

# **Relational Association Paradigm**

Software Programming Paradigm Specification Proposal

*Second Draft Edition, First Revision*

© 2015 Aaron Sami Abassi  
Licensed under the Academic Free License version 3.0

## Introduction

This document is intended to specify compliance with the relational association paradigm as a software programming paradigm. The relational association paradigm is an extension of the procedural and imperative programming paradigms, incorporating inter-dependent (relational) modular (association) software definition spaces (source code) and call by function reference table (objectification) based procedural invocation for large scale, high performance software systems programming.

## Related Materials

Included with this document are two source code examples implemented in 5 language versions. The first example demonstrates fractionalization in the Tcl scripting language, Visual Basic, C, C++98 and C++14 which are all relational association paradigm capable languages. The second demonstrates generic trajectory through spaces in both C++98 and C++14.

## Definitions

Element	Page Number
<b>Association</b>	3
<b>Relation</b>	4
<b>Conformation</b>	4
<b>Information</b>	4
<b>Abstraction</b>	5
<b>Function</b>	5-6
<b>Situation</b>	7
<b>Location</b>	8
<b>Classification</b>	8
<b>Objectification</b>	9

## Association

- Is an associative programmatic scope of reference
- Defines conformation, abstraction and classification type elements
  - May define categorical situation type elements
- Tables information, function and objectification elements by identifier
  - May table categorical location definitions by identifier
- Is available within during the definition of the associative elements

## Relation

- Is an elementary directional relation of an associative element being referenced in any scope of reference of another association
- The related element may be addressed in the receiving associative source definition either by explicit specification of the associative relation or by implicit scope of reference resolution

## Conformation

- Identifies an information type
- Is either elementary or composite
  - Member elements each identify an information type
- Includes both language provided types and user defined types
- Not required in typeless languages
- Conformation Template
  - Term must be used when the identifier identifies a template

## Information

- Identifies an addressable unit of information
- May be treated as constant or variable
- May be supported by language provided operations (operators, etc.)
- Information Template
  - Term must be used when the identifier identifies a template

## Abstraction

- Identifies a function call interface
- Defines the expected result and parametric types of the call interface
- May also define the parametric form
  - Any default argument values
  - Declaration of support for a variable number of arguments
- Not required in typeless languages
  - Call parameter/return semantics are matched instead
- Abstraction Template
  - Term must be used when the identifier identifies a template

## Function

- Identifies an addressable unit of function
- May define a set of parametric arguments
  - Parametric types where required are drawn from the function's association and relations
- May return a result
  - Return types where required are drawn from the function's association and relations
- May contain a table of information and objectification used by all invocations of the function
  - Types where required are drawn from the function's association and relations
- May contain a table of (nested) function locals
  - Outer function's locals are available to nested functions
- Contains a table of information and objectification temporary locals used exclusively by each invocation of the function
  - Types where required are drawn from the function's association and relations
- Defines sequential programmatic instructions required to complete the intended procedure
- Must comply with structured programming execution flow control
  - No direct programmatic control of statement execution
  - Defines a sequence of programmatic statements
  - The statements are followed sequentially except during conditional jumps over statements
  - Strict constructs for any forward jumps through the sequence of statements
    - A subset sequence of the statements are followed or skipped based on a specified conditional evaluation
  - Strict constructs for any reverse jumps back through the sequence of statements
    - Looping conditions must be enunciated in the source code
    - A subset sequence of the statements are followed in a loop based on a specified conditional evaluation

## Function (Continued)

- Must comply with procedural programming
  - Procedural invocations through tabled function identifier
    - Calls are made to a function tabled in the calling function's association
    - Calls are made to a function tabled in another association by relation to the calling function or by relation to the calling function's association
    - Calls may be made to a (nested) function tabled in the calling function's locals
- Must comply with procedural objectification programming
  - Procedural invocations through objectification of function by location
    - Calls are made to a function by location in an objectification tabled in the calling function's association
    - Calls are made to a function by location in an objectification tabled in another association by relation to the calling function or by relation to the calling function's association
    - Calls are made to a function by location in an objectification tabled in the calling function's locals
- Receives arguments as parameters passed from the calling function
- Execution follows the instructional statements of the function
- Function Template
  - Term must be used when the identifier identifies a template

## Situation

- Is a categorical set of reference types
  - Conformation Situation
    - Identifies a reference to an information type, if supported
  - Conformation Situation (...) Situation
    - Identifies a reference to a reference (...) to an information type, if supported
  - Abstraction Situation
    - Identifies a reference to a function type, which must be supported
  - Abstraction Situation (...) Situation
    - Identifies a reference to a reference (...) to a function type, if supported
  - Classification Situation
    - Identifies a reference to an objectification type, if supported
  - Classification Situation (...) Situation
    - Identifies a reference to a reference (...) to an objectification type, if supported
- Situation Template
  - Term must be used when the identifier identifies a template

## Location

- Is a categorical set of references
  - Information Location
    - Identifies a reference to an information
  - Information Location (...) Location
    - Identifies a reference to a reference (...) to an information
  - Function Location
    - Identifies a reference to a function
  - Function Location (...) Location
    - Identifies a reference to a reference (...) to a function
  - Objectification Location
    - Identifies a reference to an objectification
  - Objectification Location (...) Location
    - Identifies a reference to a reference (...) to an objectification
- May be treated as constant or variable
- May be supported by language defined operations (operators)
- Location Template
  - Term must be used when the identifier identifies a template

## Classification

- Identifies an objectification type
- Is either elementary or composite
- Must be composed of one or more abstraction situation
- Not required in typeless languages
- The language must provide a means of calling a function by location
  - Through a primitive objectification where applicable
  - Through a member element traversal for composite objectification where applicable
- Classification Template
  - Term must be used when the identifier identifies a template



## Objectification

- Identifies an addressable unit of objectification
- Must be composed of one or more function location
- May be composed of other information, location or objectification
- May be treated as constant or variable
- Objectification Template
  - Term must be used when the identifier identifies a template