Advanced Networked Systems Architecture

ANSA

24 Hills Road
CAMBRIDGE
United Kingdom CB2 1JP

TELEPHONE: Cambridge (0223) 323010
INTERNATIONAL: +44 223 323010
TELEX: 817343 BLUCAM G

On the Building of Models

or

How to solve an N-dimensional Towers of Hanoi puzzle

Number: AO.17.01

Date: 3rd September 1986 10:44 am

Area: AO (Architectural Overview)

Status: P (Discussion Paper)

Classification: C (**COMMERCIAL-IN-CONFIDENCE**)

Distribution: T (ANSA Team)

Author: HJW (John Winterbotham)

CONTENTS

1 INTRODUCTION

This document follows the description of the multidimensional nature of architectures [1] and describes a basic description of objects, their placement within the problem space, the relations and interfaces between the objects, and the construction of models.

2 TERMINOLOGY

Before describing objects, relations, and interfaces, we must define some basic terminology.

2.1 Type

Within ANSA there are many sorts of things. These things are separable into families where the members of each family have much in common. Some things may belong to several families. The common parts to each member of a family are known as the **type** of that family. Thus most parts of the architecture have a type, whether they are objects, relations, interfaces, or properties. (To a first order these descriptive terms form the beginnings of the type definitions, as they have meanings imported from the different cultures contributing to ANSA. This importation can be dangerous as it leads to misunderstandings and so we will work with explicit type definitions.).

Types must be referred to, and so must have names (see below). We adopt the convention that the name of a type (as opposed to an instance of the type) is CAPITALISED.

2.2 Name

Names are ubiquitous, and a little clarification will avoid confusion later.

A name is a sequence of printable characters that is used to identify something. We may therefore write -

NAME::=PRINTABLE_STRING -- In some suitable alphabet.

We will also need to refer to the set of allowed names within the architecture, since some additional restrictions will be placed upon the names actually used (EG must be readable, meaningful, ...). We therefore define -

ANSA_NAME::={n}:n∈NAME∧ansa_conforming(n)

In order to assist in the understanding of the various definitions, the set of ANSA names will be subdivided into names for objects, names for relations, and so on. These names will be distinguished from each other by some as yet undefined structuring of the name string itself. We can therefore say, for example -

OBJECT_NAME::= {o} : o∈ANSA_NAME
∧ o ∉ {∪ the other subsets } -- IE Unique
∧ is_an_object(o)

3 PROPERTIES

The Object is the fundamental abstraction for describing the components of the architecture, These Objects will have a number of intrinsic **properties** that define the characteristics of the objects themselves rather than their interactions with other objects. The set of properties will be common to all of the objects within the architecture, and the type definitions of the objects (see below) will specify value expressions determining the permitted values for each property of the object. As a simplification, default values are defined so that the object type definitions need only specify those properties that differ.

3.1 Property names

The properties will be named-

PROPERTY_NAME::= {p} : p∈ANSA_NAME
∧p ∉ {∪ the other subsets} -- IE Unique
∧ is_a_property(p)

Thus the set of property names is universal, and common to everything within the architecture.

4 DIMENSIONS

As described in [1] and [3], the architecture exists within a multi-dimensional problem space. The dimensions of this space,. and the coordinates defined along each dimension, are linked directly to selected properties of the objects that are to be placed within the space and which form the architecture.

4.1 Selection of properties

Detailed discussion of how a subset of the properties were selected as being fundamental to the architecture, and so forming the dimensions of the problem space is given in [3]. The dimensions resulting from this selection all have discrete coordinate systems, which we represent as regions along each dimension.

Since each dimension is mapped onto a single property, we can adopt a shorthand terminology by which we name the dimensions with the same name as the associated property -

DIMENSION_NAME::= {d} : d∈ANSA_NAME
∧o ∉ {∪ the other subsets} -- IE Unique
∧ is_a_dimension(d)

4.2 Placement

Since the objects within the architecture exist within the problem space with its dimensions and coordinate system, each may be **placed** in relation to the coordinates. Placement occurs at a **location** within the problem space at which the values of the dimensional properties are the same as the values of the same properties of the object -

∀p∈DIMENSION_NAME : location_of_object(Object,p) =
(property_of_object(Object,p)
∨PROPERTY_DEFAULT[p])

4.3 Boxes

Since all of the dimensions that we are using have parameretised coordinates, and we have choosen to represent each value by a region along the appropriate coordinate, each unique combination of dimensional values corresponds to a **box** within the problem space. Objects are then placed within the same box if they have the same set of values for the dimensional properties. This raises the question of the meaning and use of the additional spatial relationships that arise.

These are two possibilities, either the relative positions within the boxes are used solely for representational convenience, or they are used to indicate variation in one or more of the non-dimensional properties. It is not clear at present which of these will be appropriate for ANSA.

5 OBJECTS

This section outlines the characteristics and properties of Objects in general terms, rather than in particular. The approach described here is modeled on the work of Lorin [2] with some changes in terminology and in the concepts of inheritance and variants.

5.1 Boundary

Objects are information hiding abstractions. When representing parts of a system as objects, the intention is to hide their internals from consideration as part of that representation. Objects may be subdivided in other representations, but are the atomic abstractions from which each architectural representation will be built.

Careful use of the information hiding aspects of objects allows the architects and designers to be concerned only with the behaviour of the object and not with how this behaviour is obtained. We therefore characterise objects by their **type**.

5.2 Type

The **type** of an object defines all of the visible aspects of the object. These aspects are the **relations** that the object may establish with other objects, and the types of the **attributes** that the object may possess together with matching value expressions delimiting any constraints on the values of the properties.

```
OBJECT_TYPE::= Attributes: {ATTRIBUTE},
               Relations:  {RELATION}

ATTRIBUTE::=   Predicate | Expression   -- about the object

RELATION::=    Predicate | Expression   -- about other objects
```

5.2.1 Relations and Interactions

Relations are statements about the relationship between an object of this type and one or more other objects. Relations may refer to the types of the objects, which therefore includes the operations they perform, the values of their attributes, or other relations.

Since relations are a part of the type definition of an object, they express the abilities of instances of objects having that type to interact with other object instances. The establishment of actual **interactions** between the object instances depends upon the design satisfying the various predicates expressed within the architectural rules (see later), and taking appropriate action.

5.2.2 Attributes

Attributes are statements about the characteristics of instances of this type of object, and do not involve other objects.

A complete set of attributes for an object must define, either explicitly or by the use of defaults, values for all of the properties of the problem space.

$$\forall p \in \text{PROPERTY\_NAME} : (p \rightarrow \text{value\_expression}) \in \text{Attributes}$$

NOTE that since the value of the property is given by a value-expression within the attributes, this value-expression may limit the value rather than determine it uniquely, and may indeed be NULL or {}. There is a tighter restriction -

$$\forall p \in \textit{DIMENSION\_NAME} : (p \rightarrow \textit{value\_expression}) \in \textit{Attributes}$$

for which value_expression may be constrained to be non-null.

5.2.3 Interfaces

The interface of an object is generally considered to be the set of operations to which that object will respond. Since the type defines the potentialities of an object, ,what it actually contains is an **interface definition**, and the **interfaces** are part of the interactions that instances of the object may establish.

Since the interface definitions are concerned with the possible interaction with other objects, they are relations rather than attributes.

An interface definition contains **operations** and a **state model.** Operations are functions that take arguements and return results (or errors). They are the only mechanism that objects have for exchanging data with other objects. The operations, arguements and results exchanged are not sufficient on their own to define all of the effects that occur and may be reflected in later operations. The interface must also define a **state model.**

The state model is the type definition of the state information that the object makes available to the outside world. This is **not** the same as the internal state which may include data not made available to the outside world, and which will be in formats private to the object. The state model therefore represents the 'conceptual' state which the object designer wishes to be used for the interactions between this object and the other objects within the system.

5.3 Classes and Inheritance

Individual definitions of all the parts of the type of the objects are necessary for checking the correctness of the models constructed. However this involves considerable overhead if complete definitions are maintained for each type, and so the **inheritance** of type definitions is introduced.

The basic principle of inheritance is that every object has one or more **parent classes** from which it inherits the complete definition of the parents types. Variations in the type for the **child** object are then made by additional definitions which add to or supercede parts of the parental type definitions. Thus the definition of the type for an object consists of a reference to the parental type definitions (with some prioritisation for conflicts), and additional definitions as required. This scheme allows the construction of inheritance **nets** consisting of parent/child derivations.

It should be noted that most parent classes will have incompletely specified types, as their main purpose is the propogation of common characteristics rather than completeness of specification.

5.4 Defaults

In order to simplify the type definitions of the objects within the architecture, **default** values are defined for all properties. These defaults are used whenever a value expression is not explicitly defined for an object. These defaults are specified in the attributes of the parent classes of the objects by non null value expressions. These will be overwritten as appropriate with actual value expressions.

6 MODELS

This section described how the various objects are positioned within the problem space, and how they interact to form models.

6.1 What is a model ?

A **model** is a set of object instances interacting with each other. This is distinct and separate from the use of example models to illustrate generic aspects of systems, in which generic instances of objects are shown interacting. These generic instances are sometimes confused with types or classes of object which leads to the erroneous conclusion that such models are about relations between types when in fact they are about interactions between instances.

The various interactions between individual object instances form longer **paths** between objects that are not necessarily interacting directly. These paths are very important to the architecture as they are the means by which the models cease to be individual object instances and become a cohesive system model possessing long range order.

The complete architecture is a model composed of objects connected by paths of various sorts.

6.2 Sorts of Objects

Within the ANSA problem domain, Objects are created and destroyed, and they either remain in one place or they move around. These behaviours lead to a useful characterisation of objects as either static or mobile (in a spatial sense), and either permanent or dynamic (in a temporal sense).

For example, the computers of a distributed system are static and permanent, while documents are mobile and dynamic. All distributed systems will have some mobile permanent services (such as directories and time), while the uinderlying operating systems provide static and dynamic processes.

6.2.1 Data Objects

A major group of dynamic objects are **data objects.** These data objects are very important to ANSA, as they hide the details of the implementation of data storage, and provide a high level abstraction that uses the same semantics for describing data as for describing the rest of the system.

6.3 Paths of interaction

As described above, objects are assembled into models by their interactions. Chains of interactions will develop within the systems. These **paths** reflect a form of interaction that extends beyond the nearest objects, and allows the designer greater freedom in matching the various parts of the design to the many and conflicting topological requirements. (An obvious example of the utility of paths may be seen in the OSI Transport Service abstraction for Layers 1 to 4 of OSI communications networks).

Paths provide a common abstraction for many forms of interaction chains, communication being the most obvious, but paths of interactions such as 'is composed of' will also occur and will provide a basis for manipulating usefull system characteristics (configuration management in this case).

Paths are much more changeable than object instances, and we can distinguish paths on the basis of the time at which they are created (and destroyed). Although there is a continuum of creation times, there are three usefull groupings. These are 'design time' for those paths that are designed into the system (such as communication paths), 'configuration time' for those paths that are set up as part of configuring particular systems, and 'execution time' for those paths that are set up as the system performs its tasks for the Users.

6.4 Selective representation

Any complete model of a system will include all object instances interacting with each other via all of the interactions. This results in a very complex representation of the model, as the positioning of all of the object instances in all of the dimensions needs to be shown. We therefore have a need for selective representation of models, and in order for such partial representations to be correctly and reproducably derived from the full model, we must establish a derivation procedure.

The problem is basically how to 'project' the full model onto a representation that shows those aspects of interest to the audience considered. In brief, this is done by selecting the dimensions that will be held constant for all of the objects in this particular representation (EG structural coordinate will be service, evolution will be release 1.0, and human interface will be conceptual model), and those that will be allowed to vary (EG communications). Next the axes of the representation (2 or 3 if using isometric representations) must be mapped onto the dimensions that are allowed to vary or onto 'separation for illustrative purposes'. The fixing of dimensional property values allows the selection of those object instances that are 'visible' in this representation, and of the interactions between them. Some interactions will have only one end visible, and these are only included in the representation if they form part of one or more paths that have both ends visible. In this case the visible paths are shown.

7 RULES

Rules are expressions that constrain, bound and bind the various parts of the architecture (and so also the models of which the architecture is composed). They are discussed in [4], and a simple example is given here to illustrate the power of the approach.

7.1 Example

This simple example of a rule defines the predicate -

is_in_communication_with(OBJ_INSTANCE$_A$,OBJ_INSTANCE$_B$)::=
OBJ_INSTANCE$_A$,OBJ_INSTANCE$_B$ $\in$ SUITABLE_OBJECT_SET $\wedge$
property_of(OBJ_INSTANCE$_A$,COMMS) =
property_of(OBJ_INSTANCE$_B$,COMMS)
$\models$ has(OBJ_CLASS$_A$,INTERFACE_1) $\wedge$
has(OBJ_CLASS$_B$,INTERFACE_2) $\wedge$
can_communicate_with(INTERFACE_1,INTERFACE_2) $\wedge$
has_path_to(OBJ_INSTANCE$_A$,OBJ_INSTANCE$_B$)

NOTE that while the syntax of this example may be at fault, the intention and expressive power is thought to represent what is possible.

8 References

[1] *On the Dimensionality of Architectures* (AO.14)

[2] "An Expanded Approach to Objects" Harold Lorin

[3] *The ANSA Climbing Frame* (AO.16)