

Attribute Description Language in the Bio-Networking Platform

Jun Suzuki, Khanh Huynh and Tatsuya Suda
{jsuzuki, suda}@ics.uci.edu
khanhph@uci.edu
<http://netresearch.ics.uci.edu/bionet/>
Dept. of Information and Computer Science
University of California, Irvine

1

Overview

- Introduction
- Language specification
- Evaluation of attribute expression

Introduction

Attribute Description Language

- The Bionet Attribute Description Language is
 - a simple and generic language to specify attributes formatted in name-value pairs.
 - used to define
 - CE's attributes
 - search criteria in discovery query message
 - c.f. documentation of the bionet social networking service
 - planned to be used to define reconfiguration constraints in the bionet reconfiguration engine in the near future.

CE's Attributes

A CE attribute is

- expressed as a pair of *name* and *value*.
 - *GUID==0101*
 - *affiliation=='UCI'*
 - *name=='Suzuki'*
- Each CE can have multiple pairs of name and value.
 - like a table structure
- encoded as a string data.
 - transferred through the IDL string type in the network
 - maintained as Java String type in Java objects

5

Discovery Query Criteria

- A discovery query criterion is
 - a *Boolean expression* over name-value pairs.
 - *GUID==0101*
 - *affiliation=='UCI' and name=='Suzuki'*
 - *serviceName=='hotelReservation' and serviceCost<10.0*
 - encoded as a string data.
 - transferred through the IDL string type in the network
 - maintained as Java String type in Java objects

Language Specification

Language Elements and Syntax

- General syntax
 - *<operand> <operator> <operand>*
 - binary operator used
 - *<operator> <operand>*
 - unary operator used
- Operands are either *attribute names* or *literals*.
 - e.g. *GUID==0101*
 - GUID is an attribute name, and 0101 is an integer literal.
 - e.g. *affiliation=='UCI'*
 - Affiliation is an attribute name, and 'UCI' is a character literal.

Literals

4 types of literals

- Integer literals
 - e.g. 10, 100, -5
- Floating-point literals
 - e.g. 10.0, 100.0, -5.0
- Boolean literals
 - true or false
- Character literals
 - e.g. 'a', 'abc'
 - A sequence of character literals is called a string literal.

9

Operators

• Comparison operators (binary operator)

- ==, !=, <, >, <=, >=
- can be applied to integer, float, character, string and boolean values
 - Same meaning as in C++/Java
 - For string and character values, comparisons use the ISO Latin-collating sequence.
 - For Boolean comparisons, *true* is greater than *false*.

Arithmetic operators (binary operator)

- +, -, *, /
- can be applied to integer and float values.
 - not applied to character and string values.
- Mixed-mode arithmetic operation supported.
 - e.g. 1 + 0.5
 - type promotion performed from integer to float.
- - can be a unary operator.
 - e.g. -(1+2)

Boolean operators

- and (binary op), or (binary op), not (unary op)
- can be applied to any Boolean statements

• existence test operator (unary operator)

- *exist* <attribute name>
 - returns *true* if a specified attribute name is defined in a given CE.
 - e.g. *exist serviceCost*
 - » testing if a given CE contains an attribute named *serviceCost*
 - e.g. *not exist qualityOfService*
 - » testing if a given CE does not contain an attribute named *qualityOfService*.

• Substring matching operator (binary operator)

- <substring> ~ <string>
 - returns *true* if <substring> appears as a substring of <string>.
 - e.g. 'google' ~ 'googlegoogle'

Operator Precedence

Precedence from higher to lower

- exist, - (unary minus)
- not
- *, /
- +, -
- ~
- ==, !=, <, >, <=, >=
- and
- or

13

Evaluation of Attribute Expression

Evaluation of Expression

The expression evaluator accepts an attribute expression(s) and always returns *true* or *false* for the accepted expression(s).

Each bionet platform maintains an evaluator.

Evaluation is performed locally on each bionet platform.

- No remote evaluation supported.

Evaluation Steps

- Evaluation steps
 - (1) Parse an expression to identify and tokenize operators, literals and attribute names.
 - (2) Check if operators and operands (literals and attribute names) are placed at right places.
 - (3) Construct an expression tree.
 - (4) Evaluate and return a Boolean result

(1) Parse an expression to identify and tokenize operators, literals and attribute names.

- e.g. *affiliation*==*'UCI'* is tokenized into 3 tokens:

- *affiliation* (IDENT)
- == (EQUAL)
- *'UCI'* (STRING_LITERAL)

– Instantiate the classes corresponding to the token types.

17

- (2) Check if operators and operands (literals and attribute names) are placed at right places.

– check if all the operators are applied to valid literals

- *affiliation*==*'UCI'*

– check if there exists attribute names in a given expression.

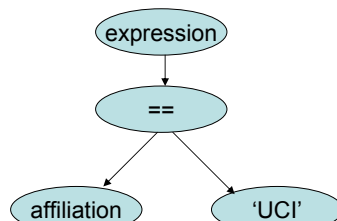
- e.g. *affiliation*==*'UCI'*

• If yes, all the identifiers should be defined in an attribute table in a CE.

(3) Construct an expression tree

– Construct a tree with the tokenized objects created at the step (1)

- Literals should be leaf nodes in the expression tree.
- Operators should be root nodes in the expression tree.



- (4) Evaluate and return boolean result

– access an attribute table in a given CE.

– check if constraint is true or false, and

– return true or false

See also...

See also the documentation about attribute-enabled CEs

- Class NVListEnabledCyberEntity