Chapter 6: Implementation details

Introduction

This chapter discusses the main building blocks that are necessary to implement ARGOS such as the life cycle information infrastructure, constraints and the design of a conceptual ARGOS. It explores the role that ARGOS could play in structured planning and design knowledge delivery. The relationships between the ARGOS, ActiveX design object and typical applications software are explored.

The basis for successful implementation is the formulation of a flexible and self-describing design language. From the previous chapters and a study of ontology it is clear that a design language with a hierarchical structure best facilitates the processing of design knowledge fragments. A successful design is the result of many different cognitive processes at both tacit and explicit levels. These processes can be augmented with many different techniques from the world of manufacturing such as Knowledge Based Design, Systems Analysis, Kansei Engineering, QFD and TRIZ. Many other techniques could be discussed such as FMEA (Failure Mode and Effects Analysis), but it would not contribute significantly to the problem under consideration, the storing of artefact design knowledge over the life cycle of a building and the secondary adaptation of designs.

Once a design is available it can be brought into the ARGOS component (container) to facilitate the positional and shape testing of design fragments. All of this must happen in a neutral environment to guarantee a long life of design information and make it possible for diverse design tools to process relevant parts of the information.

6.1 Life cycle Information infrastructure

6.1.1 Introduction

Of all the possible candidates investigated Extensible Mark-up Language (XML) proved to be most useful language to solve the stringent requirements for the problem under consideration.

XML, describes a class of data objects called XML documents and partially describes the behaviour of computer programs which process them. XML is an application profile or restricted form of the Standard Generalised Mark-up Language (SGML). XML documents are made of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data and some, which form the mark-up structure. Mark-up encodes a description of the document's storage layout and logical structure. XML also provides a mechanism to impose constraints on the storage layout and logical structure. A software module called an *XML processor* is used to read XML documents and provide access to their content and structure.

XML was developed by an XML Working Group formed under the auspices of the World Wide Web Consortium (W3C) in 1996. It was chaired by Bosak of Sun Microsystems with the active participation of an XML Special Interest Group also organised by the W3C.

The primary design goals for XML are:

- XML shall be straightforwardly usable over the Internet.
- XML shall support a wide variety of applications.
- XML shall be compatible with SGML.
- It shall be easy to write programs which process XML documents.

- The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
- XML documents should be in human-legible form and reasonably clear.
- The XML design should be prepared quickly.
- The design of XML shall be formal and concise.
- XML documents shall be easy to create.
- Terseness in XML mark-up is of minimal importance.

6.1.2 XML as a design language

Consider Code Fragment 1 below. This is a trivial example of how a materials library could be structured by means of XML. This offers the immediate advantage that the information can be used in other applications and downloaded from the Internet. This structure could be used to implement the Materials Library as detailed in Figure 52 [E1]. In this case the material library starts with the `<MATERIAL_LIBRARY>` label and ends with the `</MATERIAL_LIBRARY>` label. Each separate material starts with the label `<MATERIAL keyword1=”METAL”>` and ends with a `</MATERIAL>`. Hierarchically nested under this is the `<DESCRIPTION>` label that contains a short description of the material.

`<DESCRIPTION>Aluminium (Al) 99.0% pure</DESCRIPTION>`

This is followed by the list of applicable attributes. Note that the attributes are grouped within the attribute label for example:

`<A unit="kg/m3" value="2650.000">Density</A>`

There are no hard and fast rules when to use child elements and when to use attributes. Generally the application developer uses whichever suits his application. A rule of thumb is that data themselves should be stored in elements. Information about the data (meta-data) should be stored in attributes (Harold 1999:101).

Code Fragment 1 could be generated by means of many different methods such as:

- Output from a relational database.
- Dynamic upon demand generation by a web based search engine or query builder.

Domain specific application software such as a design scenario builder could use the basic information contained in the database or could present it in neatly formatted document for reference purposes.

```
<?xml version="1.0" standalone="yes"?>
<?xml-stylesheet type="text/xsl" href="Material_fragment.xsl"?>
        <MATERIAL_LIBRARY>
   <MATERIAL keyword1="METAL">
                <NAME>ALUMINIUM</NAME>
                <DESCRIPTION>Aluminium (Al) 99.0% pure</DESCRIPTION>
                <ATTRIBUTES>
                        <A unit="kg/m3" value="2650.000">Density</A>
                        <A unit="Deg C" value="660.000">Melting point</A>
                        <A unit="N/mm2" minvalue="68300.000">Modulus of elasticity (minimum)</A>
                        <A unit="N/mm2" value="70350.000">Modulus of elasticity (average)</A>
                        <A unit="N/mm2" maxvalue="72400.000">Modulus of elasticity (maximum)</A>
                        <A unit="W/m deg C" value="214.000">Thermal conductivity (k)</A>
                </ATTRIBUTES>
        </MATERIAL>
        <MATERIAL keyword1="METAL">
                <NAME>ALUMINIUM BRONZE</NAME>
                <DESCRIPTION>Aluminium-Bronze Cu 5-10%: Al</DESCRIPTION>
                <ATTRIBUTES>
                        <A unit="kg/m3" minvalue="7570.000">Density (minimum)</A>
                        <A unit="kg/m3" value="2650.000">Density (average)</A>
                        <A unit="kg/m3" maxvalue="8150.000">Density (maximum)</A>
```

```
                        <A unit="Deg C" minvalue="1041.000">Melting point (minimum)</A>
                        <A unit="Deg C" value="1052.000">Melting point (average)</A>
                        <A unit="Deg C" maxvalue="1063.000">Melting point (maximum)</A>
                        <A unit="N/mm2" value="120000.000">Modulus of elasticity</A>
                        <A unit="W/m deg C" minvalue="64.000">Thermal conductivity (k) (minimum)</A>
                        <A unit="W/m deg C" value="74.500">Thermal conductivity (k) (average)</A>
                        <A unit="W/m deg C" maxvalue="85.500">Thermal conductivity (k) (maximum)</A>
                </ATTRIBUTES>
        </MATERIAL>
        <MATERIAL keyword1="METAL">
                <NAME>BRASS</NAME>
                <DESCRIPTION>Brass Cu 60%: Zn 40%</DESCRIPTION>
                <ATTRIBUTES>
                        <A unit="kg/m3" value="8380.000">Density</A>
                        <A unit="Deg C" value="904.000">Melting point</A>
                        <A unit="N/mm2" value="103000.000">Modulus of elasticity</A>
                        <A unit="W/m deg C" value="129.000">Thermal conductivity (k)</A>
                </ATTRIBUTES>
        </MATERIAL>
        <MATERIAL keyword1="WOOD" keyword2="CONSTRUCTION">
                <NAME>PINE</NAME>
                <DESCRIPTION>British Columbia pine</DESCRIPTION>
                <ATTRIBUTES>
                        <A unit="kg/m3" minvalue="480.000">Density (minimum)</A>
                        <A unit="kg/m3" value="600.000">Density (average)</A>
                        <A unit="kg/m3" maxvalue="720.000">Density (maximum)</A>
                        <A unit="years" minvalue="10.0">Durability (minimum)</A>
                        <A unit="years" value="12.5">Durability (average)</A>
                        <A unit="years" maxvalue="15.0">Durability (maximum)</A>
                </ATTRIBUTES>
        </MATERIAL>
</MATERIAL_LIBRARY>
```

Code Fragment 1: Suggested XML structure for the storage of material definitions (Author)

Code Fragment 1 could be formatted, for reporting purposes, at the most basic level by means of Cascading Style Sheets (CSS). CSS styles only apply to XML element content not to attributes in the elements. If CSS were applied to Code Fragment 1 in an Internet Explorer then the attributes would be invisible rendering most of Code Fragment 1 data invisible. However there is an alternative style sheet language that allows the user to access and display attribute data as well. This language is Extensible Style language (XSL). XSL is divided into two main sections:

- Transformations
- Formatting

Consider Code Fragment 2 for an example of a typical XSL that could be used to convert Code Fragment 1 into a neatly formatted output for a web page.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
        <xsl:template match="/">
                <html>
                        <xsl:apply-templates/>
                </html>
        </xsl:template>

        <xsl:template match="/MATERIAL_LIBRARY">
                <html>
                        <body>
                                <h1>Example Material Library</h1>
                                <xsl:apply-templates/>
                        </body>
                </html>
        </xsl:template>
```

```
        <xsl:template match="MATERIAL">
                <p>
                        <h3><u><xsl:value-of select="NAME"/></u></h3>
                        <xsl:apply-templates/>
                        <br><hr></hr></br>
                </p>
        </xsl:template>

        <xsl:template match="ATTRIBUTES">
                        <xsl:apply-templates/>
        </xsl:template>

        <xsl:template match="A">
                 <br><i><xsl:value-of select="."/> = </i>
                 <b><xsl:value-of select="@minvalue"/></b>
                 <b><xsl:value-of select="@value"/></b>
                 <b><xsl:value-of select="@maxvalue"/></b>
                 <xsl:value-of select="@unit"/></br>
        </xsl:template>
</xsl:stylesheet>
```

Code Fragment 2: Typical style sheet to format XML data for web page display (Author)

If the XSL in Code Fragment 2 is applied to Code Fragment 1 the output looks like Code Fragment 3. At this stage the XML data in Code Fragment 1 can be used for two entirely different purposes:

- The transfer of structured material attributes for design purposes
- The display of the material characteristics in a web page

Example Material Library

ALUMINIUM

Density = **2650.000** kg/m3

Melting point = **660.000** Deg C

Modulus of elasticity (minimum) = **68300.000** N/mm2

Modulus of elasticity (average) = **70350.000** N/mm2

Modulus of elasticity (maximum) = **72400.000** N/mm2

Thermal conductivity (k) = **214.000** W/m deg C

ALUMINIUM BRONZE

Density (minimum) = **7570.000** kg/m3

Density (average) = **2650.000** kg/m3

Density (maximum) = **8150.000** kg/m3

Melting point (minimum) = **1041.000** Deg C

Melting point (average) = **1052.000** Deg C

Melting point (maximum) = **1063.000** Deg C

Modulus of elasticity = **120000.000** N/mm2

Thermal conductivity (k) (minimum) = **64.000** W/m deg C

Thermal conductivity (k) (average) = **74.500** W/m deg C

Thermal conductivity (k) (maximum) = **85.500** W/m deg C

BRASS

Density = **8380.000** kg/m3

Melting point = **904.000** Deg C

Modulus of elasticity = **103000.000** N/mm2

Thermal conductivity (k) = **129.000** W/m deg C

Code Fragment 3: Output generated by Code Fragment 2 applied to Code Fragment 1 (Author)

Consider the formatted output in code fragment 3 that is achieved by means of the XSL code in Code Fragment 2. In this case the output generated from the XML in Code Fragment 1 is HTML format that makes it suitable for direct display in web pages.

Code Fragment 4 below is an example of how the information of a CAD system such as MicroGDS 6.0 is expressed in XML. This system was the first to offer the ability to translate CAD drawings into XML. The XML output is translated into Vector Mark-up Language (VML) by means of a style file. VML is an XML application that combines vector

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE MicroGDS PUBLIC "+//IDN informatix.co.uk//MicroGDS600 DTD//EN" "file://c:\usr\PhD\Xml\MicroGDS PhD XML\MicroGDS600.dtd">
<MicroGDS>
<StylePath/>
<Aliases/>
<Styles>
<CVCharstyle Name="18" Height="1.8" FontName="DEFAULT"/>
<CVCharstyle Name="25" Height="2.5" Width="2.5" FontName="DEFAULT"/>
<CVCharstyle Name="35" Height="3.5" Width="3.5" FontName="DEFAULT"/>
<TTCharstyle Name="AR06" Height="6E-1" Width="2.8E-1" FontName="Arial" Weight="Normal" Underline="false" StrikeOut="false" Italic="false" Pitch="Variable"
Family="Swiss"/>
<TTCharstyle Name="AR10" Height="3.5" Width="1.48" FontName="Arial" Weight="Normal" Underline="false" StrikeOut="false" Italic="false" Pitch="Variable"
Family="Swiss"/>
<TTCharstyle Name="AR12" Height="2.5" Width="1" FontName="Arial" Weight="Normal" Underline="false" StrikeOut="false" Italic="false" Pitch="Variable" Family="Swiss"/>
...
<Linestyle Name=".00" Font="DEFAULT" Border="true" Opaque="false" SymbolHeight="2.5" Pen="0" Gap="2"/>
<Linestyle Name=".18" Font="DEFAULT" Border="true" Opaque="false" LeftOffset="4E-2" RightOffset="-4E-2" SymbolHeight="2.5" Pen="0" Gap="2"/>
<Linestyle Name=".25" Font="DEFAULT" Border="true" Opaque="false" LeftOffset="9E-2" RightOffset="-9E-2" SymbolHeight="2.5" Pen="0" Gap="2"/>
…
<Linestyle Name="CENT1" Font="SYMBOL" Border="true" Opaque="false" LeftOffset="5E-2" RightOffset="-5E-2" SymbolHeight="5" Phasing="Line">
<FixedLine Length="7"/>
<EndOfStart/>
<VariableGap Length="1"/>
<VariableLine Length="2"/>
<VariableGap Length="1"/>
<VariableLine Length="2"/>
<VariableGap Length="1"/>
<VariableLine Length="8"/>
<StartOfEnd/>
</Linestyle>
…
<TextMnemonic Name="RGUI" Prompt="Attach  Microsoft global unique identifier" MinLineLength="1"/>
</Styles>
…
<Layer>
<Layer Name="STRUC" Label="" LinkNumber="34" HighestObjectLinkNumber="290">
<Extent LX="0" LY="-2.0927500247955322E2" LZ="0" HX="1.7490625047683716E2" HY="2.0543749952316284E2" HZ="0"/>
…
```

```
<OCD Name="STRUC:STAIR">
<Object LinkNumber="281" HighestPrimitiveLinkNumber="41" Lightstyle="NONE" ContainsItems="false">
<Extent LX="1.0527500009536743E2" LY="-1.2537845563888855E2" LZ="0" HX="1.4290635883501973E2" HY="-
9.465041399002075 2E1" HZ="0"/>
<Axes Y="-3" RZ="-7.853981475830078 6E-1" S="5E1"/>
<TextPrimitive LinkNumber="41" Mirrored="false" Box="false" Dim="false" Data="false" YFactor="1.1281112432479858"
Charstyle="TNR25" Justification="BC">
<Extent LX="1.2659249997138977E2" LY="-1.2327500009536743E2" LZ="0" HX="1.3083250021934509E2" HY="-
1.1947499952266084E2" HZ="0"/>
<Axes X="1.2871250009536743E2" Y="-1.2327500009536743E2" S="8.864375359234303 3E-1"/>
<DefinitionText>UP
</DefinitionText>
</TextPrimitive>
<LinePrimitive LinkNumber="40" Mirrored="false" Linestyle=".00" StartMark="true" EndMark="true">
<Extent LX="1.1563856649398804E2" LY="-1.0827500152587891E2" LZ="0" HX="1.1681673431396484E2" HY="-
1.0694840526580811E2" HZ="0"/>
<Polyline>
<Point X="1.1681673431396484E2" Y="-1.0764382553100586E2"/>
<Point X="1.1576224994659424E2" Y="-1.0827500152587891E2"/>
<Point X="1.1563856649398804E2" Y="-1.0694840526580811E2"/>
</Polyline>
...
</Object>
</OCD>
</Layer>
...
</MicroGDS>
```

Code Fragment 4: Structure of a MicroGDS 6.0 CAD file described with XML (Author)

information with CSS markup to describe vector graphics that can be embedded in Web pages in stead the bitmapped GIF and JPEG images loaded by HTML's IMG element. VML is supported by the various components of Microsoft Office 2000 as well as by Internet Explorer 5.0.

The W3C has received four different proposals for vector graphics in XML from a wide variety of vendors. It's formed the Scalable Vector Graphics (SVG) working group composed of representatives from all these vendors to develop a single specification for an XML representation of Scalable Vector Graphics. When SVG is complete it should provide everything VML currently provides plus a lot more including animation, interactive elements,

filters, clipping, masking and pattern fills. A full SVG specification and the software that implements the specification are some time away.

The World Wide Web Consortium released the first working draft of SVG in February 1999 and revised the draft in April 1999. A well advanced working draft appeared 29 June 2000. Microsoft has stated publicly that they intend to ignore any Web graphics efforts except VML.

Code Fragment 4 contains a portion of an XML file that encodes the graphics of a CAD drawing. The XML data starts with the `<MicroGDS>` label and ends with `</MicroGDS>`. In this case the `<StylePath/>` and `<Aliases/>` labels are empty. The next section between the `<Styles>` and `</Styles>` labels defines the various character and linestyles as well as the mnemonics for the attribute data that could be attached to drawing objects (coloured in blue). The particular CAD system under consideration supports both vector type and true type character styles. The former is indicated in a label such as

```
<CV Charstyle Name=”18” Height=”1.8” FontName=”Default”/>
```

and the latter by

```
<TT Charstyle Name=”AR06” Height=”6E-1” Width=”2.8E-1”
FontName=”Arial” Weight=”Normal” Underline=”false”
Strikeout=”false” Italic=”false” Pitch=”Variable”
Family=”Swiss”/>
```

Linestyles could be simple or complex. A typical simple linestyle of .18 mm thickness is described by:

```
<Linestyle Name=”.18” Font=”DEFAULT” Border=”true”
Opaque=”false” Leftoffset=”4E-2” RightOffset=”-4E-2”
SymbolHeight=”2.5” Pen=”0” Gap=”2”/>
```

A more complex linestyle that contains patterns is described by:

```
<Linestyle Name=”CENT1” Font=”SYMBOL” Border=”true”
Opaque=”false” LeftOffset=”5E-2” RightOffset=”-5E-2”
SymbolHeight=”5” Phasing=”Line”>
<FixedLine Length=”7”/>
<EndOfStart/>
<VariableGap Length=”1”/>
<VariableLine Length=”2”/>
<VariableGap Length=”1”/>
<VariableLine Length=”2”/>
<VariableGap Length=”1”/>
<VariableLine Length=”8”/>
<StartOfEnd/>
</Linestyle>
```

The `<TextMnemonic>` label contains the definition of attribute data templates that could be used in this particular case to attach non-graphical information to the graphical objects. In this example a mnemonic called RGUI has been defined. The R in RGUI indicates that the data will apply to a specific instance (reference) of a graphical object. GUI is a mnemonic for Global Unique Identifier. This method has been used in the precedent system AEDES to connect alphanumeric data and graphical data. This particular aspect will have to be developed much further to accommodate the various levels of specificity required for CBR as well as to facilitate constraint propagation, tacit and explicit requirements of design.

The actual graphical data are contained between the <Layer> and </Layer> labels. In this system graphical data must occur on a layer although it is not a layer-based system. In this case the layer under consideration is "STRUC" that indicate that graphical and textual entities related to the structure of the building should be on this layer. The first graphical object is indicated by the <OCD Name="STRUC:STAIR"> label. This label is closed by the matching </OCD> label lower down. Within the bounds of the <Object> and </Object> labels the graphical text and lines are defined. The part related to text is indicated in blue and the part related to the graphical entities such as polylines in red. The text part is bounded by the <TextPrimitive> and </TextPrimitive> labels. The polylines are bounded by the <Polyline> and </Polyline> labels.

The hierarchical nature of the graphical example object conceptually follows the hierarchical structure of:

```
<Layer>
      <OCD>
            <Object>
                  <TextPrimitive>
                  </TextPrimitive>
                  <Polyline>
                  </Polyline>
            </Object>
      </OCD>
</Layer>
```

This forms a useful basis for a design language on which the more extensive requirements of a CBR system that supports design scenario generation and suspension of partially completed designs can be built.

The integrity of Code Fragment 4 is supported by an extensive Document Type Definition (DTD). A DTD provides a list of the elements, attributes, notations and entities contained in a document as well as their relationships to one another. DTDs specify a set of rules for the structure of a document. The DTD accomplishes this with a list of mark-up declarations for particular elements, entities, attributes and notations.

Consider Code Fragment 5 below for a shortened example of a DTD that ensures the integrity of the XML in Code Fragment 4. Only the entities used in Code Fragment 4 are included. The DTD is not necessary if the output is generated by an application. If XML fragments are obtained from other external sources then the DTD ensures conformance to the design language.

```
<!-- Styles -->

<!ELEMENT Styles (
    CVCharstyle|TTCharstyle|
    Linestyle|Material|Lightstyle|
    TextMnemonic|WordMnemonic|DoubleMnemonic|SingleMnemonic|IntegerMnemonic
)*>

<!-- Character Styles
A character style is either defined as a CAD Vector font or a (Windows)
True-Type font.
-->

<!-- Attributes common to character styles -->

<!ENTITY % CharstyleAttributes '
    Name        CDATA           #REQUIRED
    Height      CDATA           #REQUIRED
    Width       CDATA           #IMPLIED
```

```
    Pen         CDATA           "1"
'>

<!-- CAD Vector Fonts. The FontName attribute is a font name. -->

<!ELEMENT CVCharstyle EMPTY>
<!ATTLIST CVCharstyle
    %CharstyleAttributes;
    FontName    CDATA           #REQUIRED
>

<!-- True-Type (Windows) font -->

<!ELEMENT TTCharstyle EMPTY>
<!ATTLIST TTCharstyle
    %CharstyleAttributes;
    FontName    CDATA           #REQUIRED
    Weight      (DontCare|Thin|ExtraLight|Light|Normal|Medium|SemiBold|Bold|
                ExtraBold|Heavy)
                                "DontCare"
    Underline   (true|false)    "false"
    StrikeOut   (true|false)    "false"
    Italic      (true|false)    "false"
    Pitch       (Default|Fixed|Variable)
                                "Default"
    Family      (Decorative|DontCare|Modern|Roman|Script|Swiss)
                                "DontCare"
    CharSet     (ANSI|Baltic|ChineseBig5|Default|EastEurope|GB2312|Greek|
                Hangul|Mac|OEM|Russian|ShiftJIS|Symbol|Turkish|Hebrew|Arabic|
                Thai)
                                "ANSI"
>

<!-- Line styles -->

<!ELEMENT Linestyle (
   (FixedLine|FixedGap|Symbol)*,
   EndOfStart,
   (FixedLine|FixedGap|VariableLine|VariableGap|Symbol)*,
   StartOfEnd,
   (FixedLine|FixedGap|Symbol)*
)?>
<!ATTLIST Linestyle
   Name         CDATA           #REQUIRED
   Font         CDATA           #IMPLIED
   VertexStart  CDATA           #IMPLIED
   VertexInternal CDATA         #IMPLIED
   VertexEnd    CDATA           #IMPLIED
   VertexMidPoint CDATA         #IMPLIED
   SegLineStart CDATA           #IMPLIED
   SegLineEnd   CDATA           #IMPLIED
   SegSegStart  CDATA           #IMPLIED
   SegSegEnd    CDATA           #IMPLIED
   FillSymbol   CDATA           #IMPLIED
   Border       (true|false)    "true"
   Opaque       (true|false)    "false"
   LeftOffset   CDATA           "0"
   RightOffset  CDATA           "0"
   SymbolHeight CDATA           "2.5"
   Pen          CDATA           "1"
   Phasing      (None|Angle|Line|Grid)
                                "None"
   Fill (None|HatchHorizontal|HatchVertical|HatchFDiagonal|HatchBDiagonal|
        HatchCross|HatchDiagCross|Solid0|Solid1|Solid5|Solid10|Solid15|Solid20|
        Solid25|Solid30|Solid35|Solid40|Solid45|Solid50|Solid60|Solid70|
        Solid80|Solid90|Solid100|BrushBDiagonal|BrushCross|BrushDiagCross|
        BrushFDiagonal|BrushHorizontal|BrushVertical|FillSymbol)
                                "None"
   Gap          CDATA           #IMPLIED
   Space        CDATA           #IMPLIED
   Shear        CDATA           #IMPLIED
   Slope        CDATA           #IMPLIED
>

<!-- Linestyle pattern elements -->
```

```
<!ELEMENT EndOfStart EMPTY>
<!ELEMENT StartOfEnd EMPTY>

<!ELEMENT FixedLine EMPTY>
<!ATTLIST FixedLine
   Length       CDATA           #REQUIRED
>

<!ELEMENT FixedGap EMPTY>
<!ATTLIST FixedGap
   Length       CDATA           #REQUIRED
>

<!ELEMENT VariableLine EMPTY>
<!ATTLIST VariableLine
   Length       CDATA           #REQUIRED
>

<!ELEMENT VariableGap EMPTY>
<!ATTLIST VariableGap
   Length       CDATA           #REQUIRED
>

<!ELEMENT Symbol EMPTY>
<!ATTLIST Symbol
   Symbol       CDATA           #REQUIRED
>

<!-- Mnemonic definitions -->

<!ENTITY % MnemonicAttributes '
    Name        CDATA           #REQUIRED
    Prompt      CDATA           ""
'>

<!ELEMENT TextMnemonic EMPTY>
<!ATTLIST TextMnemonic
    %MnemonicAttributes;
    MaxLines      CDATA         "1"
    MinLineLength CDATA         "0"
    MaxLineLength CDATA         "132"
>

<!-- Layers -->

<!ELEMENT Layer (Extent?, (Attribute|OCD)*)>
<!ATTLIST Layer
    Name        CDATA           #REQUIRED
    Label       CDATA           ""
    LinkNumber  CDATA           #IMPLIED
    HighestObjectLinkNumber CDATA #IMPLIED
    GUID        CDATA           #IMPLIED
>

<!-- OCD
This is the top-level element for an Object which is a container for the object
name. OCD is short for Object Code (ie name) Definition.
-->

<!ELEMENT OCD (Attribute|Object|ObjectInstance)*>
<!ATTLIST OCD
    Name        CDATA           #REQUIRED
>

<!-- Objects -->

<!ELEMENT Object (Extent?, Axes,
    (Attribute|LinePrimitive|TextPrimitive|RasterPhotoPrimitive|
    WindowPhotoPrimitive|OlePhotoPrimitive|ClumpPrimitive)*
)>
<!ATTLIST Object
    LinkNumber  CDATA           #IMPLIED
    HighestPrimitiveLinkNumber CDATA #IMPLIED
    Lightstyle  CDATA           "NONE"
    ContainsItems (true|false)  "false"
>
```

```
<!-- Line Primitive -->

<!ELEMENT LinePrimitive (%PrimitiveContent;, Polyline)>
<!ATTLIST LinePrimitive
     %PrimitiveAttributes;
    Linestyle   CDATA           "DEFAULT"
    StartMark   (true|false)    "true"
    EndMark     (true|false)    "true"
>

<!-- Text Primitive -->

<!ELEMENT DefinitionText (#PCDATA)>
<!ELEMENT ExpandedText   (#PCDATA)>

<!ELEMENT TextPrimitive (
    %PrimitiveContent;, Axes,
    DefinitionText, ExpandedText?
)>

<!ATTLIST TextPrimitive
    %PrimitiveAttributes;
    Charstyle   CDATA           "DEFAULT"
    Linestyle   CDATA           #IMPLIED
    Justification (TL|TC|TR|CL|CC|CR|BL|BC|BR)
                                "BL"
    Box         (true|false)    #IMPLIED
    Dim         (true|false)    #IMPLIED
    Data        (true|false)    #IMPLIED
    YFactor     CDATA           "1"
>

<!-- Points simply consist of x,y,z coordinates -->

<!ELEMENT Point EMPTY>
<!ATTLIST Point
    X           CDATA           "0"
    Y           CDATA           "0"
    Z           CDATA           "0"
>

<!-- As far as the DTD is concerned, a vector is equivalent to a point -->

<!ELEMENT Vector EMPTY>
<!ATTLIST Vector
    X           CDATA           "0"
    Y           CDATA           "0"
    Z           CDATA           "0"
>

<!-- BulgeAxis - this is a bulge factor
The B attribute represents the bulge factor, a value between 0 and 1. -->

<!ELEMENT BulgeAxis EMPTY>
<!ATTLIST BulgeAxis
    B           CDATA           "0"
    X           CDATA           "0"
    Y           CDATA           "0"
    Z           CDATA           "0"
    A           CDATA           #IMPLIED
>

<!-- Polyline
Start point, followed by a sequence of line segments (curved or straight).  If
the first and last points are the same, the polyline is closed.
-->

<!ELEMENT Polyline (Point, (BulgeAxis?, Point)*)>
```

Code Fragment 5: Partial MicroGDS 6.0 XML Document Type Definition (DTD) described with XML (Author)

At this stage it is possible to implement the conceptual design processor illustrated in Figure 46.

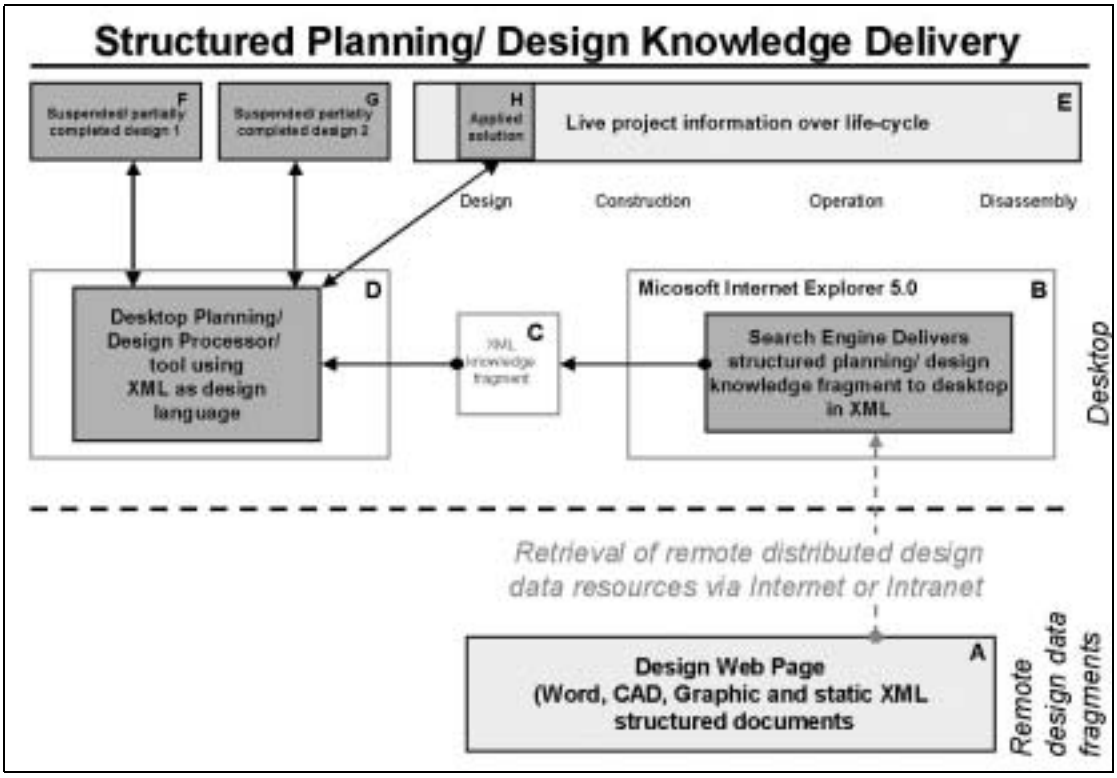

Figure 1: Structured Planning/ Design Knowledge Delivery (Author)

The design knowledge delivery system will conceptually work as detailed in Figure 46. A designer that wants to design a facility or solve a specific operational problem will activate a purpose made search engine [B] in Microsoft Internet Explorer. The search engine [B] will enable the user to set basic constraints and search criteria in order to expedite information retrieval. If the relevant information is found it will be packaged in the form of XML design knowledge fragments. The user can first view the result in Internet Explorer and if he is satisfied ask the system to download it to the desktop. The desktop planning/ design processor [D] will retrieve the downloaded XML knowledge fragment [C]. Due to the fact that design takes place in an open world it is expected that many different planning concepts might exist that need to be explored. These partially completed scenarios are stored in [F] and [G] again in XML format. Once the planner is satisfied the solution can be plugged into a live project environment [H]. It is also possible to publish good designs back into an office web page [A] to make them available to other designers.

[D] could be seen as *working memory* (WM), [F] and [G] as *long term memory* (LTM) (Simina 1999:39-43). The main purpose of WM is:

- Promote synergy among design parts
- WM facilitate external and internal event detection and processing
- WM keeps a limited store of recently accessed artefacts

The purpose of LTM is:

- Main repository of past design or design fragments
- Retrieval from LTM could be based on any combination constraints or functions

The XML Fragment interchange working draft (W3C 1999) defines a way to send fragments of an XML document to an XML user, in this case the designer using the desktop/ planning processor. It must be emphasised that although Figure 46 is an oversimplified example the following important principles are used:

- The designer remains in full control of the ultimate solution at all times
- Design experience is stored in a structured format (the beginning of CBR)
- Most information required in the planning and design environments are basically hierarchical and occur at various levels of specificity
- XML supports the inclusion of non-XML data and can act as an integrator of diverse data sources
- XML supports distribution of data as well as data hyper linking
- XML supports multi-media data sources
- The example attempts to support design as a pragmatic as well as a cognitive activity
- The solution assumes that planning and design requires a continuum of design methods that use model based, rule based and case-based reasoning. It is ultimately up to the designer to decide what method he prefers
- Current relational databases such as Oracle already support the generation of XML data from a relational query

By means of a style sheet defined in XSL it is possible to display the XML such as Code Fragment 4 in vector format in a web page (Figure 47). For a complete listing of the style sheet please consult Appendix D. The style sheet converts the XML code into Microsoft VML format that makes the display in a web page possible.

The display as illustrated in Figure 47 was done in the smallest possible custom developed web browser for the following reasons:

- To test the feasability of a thin browser developed in Visual Basic by means of the convenient *Inet* ActiveX control.
- To facilitate retrieval of XML code fragments in the ARGOS autonomous design objects a small Internet Explorer is required that has the ability to interpret the XML, stylesheets and DTDs.

The actual code required to implement the minimal browser is included in Appendix F. The browser was tested by means of a small test web page run on a personal computer by means of the Personal Web Server provided with Microsoft Windows 98. Only minimal functionality is provided but enough to facilitate connection to any potential design site in the world or design knowledge fragment on the personal machine or Intranet.

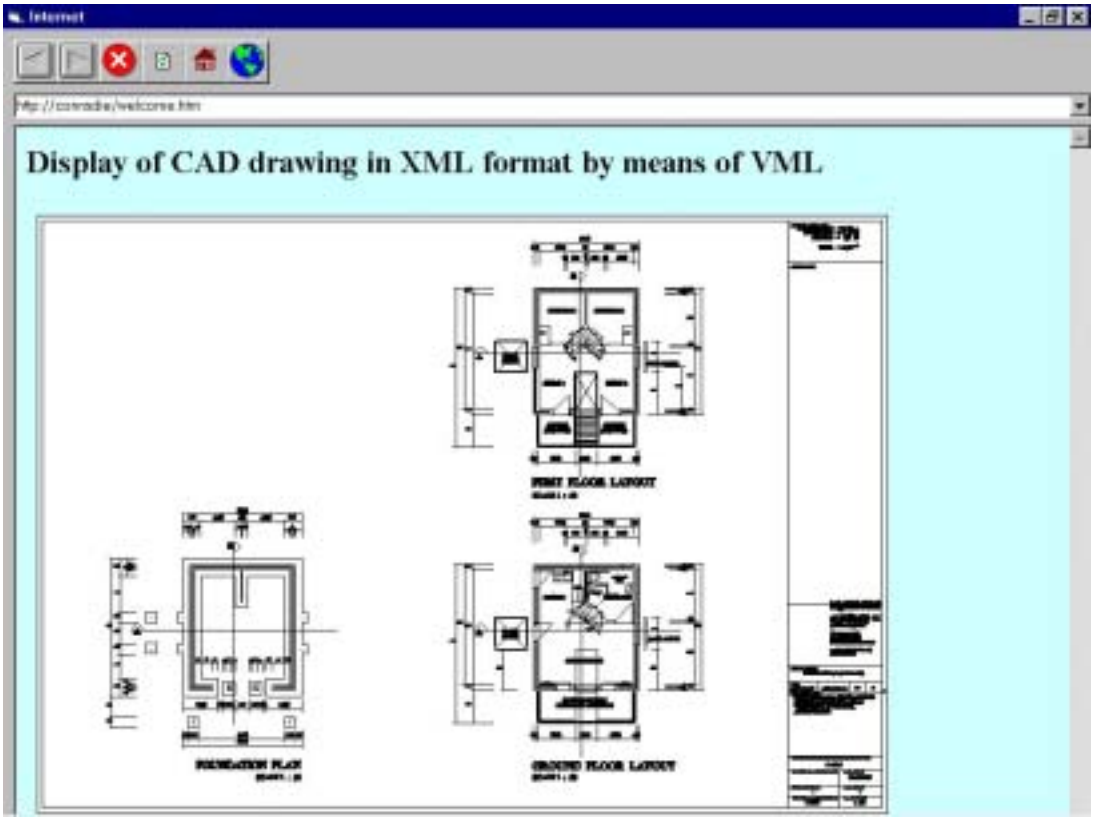

Figure 2: Display of CAD drawing in XML format by means of VML (Author)

Careful analysis of the display reveals numerous small errors such as inaccurate text display, and problems with the interpretation of the bulge factor to display circles or arcs. Bit map images, although saved in the XML file are not displayed at all in the web page. At this stage the display capabilities of an AutoCAD Whip file in the web environment are superior to what is offered by the static VML display. However the XML provides a structured and accessible data format that can be processed further whereas the Whip[1] format is closed and proprietary.

6.2 Packaging and retrieval of design knowledge

6.2.1 Introduction

The author proposes a totally new approach to architectural design knowledge packaging that would require the lowest possible level of platform technology, such as a spreadsheet, as the entry level. Many ambitious attempts have been made in the past to define universal Building Product Models. At this stage none of them are entirely satisfactory due to complexity of the artefact creation world. All indications are that conscensus will be reached soon (Eastman 1999)

The portable nature of Microsoft ActiveX controls makes it possible to support a wide range of platforms without being tied into particular CAD systems, databases or design software. It also ensures a cost effective design environment. By means of ActiveX controls that are embedded into web pages it is possible for service providers to offer a subscription service of design tools such as lightweight cases (architectural design kits) to designers. The designer could then use design software in his Internet explorer without even installing or buying

[1] Whip is a proprietary format that facilitates the display of CAD drawings in a web page

expensive software. The user could then purchase time from the software service provider only when required. It is proposed that the approach that has been followed in the development of the precedent systems AEDES and PREMIS up to date be completely changed around. The approach in the past was an application centric approach with particular emphasis on specific database technology and CAD systems. It is proposed to use a document-based approach (Figure 52). Designers and Architects are used to the concept of documents. This will ensure that anybody that has Microsoft OLE, COM and DCOM compliant software can significantly benefit from the approach. Microsoft developed these technologies specifically to support the intelligent use of documents in a collaborative environment.

The architectural design starter kits, developed by the Division of Building Technology, over a long period of time provided a useful starting point for AEDES and the present research. These starter kits have already contributed significantly towards an accelerated and more efficient design process in the domain of health facilities. These starter kits are available in CAD format and contains “empirical ideal” total facility layouts. The author recently wrote a prototype web page to test the technical feasibility of the distribution of these design kits via the Internet.

The main shortcomings of the present CSIR starter kits (cases) are:

- They contain no traceability of the design process.
- Although staffing required, fixed and loose equipment are available in supporting design documents, it is not in a structured way that could be used in Case-Based Reasoning.
- No distinction is made between neutral and localised information. Heidegger described this as the *“Dasein”* of tools (Biemel 1976:38).
- No object naming conventions have been used that can facilitate connection with other data sources.
- No intelligence is available to predict operational performance.
- No integration with the total life cycle information infrastructure.
- Starter kits should contain knowledge from both the tacit and explicit levels of knowledge management to give future users an idea what the design rationale was.

The use of structured methods such as QFD, Kansei and System Engineering is explicitly time-consuming (Cohen 1995:31). In order to achieve the best possible future use of the design knowledge it is important that knowledge can be reused. The object technologies presently available are already mature enough to support this need well.

The AEDES prototype software solved some of the abovementioned knowledge packaging challenges. However a few fundamental matters are still unresolved such as support for the Internet, complete object encapsulation and a low-level entry platform. In the prototype CAD drawings were embedded into an OLE field into an Oracle form field in an attempt to encapsulate the various types of knowledge required. The main disadvantages of this approach were:

- The object data is not persistent.
- Low-level users would require a database such as Oracle, Microsoft Access or SQL Server as a minimum to use the starter kit.
- It would be difficult to distribute the design globally.
- The response by means of Visual Basic during interrogation of the embedded CAD objects is presently very slow. This improved significantly in Oracle 8.0i (The latest Oracle RDBMS release).

- It is difficult and inconvenient to interface the alphanumeric and graphic contents of the starter kit with other applications.
- It would be difficult for third party companies to build starter kits independently. This is a prerequisite if the starter kits are to gain widespread commercial acceptance in future.
- It is particularly difficult to profile or deliver design data to suit the specific needs of the designer, planner or reasoner.

6.2.2 Constraints

Constraints form an important part of planning and design in general and should be supported by ARGOS. The following different main categories of constraints can be identified that could be supported by ARGOS:

- *Formulation*. This is the process of adding or creating new constraints based on decisions. Constraints could originate from the designer, propagation of second order constraints and by inheritance.
- *Propagation*. This is the process of inferring values and constraints from other values and constraints. This is achieved by means of functions associated with the constraint type.
- *Satisfaction*. This is the process of finding values that satisfy a constraint set. Different constraints can have different functions associated with the particular constraint.

The implementation of constraints in an open world is subject to several requirements:

- *Sensitivity to incomplete knowledge*. It is possible that constraints need to be evaluated with some arguments missing. Hinrichs (1991:98) suggests that two evaluation functions are used in this situation, one that is optimistic about the missing information and one that is pessimistic.
- *Ability to relax preferences*. Since design problems may have satisficing solutions, the design processor needs to be able to relax constraints. To facilitate this the importance of a specific constraint needs to be known.
- *Flexibility of propagation.* The constraint poster should be able to propagate constraints between different sets of variables in a problem.
- *Protection of problem-independent constants.* The flexibility of propagation necessitates the restriction of what counts as a variable in a problem.

The constraints determine the class of problems that can be represented. Figure 48 illustrates the taxonomy of constraint types that could be used in a design processor. The constraints fall into five main categories:

- *Logical Connectives* permit recursive combinations of constraints.
- *Nominal Constraints* relate identities of values.
- *Ordinal Constraints* capture relationships between continuous valued quantities.
- *Structural Constraints* constrain the existence of variables rather than their values.
- *Functional Constraints* degenerate constraints (rules) that propagate only in one direction.
- *Second-Order Constraints* are constraints on other constraints.

In the descriptions below, the term *variable* refers to a slot in some frame and *argument* refers to an actual argument to the constraint, which could be either a variable or a constant (Figure 48).

Same. Two arguments are constrained to be identical. This is typically used to connect two variables together. It could also be used to restrict a variable to a constant.

Instance. The first argument must be a frame subtype of the second. In this case instances and subtypes are treated equivalently.

Compatible. The arguments must be frames, in which neither is represented as being *incompatible* with the other.

Inverse. The first argument must be the logical or functional inverse of the second.

Member. The first argument is a member of the set designated by the second argument.

Contains. The second argument is an ingredient or component of the first argument. This is a transitive relationship.

Does-not-Contain. The second argument is not an ingredient or component of the first argument. This is a transitive relationship.

Within. The first argument is in the numerical range designated by the second argument.

At-least. The first argument is greater than or equal to the second.

At-Most. The first argument is less than or equal to the second.

Max. The first argument is the maximum of all subsequent arguments.

Min. The first argument is the minimum of all subsequent arguments.

Same-Structure. The variables in the first argument are the same as the variables in the second argument. The arguments to structural constraints of this sort are effectively quoted[1] such that variables themselves are returned, rather than the values of those variables. This permits constraints on structure as well as on content.

Struc-Member. The variable in the first argument is a member of the variables in the second argument.

Same-Constraints. Every constraint on the internal slots of the value of the variable is present on the corresponding slots of the frame containing the variable.

Constraints. The constraints on the first argument are propagated to all the variables designated by the second argument.

To carry out a planning and design activities certain information must be available. In addition, certain conditions, states or evaluations may apply to the data. Eastman (1999: 343) calls this the *Readset* and *Before Constraints*. When an activity is completed data will be added or modified. That design data will possibly have new conditions, constraints or states associated with it. Eastman call this the activities' *Writeset* and *After Constraints*. Together they define an activity Φ that has the following general structure:

$$\Phi = (\{E\}^R, \{E\}^W, \{C\}^B, \{C\}^A)$$

where $\{E\}^R =$ the set of entities to be read into the application

$\{E\}^W =$ the set of entities that are written by the application

[1] This term refers to a convenient LISP construct. LISP was a prominent language in AI ten years ago.

$\{C\}^B =$	the set of constraints that must be satisfied before the application can be executed
$\{C\}^A =$	the set of constraints that are satisfied within the application and can be relied on by later operations

The before constraints and after constraints specify a logical relationship between activities and the information and the conditions that the activities require. The *Readsets* and *Writesets* define data dependancies. The before constraints and after constraints identify process dependencies.

Constraints, as defined here, can have one of four values (Eastman 1999:343):

$< T >== True$	implies that it has been satisfied
$< F >== False$	implies that it has been evaluated and has failed
$< U >== Unknown$	implies that it has not been evaluated, possibly because it is not available to do so
$< X >== Blank$	implies that changes have been made to the context, so that the state of the constraint is uncertain

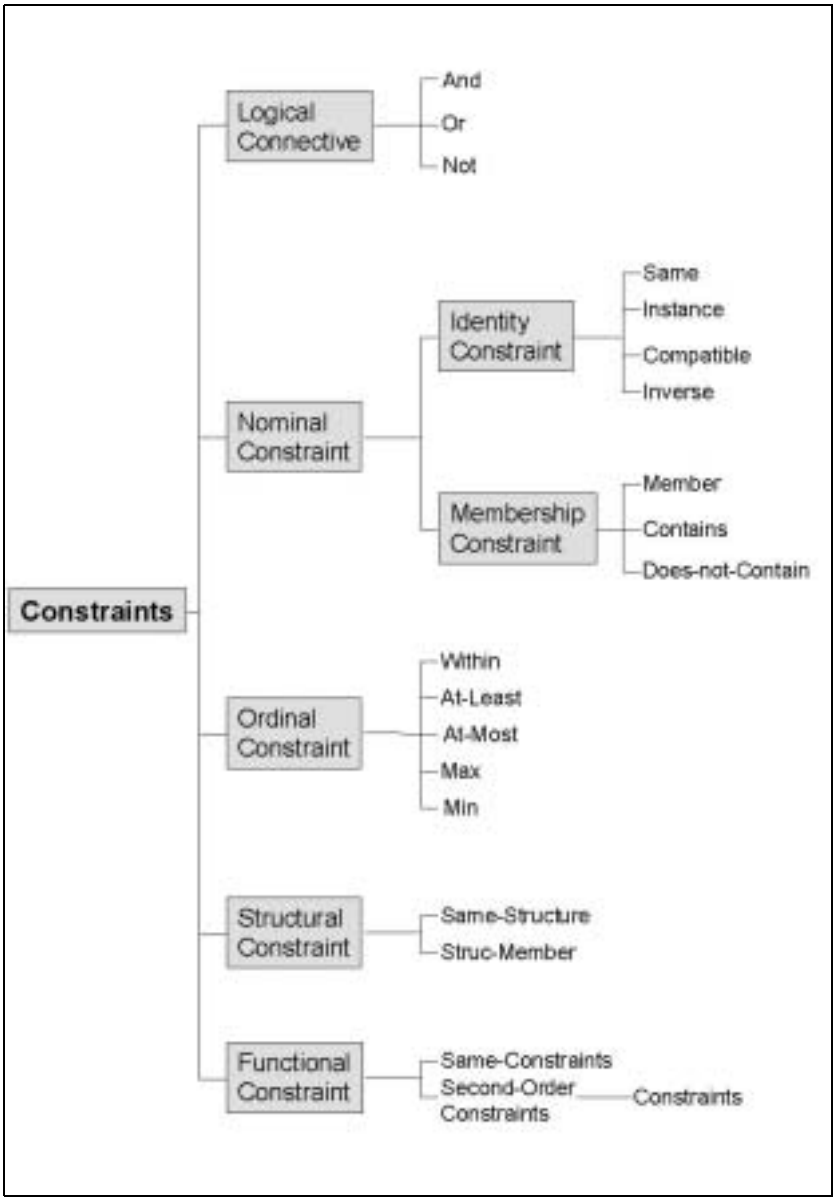

Figure 3: Taxonomy of constraint types (Hinrichs 1991:99)

6.2.3 The design of the ARGOS intelligent component

In order to conveniently process design fragments on the desktop without the use of CAD requires intelligent components that can encapsulate the design fragments. The component should also be able to retrieve design fragments from anywhere. To test the idea a prototype control was built. Consider Figure 49 for an example of the control running in Internet Explorer 5.0 The component has the ability to be resized in the x and y axis whilst in two dimensional mode and the x and z axis whilst in three dimensional mode. Appendix F contains the actual code that connects the various parts of the control parametrically together. The intention is that a designer might place a number of the controls in a spreadsheet to test the relationship between architectural design units at any level of specificity. Each component is an autonomous encapsulated world on its own.

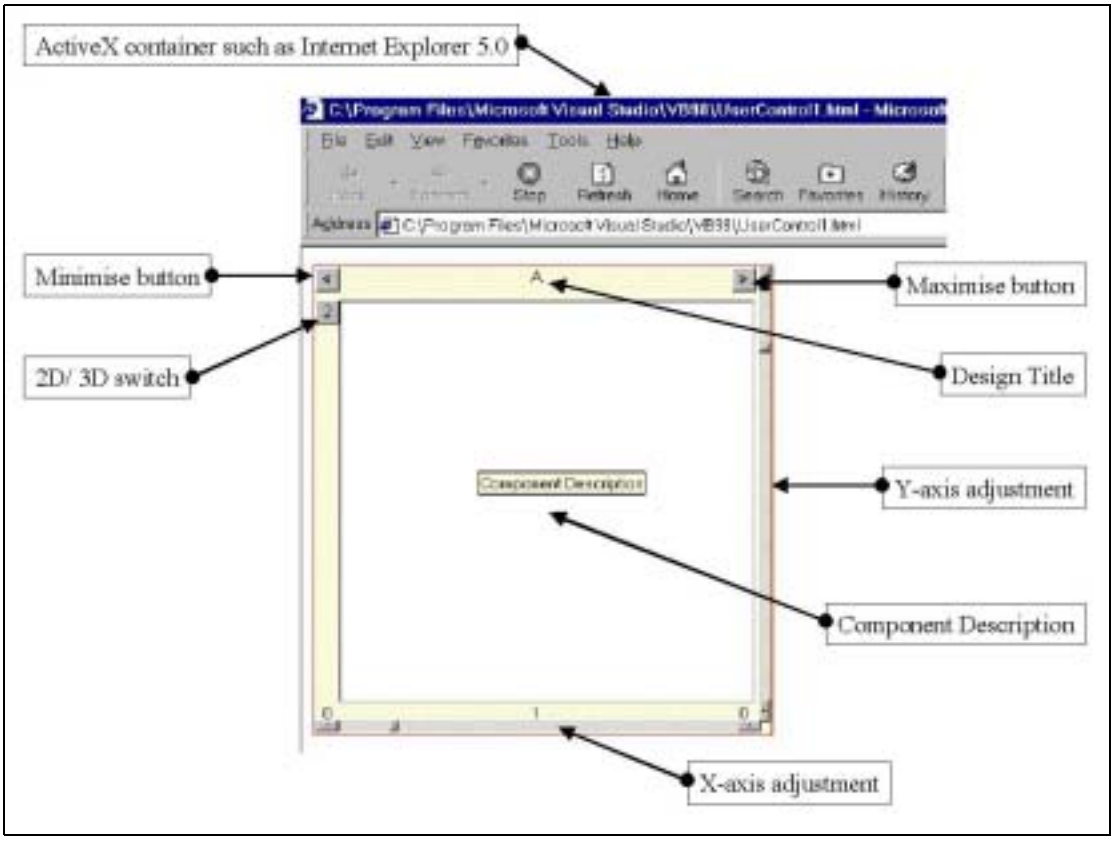

Figure 4: ARGOS object in 2D mode (Author)

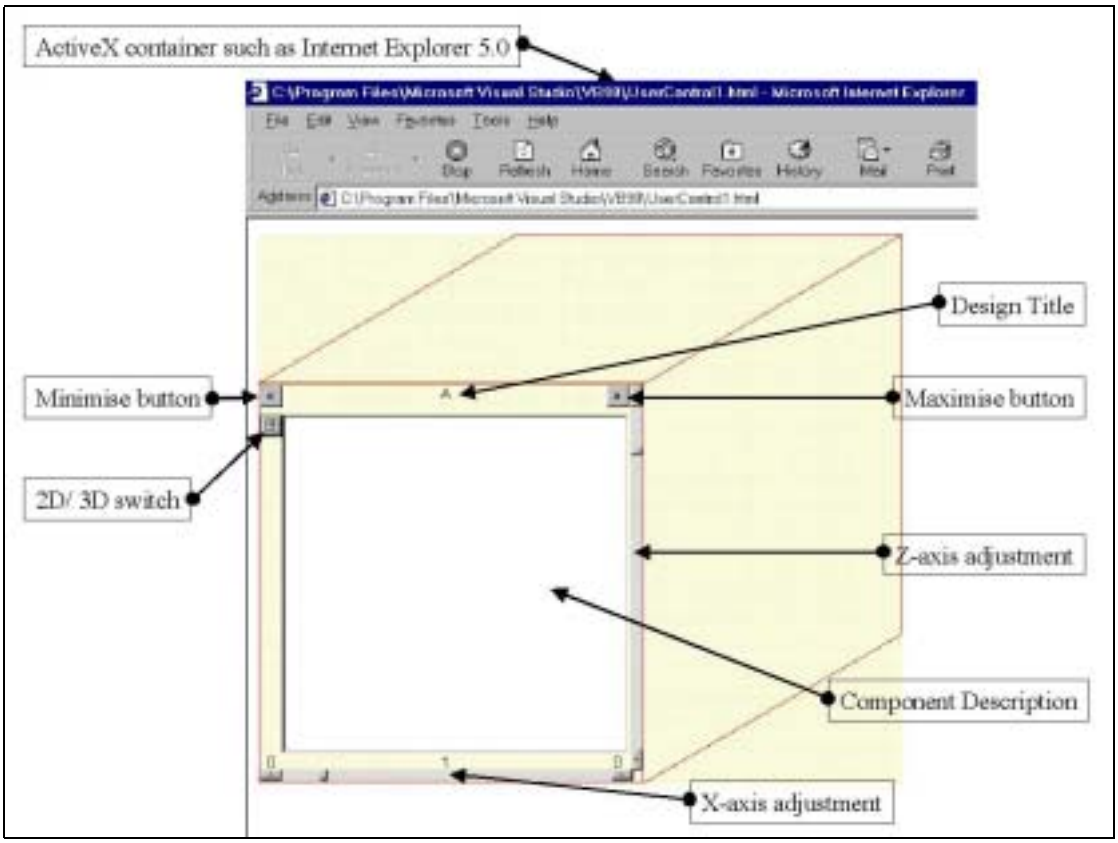

Figure 5: ARGOS object in 3D mode (Author)

Synchronisation between the autonomous components can achieved by means of very simple Visual Basic for Application code if it is used in a spreadsheet.

Figure 50 illustrates the ARGOS control in 3D mode. The 3D mode enables a designer to get a feeling for volume in a basic way. The z-axis adjustment facilitates the adjustment of the height. The controls can be made highly sophisticated by adding automatic volume calculation and readout of pertinent design parameters. It is envisaged that many different variants of ARGOS can be built such as:

- Controls that are unlocked, leaving it up the user to place, size and populate them with design information
- Controls that contain cases from the past at various levels of specificity
- Rule-Based controls that model certain well known design characteristics such as energy use
- Model-Based controls that model the constructional performance of a structure such as the forces on a slab

6.2.4 Classification and knowledge organisation in a packaged environment

If the packaging of architectural design knowledge in the form of encapsulated Microsoft ActiveX controls is to be successful then it is important that a designer can easily find relevant controls anywhere in the world. It is also important to realise that a control that has not been brought into the specific environment where it will be used should contain knowledge that is neutral. Once it arrives in the specific environment where it will be used it should take on the localised qualities. An example of this is the cost of plant, labour, construction materials, temperature and soil conditions.

The core problem in Information Science (IS) is seen as information seeking and “information retrieval (IR). The design of information systems and knowledge organisation by classification and indexing is a means to that end.

Hjörland (ISKO 1994:91) identifies nine principles on the organisation of knowledge.

1. Naïve-realistic perception of knowledge structures is not possible in more advanced sciences. The deepest principle on the organisation on knowledge rests upon principles developed in and by scientific disciplines.

2. Categorisations and classifications should unite related subjects and separate unrelated ones. In naïve realism, subject relationships are based on similarity. Two things or subjects are seen as related if they are “alike”, that is if they have common properties or descriptive terms ascribed.

3. For practical purposes, knowledge can be organised in different ways and with different levels of ambition.

    - Ad-hoc classification (categorisation) reflects a very low level of ambition in knowledge organisation. Every time you arrange flowers in your private home, you use a kind of “ad-hoc classification” determined by your private taste, the colours of your rooms, what other objects they should match with.
    - Pragmatic classification reflects a middle level of ambition in knowledge organisation. It is a compromise between ad hoc classifications and scientific classifications. Amateur gardeners or horticulturists have other criteria for categorising proteas and azaleas than the biologist would imply.

- Scientific classification reflects a very high level of ambition in knowledge organisation. It is highly abstract and generalised way of organising knowledge. An example of this is the classification of animals and plants according to biological taxonomies.

4. Any given categorisation should reflect the purpose of that categorisation. It is very important to teach the student to find out the lie of the land and apply ad hoc classifications, pragmatic classifications or scientific classifications when appropriate.

5. Concrete scientific categorisations and classifications can always be questioned. The concept of "science" has more than one meaning.

   - Science as a social institution, consisting of people paid to do research. This is the cultural concept of science.
   - Science as a normative, epistemological concept (to argue in a scientific way). What constitutes science in this respect is a matter of continuous development, argument and criticism in methodology and theory of science and in the development of science itself.

6. The concept of "polyrepresentation" is important. In typical information seeking situations, some categorisations are useful to some degree, others to some other degree.

7. To a certain degree different arts and sciences could be understood as different ways of organising the same phenomena.

8. The nature of disciplines varies. The distinction between "hard sciences" and "soft sciences" is well known, but perhaps not fruitful.

9. Many authors indicated the important problem that the quality of knowledge production in many disciplines is in great trouble. It seems if the priorities become more and more short-sighted, that less effort are made to develop long-sighted, well organised and well-cared for bodies of knowledge and literature. This means, that the integrity of scientific knowledge as well as other forms of knowledge is threatened.

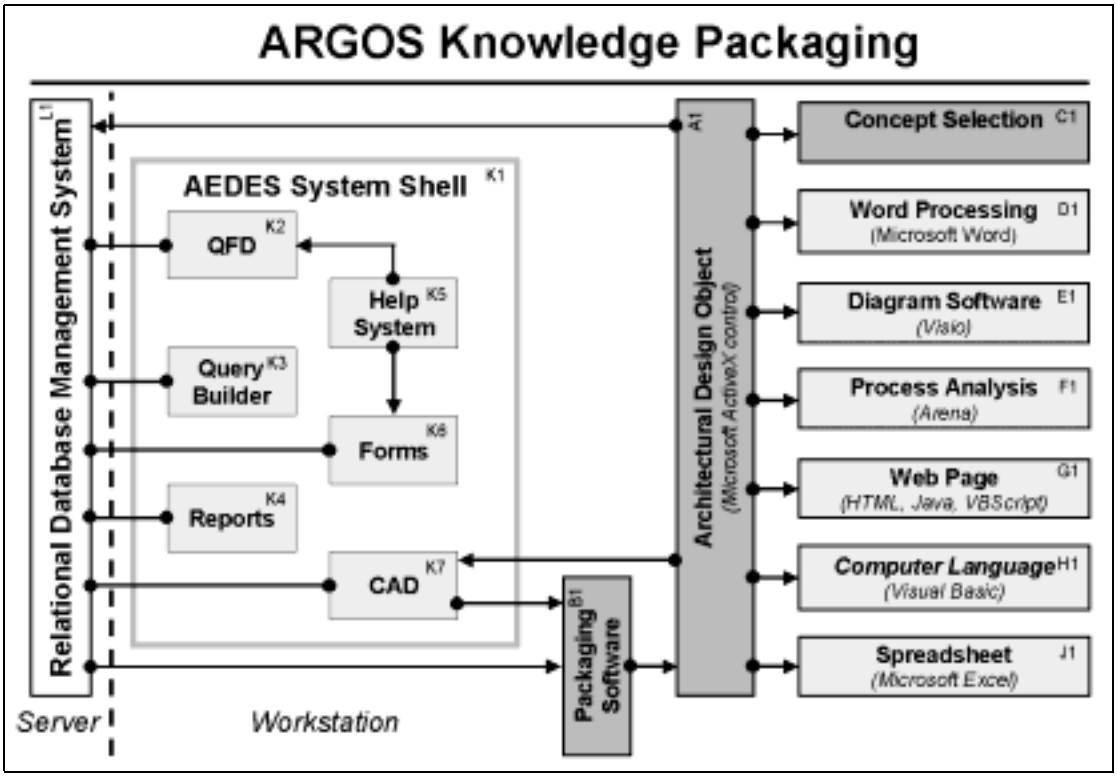

Figure 6: The relationship between the ARGOS, ActiveX design object and the applications software (Author)

6.2.5 The co-existence of ARGOS with other software

The Architectural General Object System (ARGOS) is a Microsoft ActiveX object with the internal design fragment stored in XML. Microsoft Visual Basic provides enough functionality to build the object (Appleman 1999). Although third party users can generate the object independently, it is recommended that a structured front end consisting of an appropriate collection of methodologies as described for the AEDES system could be used. This will ensure that the object is optimal for the given set of requirements. COM software components such as ActiveX controls can be developed with several different programming languages. The most common choice, if Web pages on the subject are any indication, is Microsoft Visual C++. Presently Visual Basic 5 and 6 also support the development of ActiveX controls very well. Using Visual C++, COM software can be written using one of three development libraries, the ActiveX Template Library, Microsoft Foundation Class Library or the BaseCtl framework. ActiveX controls can use a variety of programming languages from Microsoft for component design in addition to Visual C++ like Visual Basic, Visual J++ and even Word or Excel's programming languages.

Currently only highly skilled programmers can build the ActiveX objects. For this reason a special module B1, Packaging Software is proposed (Figure 51). This software tool takes the final design fragment and encapsulates it into a single object. Once the object is created it can be distributed in many different ways and used in a wide variety of environments. The contents of the object can be imported back into the original environment that created it. However the object can be used in many other environments such as spreadsheets, Web pages and process analysis.

6.2.5.1 Concept selection

The process of concept selection is important in the product development environment and therefore architectural design. In Architecture it is often necessary to compare alternative architectural design concepts, especially during the early phases of design. To this end the ARGOS kits could be inserted into a spreadsheet. The designer could then conveniently analyse various design aspects in the familiar environment of a spreadsheet without doing any programming. In this case the controls containing the likely concepts would be drawn into a spreadsheet or a simple Visual Basic program. The ratings from the different concepts are derived from the controls and subsequently compared with one another.

6.2.5.2 Spreadsheets

Spreadsheets such as Microsoft Excel support the use of ActiveX controls. Many people use spreadsheets and it is a convenient environment for initial project planning tasks such as cost estimating, area and energy analysis. In this environment there is no need to be connected to a database, although the proposed design of the ARGOS object includes links to material and product databases.

In order to use the control in this environment, a user simply has to insert the control into the spreadsheet. To access the list of properties and methods provided in the control in the spreadsheet, the user has to connect the desired property in the control to a cell(s) in the spreadsheet. This can be achieved by the example code fragments below (Code Fragment 3).

In this case the cells are manipulated by the Visual Basic *GotFocus* and *LostFocus* events. In the case of the function *ArgosAB_GotFocus* a range of cells *Range("A1:A10")* on *Worksheet("Sheet1")* is set to the value of the *GrossArea* property retrieved from object instance *ArgosAB*. Note that during the creation of the object certain properties were set to read only. In a similar way the function *ArgosAB_lostFocus* sets the *value* of a range of cells *Range("A1:A10")* to an empty string. At the same time a property *text* of the text box *txtArgos* is set to the text string "RESET TO EMPTY". The control is a totally encapsulated world that contains many properties. These autonomous controls need to be connected together in order to do something useful with it. This can be achieved in any ActiveX compliant container environment.

```
Private Sub ArgosAB_GotFocus()
    Worksheets("Sheet1").Range("A1:A10").Value = ArgosAB.GrossArea
    txtArgos.Text = ArgosAB.GrossArea
End Sub

Private Sub ArgosAB_LostFocus()
    Worksheets("Sheet1").Range("A1:A10").Value = ""
    txtArgos.Text = "RESET TO EMPTY"
End Sub
```

Code Fragment 3: Communication between an ARGOS ActiveX control and Excel Spreadsheet cells (Author)

6.2.5.3 Computer languages

A systems integrator or software tool designer can use the ActiveX controls (objects) in exactly the same way. However he can implement the objects in far more advanced environments. A typical scenario would be where a suggested method such as *ArgosAB.UnpackFunction* or *ArgosAB.UnpackCAD* could be invoked. This tells the particular

instance of the design component (*ArgosAB*) that the user wants to inspect the particular design functions embodied into the design or want the design object to download the CAD drawing to start with CAD based layout planning.

6.2.5.4 Process analysis

In an environment such as offered by *Arena* users can use the control to extract the desired properties that he wants to analyse. The capabilities of *Arena* can be utilised to optimise flow of people in the specific layout. *Arena* uses Visual Basic for Applications (VBA) as its command language.

6.2.6 The design of the ARGOS object

The ARGOS object [A1] can be placed inside any ActiveX container [B1] such as supported by Excel, Word, Visio or World Wide Web pages. Due to the intrinsic information that is built into the object the designer can use the object immediately without connecting to any outside information sources. However to realise the full power of this approach it is recommended that a user connects to the Internet to access convenient outside data sources to provide information such as product data [D1], material characteristics [E1], other existing cases [F1] and Facilities Management cost models. Figure 52 illustrates this concept as well as the relationship of the object with such remote data sources.

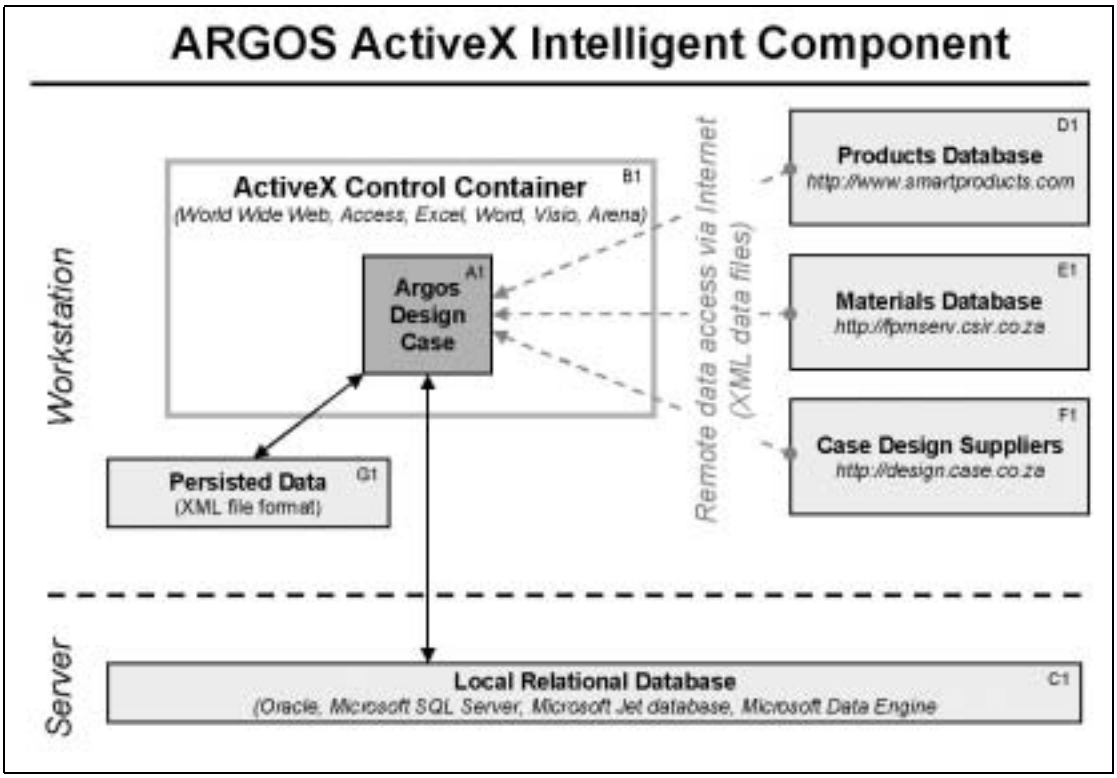

Figure 7: The relationship of the ARGOS object to other intelligent data sources (Author)

Internally the ARGOS design case contains 4 main types of design knowledge that consists of both alphanumeric and graphic information:

- Tacit design information
- Explicit design information

- Graphic information in the form of a design drawing that should preferably be in a neutral data format
- Functional design information such as the design functions and their allocation to physical design elements in a structured format. The W3C, XML format is ideal for this

Some of the information in the first two groups is exposed directly as ActiveX properties. In this case the design object properties is a synonym for surface features[1]. The indices of a case are those combinations of features that distinguish it from other cases, because they are *predictive* of something important in the case. In addition to be being predictive of something important, indices need to be *concrete* enough to be recognisable and *abstract enough* to make a case useful in a variety of future situations. This enables a designer to assess the applicability of the design or to estimate approximate cost. If the design appears to be suitable then the detailed functional design can be inspected.

Again it is important to note that it is an incorrect assumption when people argue about surface features, deep features, structural features, pragmatic features and thematic features in the sense of designing retrieval methods for cases based on one of those. To build a good index it is important to choose from all these levels and make sure that it has the important properties (Kolodner 1996:357). Those descriptors describe where a feature lies in a representation or what its content is. Sengupta *et al.* (1999) gives an indication of the usefulness of the W3C, XML standard for the representation of a case structure and describes methods to translate between relational databases and XML. The author is of the opinion that XML is almost ideal for the structured documentation of the intrinsic artefact design functions. By structured the following is assumed:

- The structure can be analysed by means of computer software.
- It is a complete documentation of the design *performance requirement*, *functions*, *allocations* to construction elements and *specifications* using systems engineering principles.
- Design function groups can be inserted into the existing structure.
- Constraint posting can be supported.
- Quality is an intrinsic part of the function structure.

If adaptation is required then the functional tree can be modified. Modification could be by means of the insertion of function fragments, elements or specifications. If existing design fragments cannot be found then the designer has to design the specific parts from first principles following a process of structured design.

Although the user definable properties that a user can set in this environment are persistent within the particular container, this persistence is destroyed the moment the object is moved to a different container environment. To overcome this problem two object methods *PersistDesignOut* and *PersistDesignIn* are introduced that will write the design data into an XML computer file on a local disk or an ftp directory on a remote project server. In this way structured design functions can be freely exchanged. As indicated in Figure 52 there is a bi-directional exchange of persistent data.

[D1] and [C1] are fictitious remote data sites that can be nominated by means of a data address within the object. This is achieved by setting the object property *DataLocation* to a valid URL. By means of the method *DisplayRemoteData* or by pressing the command button, these data will be displayed. The designer can then select the record and apply it to the current design object. Note that the data flow from remote data sources is uni-directional at this stage.

---

[1] There is a difference between easily available and surface features. Surface features make good indices to the extent that they are predictive of something important or useful.

The connection to the local database is conveniently achieved by means of the Microsoft ActiveX Data Object.

6.3 World Wide Web Implementation

Microsoft Internet Explorer supports the ActiveX controls. When the ARGOS design object is inserted into a web page the code looks like in Code fragment 4. The object starts with the label

```
<object classid="clsid:59DF65DF-632C-11D3-8D31-4854E8284FB0"
id="UserControl11" width="250" height="467">
```

and ends with the label

```
</object>
```

The `classid` is particularly important, because it is a totally unique code that is used to identify the particular class of the ActiveX control. This code is guaranteed to be unique in the world. This object was labelled with this code during the design and programming of the object. The `id` is the name that will appear on the list of possible controls when a user wants to insert an ActiveX control into his container software. In this case the `id` is `UserControl1`. The properties available for the object is exposed with the statements that read `<param name="_ExtentX" value="5292">`. In this example the ARGOS object contains 15 user definable properties. These properties fall into two main groups:

- Explicit
- Tacit

The explicit attributes have the prefix *AE_* and the tacit ones *AT_*. The explicit properties contain surface features (in CBR terminology) such as *gross area*, *net area*, *rentable area*, *construction area*, *volume*, *shape*, *durability*, *energy use* and *cost*. The tacit properties contain factors that were identified in Chapter 3, 3.5 where Kansei engineering was discussed in detail. This gives an indication of the sensory aspects of the design such as *sight*, *hearing*, *taste*, *smell*, *internal sensitivity* and *recognition*. Architecture has lot to do with the sensory aspects such as feeling of space, colour and acoustics.

It is apparent from the code fragment that the design detail is hidden away from the designer at this stage. The directly available properties make it possible to do basic preliminary feasibility studies. To make the detail visible the user will have to press the command buttons for CAD or Function that will unload the CAD drawing or the XML function tree. The ARGOS object also contains two buttons that a user can use to maximize or minimize the object. If a user wants to perform a specialised task he can invoke one of several object *methods* available.

```
<html>

<head>
<meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-1">
<meta name="Template"
content="C:\PROGRAM FILES\MICROSOFT OFFICE\OFFICE\html.dot">
<meta name="GENERATOR" content="Microsoft FrontPage 4.0">
<title>AEDES</title>
</head>
```

```
<body background="aedes_b.gif" link="#0000FF" vlink="#800080"
bgproperties="fixed">

. . .

<object classid="clsid:59DF65DF-632C-11D3-8D31-4854E8284FB0" id="UserControl11"
width="250" height="467">
      <param name="_ExtentX" value="5292">
      <param name="_ExtentY" value="9885">
      <param name="BackColor" value="0">
      <param name="ForeColor" value="0">
      <param name="Enabled" value="0">
      <param name="BackStyle" value="0">
      <param name="BorderStyle" value="0">
      <param name="AE_grossarea" value="0">
      <param name="AE_nettarea" value="0">
      <param name="AE_rentable_area" value="0">
      <param name="AE_construction_area" value="0">
      <param name="AE_volume" value="0">
      <param name="AE_shape" value="0">
      <param name="AE_durability" value="0">
      <param name="AE_energy_use" value="0">
      <param name="AE_cost" value="0">
      <param name="AT_sight" value="0">
      <param name="AT_hearing" value="0">
      <param name="AT_taste" value="0">
      <param name="AT_smell" value="0">
      <param name="AT_internal_sensitivity" value="0">
      <param name="AT_recognition" value="0">
      <param name="AF_function" value="0">
</object>
. . .

<hr size="1" noshade color="#0000FF">

<p align="left"><font color="#0000FF" size="2" face="Arial"><!--webbot
bot="Timestamp" startspan s-type="EDITED"
s-format="%d %B %Y %I:%M %p" -->12 February 2000 06:46 AM<!--webbot bot="Timestamp"
i-CheckSum="54291" endspan --></font></p>
</body>
</html>
```

Code Fragment 4: ARGOS object placed in a web page (Author)

6.4 Hypothetical use of ARGOS

Due to time and financial constraints it is not the intention to develop a full commercial system in this study. However this section provides a run-through of how a designer might use the system.

It is assumed that a designer wants to design a new 16-bed male/female/paediatric in-patients section. The designer decides to see whether a previous conceptual layout of this type of facility exists. Unfortunately nothing exists in the office and a search of the web is also unsuccessful. It is also assumed that the ARGOS ActiveX control set is installed and available on the design workstation.

The designer decides that he will be using his Microsoft Excel Spreadsheet as a blackboard, because this is convenient for the type of design testing that he wants to do. The design brief specifies a design of not more than 260 m² and the cost should be below R 780 000-00. The accommodation requirement for the design is the following:

Capacity of 16 beds
Staff WC
Patient ablution

Sit bath
Nurse station
Duty room
Clean linen storage
Clean utility room
Ward kitchen
Dirty utility room
Store

The client states that it is a specific requirement that energy be saved especially with regards air conditioning.

The designer starts the process by calling up an Excel Spreadsheet with a default ARGOS control panel. He selects a minimal parametric ARGOS control from the Control Toolbox (ARGOS.CBR) and inserts it into the spreadsheet (Figure 53).

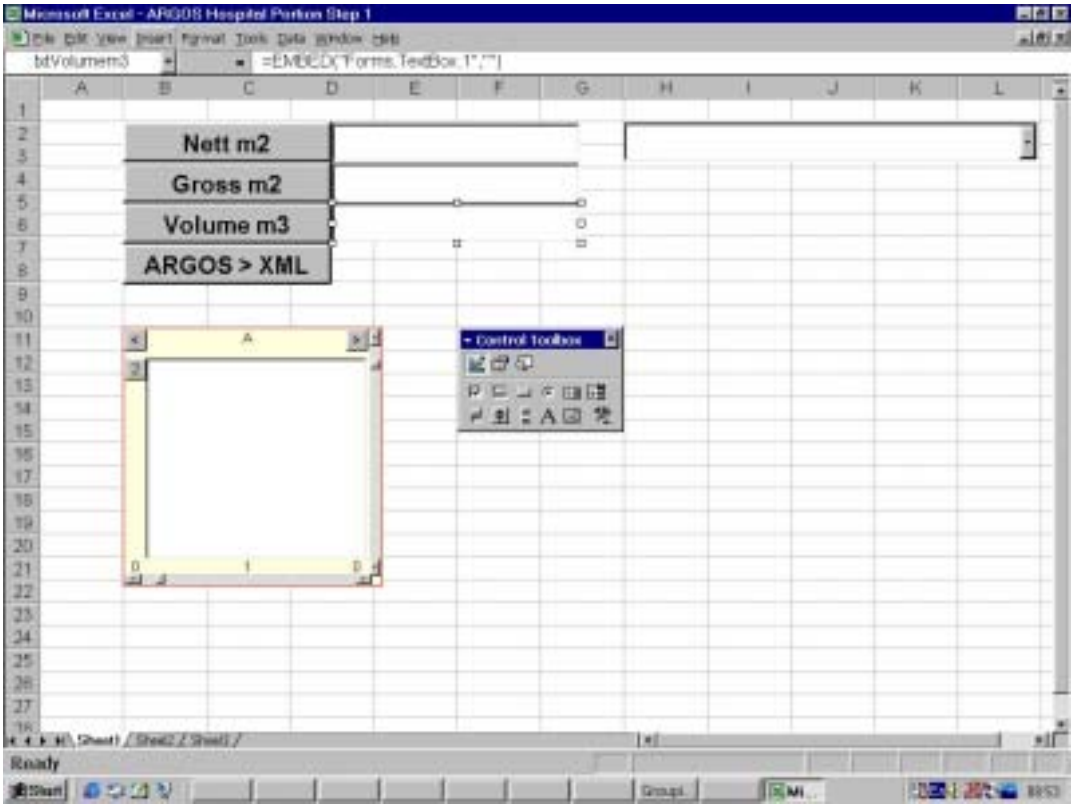

Figure 8: Design of 16 bed male/ female/ paediatric in-patients section step 1 (Author)

At this stage the spreadsheet contains four command buttons that enable the designer to calculate net m^2, gross m^2, volume m^3 and a special button that enables him to export the design in XML format to an XML aware CAD system for subsequent detailed design. For convenience a combobox is also included where the designer can list the spaces in the design. At the moment the ARGOS control is still default size and the internal properties all have default or undefined values.

The designer now continues to develop the design according to the brief and his experience. After some time the design looks like in Figure 54. The design contains 14 ActiveX controls of varying size. The designer adjusts some of the properties that will be important for subsequent retrieval of previous design cases. He sets the wall thickness in the *M_wall1*,

M_wall2, *M_wall3* and *M_wall4* properties to respectively 55, 220, 55 and 220. Seeing that this is a special section intended for paediatrics some Kansei adjectives are added in the *AT_hearing*, *AT_internal_sensitivity*, *AT_recognition*, *AT_sight*, *AT_smell* and *AT_taste* properties.

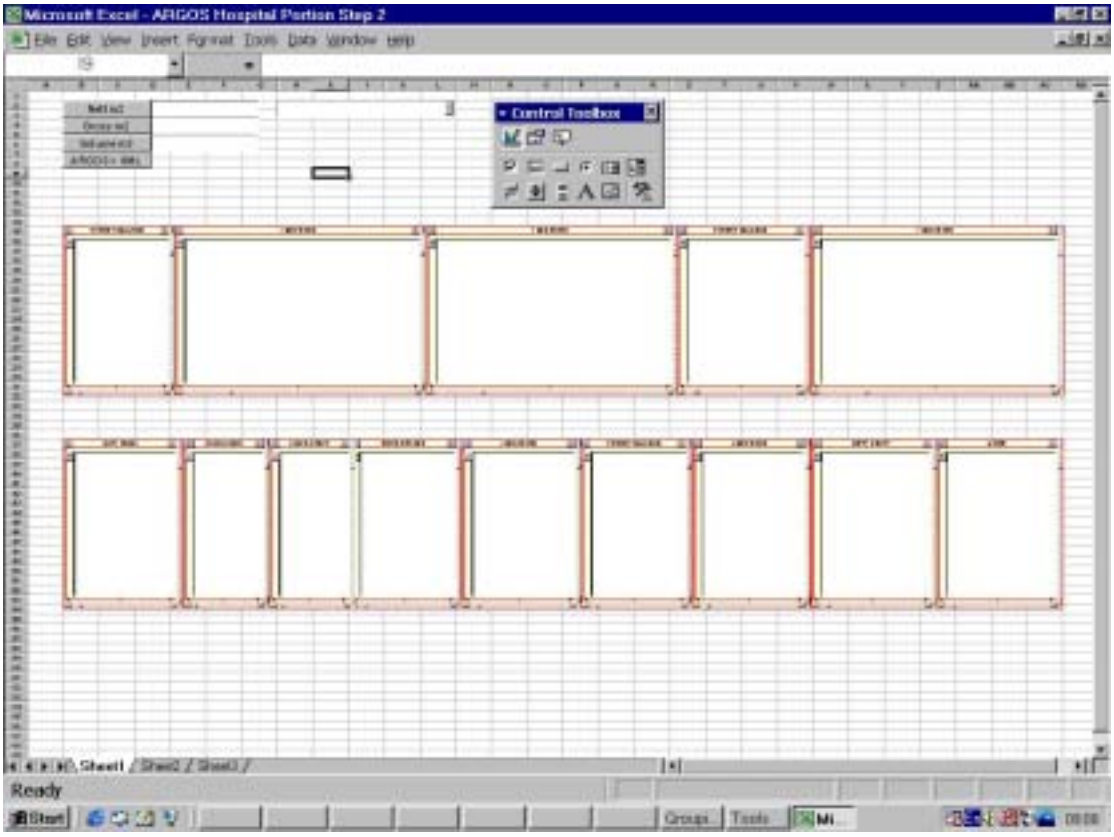

Figure 9: Design of 16 bed male/ female/ paediatric in-patients section step 2 (Author)

Some useful adjectives that could be used are listed in Figure 27. The Kansei properties now read:

AT_hearing = quiet
AT_internal_sensitivity = warm, tranquil, cheerful
AT_recognition = cute, elegant
AT_sight = cute, elegant
AT_smell = pleasant
AT_taste =

These properties are very important for subsequent retrieval of possible previous design experience. The ARGOS properties can be defined by typing directly into the relevant property, set by program or indirectly adjusted by means of the *x*, *y* or *z* slide controls. The properties are also useful because they can be directly transferred to CAD systems that support the definition of attributes such as MicroGDS or AutoCAD.

At this stage a typical property list for the 4 Bed Ward would look like the one illustrated in Figure 55. Note the Kansei definitions at the top of the list and the various wall thickness properties at the bottom of the list.

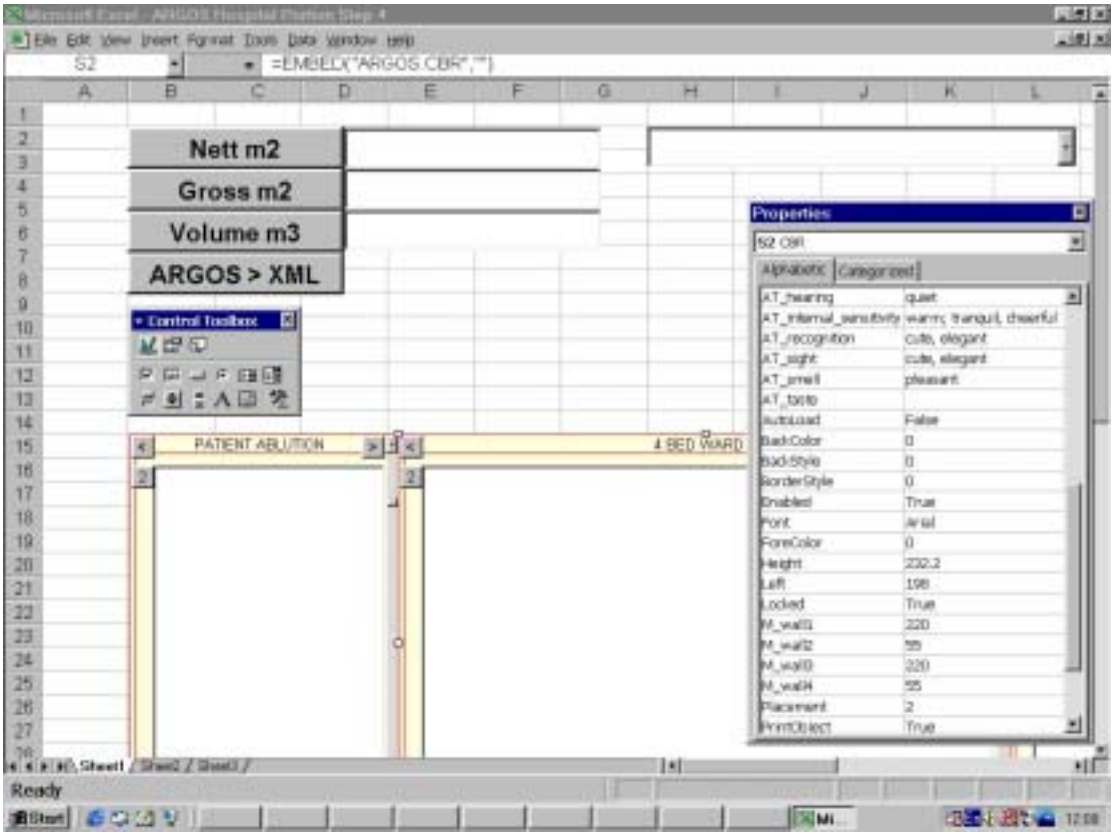

Figure 10: Setting design properties of a paediatric ward (Author)

At this stage the designer would like to know the gross and net area. This is accomplished by selecting the relevant command buttons that start Visual Basic routines that scan through all the design controls present and retrieve all the areas. In a similar way the volume is determined.

The system reports the following (Figure 56):

Net m² = 221.256
Gross m² = 251.5808
Volume m³ = 639.42984

At this stage he is not sure what the construction cost per m² is and activate his Microsoft Internet Explorer. He accesses the http://design.case.co.za web page that is one of the available design information sites to search for an estimated construction cost/m² for this type of facility. This is conceptually illustrated in Figure 52. He finds this cost to be R 2800-00/m². On the basis of this it is estimated that it would cost R 704 426-24 to build this facility. In the meantime the air conditioning engineer is analysing the design from an energy point of view. He finds that the current volume would require a significantly larger installation than originally anticipated. In an attempt to solve this the ceiling height is lowered from 2 890mm (34 brick courses) to 2 720mm (32 brick courses) (Figure 57). This is accomplished by setting the *AA_zdim* property to 2 720 for each ARGOS component. The recalculation indicates that the volume has now in fact been reduced from 639.43 m³ to 604.29 m³. This is an immediate saving of 5,5%. In a similar way other direct design parameters can be adjusted and tested.

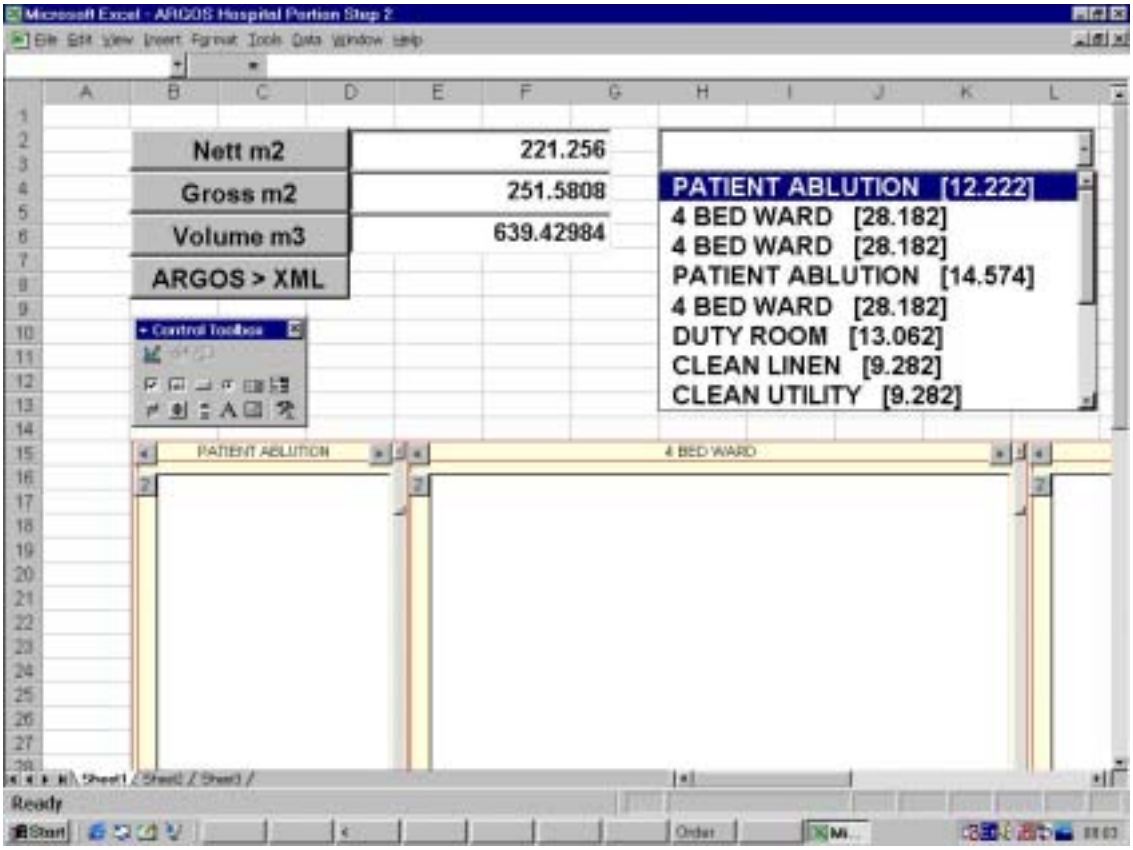

Figure 11: The calculation of area and volume (Author)

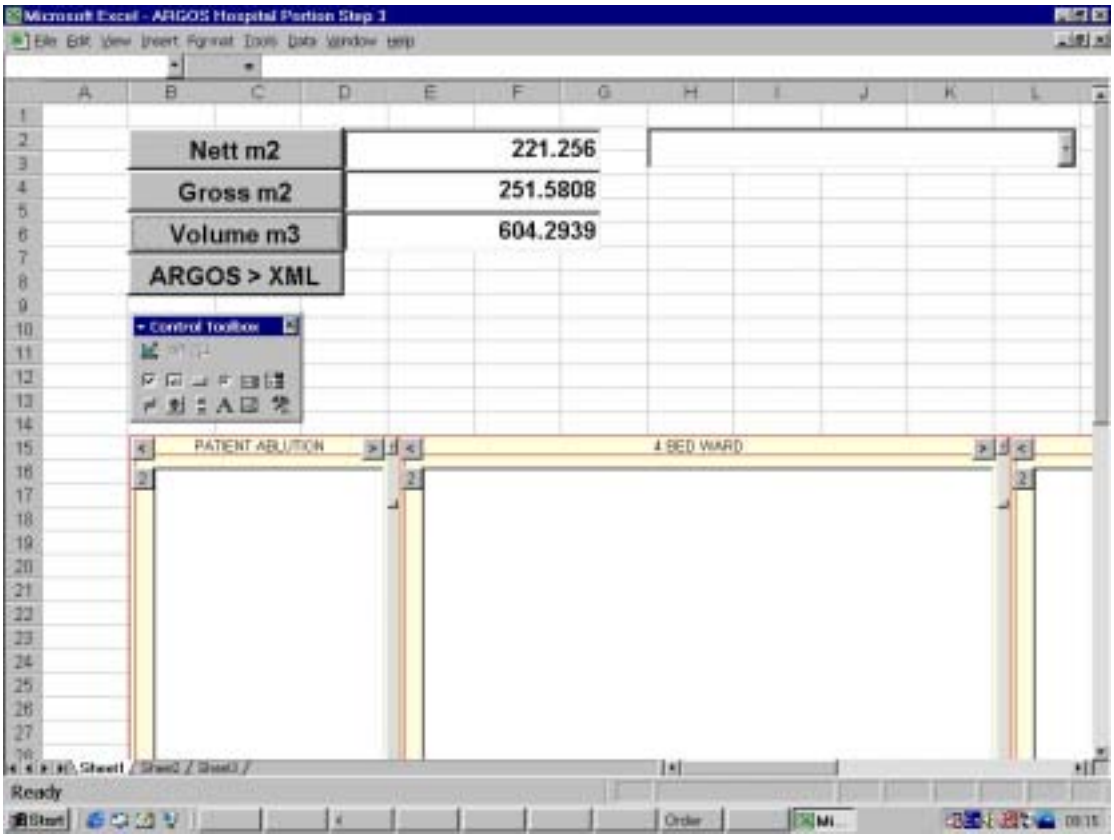

Figure 12: The reduction of volume by lowering the ceiling (Author)

The designer is now satisfied with the basic design and continues with the detailed design. By double clicking[1] on the ward the search engine is invoked to search for previous design cases that fit the type and dimensions previously captured. Initially only the main description is used to search for a list. The system reports that three types of ward is available:

Two Bed Ward
Four Bed Ward
Observation/ Trauma Ward

He selects the Four Bed Ward. ARGOS now uses the *AA_xdim*, *AA_ydim*, *AA_zdim* as well as the set of Kansei descriptions such as *AT_hearing*, *AT_internal_sensitivity*, *AT_recognition*, *AT_sight*, *AT_smell* and *AT_taste* as search parameters. This is implemented with the dynamic linguistic variable method described in detail in Chapter 3. Only one solution is found and placed into the design (Figure 58). In a similar way other parts of the design can be developed and further refined. Once the designer is satisfied he can export the entire design to an XML aware CAD system or a rendering/ visualisation package for detailed design and the production of working drawings.

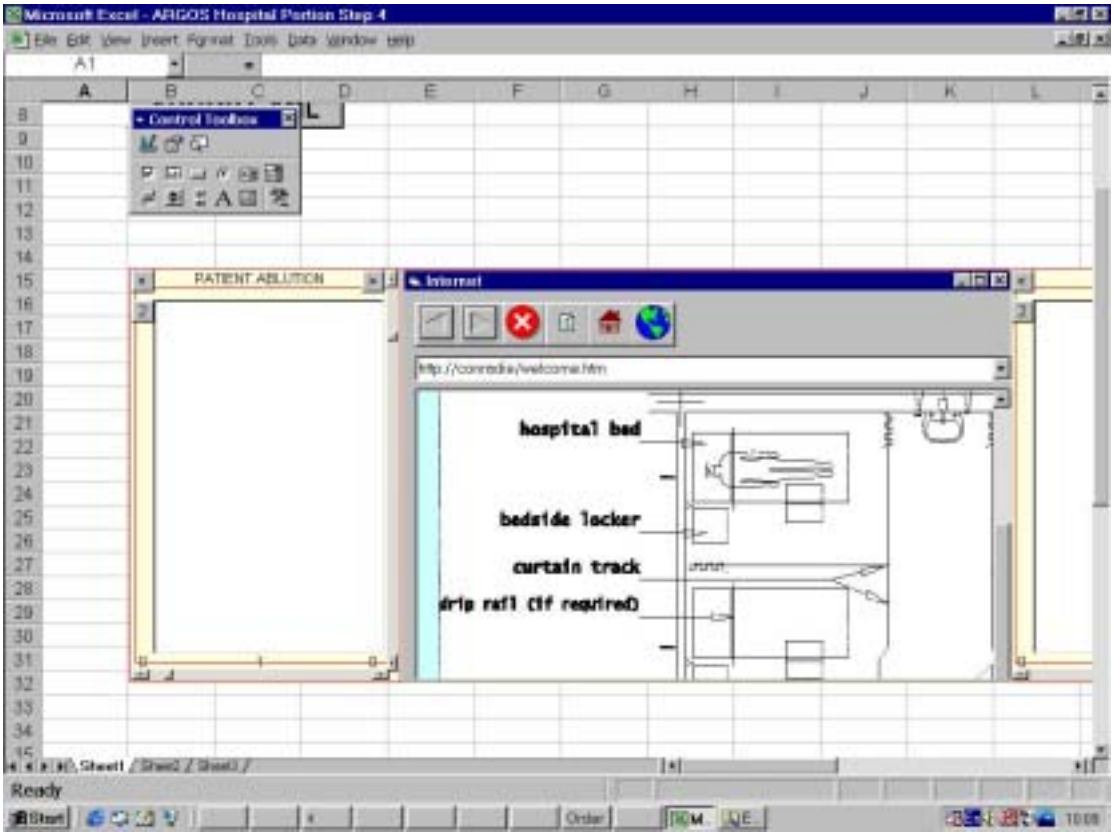

Figure 13: The retrieval and insertion of a Four Bed Ward detailed case (Author)

6.5 Empirical response tests

To ensure that the proposed system would be scalable and could eventually be applied to real problems in the architectural design domain a series of response tests were conducted. A recently completed very large shopping centre analysed to establish the needs of the

[1] Two possibilities exist to implement the CBR retrieval in ARGOS. The basic ARGOS can switch to CBR mode or a special separate ARGOS control can be written to handle only this aspect. The final implementation will become clearer with continued research.

professional team consists of 57 spaces on the lower first floor, 148 on the ground floor and 73 on the upper first floor. The proposed component system should be capable of supporting the following types of design activities in large and complex designs:

- Concept selection
- Retrieval of design experience
- Test of spatial relationships
- Scenario planning
- Collaboration on a global basis
- Modelling and simulation

To support and implement these activities require the ARGOS components to support any combination of parametric, *Rule-based*, *Model-based* and *Case-Based* methodologies. The tests concentrated on how responsive the components are to return direct and derived parametric values such as gross area and wall-space ratio. The prototype component presently supports 30 primary design properties, inter linked where appropriate.

The efficient response is primarily due to the fact that the parametric calculations are performed inside the ARGOS components where it is optimal and in a compiled form. The software that interrogated the components was, in this case, Visual Basic for Applications running at a more moderate interpreted speed than compiled Visual Basic. Although the prime purpose of the system is not efficient response, but rather opportunistic control, flexibility and interfacing to external software these tasks need to be accomplished within reasonable time.

The tests were conducted on a Microsoft Excel spreadsheet used as a blackboard, because it is so widely used and offers a convenient interface to spreadsheet capabilities and analysis software. It is clear that the slow computer (266 MHz CPU with 32 MiB[1] of RAM) is efficient up to about 75 components, whereas the moderate and fast computers are still efficient well beyond 100 components. The tests consisted of a Visual Basic program requesting parametric values that could be used in complex external analysis programs (Figure 59).

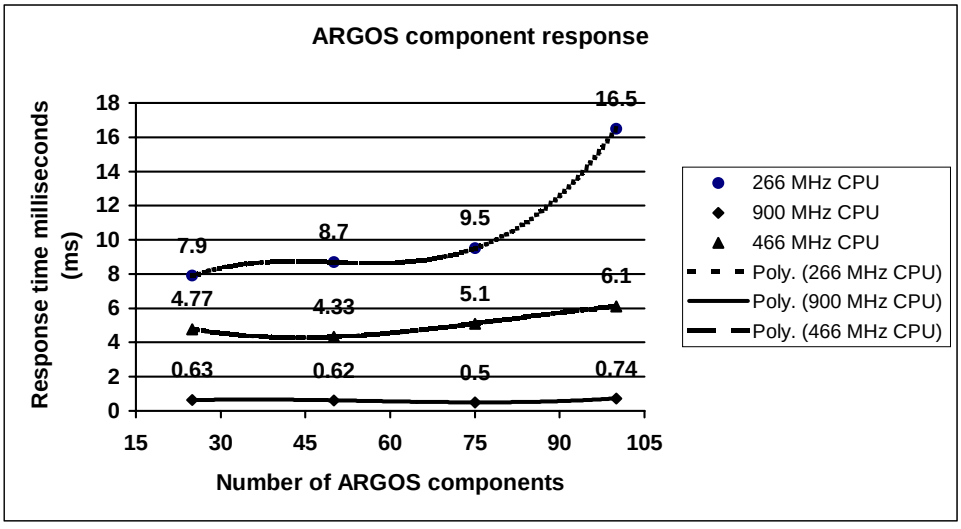

Figure 14: ARGOS component response (Author)

[1] One mebibyte (MiB) is equivalent to 1 048 576 bytes whereas one megabyte is equivalent to 1 000 000 bytes.

The increase in file size is linear (Figure 60). It should be noted that the size is expressed in kibibytes[2] as recommended by the International Electrotechnical Commission for binary multiples in December 1998.

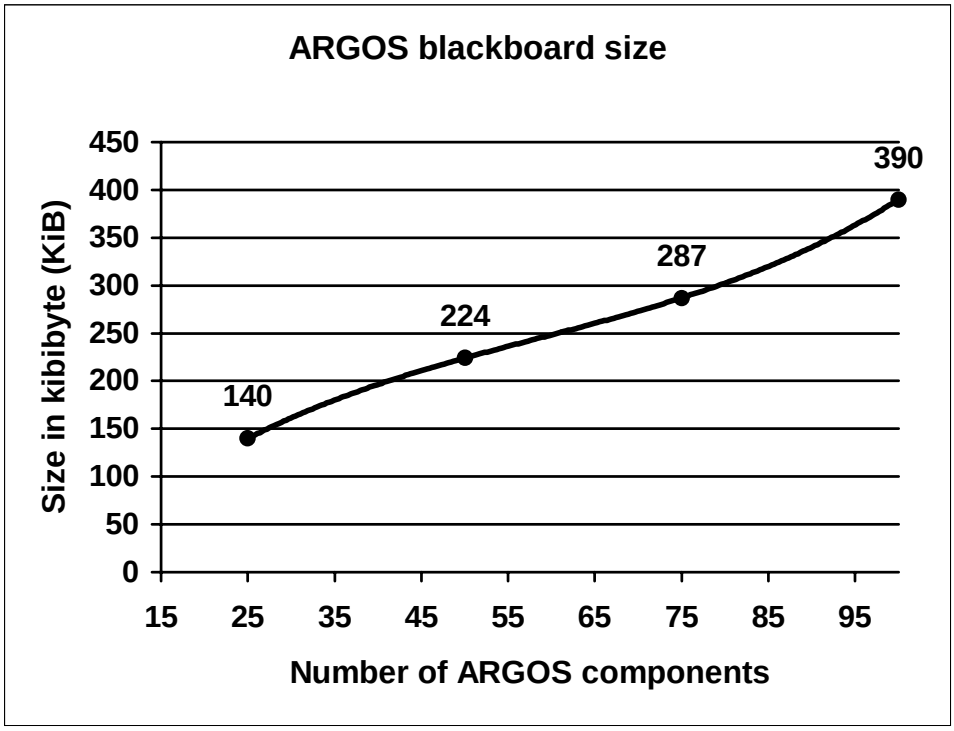

Figure 15: ARGOS blackboard size (Author)

Summary

A life cycle information infrastructure based on XML is used as a basis for ARGOS. The use of XML as a design language facilitates design knowledge delivery to users. The use of Cascading Style Sheets, XSL and VML was explored. This proves the versatility of XML beyond doubt. The storage of a CAD drawing in XML was analysed in detail.

Due to the generic nature of ARGOS the range of possible applications is large. The role in structured planning and design knowledge delivery is proposed. The relationships of ARGOS to other intelligent data sources were explored.

A detailed parametric ARGOS object was written with the ability to switch between 2D and 3D modes. A compact miniature Internet browser was developed that could be combined with the basic ARGOS component. This will enable unlimited data access.

Internally the ARGOS design case should contain four main types of design information consisting of both alphanumeric and graphic information:

- Tacit design information
- Explicit design information
- Graphic information in the form of XML
- Functional design information and constraints

[2] One kibibyte (KiB) is equivalent to 1 024 bytes whereas one kilobyte (kB) is equivalent to 1 000 bytes.

Finally a short hypothetical run-through of how the ARGOS system might be used is described. Empirical response tests were also conducted for a blackboard (spreadsheet) with 25, 50 and 100 controls on different types of computer. This indicates that the ARGOS blackboard type of architecture using a spreadsheet is effective.