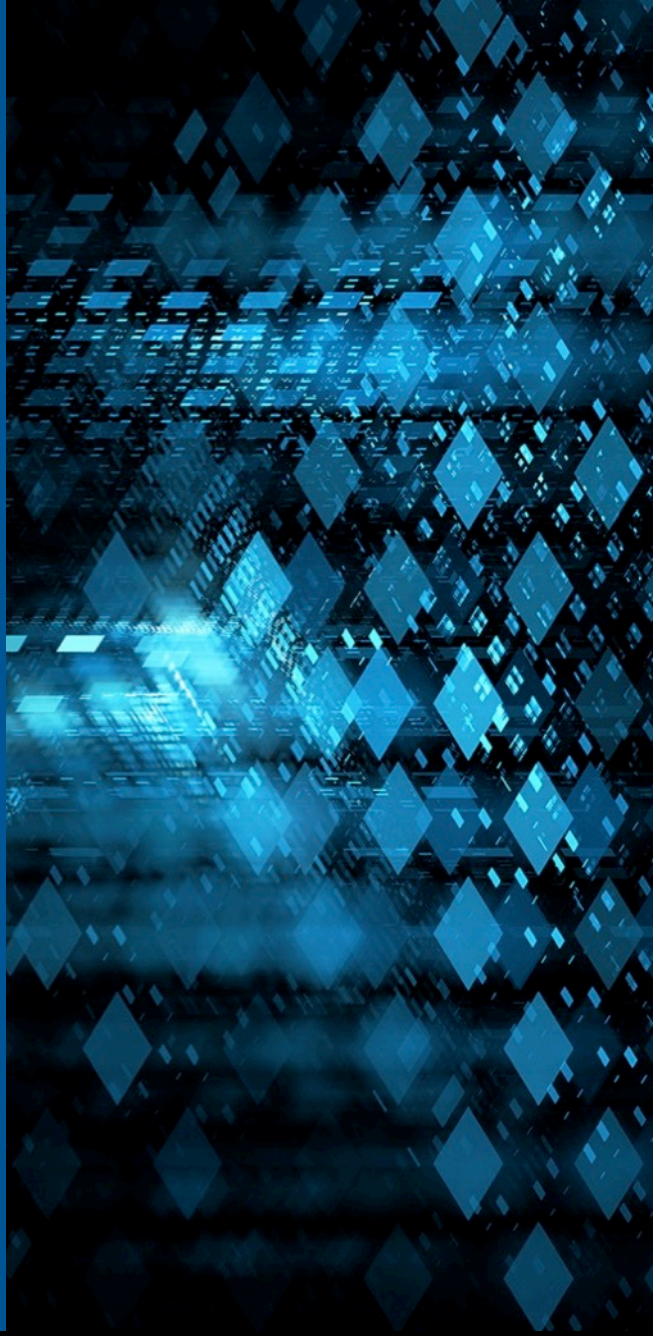
Type System Unification

Peter Feiler

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213



This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM18-0078

Type System Unification

Unification of type systems and expression languages (Peter, Lutz*, Alexey, Brian, Serban)

- Alisa ReqSpec et.al.: types, assign once variables, computed variables., property types, Resolute types, Java type mapping
- Property language V3
- Constraint language
- BLESS
- Data Model annex
- Resolute, Scripting languages (Python, Ease)

Type System Unification

Types

- Data types, property types, constraint language variable types
  - Property types available as data types
  - Data types available as property types
- Base types: integer, real, string, Boolean
  - No more **aadlinteger** keyword
- Handling of units: part of value, association via property
  - Integration of proposed Units system (ISO, SysML)
  - Unit assumptions vs. units passed as part of value
- Sequences & sets: Set with unique element semantics
- Union of types
- Type conversion: explicit casting and implicit for numerics
  - Real without .0 is accepted
  - Numeric and numeric range
- Types like time: when to use integer vs. real
- Support for type inference from value? Require type

Type System Unification Approach

Common type system available for use as data types, property types, annex sublanguage types

- Types can have properties

Base types

- Numeric, Boolean, string, enumeration, units

User defined types

- Int16: **type** Integer { Data_Size => 16 Bits};
- Temperature: **type** real **units** Celsius;
- Speed: **type** integer [0 .. 200 kph] **units** SpeedUnits;

Composite types

- Unions and aggregates
- Aggregates: records, arrays, sets, multiset (bag), list(sequences), map, graph
- Personel_Record: **record** (first: string; last: Address;);

Provide subtyping

- Same as subclassing
- Equivalent of type extension for data types

Type System Usage

Port types

- P1: **in port** Temperature;

Data components

- DataObject: **data** Personel_Record;
  - Subtype substitution/match (Type_Extension)

Properties

- Property definitions reference types

Data Annex

- Characterization via properties vs. partial specification
- **Data** personel_record { Data_Representation => Struct; };
- Personel_Record: **type record** () { Source_Name => PersonnelRecord;};

- Personel_Record: **refined to type record** (first: string; last : string;);

Representation of Transferred Data

Example

- BodyTemperature: **type** integer [30..50 C] **units** TemperatureUnits;
- P1: **out port** BodyTemperature;

Is unit included in transferred data or is a unit assumed?

Non-zero reference point for transferred value

Transfer representation may be different from in memory representation

Alternatives:

- Protocol specification
  - As virtual bus
  - Mapping into bit representation (see 429 protocol example in SAVI demo)
- Assumed unit as property on port

Representation of Types

Example

- BodyTemperature: **type** integer [30..50 C] **units** TemperatureUnits;
- P1: **out port** BodyTemperature;

Digital representation

- Base_Type property in Data_Model
- Associated with type or with port

Physical representation

- Dynamic behavior
- Specified as part of type or specific to each use site
  - Associated with feature