SOFTWARE COMMUNICATIONS ARCHITECTURE SPECIFICATION

APPENDIX E - PLATFORM SPECIFIC MODEL (PSM) - TRANSFER MECHANISMS AND ENABLING TECHNOLOGIES

20 August 2015
Version: 4.1

Prepared by:

Joint Tactical Networking Center (JTNC)
33000 Nixie Way
San Diego, CA 92147-5110

REVISION SUMMARY

Version	Revision	Date
Next <Draft>	Initial Draft Release	30 November 2010
Candidate Release	Initial Release	27 December 2011
4.0	ICWG Approved Release	28 February 2012
4.0.1	Incorporated transition to JTNC and applied SCA 4.0 Errata Sheet v1.0	01 October 2012
4.1<DRAFT>	Changed appendix title (was previously titled "Appendix E - Platform Specific Model (PSM) – Transfer Mechanisms and Enabling Technologies"), Incorporated IDL Changes and pulled in content from SCA 4.0.1 Appendix E-3 PSM OMG IDL.	31 December 2014
4.1	Updated References Fixed technical redlines in CFPrimitiveTypes.h Added header files for technology neutral sequence templates and types (section E.8.2 and E.8.3) **ICWG Approved**	20 August 2015

TABLE OF CONTENTS

TABLE OF TABLES

APPENDIX E MODEL DRIVEN SUPPORT TECHNOLOGIES

E.1 SCOPE

This appendix identifies the approved set technology mappings, Platform Independent Model (PIM) transformations, and model representations used to achieve conformance with the SCA PIM as defined in the main specification. As additional transfer mechanisms and enabling technologies are approved for SCA usage they will be added as attachments to this appendix. The appendix also provides design guidelines for maximizing the technology neutral attributes of SCA compliant implementations; however, it does not prescribe any technology neutral design patterns.

E.1.1 Overview

The SCA provides an architectural framework that can be used across a wide variety of target platforms and technologies. An SCA compliant solution can be deployed within many differing end user architectures. The architectures should be designed to align as closely as possible to their associated mission requirements. The requirements often play a large role in dictating items such as implementation language, desired performance characteristics and processor architecture. SCA is not prescriptive on the implementing technology, but a key SCA enabler is the preservation of its interface definitions across the realizing technologies.

The SCA includes transfer mechanisms to provide standardized client/server operations. Client/server communications may be co-located or distributed across different processors. The transfer mechanism structure may be comprised of object request semantics, transfer and message syntax, and transports. SCA products can be realized using a variety of transports and technologies (e.g. CORBA, POSIX (IPC) Mechanisms such as Queues or Shared Memory, SOAP, Data Distribution Service (DDS), Modem Hardware Abstraction Layer (MHAL) Communication Service, etc.).

Most SCA products will use a transfer mechanism (e.g. CORBA) to connect (invoke methods in another component and transport associated data). However, some products may link components directly whereby no transfer mechanism is required (e.g., components are in libraries and methods invoked by function call). The available SCA transfer mechanisms are identified in E.9.2.

One or more Enabling Technologies, identified in E.9.3, are used. Some listed technologies may only be appropriate when used with specific Transfer Mechanisms.

E.2 CONFORMANCE

Conformance with the IDL PSM representation of the SCA can be achieved by using the IDL d**o**cumented in SCA Appendix C.

SCA Product conformance with the Transfer Mechanisms and Enabling Technologies is described in sections E.9.2 and E.9.3.

The elements of this specification are not required to be used solely for a particular platform or application. The "SCA model and architecture" may be realized by many different enabling technologies (i.e. technology specific representations). As technology specific mappings are introduced it is the intent that they will be functionally equivalent to the existing representations.

Conformance for an SCA product is at the level of usage as follows:

- An SCA product needs to be conformant with the semantics, design and mandatory elements specified in a defined profile of a technology specific representation identified within this appendix.

E.2.1 Sample Conformance Statement

An SCA product can be identified as being conformant to a specific version of the SCA and the specific technology that the product realizes.

- "Product A is an SCA conformant waveform application in accordance with the LwAEP, SCA Lightweight CORBA Profile and the IDL/XML DTD platform."

E.3 CONVENTIONS

N/A.

E.4 NORMATIVE REFERENCES

The following documents contain provisions or requirements which by reference constitute requirements of this appendix (required mappings are identified in section E.6). Applicable versions are as stated.

[1] OMG Document formal/2011-11-01, Common Object Request Broker Architecture (CORBA) Specification, Version 3.1.1 Part 1: CORBA Interfaces, Version 3.2, November 2011.

E.5 INFORMATIVE REFERENCES

[2] OMG Document formal/2002-04-01, UML™ Profile for CORBA™ Specification, Version 1.0, April 2002.

[3] OMG Document formal/2008-11-06, Common Object Request Broker Architecture (CORBA) for embedded Specification, Version 1.0, November 2008.

E.6 PLATFORM INDEPENDENT MODEL

This section provides a mapping of the Unified Modeling Language (UML) representation documented in the SCA PIM to its equivalent IDL representation.

The SCA technology specific mapping to OMG IDL consists of IDL that is based upon the UML Profile for CORBA [2]. The UML to IDL transformation rules are not universal rules for creating *any* technology specific mapping, but only for those within this appendix. This section defines a non-normative reference mapping of the UML constructs used within the SCA,

a summary of which is represented in Table 1. The rule set for transforming UML packages, interfaces, types, and exceptions into IDL constructs are as follows:

1. UML interfaces and interface extensions map to a <<CORBAInterface>>. The <<CORBAInterface>> names, and subsequently the interface name in the IDL, are equivalent to the interface names within the SCA.
2. UML attributes with readonly and readwrite map to <<CORBAInterface>> attributes and interface attributes within the IDL.
3. UML classes without operations that are not stereotyped and used as type definitions map to a <<CORBAStruct>> in a <<CORBAInterface>> or <<CORBAModule>> and a struct within the generated IDL. The designated classes name does not get translated into a CORBA type, the class attributes are added as fields within the IDL struct definition.
4. UML <<datatype>> maps to CORBA basic types. Primitive types are mapped to the corresponding IDL primitive types and primitive sequence types are mapped to a <<CORBATypedef>> and IDL typedef of primitive sequence types.
5. UML exceptions and exception extensions map to a <<CORBAException>> and an IDL exception. There is no specialization of exceptions in IDL so the SystemException definition does not appear in the generated IDL interfaces but all the specialized exceptions of SystemException are in the CORBA object module.
6. UML attributes that have a cardinality of many [*] map to a <<CORBATypedef>> of an IDL sequence type.
7. UML operations map to operations in the <<CORBAInterface>> definition and within the IDL interfaces.
8. Transformations are only performed for concrete classes, not for template classes.
9. For interfaces that reference a component stereotype for a type, the "component" qualifier is removed from the name within the IDL definition. For example, the FileManagerComponent would become *FileManager* as the type for the parameter or attribute.
10. UML attributes with constant stereotype map to a <<CORBAConstant>> in a <<CORBAInterface>> and within the generated IDL interface.
11. Basic types (e.g., Any, Object) map to an equivalent CORBA and IDL type.

The SCA UML model maps to a <<CORBAModule>> and IDL module named CF (Core Framework). The CF IDL module is broken up into multiple files in order to allow for a minimal memory foot print size for the embedded environment.

Table 1: SCA UML to OMG IDL PSM Mapping

UML Representation	OMG IDL Representation
Interface	Interface
exception	exception
Object	Object
Primitive Types (integer, Boolean, string, unlimited natural); others are constructed by layering formatting or constraints on top of primitives	void, boolean, char, double, long, float, long long, short, octet, string, unsigned long long, unsigned short, wchar, wstring
Any	Any data type
Struct	Struct
Sequence	Sequence definition
Type	Typedef
Package	Module
No return value from an operation	void
Enumeration	enum
A constrained integer data type that corresponds to an octet	octet
Attribute	attribute

E.7 IDL TECHNOLOGY NEUTRAL TYPES

This section defines the technology neutral recommendations for component interfaces using IDL. IDL was selected for this representation because several Standard IDL to programming language mappings exist. The technology neutral interface definitions avoid using CORBA types, thus allowing communication mechanisms other than CORBA to be used. CORBA types are replaced by CF primitive and primitive sequence types. Those primitives or user defined types should be used within component definitions to increase the creation of technology neutral interfaces. The use of CORBA types such as Object and Any should be avoided within interfaces if the objective is to be technology neutral.

In addition, XML primitive sequence types are mapped to the CF primitive types described below.

E.7.1 CF Primitive Types

```
//Source file: CFPrimitiveTypes.idl
#ifndef __CFPRIMITIVETYPES_DEFINED
#define __CFPRIMITIVETYPES_DEFINED

module CF {

   /* These primitive types to avoid the usage of CORBA qualified types
      in user-defined interfaces and types in generated code. */

   typedef boolean BooleanType;

   typedef char CharType;

   typedef long long LongLongType;

   typedef long LongType;

   typedef short ShortType;

   typedef unsigned long long ULongLongType;

   typedef unsigned long ULongType;

   typedef unsigned short UShortType;

   typedef float FloatType;

   typedef double DoubleType;

typedef string StringType;

   typedef octet OctetType;

};
#endif
```

E.7.2 CF Primitive Sequence Types

```
//File: CFPrimitiveSeqTypes.idl
#ifndef __CFPRIMITIVESEQTYPES_DEFINED
#define __CFPRIMITIVESEQTYPES_DEFINED

#include "CFULongLongSeq.idl"
#include "CFBooleanSeq.idl"
#include "CFULongSeq.idl"
#include "CFLongLongSeq.idl"
#include "CFCharSeq.idl"
#include "CFUShortSeq.idl"
#include "CFLongSeq.idl"
#include "CFDoubleSeq.idl"
#include "CFShortSeq.idl"
#include "CFFloatSeq.idl"
#include "CFOctetSeq.idl"
#include "CFStringSeq.idl"
```

```
#endif
```

E.7.2.1 CF BooleanSeq

```
//File: CFBooleanSeq.idl
#ifndef __CFBOOLEANSEQ_DEFINED
#define __CFBOOLEANSEQ_DEFINED
#include "CFPrimitiveTypes.idl"

module CF {
    /* This type is a unbounded sequence of booleans. */
    typedef sequence <CF::BooleanType> BooleanSeq;
};

#endif
```

E.7.2.2 CF CharSeq

```
//File: CFCharSeq.idl
#ifndef __CFCHARSEQ_DEFINED
#define __CFCHARSEQ_DEFINED
#include "CFPrimitiveTypes.idl"

module CF {
    /* This type is a unbounded sequence of characters. */
    typedef sequence <CF::CharType> CharSeq;
};

#endif
```

E.7.2.3 CF DoubleSeq

```
//File: CFDoubleSeq.idl
#ifndef __CFDOUBLESEQ_DEFINED
#define __CFDOUBLESEQ_DEFINED
#include "CFPrimitiveTypes.idl"

module CF {
    /* This type is a unbounded sequence of doubles. */
    typedef sequence <CF::DoubleType> DoubleSeq;
};

#endif
```

E.7.2.4 CF FloatSeq

```
//File: CFFloatSeq.idl
#ifndef __CFFLOATSEQ_DEFINED
#define __CFFLOATSEQ_DEFINED
#include "CFPrimitiveTypes.idl"

module CF {
    /* This type is a unbounded sequence of floats. */
    typedef sequence <CF::FloatType> FloatSeq;
};
```

```
#endif
```

E.7.2.5 CF LongSeq

```
//File: CFLongSeq.idl
#ifndef __CFLONGSEQ_DEFINED
#define __CFLONGSEQ_DEFINED
#include "CFPrimitiveTypes.idl"

module CF {
    /* This type is a unbounded sequence of longs. */
    typedef sequence <CF::LongType> LongSeq;
};

#endif
```

E.7.2.6 CF LongLongSeq

```
//File: CFLongLongSeq.idl
#ifndef __CFLONGLONGSEQ_DEFINED
#define __CFLONGLONGSEQ_DEFINED
#include "CFPrimitiveTypes.idl"

module CF {
    /* This type is a unbounded sequence of long longs. */
    typedef sequence <CF::LongLongType> LongLongSeq;
};

#endif
```

E.7.2.7 CF OctetSeq

```
//File: CFOctetSeq.idl
#ifndef __CFOCTETSEQ_DEFINED
#define __CFOCTETSEQ_DEFINED
#include "CFPrimitiveTypes.idl"

module CF {
    /* This type is a unbounded sequence of octets. */
    typedef sequence <CF::OctetType> OctetSeq;
};

#endif
```

E.7.2.8 CF ShortSeq

```
//File: CFShortSeq.idl
#ifndef __CFSHORTSEQ_DEFINED
#define __CFSHORTSEQ_DEFINED
#include "CFPrimitiveTypes.idl"

module CF {
    /* This type is a unbounded sequence of shorts. */
    typedef sequence <CF::ShortType> ShortSeq;
};

#endif
```

E.7.2.9 CF StringSeq

```
//File: CFStringSeq.idl
#ifndef __CFSTRINGSEQ_DEFINED
#define __CFSTRINGSEQ_DEFINED
#include "CFPrimitiveTypes.idl"

module CF {
    /* This type is a unbounded sequence of strings. */
    typedef sequence <CF::StringType> StringSeq;
};

#endif
```

E.7.2.10 CF ULongSeq

```
//File: CFULongSeq.idl
#ifndef __CFULONGSEQ_DEFINED
#define __CFULONGSEQ_DEFINED
#include "CFPrimitiveTypes.idl"

module CF {
    /* This type is a unbounded sequence of unsigned longs. */
    typedef sequence <CF::ULongType> ULongSeq;
};

#endif
```

E.7.2.11 CF ULongLongSeq

```
//File: CFULongLongSeq.idl
#ifndef __CFULONGLONGSEQ_DEFINED
#define __CFULONGLONGSEQ_DEFINED
#include "CFPrimitiveTypes.idl"

module CF {
    /* This type is a unbounded sequence of unsigned long longs. */
    typedef sequence <CF::ULongLongType> ULongLongSeq;
};

#endif
```

E.7.2.12 CF UShortSeq

```
//File: CFUShortSeq.idl
#ifndef __CFUSHORTSEQ_DEFINED
#define __CFUSHORTSEQ_DEFINED
#include "CFPrimitiveTypes.idl"

module CF {
    /* This type is a unbounded sequence of unsigned shorts. */
    typedef sequence <CF::UShortType> UShortSeq;
};

#endif
```

E.8 HEADER FILE TECHNOLOGY NEUTRAL TYPES

This section defines the technology neutral recommendations for component interfaces using header files. The technology neutral interface definitions avoid using CORBA types, thus allowing communication mechanisms other than CORBA to be used. The identified types should be used within component definitions to enhance the development of technology neutral interfaces.

E.8.1 CF Primitive Types

```
/* File: CFPrimitiveTypes.h */
#ifndef __CFPRIMITIVETYPES_DEFINED
#define __CFPRIMITIVETYPES_DEFINED
#include <stdint.h>

namespace CF
{
    typedef bool BooleanType;
    typedef BooleanType& Boolean_out;
    typedef BooleanType* Boolean_ptr;

    typedef char  CharType;
    typedef CharType & Char_out;
    typedef CharType * Char_ptr;

    typedef uint8_t OctetType;
    typedef OctetType&  Octet_out;
    typedef OctetType*  Octet_ptr;

    typedef int16_t ShortType;
    typedef ShortType&  Short_out;
    typedef ShortType*  Short_ptr;

    typedef uint16_t UShortType;
    typedef UShortType&  UShort_out;
    typedef UShortType*  UShort_ptr;

    typedef int32_t LongType;
    typedef LongType&   Long_out;
    typedef LongType*   Long_ptr;

    typedef uint32_t ULongType;
    typedef ULongType&   ULong_out;
    typedef ULongType*   ULong_ptr;

    typedef float  FloatType;
    typedef FloatType& Float_out;
    typedef FloatType* Float_ptr;

    typedef double  DoubleType;
    typedef DoubleType& Double_out;
    typedef DoubleType* Double_ptr;

    typedef int64_t LongLongType;
```

```
    typedef LongLongType& LongLong_out;
    typedef LongLongType* LongLong_ptr;

    typedef uint64_t ULongLongType;
    typedef ULongLongType& ULongLong_out;
    typedef ULongLongType* ULongLong_ptr;

    typedef char* StringType;
    class String_var;
    class String_mgr;
    class String_out;
}

#endif //__CFPRIMITIVETYPES_DEFINED
```

E.8.2 CF Sequence Templates

```
/* File: CFSequenceTemplates.h */
#ifndef __CFSEQUENCETEMPLATES_DEFINED
#define __CFSEQUENCETEMPLATES_DEFINED
#include "CFPrimitiveTypes.h"

namespace CF
{
       template <class T>
       class Sequence
       {
         public:
              Sequence();
              Sequence(CF::ULongType max);
              Sequence(
                           CF::ULongType max,
                           CF::ULongType length,
                           T *value,
                           CF::BooleanType release = 0
                           );
              ~Sequence();

              Sequence(const Sequence&);
              Sequence &operator=(const Sequence&);

              T &operator[](CF::ULongType index);
              const T &operator[](CF::ULongType index) const;

              CF::ULongType length() const;
              void length(CF::ULongType);
              CF::ULongType maximum() const;

              CF::BooleanType release() const;

              void replace(
                     CF::ULongType max,
                     CF::ULongType length,
                     T *data,
                     CF::BooleanType release = 0
```

```
                    );

            const T* get_buffer() const;
            T* get_buffer(CF::BooleanType orphan = 0);

            static T *allocbuf(CF::ULongType nelems);
            static void freebuf(T *);
      };

      template <class T>
      class Sequence_String : public Sequence<T>
      {
        public:
            const T operator[](CF::ULongType index) const;
      };
}

#endif //__CFSEQUENCETEMPLATES_DEFINED
```

E.8.3 CF Primitive Sequence Types

```
/* File: CFPrimitiveSeqTypes.h */
#ifndef __CFPRIMITIVESEQTYPES_DEFINED
#define __CFPRIMITIVESEQTYPES_DEFINED

#include "CFULongLongSeq.h"
#include "CFBooleanSeq.h"
#include "CFULongSeq.h"
#include "CFLongLongSeq.h"
#include "CFCharSeq.h"
#include "CFUShortSeq.h"
#include "CFLongSeq.h"
#include "CFDoubleSeq.h"
#include "CFShortSeq.h"
#include "CFFloatSeq.h"
#include "CFOctetSeq.h"
#include "CFStringSeq.h"

#endif __CFPRIMITIVESEQTYPES_DEFINED
```

E.8.3.1 CF BooleanSeq

```
/* File: CFBooleanSeq.h */
#ifndef __CFBOOLEANSEQ_DEFINED
#define __CFBOOLEANSEQ_DEFINED
#include "CFSequenceTemplates.h"

namespace CF
{
      typedef CF::Sequence<CF::BooleanType> BooleanSeq;
}

#endif //__CFBOOLEANSEQ_DEFINED
```

E.8.3.2 CF CharSeq

```
/* File: CFCharSeq.h */
#ifndef __CFCHARSEQ_DEFINED
#define __CFCHARSEQ_DEFINED
#include "CFSequenceTemplates.h"

namespace CF
{
	typedef CF::Sequence<CF::CharType> CharSeq;
}

#endif //__CFCHARSEQ_DEFINED
```

E.8.3.3 CF DoubleSeq

```
/* File: CFDoubleSeq.h */
#ifndef __CFDOUBLESEQ_DEFINED
#define __CFDOUBLESEQ_DEFINED

#include "CFSequenceTemplates.h"

namespace CF
{
	typedef CF::Sequence<CF::DoubleType> DoubleSeq;
}

#endif //__CFDOUBLESEQ_DEFINED
```

E.8.3.4 CF FloatSeq

```
/* File: CFFloatSeq.h */
#ifndef __CFFLOATSEQ_DEFINED
#define __CFFLOATSEQ_DEFINED
#include "CFSequenceTemplates.h"

namespace CF
{
	typedef CF::Sequence<CF::FloatType> FloatSeq;
}

#endif //__CFFLOATSEQ_DEFINED
```

E.8.3.5 CF LongSeq

```
/* File: CFLongSeq.h */
#ifndef __CFLONGSEQ_DEFINED
#define __CFLONGSEQ_DEFINED
#include "CFSequenceTemplates.h"

namespace CF
{
	typedef CF::Sequence<CF::LongType> LongSeq;
}

#endif //__CFLONGSEQ_DEFINED
```

E.8.3.6 CF LongLongSeq

```
/* File: CFLongLongSeq.h */
#ifndef __CFLONGLONGSEQ_DEFINED
#define __CFLONGLONGSEQ_DEFINED
#include "CFSequenceTemplates.h"

namespace CF
{
	typedef CF::Sequence<CF::LongLongType> LongLongSeq;
}

#endif //__CFLONGLONGSEQ_DEFINED
```

E.8.3.7 CF OctetSeq

```
/* File: CFOctetSeq.h */
#ifndef __CFOCTETSEQ_DEFINED
#define __CFOCTETSEQ_DEFINED
#include "CFSequenceTemplates.h"

namespace CF
{
	typedef CF::Sequence<CF::OctetType> OctetSeq;
}

#endif
```

E.8.3.8 CF ShortSeq

```
/* File: CFShortSeq.h */
#ifndef __CFSHORTSEQ_DEFINED
#define __CFSHORTSEQ_DEFINED
#include "CFSequenceTemplates.h"

namespace CF
{
	typedef CF::Sequence<CF::ShortType> ShortSeq;
}

#endif //__CFSHORTSEQ_DEFINED
```

E.8.3.9 CF StringSeq

```
/* File: CFStringSeq.h */
#ifndef __CFSTRINGSEQ_DEFINED
#define __CFSTRINGSEQ_DEFINED
#include "CFSequenceTemplates.h"

namespace CF
{
	typedef CF::Sequence_String<CF::StringType> StringSeq;
}

#endif //__CFSTRINGSEQ_DEFINED
```

E.8.3.10 CF ULongSeq

```
/* File: CFULongSeq.h */
#ifndef __CFULONGSEQ_DEFINED
#define __CFULONGSEQ_DEFINED
#include "CFSequenceTemplates.h"

namespace CF
{
	typedef CF::Sequence<CF::ULongType> ULongSeq;
}

#endif //__CFULONGSEQ_DEFINED
```

E.8.3.11 CF ULongLongSeq

```
/* File: CFULongLongSeq.h */
#ifndef __CFULONGLONGSEQ_DEFINED
#define __CFULONGLONGSEQ_DEFINED
#include "CFSequenceTemplates.h"

namespace CF
{
	typedef CF::Sequence<CF::ULongLongType> ULongLongSeq;
}

#endif //__CFULONGLONGSEQ_DEFINED
```

E.8.3.12 CF UShortSeq

```
/* File: CFUShortSeq.h */
#ifndef __CFUSHORTSEQ_DEFINED
#define CFUSHORTSEQ_DEFINED
#include "CFSequenceTemplates.h"

namespace CF
{
	typedef CF::Sequence<CF::UShortType> UShortSeq;
}

#endif //__CFUSHORTSEQ_DEFINED
```

E.9 ATTACHMENTS

This appendix includes the following platform independent and platform specific guidance, each defined in their associated attachment. Additional technologies and mechanisms may be included in future versions of this appendix.

E.9.1 Portable Design Guidance

- Appendix E-1: Application IDL PIM Profiles

E.9.2 Transfer Mechanisms

- Appendix E-2: PSM - CORBA

E.9.3 Enabling Technologies

- Appendix E-3: PSM – Language Specific Mappings