

# Type System Unification

Peter Feiler

Software Engineering Institute  
Carnegie Mellon University  
Pittsburgh, PA 15213

Copyright 2018 Carnegie Mellon University. All Rights Reserved.

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.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at [permission@sei.cmu.edu](mailto:permission@sei.cmu.edu).

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

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