



*TCI / TRI :*  
*C++ Language Mapping*

# Index

---

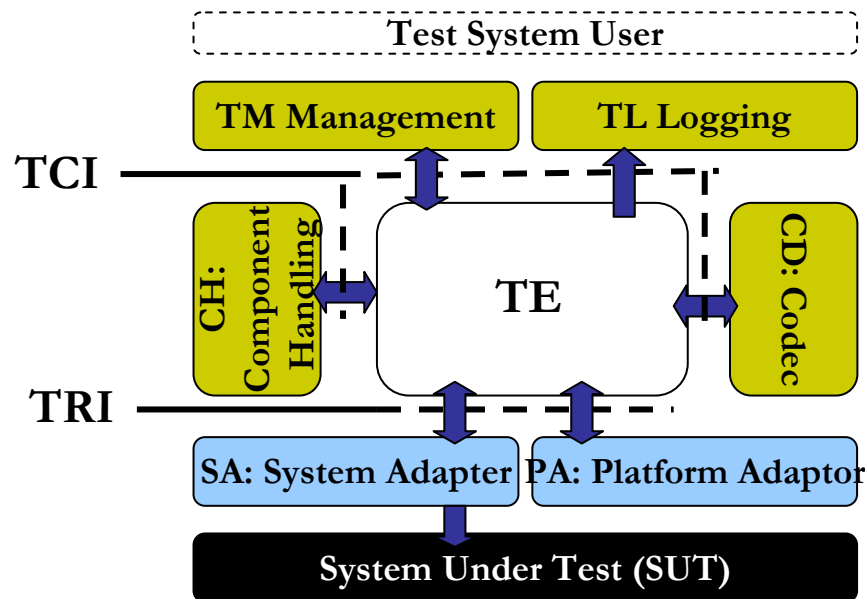


1. Introduction to TCI and TRI
2. ABC model
3. Memory management
4. Resource Locator
5. Conclusions



# Introduction to TCI and TRI

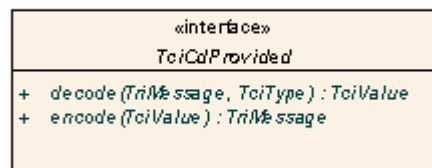
- TCI and TRI are part of TTCN3 standard especification.
- Both standards define the Architecture of a TTCN-3 testsystem.
- Testsystem is a program that contains executable TTCN3 testcases.



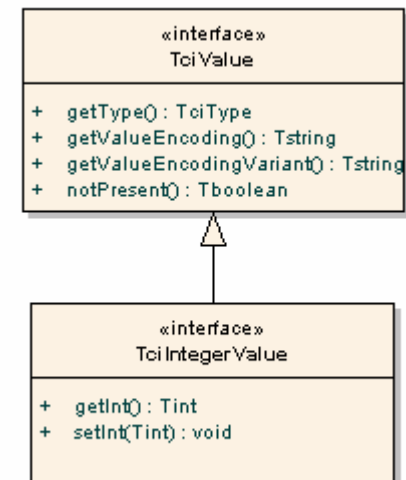
# Introduction to TCI and TRI (II)

- Both TCI and TRI are structured into 2 parts: operations and data.
- Tci Operations define operations between test system entities.
- Tci Data are used as parameter in Tci Operations.

## OPERATIONS



## DATA MODEL



# Base Line

- TTCN-3 c++ mapping is based on java mapping and c mapping
- Names have been respected to ensure naming convention.

## TTCN-3 Java Mapping (OO)

- Name of interfaces
- Name of methods

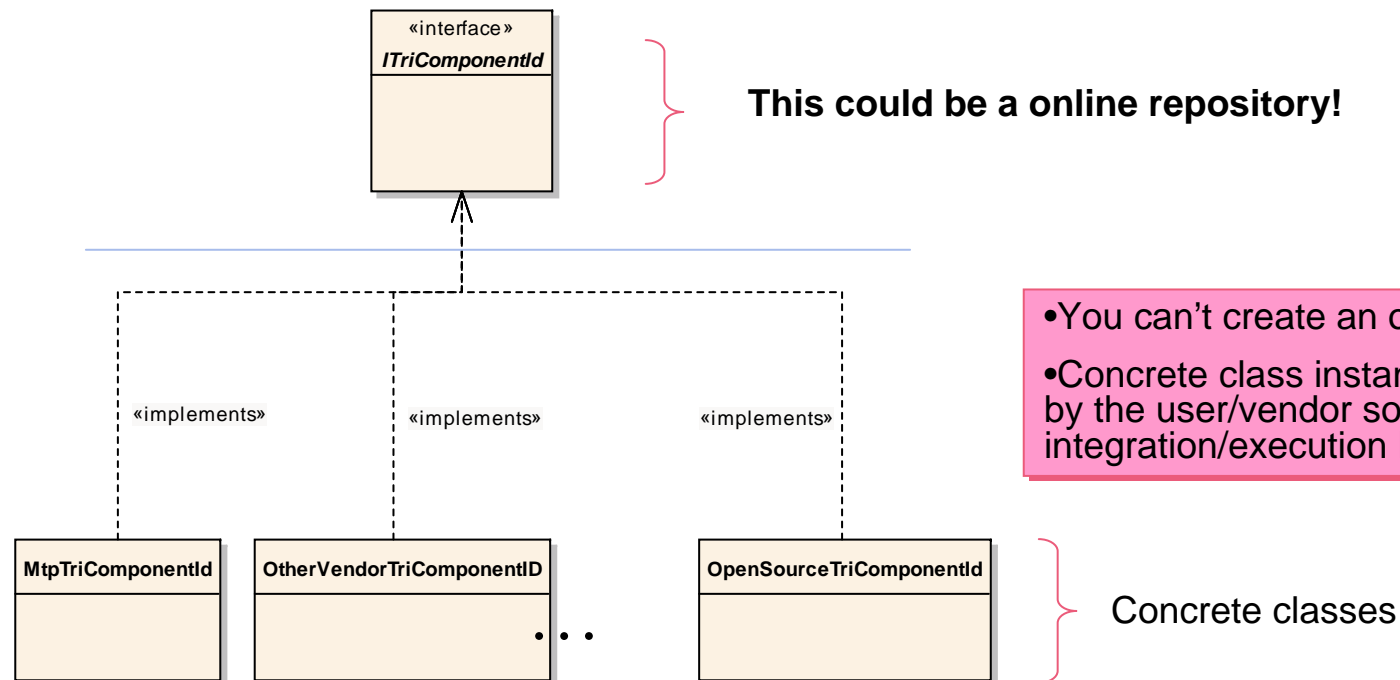
## TTCN-3 ANSI C Mapping

- data types used as parameters

easy 3rd party  
integration

# ABC model (I)

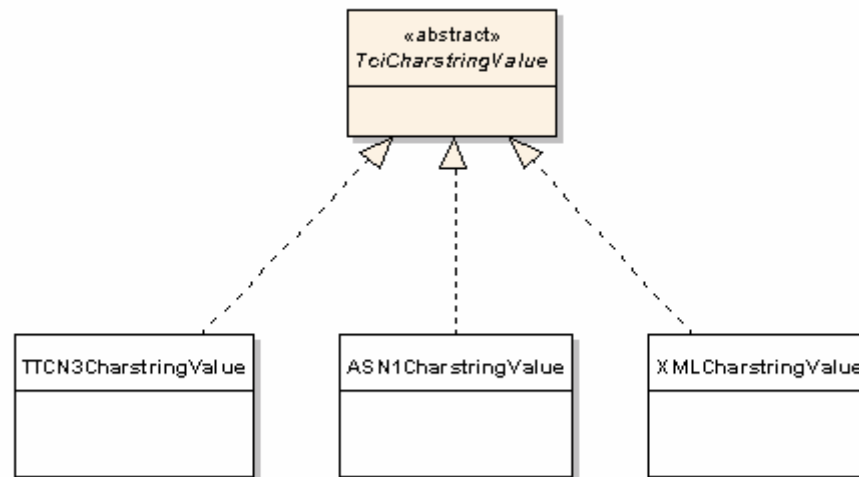
- Methods defined as pure virtual.
- All Implementors are obey to follow the interface.
- ABC model is common in component-based programming paradigms.



# ABC model (II)

## ABC Benefits

- ✓ Decouple dependencies between vendors
- ✓ Increase Flexibility and Scalability
- ✓ make easy the integration of code from different vendors.



# ABC Model (III)

- Getter methods has been defined to retrieve the properties of the types.
- Setter methods: non const methods to set properties of objects.
- All the formal parameters have been defined as references where it was possible.

```
class QualifiedName{  
public:  
virtual ~QualifiedName() = 0;  
virtual const TString &getObjectName() const = 0;  
virtual void setObjectName( const TString & ) = 0;  
};
```

# ABC Model (IV)

- Enumerated types such as TciTypeClass and TriStatus are proposed according to Type-safe design pattern.

## Benefits:

- Clients use a type that defines the allowed values instead of using an integer or a char that emulate the enumeration.
- Avoid checking if the actual value is a valid value.
- They can encapsulate complex data.

TriStatus
+ <u>TRI_ERROR</u> : TriStatus {readOnly}
+ <u>TRI_OK</u> : TriStatus {readOnly}
+ equals(p_other): Tbool
- TciTypeClass()
+ toString(): Tstring

# ABC Model (V)

- Namespace declaration to avoid name collisions.  
ORG\_ETSI\_TTCN3\_TRI  
ORG\_ETSI\_TTCN3\_TCI
- Encapsulated C++ built-in types.

```
typedef bool Tboolean
typedef long int Tint
typedef char Tchar
typedef std::string Tstring
typedef wchar_t TuniversalChar
typedef std::wstring TuniversalString
typedef double Tfloat
typedef unsigned long int Tsize
typedef unsigned long int Tposition
typedef unsigned char Tbit
typedef unsigned char Tbyte
typedef unsigned char Toctetchar
typedef wchar_t Twidechar
typedef std::wstring Twidestring
```

# Memory Management

- Problem: No GarbageCollector in C++.
- Common memory errors.

## MEMORY LEAK



## DANGLING REFERENCE



- Every project in C++ need a user defined memory management strategy.

# Use of STL Library (I)

- c-style strings and c-style vectors lead to memory leaks.
- `std::string` and `std::wstring` instead of `char *` and `wchar_t*`.

```
class QualifiedName{  
public:  
virtual ~QualifiedName() = 0;  
virtual const std::string &getObjectName() const = 0;  
};
```

# Use of STL Library (II)

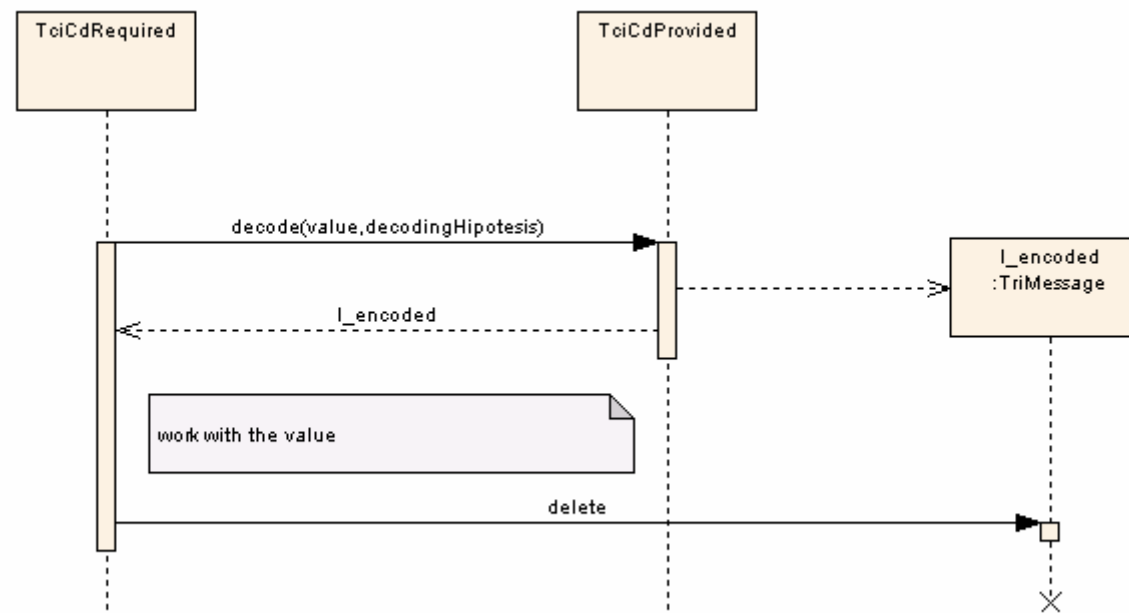
- c-style vectors lead to memory leaks.
- `std::vector` instead of c-style vectors/lists.

```
class TriPortIdList{  
public:  
virtual ~TriPortIdList()=0;  
virtual const std::vector<const TriPortId *>  
    &getPortId() const = 0;  
};
```

# Memory Management

## *Functions that return new instances*

- *Policy*: The One who creates memory is responsible to delete it.
- *Exception*: functions that return dynamic memory.
- *Solution*: The client of the function is responsible to delete the memory.

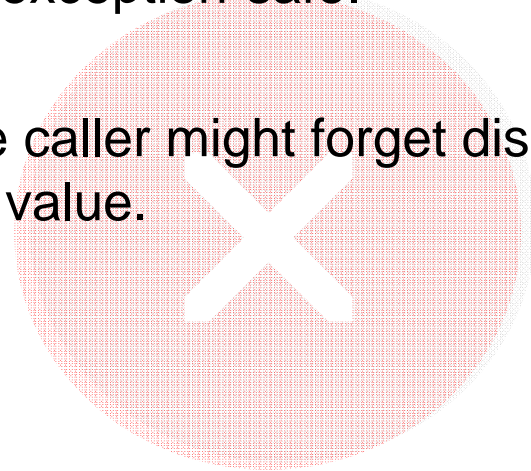


# Functions that return new instances

## *Memory Management*

### Return Raw Pointers

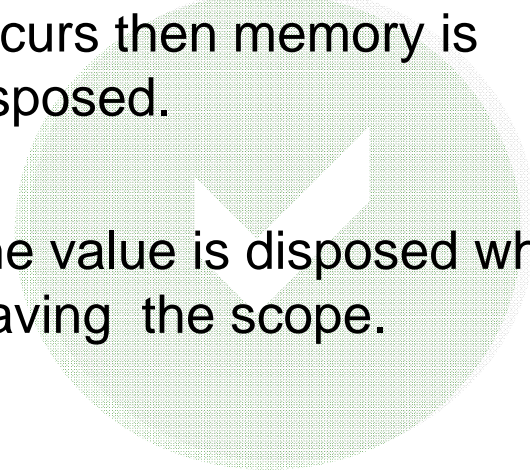
- No exception-safe.
- The caller might forget dispose the value.



### Return Smart Pointers

`std::auto_ptr`

- Exception safe: If exception occurs then memory is disposed.
- The value is disposed when leaving the scope.



# Functions that return new instances

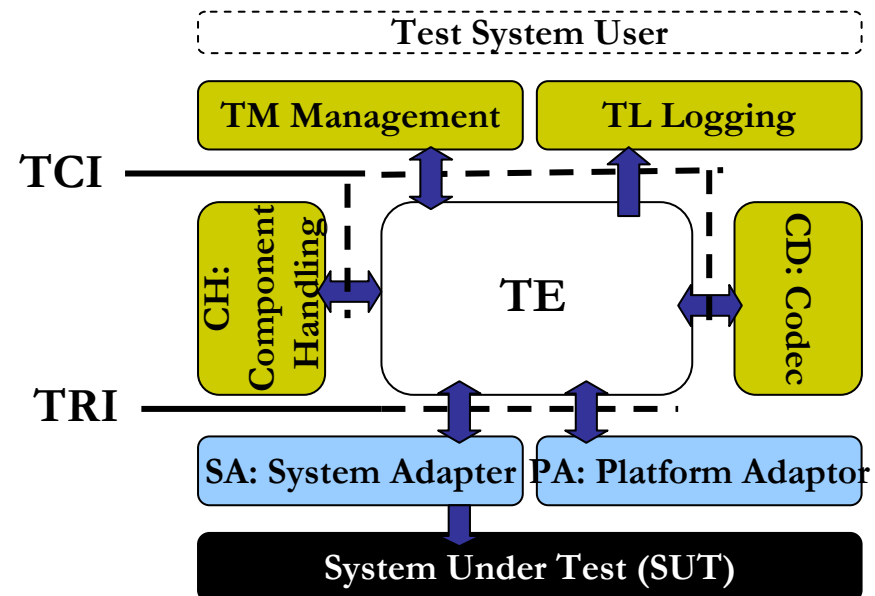
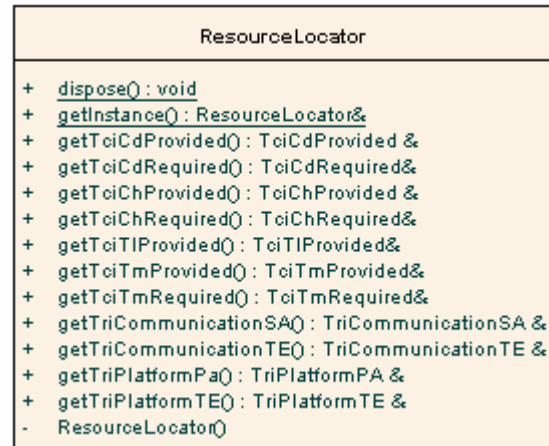
## *Memory Management*

### The approach using smart pointers

```
class TciCdRequried{  
public:  
virtual ~TciCdRequried( ) =0;  
virtual std::auto_ptr<TriMessage>  
    encode(const TciValue &)=0;  
};
```

# Resource Locator

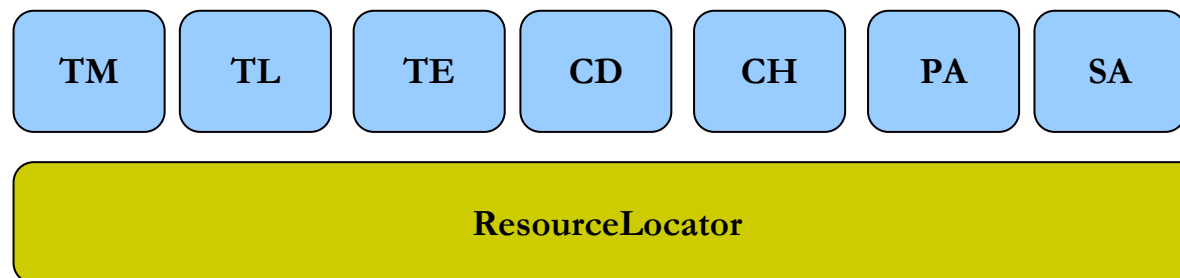
- Initialize, finalize, and configure the testsystem.
- Responsible of the lifecycle of the TM, TL, CD, CH, SA, PA and TE.
- Entities get a reference to the entities that they need once by requesting the Resource Locator.



# Why Resource Locator

## Benefits:

- Make easy the integration of entities from diferents vendors.
- Easy interface to entities.



# Conclusions

---

- TCI/TRI c++ mapping soon will be part of the TCI and TRI estandar especifications.
- ABCmodel assure flexibility and scalability of both data types and interfaces/methods.
- Memory management staff is a pretty important deal to be faced by tool providers and 3rd parties. It is needed a policy that regulates it.
- ResourceLocator makes easy the integration of entities from different vendors.

MÉTODOS Y TECNOLOGÍA



***CONTACT:***

*Adrián Benéitez Turrión*

*Senior Consultant*

adrian.beneitez@mtp.es

**Thanks for your attention!**

Paseo de la Castellana 182 - 10ª planta, 28046 Madrid. Tel: +34 913531564

www.mtp.es