On problems with data components during low-level modelling

Denis Buzdalov
ISPRAS

January 19, 2016

1 Introduction

In this paper we present some problems that can appear while using AADL in applications where data components are needed to be modelled precisely.

2 Background

The background of the problems was the simulation of systems modelled with AADL extended with behaviour specifications.

Behavior Annex specification language was used for behaviour of most threads and devices. Behaviour of hardware components was modelled with Java code using special simulation library.

Both approaches are abstract enough to use values of data components modelled with data representations defined in Data Modeling Annex (e.g. general integers of different sizes with or without signs, structures, strings and etc.).

Afterwards, a lower level behaviour specification were considered to be used. As an example, a real operating systems running in QEMU was used as the behaviour specification for processor components.

Binary representation of data values cannot be avoided at this level of abstractness. It was realized that semantics for data representation defined in Data Modeling Annex sometimes is not enough to perform binary transformations.

3 Problems

3.1 Endianness

Both AADL core and Data Modeling Annex lacks ability of definition of endianness of data components.

On the one hand, it is okay since AADL-level of modelling seems to be usually too abstract to be aware of endianness problems. But on the other hand, this actually leads to unability to check system properties related to interconnection of systems with different endianness.

Consider a model with the following properties:

- model contains two connected processors running a single process each;
- one process sends to another one data values of type `base_types::Integer_64` (defined in Data Modeling Annex) along a port connection;
- these two processors and, as a consequence, processes in fact use different endiannesses (but this information does not actually appear in the model).

During modelling of such system, we will not catch any errors because we have to interpret data going from output port of process as just an general 64-bit integer which is going to an input port of another process.

But if we build a real system based on this model, we will run into a problem of misinterpretation because of wrong endianness of different interconnected processors.

So, it means that neither AADL core, nor Data Model Annex do not contain any facilities to represent (and thus analyze) endianness-aware properties of models.

3.2 Order of subcomponents

Another problem is that order of subcomponents of data components is not defined (and is not actually clear, especially in some cases). The order can be significant, for example, for binary transformations of structure fields.

All such problems appear when we refine data components that represent structures. Consider a simple data structure definition which we will refine in some ways showing the problems.

```
data DataExample
properties
    data_model::data_representation => struct;
end DataExample;

data DataExample.basic
subcomponents
    a: base_types::Integer_32;
    b: base_types::Integer;
    c: base_types::Integer_8;
    s: base_types::String;
end DataExample.impl;
```

3.2.1 Addition of new element

Consider a child of a data component that adds a new element to it.

```
data DataExample.ext extends DataExample.basic
subcomponents
    d: base_types::Boolean;
end DataExample.ext;
```

The question is how the subcomponent `d` will be put in order with subcomponents `a`, `b`, `c` and `s`. Neither AADL core, nor Data Modeling Annex defines this.

3.2.2 Refinement of an existing element

We run into the very similar situation when we simply refine some subcomponent.

```
data DataExample.mod extends DataExample.basic
subcomponents
    b: refined to base_types::Integer_16;
end DataExample.mod;
```

Standard does not define the order when some element is refined.

This problems raises especially when subcomponents refinings are listed in some other order than they were declared initially.

```
data DataExample.mod2 extends DataExample.basic
subcomponents
    s: refined to base_types::String {
        source_data_size => 20 bytes;
    };
    b: refined to base_types::Integer_16;
end DataExample.mod2;
```

3.2.3 Substitution and match

We have classifiers and prototype substitution and classifiers matching rules in the standard. Despite "signature match" variant was being discussed to be discarded, it should be considered that order might influence on decisions of substituting and matching using "signature match", "equivalence" and "subset" rules.

Of course, order influences to these rules only when we model data values precise enough (i.e. on the later steps of the modelling process).

3.3 Data structures alignment

Another problem we can run into trying to do low-level modelling of values of data components is the data alignment.

Lots of real data structures are actually aligned. AADL data components do not have any standard properties that allows to model alignment when we need it (and we need it, for example, when we are modelling data structures precisely to be able to perform binary conversions).

Actually, this problem can be worked around at the current state of the standard (considering there are no problems with structure's subcomponents order). We can always insert padding subcomponents with `source_data_size` set for particular alignment. But this workaround looks ugly, nay it can easily lead to lots of errors, for example during change of architecture (thus, changing paddings in a lot of places).

4 On solutions

4.1 Endianness and alignment

For both problems the similar solution is proposed: it is proposed to simply add corresponding properties to Data Modeling Annex.

4.1.1 Endianness

It is proposed for potential endianness property to not to have any default value. When endianness is not set, it is proposed to be considered too abstract for this data component (i.e. this component is not applicable to binary transformations and precise-level data compatibility analysis).

Considering possible values for the endianness property, big-endian and little-endian should be considered doubtlessly. It is an open question how to represent different variants of mixed endiannesses (since the count of mixed endiannesses variants grows with the size of a number; and at the modern systems mixed endiannesses are rarely used).

This property should be definitely considered to be applicable to processors, not only for data components. It seems to be reasonable for this property to be inherited (in AADL terms).

4.1.2 Alignment

It seems that alignment property should have default value corresponding to no actual alignment.

In the simplest variant, alignment property can have type of memory size. Its meaning is a size, which a size of a structure field (including padding) have to be a multiple of.

4.2 Order of subcomponents

Considering the possible solutions of the problem of subcomponents order, at first, it should be decided whether data components should be considered to be always ordered or not. Both decisions have advantages and disadvantages.

Always If data values are considered to be always ordered, it makes rules defining the order more straight. But this can lead to breakage of some relations that are being used in models (like relations of equivalence or complementing).

Not always If data values are allowed to be not ordered, it allows the preciseness level of modelling to be set explicitly. At this case, standard must contain clear rules for determining whether this or that data value is considered to be ordered or not.

When data values are modelled with enough preciseness, they should be considered to have actual order. Presence of order makes such data values to be able to be transformed into and from the binary representation.

When data values are not modelled precise enough, no binary transformations and low-level analysis can be performed. Standard should explicitly set this point.

The disadvantage of this approach is that we need to define more complicated rules considering that data value can be ordered or not. Analyzers that are aware of binary representation of data values, will need to have a precondition on the model that data values have to be ordered.

But this approach also has an advantage that the current (abstract enough) models do not have to be changed or reconsidered according to addition of order rules.

Talking about semantics of “not always” variant, for some particular variants of data values it is a very important to decide whether special semantics of ordering should be

defined or this case should be considered unordered. We can illustrate it on the example of `DataExample.mod2`. Consider two possible decisions:

- primitive decision: to consider this case to not to have an order;
- decision with preserving of extension compatibility: to consider the original order of `DataExample.basic` to be unchanged on refinements.

The second decision is consistent with idea of classifiers extension: since `DataExample.mod2` extends `DataExample.basic`, we can use instance of `DataExample.mod2` as an instance of `DataExample.basic`. This principle implies a decision to have a rule about preserving the order on extension of components.

Actually, there are a lot of similar questions that should be decided deliberately and consequently.

5 Conclusion

There are some problems in precise data modelling in AADL. Both AADL core and Data Modeling Annex miss semantics for this.

These problems looks decidable. Some steps of solution were considered.

It is suggested to discuss these problems and to prepare some plan for a solution.