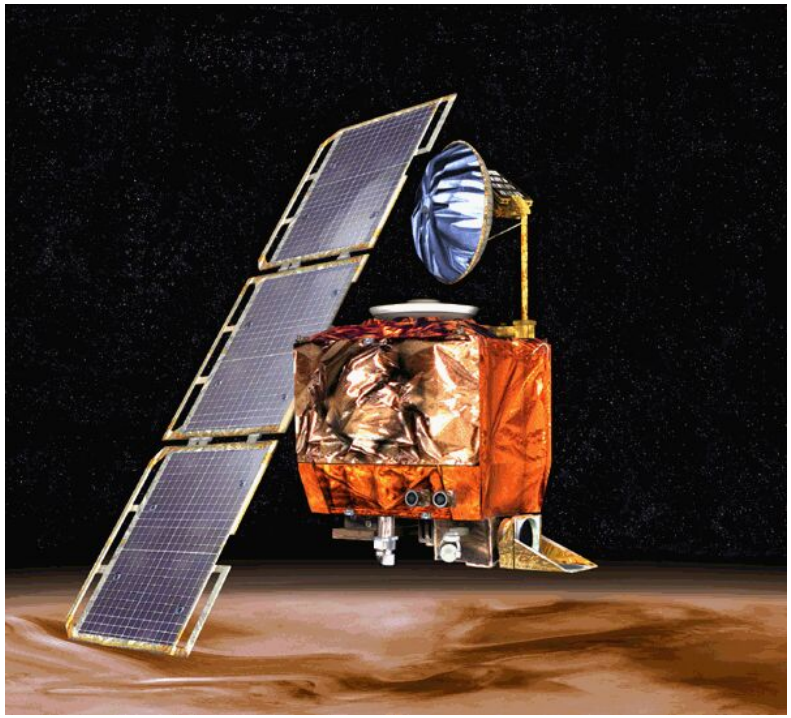
Expressing Measurement Units in Interfaces for Scientific Component Software

Kosta Damevski

Department of Mathematics and Computer Science
Virginia State University

November 16, 2009

The 1999 NASA Mars Orbiter Crash



"The 'root cause' of the loss of the spacecraft was the failed translation of English units into metric units in a segment of ground-based, navigation-related mission software, as NASA has previously announced," said Arthur Stephenson, chairman of the Mars Climate Orbiter Mission Failure Investigation Board

Two Separate Teams Were Involved

Mars Climate Orbiter Mishap Report

Specifically, thruster performance data in English units instead of metric units was used in the software application code titled SM_FORCES (small forces). A file called Angular Momentum Desaturation (AMD) contained the output data from the SM_FORCES software. The data in the AMD file was required to be in metric units per existing software interface documentation, and the trajectory modelers assumed the data was provided in metric units per the requirements.

In This Talk...

- Convince you that dimensional unit (e.g. meters, seconds, etc.) information should be expressed at the component interface level
- Select a set of suitable requirements for a component-based unit system
- Describe one implementation of a unit system (using Babel and CCA)

Dimensional Units in Computational Science

- Unit information is left out of most programs
  - Dimensional mismatch errors are left up to the programmer
- Some programs use language-level extensions to express unit information
  - A few tools exist (e.g. SIUnits(FNAL), UDUnits(UCAR), and others)
  - Can be tedious to the programmer and may significantly affect performance

Dimensional Units in Computational Science

- Unit information is left out of most programs
  - Dimensional mismatch errors are left up to the programmer
- Some programs use language-level extensions to express unit information
  - A few tools exist (e.g. SIUnits(FNAL), UDUnits(UCAR), and others)
  - Can be tedious to the programmer and may significantly affect performance

Hypothesis

Unit mismatch errors typically occur between two teams. Therefore it sense for them to be handled at the component interface level.

Units in Component Interfaces

Strengths:

- Likely less of a performance overhead than language extensions
- Easier for a programmer to express language extensions
- Likely would have caught the Mars Orbiter bug

Weaknesses:

- Vulnerable to intra-component dimensional mismatch

Objectives

1. Clearly express measurement unit information in interfaces

Objectives

1. Clearly express measurement unit information in interfaces
2. Components that use a particular interface (port) should be able to express unit information programatically
   - To a location service or directly to the providing component

Objectives

1. Clearly express measurement unit information in interfaces
2. Components that use a particular interface (port) should be able to express unit information programatically
   - To a location service or directly to the providing component
3. Automatically convert commensurate quantities (e.g. miles into meters)

Objectives

1. Clearly express measurement unit information in interfaces
2. Components that use a particular interface (port) should be able to express unit information programatically
   - To a location service or directly to the providing component
3. Automatically convert commensurate quantities (e.g. miles into meters)
4. Provide a convenient way to express derived units (such as meters per second) and to extend the system with new units.

Complexity of Representing Units

Le Systeme International D'Unites a.k.a. SIUnits is an international standard that describes dimensions and their measurement units

- Seven mutually independent dimensions
  - mass, length, time, electric current, thermodynamic temperature, amount of substance, luminous intensity
- Each dimension has a corresponding base unit
  - kilogram, meter, second, ampere, kelvin, mole, and candela
- Twenty-three derived units, which may span several dimensions
  - newton (kilogram meter per seconds squared) (force)
- Factors on existing units may have different names
  - 10^{-3} kilograms is a gram
- Non-SI unit systems

Difficult to determine compatibility (comenssurability) between units

Determining Unit Commensurability

Approach used by SIUnits (FNAL)

- Implement units as C++ templates
- Interesting approach to commensurability
  - Encode all units with a numerical value per each SI dimension

unit	mass	length	time	substance	current	thermodynamic temp.	luminous intensity
meter	0	1	0	0	0	0	0
candela	0	0	0	0	0	0	1
$meter / second^2$	0	1	-2	0	0	0	0

Table: Encoding units according to their seven SI base dimensions.

Borrow this approach in our design

Unit Conversion

We still need a way to convert quantities between two commensurable units...

Approach used by Allen et al.

- Units as an extension to Java
- Convert between units by using a primary unit as a intermediate value
  - Part of the *Unit* base class

Again, borrow this approach in our design

Simple Approach

A simple class per Unit model

Unit Definitions (in SIDL)

```
interface Unit {
  void setQuantity(in double q);
  double getQuantity();
  double inPrimaryUnit();
  array<int,1> getDimensions();
  bool isCompatible(in Unit impl);
}

class Mile implements-all Unit {}
class Second implements-all Unit {}
class Kilometer implements-all Unit {}
...
```

Simple Approach (continued)

Example SIDL

```
class Earth {
  void setCircumference(in Mile m);
}
```

Example Call (using Kilometers)

```
Earth earth = Earth::_create();
KiloMeter circumference = Kilometer::_create();
circumference.setQuantity(40041);
earth.setCircumference(circumference);
```

Simple Approach (continued)

Example SIDL

```
class Earth {
  void setCircumference(in Mile m);
}
```

Example Call (using Kilometers)

```
Earth earth = Earth::_create();
KiloMeter circumference = Kilometer::_create();
circumference.setQuantity(40041);
earth.setCircumference(circumference);
```

- The above would not be possible with Babel since Kilometer and Mile are separate classes

General Approach

Generalize application interfaces so that we can pass multiple units.

Example SIDL

```
class Earth {
  void setCircumference(in Unit m);
}
```

Example Call (using Kilometers)

```
Earth earth = Earth::_create();
KiloMeter circumference = Kilometer::_create();
circumference.setQuantity(40041);
earth.setCircumference(circumference);
```

General Approach (continued)

Example SIDL

```
class Earth {
  void setCircumference(in Unit m);
}
```

Example Implementation

```
void
Earth::setCircumference(const Unit& u) {
  if ( ! u.isCompatible(Unit::Mile::_create()) ) {
    //ERROR: incompatible units
  }
  ...
}
```

- User has to implement an explicit unit mismatch check. Also, interface readability is diminished

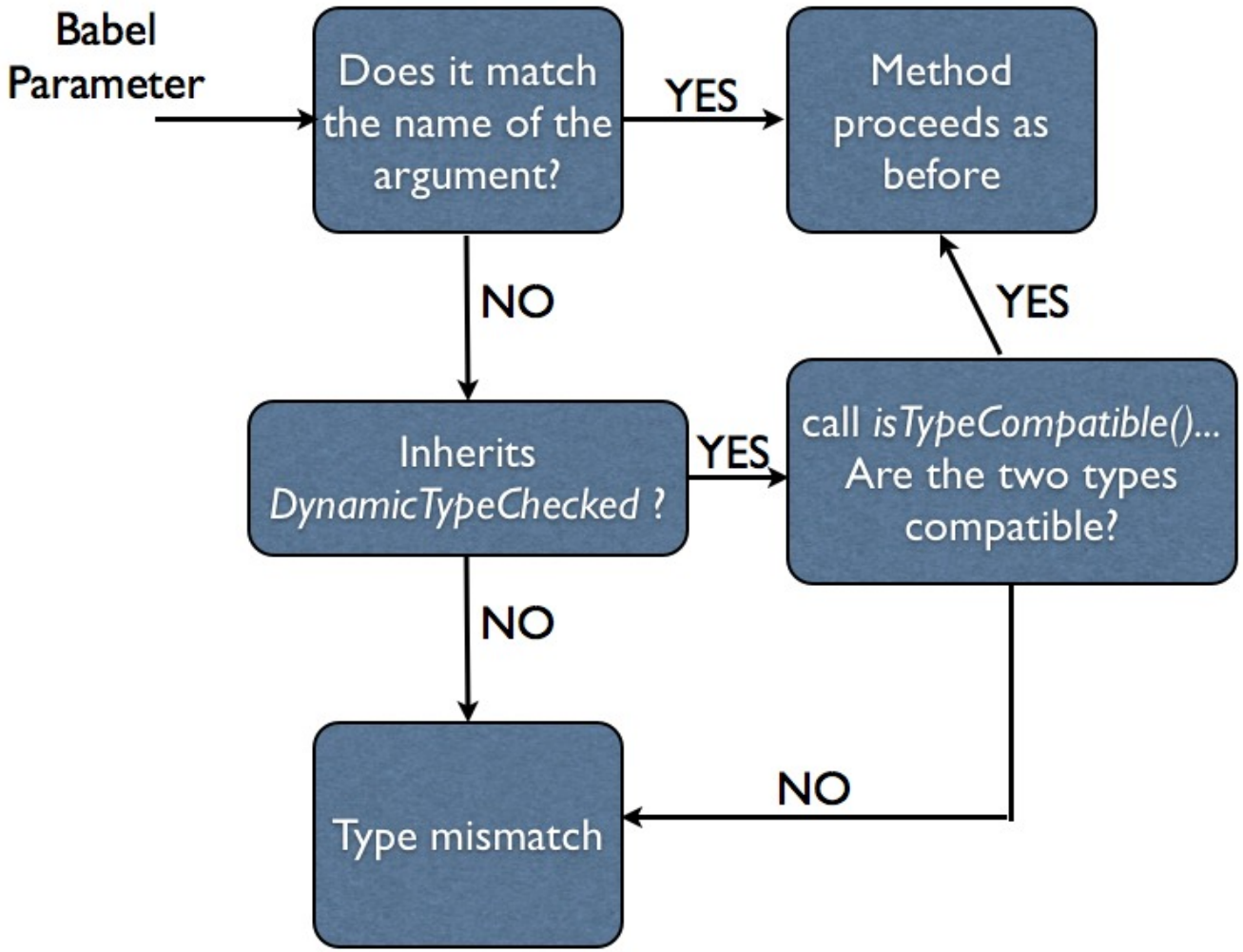
Our Approach: Add Dynamic Type Checking to Babel

- Define another interface in Babel's runtime called *sidl.DynamicTypeChecked*

```
package sidl {
  interface DynamicTypeChecked {
    bool isTypeCompatible(in BaseClass class,
                          out BaseClass newClass);
  }
}
```

- To determine type compatibility, Babel invokes the *isTypeCompatible* method in classes implementing this interface

Parameter (with Dynamic Type Checking) Flowchart



Units with Dynamic Type Checking

```
abstract class Unit implements
                    sidl.DynamicTypeChecked {

  bool isTypeCompatible(in BaseClass class,
                    out BaseClass newClass);
  void setQuantity(in double q);
  double getQuantity();
  abstract array<int,1> getDimensions();
}

class Meter extends Unit {}
class Second extends Unit {}
...
```

One implementation of *isTypeCompatible* is enough for all Units.

Summary

- Expressing dimensional units at the component interface level is a reasonable choice
- Ability to provide the capacity to automatically convert between commensurable units is a convenient addition
- Implement units by introducing dynamic type checking to Babel

Future Work

- This approach to units in component interfaces requires a change in the application code
  - Though minimal, this probably would impede acceptance of units
- Investigate ways of catching dimensional mismatch errors based solely on interface information
  - May be able to produce a warning when two incompatible but commensurable units are encountered in the same component

Acknowledgements

- CCA Forum (http://www.cca-forum.org)
- Babel Team (http://computation.llnl.gov/casc/components/babel.html)

QUESTIONS?