Groundwork

Lambert Meertens

Department of Algorithmics and Architecture, CWI, Amsterdam, and

Department of Computing Science, Utrecht University, The Netherlands

lambert@cwi.nl

June 23, 1998

What this is about

This documents lists some infrastructural needs in component-oriented software construction. These needs appear to form an intricate mesh of interrelated issues. An attempt has been made to unravel them, but more unraveling effort is needed in order to obtain greater conceptual clarity. The hope is that this will lead to a collection of infrastructural formalisms that can be used and maintained independently, but that have been designed in such a way that they can easily work together.

The aims of this kind of infrastructure include the following, partially overlapping, items:

- improved software productivity by building upon proven technology;
- improved interoperability by building upon shared technology;
- improved flexibility and extensibility;
- distributed and mobile functionality.

Some of the needs mentioned may be satisfactorily covered by existing technology. Their inclusion here should then be ascribed to ignorance rather than a desire to re-invent. However, proprietary, not freely available software will not do. Note also that a problem of some software is that it does "too much", rather than too little.

In this document the term *formalism* typically comprises, ideally:

- a collection of well-defined notions and attendant terminology;
- an interface description (such as formats and protocols);
- libraries implementing this.

1 Windowing interface

A platform-independent windowing interface.

2 Linear inequation solver

What is needed here is a *fast* incremental solver of a set of linear inequations. It is purely ancillary to presentation issues (3.5). The reason it is singled out is that such a solver can be designed and implemented in isolation. If the set of variables is $x_0, x_1, \ldots$, inequation i can be written in the form

$$\sum_j \beta_{ij} x_j \geq \lambda_i$$

Equations can be expressed as a pair of inequations. The set of inequations may be unsatisfiable. The solver should minimise the expression

$$\sum_{ij} (\beta_{ij} x_j \ominus \lambda_i)^2 \quad \text{where} \quad u \ominus v \;=\; \begin{cases} 0 & \text{if} \;\; u \geq v \\ |u - v| & \text{if} \;\; u < v \end{cases}$$

The sets of variables and of inequations may be large, but the matrix of βs is typically sparse. The solver should be robust (able to cope with rather ill-conditioned systems) and factor out independent subsystems.

3 Objects

Possibly most or all of the following is covered by CORBA2.

An *object* is something that has an *identity* and a current *value*. The notion of object identity makes it meaningful to say that the value of an object changes. No current value can be seen as a current value none.

3.1 Structure

We need a formalism for specifying structure. Structure is needed to assist the editing of objects (for example, it should be possible to cut and paste a 'section' of a document as a single entity), to specify presentation rules (see Section 3.5), and to ensure well-formedness of values that are submitted to further processing (see Section 6). The general idea is rather like that of a *type* in programming languages, and we use that term as shorthand for structure specification.

We need to distinguish between the type of an object and the type of a value it possesses. The type of an object might be Any, while its current value has type Text. In a consistent world, however, the type of the value of an object should at any time "fit" the type of the object.

The structure formalism should be flexible enough to allow the description of:

- conventional text (title, author, table of contents, sections, appendices, index);
- pictures (diagrams, bitmaps);
- tuples, tables, sets, bags, lists;
- music;
- mathematics;
- references (hyperlinks, entries of 'folder index');
- widgets (buttons, "hot spots", etc.);
- any combination of the above.

It should cover a range from large freedom (with Any as the supremum) to complete fixation, with anything else in between. In addition, it is desirable to be able to specify:

- overridable default types;
- constraints between elements or subobjects, both within one object and as inter-object constraints;
- attributes.

In terms of the implementation, it should be possible to represent an object by a reference to it (see Section 3.3), to be expanded "by need".

Research questions:

- Is the type of a given object immutable? That is not very flexible. But if it can change, what are the implications, and how can these be handled?
- Can an object have more than one type?
- Do subobjects necessarily have a unique parent?

3.2 Object Management

It is not clear what belongs specifically here, but part of it would be a notion of "obligatory" attributes, among which something like the type, the capabilities (see Section 5) and a (possibly virtual) origin document giving information about how the object came into existence, what its purpose is, etc.

3.3 References

A reference is like an URL; in fact, in our reference formalism URLs should be acceptable references. However, more is needed. A reference should not necessarily specify the exact location of an object, but be able to use a name server. Finally, it should be possible to use pointers (in the C sense) as a local, non-exportable reference.

As with CGI scripts, the object referred to may possibly be created in response to an operation deref.

Among the "exceptions" of the deref protocol should be something like: *Request received; object will be shipped in an estimated 15 seconds*.

Research questions:

- Wide-area distributed garbage collection.
- Robust references for mobile objects.
- Caching; replication; marshalling.

3.4 Surgery

The term *surgery* refers to "edit" operations on objects. The term *editing* will be reserved for visual, interactive editing.

Needed is a subformalism for superseding objects with new values, including replacing subobjects.

Research questions:

- Serialisability, concurrency control etc.
- Undo.
- Notification services.

3.5 Presentation

Needed is a formalism for a special kind of objects (presentation objects) embodying a sort of picture algebra; in first approximation boxes combined with glue. Additionally, it must be possible to specify positional constraints as linear inequalities (see Section 2).

Next, we need a formalism for specifying the presentation of objects, that is, rules for deriving a presentation object from a given object, such that it can be determined which original subobject(s) give rise to a selected "part" of a presentation. This requires a subformalism of *selections*.

Selections should not necessarily be contiguous.

A given object may have many presentations, and within a presentation object the presentation of subobjects should be changable, as with the template versions of Mathspad. An original subobject may be involved in incomparable selections, for example a matrix entry in a row or column selection.

4 Editing

Needed is one customisable but *generic* editor. Although built on top of some presentation formalism (see Section 3.5) and some object-surgery formalism (see Section 3.4), it should ideally not have to know the details, but have a thin interface with these.

For plain text, the generic formalism should *automatically* specialise to a usable plain-text editor. For objects with a rigid presentation, such as forms, the editor should likewise specialise to a forms editor. For immutable objects the editor is still a convenient browser.

Research questions:

- "Generalised paste": `paste` should succeed both when the types fit but the presentations don't, and when the presentations fit but the types don't, and this recursively.
- Scripting.
- Write-through control.

5 Capabilities

A formalism for access and modification rights.

6 Directional constraints

A formalism for maintaining a (distributed) constraint network.

Research questions:

- Incremental change propagation, generalising the current notion of *structure watchers*.
- Cycles.
- Serialisability, concurrency control, undo.
- Granularity control.
- Normalisation.
- Error handling.

7 Text

A formalism for dealing with notions like *character*, *glyph*, *ligature*, *diacritics*, etc.

A formalism for font families, styles, etc.

A formalism for language-dependent issues like hyphenation and collation (sorting).