Department of Computer Science
Adran Cyfrifiadureg

BRITISH AEROSPACE
Sowerby Research Centre

A Java/CORBA Based Visual Program Composition Environment

Matthew Shields, David Walker, Omer Rana
Department of Computer Science, Cardiff University,
PO Box 916, Cardiff CF2 3XF, UK

David Golby
Department of Mathematical Modeling
British Aerospace Sowerby Research Center
PO Box 5, Filton, Bristol BS12 7QW, UK

Project Objectives

A Problem Solving Environment (PSE) is a complete, integrated computing environment for composing, compiling and running applications in a specific problem area or domain. This poster describes a Visual Programming Composition Environment (VPCE), the user interface for a PSE, that uses Java and CORBA to provide a framework of tools, enabling the construction of scientific applications from components.

The VPCE consists of a component repository, from which the user can select off-the-shelf or in-house components, a “scratch” (graphical composition) pad on which components can be combined, and various tools that facilitate the configuration of components, the integration of legacy codes into components and the design and building of new components.

The VPCE produces output using dataflow techniques in the form of a taskgraph, annotated with a performance model plus constraints for each component, expressed in XML. In addition the VPCE supports a domain specific Expert System based on JESS that will guide the user in component selection and perform integrity checking.

Use Cases

- Running a legacy application as a wrapped component. The interface to the legacy application is provided in C-XML. The application may be a sequential code, or may contain internal parallelism using MPI or PVM.
- Performing paramter runs on an existing or new applications, to study the effect on parameter ranges.
- Combine various third party components to generate a new application. The application can itself be stored as a seperate component in the Component Repository
- Searching for suitable components in various repositories maintained on the internet at a remote site, where each component adheres to the C-XML specification.
- Developing a new application using the EXPERT ADVISOR, to either select new components, or developing an application on a different platform. In the latter case, the EA is used to analyse effects of platform constraints on a given application code.
- Visualising results of an application remotely.
- Enabling and supporting Computational Steering

References

- WebFlow
http://www.npac.syr.edu/Projects/WebSimulation/

- JESS: Java Expert System Shell
http://herzberg.ca.sandia.gov/jess/

- XML: Extensible MarkUp Language
http://www.w3c.org/

- A Problem Solving Environment for Network Computing
Haluk Topcuoglu, Salim Hariri, Wojtek Furmanski, Dongmin Kim, Yoonhee Kim, Xue Bing, Baoqing Ye, Ilkyeun Ra, Jon Valente
In Problem Solving Environments, editors E. Houstis, R. Bramley, and E.Gallopoulos, IEEE Computer Society Press, 1998

- Parsec: A Parallel Simulation Environment for Complex Systems
Rajive Bagrodia and Richard Meyer and Mineo Takai and Yu-an Chen and Xiang Zeng and Jay Martin and Ha Yoon Song,
IEEE Magazine October 1998

- PSDM: An Efficient Parallel Software Design Model
Kok K. Kee, Salim Hariri
High Performance Computing Conference '94, Singapore, September 1994

- GRADE - Graphical Environment for Parallel Programming
Peter Kacsuk, Sandor Forrai, ERCIM News No. 36

- Visual Programming Environments for Multi-Disciplinary Distributed Applications
Matthew Shields, Internal Report, Cardiff University, UK

- Iris Explorer
http://www.nag.co.uk/Welcome_IEC.html

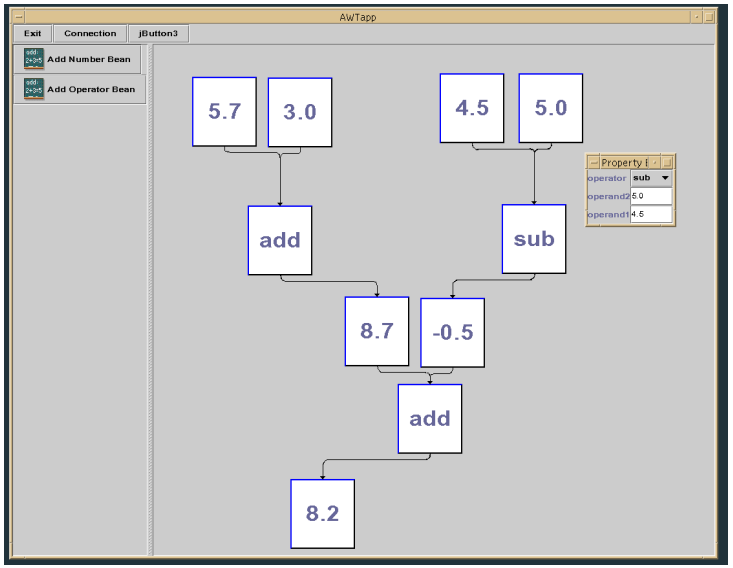


Figure 1

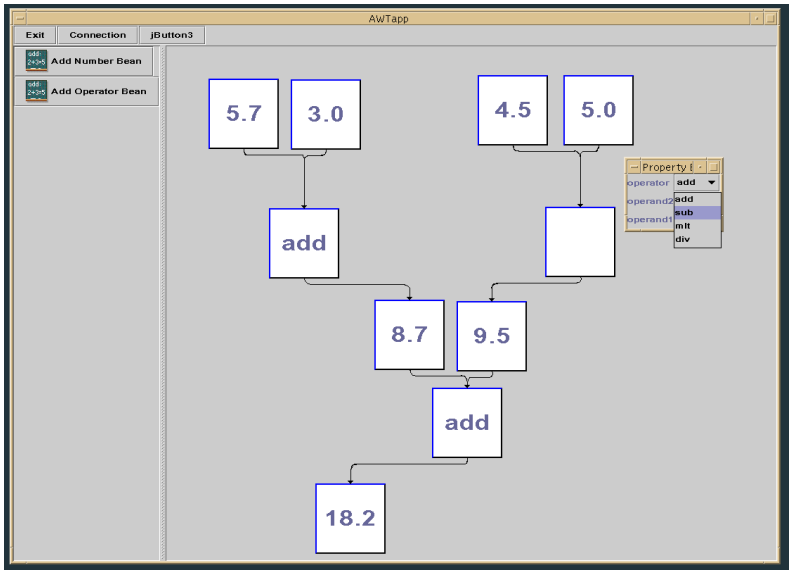


Figure 2

These screen shots show a simple prototype interface that we are using as a demonstrator. Each of the screen shots show a small system of interconnected components assembled from the component repository.

Figure 1 shows the property sheet for one of the operator components, with its two input values and the operator that acts on them.

Figure 2 shows the operator being changed on one of the “operator” Beans.

Figure 3 shows the property value of a bean being edited.

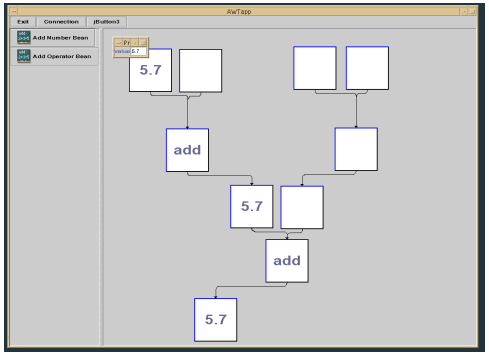
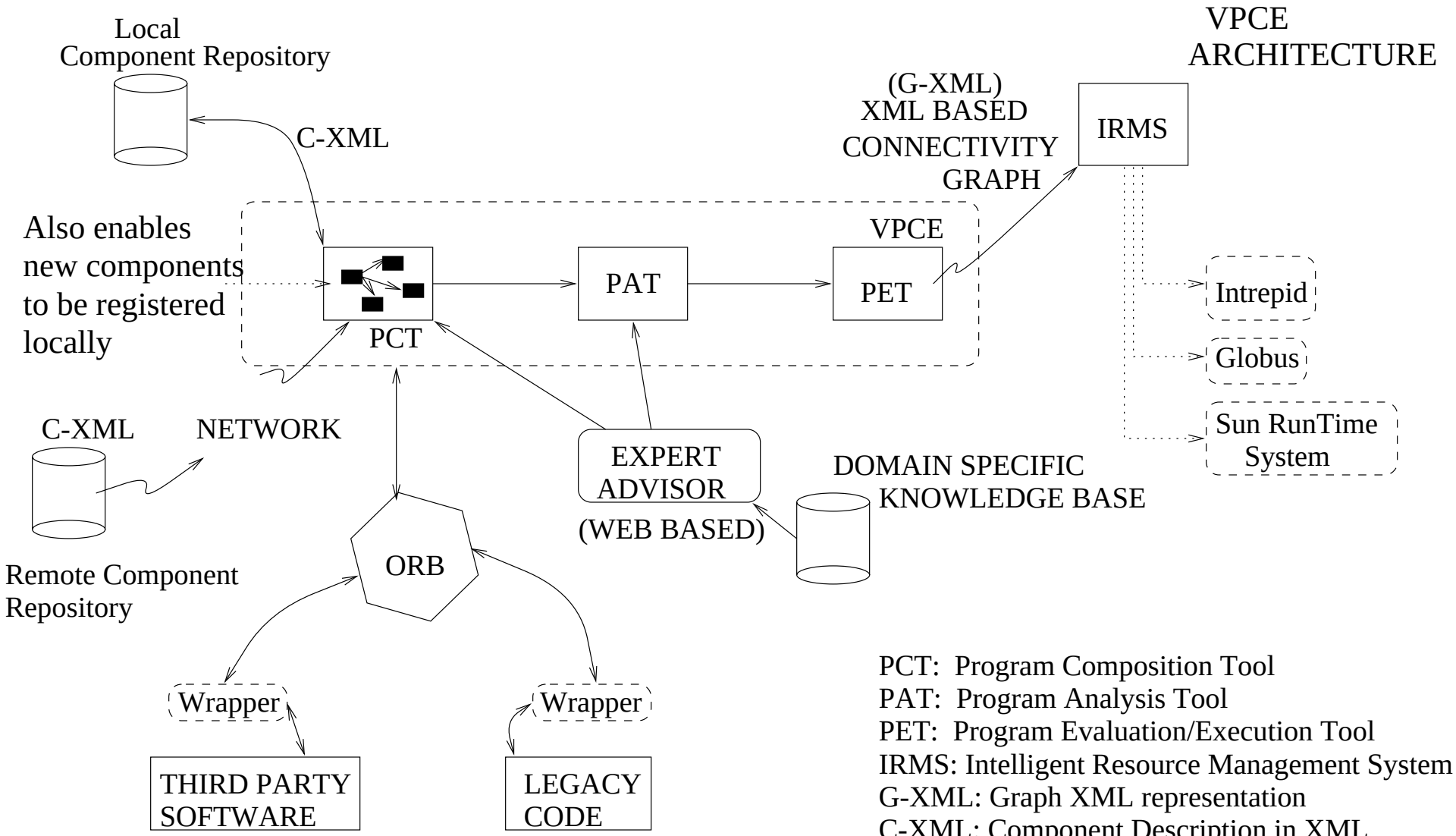


Figure 3

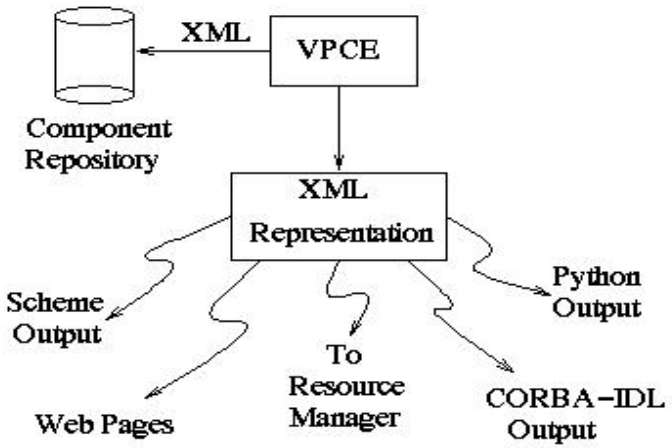
XML Connectivity

```
<preface>
  <name alt=OD id=OD01>Operand</name>
  <pse-type>Generic</pse-type>
  <hierarchy id=parent></hierarchy>
  <hierarchy id=child>OP01</hierarchy>
</preface>

<preface>
  <name alt=OP id=OP01>Operator</name>
  <pse-type>Generic</pse-type>
  <hierarchy id=parent>OD01</hierarchy>
  <hierarchy id=child>OD02</hierarchy>
  <hierarchy id=child>OD03</hierarchy>
</preface>
```

An example of the XML connectivity for the top most two elements in the task graph.


VPCE
ARCHITECTURE
Local
Component Repository
C-XML
(G-XML)
XML BASED
CONNECTIVITY
GRAPH
IRMS
Also enables
new components
to be registered
locally
VPCE
PAT
PET
PCT
Intrepid
Globus
Sun RunTime
System
C-XML
NETWORK
EXPERT
ADVISOR
(WEB BASED)
DOMAIN SPECIFIC
KNOWLEDGE BASE
ORB
Remote Component
Repository
Wrapper
Wrapper
THIRD PARTY
SOFTWARE
LEGACY
CODE
PCT: Program Composition Tool
PAT: Program Analysis Tool
PET: Program Evaluation/Execution Tool
IRMS: Intelligent Resource Management System
G-XML: Graph XML representation
C-XML: Component Description in XML

```
<?xml version="1.0" href=URL?>

<preface>
  <name alt=DA id=DA01>Data Analyser</name>
  <pse-type>Generic</pse-type>
  <hierarchy id=parent>Tools.Data.Data
Analyser</hierarchy>
  <hierarchy id=child></hierarchy>
</preface>

<ports>
 <inportnum>2</inportnum>
 <outportnum>1</outputnum>
  <inportype id=1>float</inportype>
  <inport id=1 type=real>
    <parameter=regression value=NIL/>
  </inport>
  <inport id=2 type=float>
    <parameter=bayesian value=NIL/>
  </inport>
 <outportype> real </outportype>
</ports>

<execution id=software>
 <type>parallel</type>
 <type>MPI</type>
 <type>SPMD</type>
 <type>binary</type>
</execution>

<execution id=platform>
 <type> </type>
</execution>

<help context=instantiate>
<href name=file:/home/pse/help/data-analyser.txt
value=NIL>
```



XML Component Model

The XML based DTD defines the following types of tags:

• Context and header tags, used to identify a component and the types of PSE that a component may usefully be employed in. These details are grouped under the preface tag. The hierarchy tag is used to identify parent and child relationships between components. A parent can have a single parent and multiple children.

• Ports, used to identify the number of input and output ports and their types. An input port can accept multiple data types, and both input and output may be from simple data types or more complex sources, such as files or network streams. In this case the ports need to define an *href* tag, rather than a specific data type.

E.g.
```
<ports>
 <inport id=1 parameter=regression type=stream value=NIL>
   <parameter=regression value=NIL/>
   <href name=http://www.cs.cf.ac.uk/PSE/ value=test.txt>
 </inport>
 </ports>
```

• Execution, a component may have execution specific details associated with it , such as whether it contains MPI code, if it contains internal parallelism etc. If there is only a binary version of the component available, then this must be specified. Such component specific details may be contained within any number of type tags. The execution tags are divided into a software and a platform part. The former identifies the internal properties of the component, while the latter identifies a suitable execution platform.

• Help, a user can specify an external containing help about a component. The help tags contain a context option which enables the association of a particular file with an option. The option can be used to specify certain help files at certain points in a components use. If no help file is specified then the XML definition of the interface is used to display the component properties.

• Configuration, used to identify a configuration file or utility that enables the component to be initialised or customised using predefined vales. This enables a component to be pre-configured given a context or allows the state of a component to be preserved. This is particularly useful when the same component needs to be used in different applications, enabling a user to share parts of the hierarchy, while defining local variations.

• Performance Model, each component has an associated performance model specified in a file. This may range from being a numerical cost of running the component on a given architecture , to being a parameterised model that can account for range and types of data it deals with, to more complex models that are specified analytically.

• Event Model, each component supports an event listener. If a source component can generate an event of type Xevent, then any listener (target) must implement an Xlistener interface. Listeners can either be separate components that perform a well defined action - such as exception handling, or can be more general and support methods that are invoked when a given event occurs. We use an event tag to bind an event to a method identifier on a particular component.

E.g.
```
<event target="ComponA" type="ouput" name="overflow" filter="filter">
    <component id=XX> ... </component>
  </event>
```

The target identifies the component to initiate when an event of a given type occurs on component with identity id, as defined in the preface tag of a component.

Main Features and Technologies

JavaBean Model

•A JavaBean is defined as a reusable software component that can be manipulated visually in a builder tool.
•A bean has exposes
- Properties, internal states that can be set and queried externally by another program.
- Methods, public methods that can be accessed by another program.
- Events. A bean may generate or receive events. A bean defines an event if it provides methods for adding and removing event listeners from a list of interested objects.

•Builder tool
- used to manipulate beans.
- must be able to dynamically load an arbitrary class.
- use introspection/reflection to discover a components properties, methods and events.
- provide a mechanism for dynamically creating the connections between components.

VPCE Event Model

•Each component supports an event listener.
- separate components that perform a well defined action -- such as handling exceptions.
- general, support methods that are invoked when the given event occurs.

```
<event target="ComponA" type="output" name="overflow" filter="filter">
 <component id=XX> ... </component>
</event>
```

- *target,* identifies the component to initiate.
- *type,* when an event of a given type occurs.
- *component id,* source of the event.
- *name,* tag is used to differentiate events of the same type
- *filter,* tag is a place-holder for JDK1.2 property change and vetoable property change event support, or a specific method in the listener interface.

• Event handling may be performed internally within a component, an event listener for each component.
• For legacy codes wrapped as components, separate event listeners may be implemented as components.
• Component listeners may be shared between components within the same PSE.
• Components that contain internal structure, and support hierarchy, must be able to register their events at the highest level in the hierarchy, if separate event listeners are to be implemented.

```
<preface>
  <name alt=DA id=DA02>Data Extractor</name>
  <pse-type>Generic</pse-type>
  <hierarchy id=parent>Tools.Data.Data_Extractor</hierarchy>
  <hierarchy id=child></hierarchy>
</preface>

<event type="initialise" name="start" filter="">
 <script>
   <call-method target="DA01" name="bayesian">
 </script>
</event>
```

• The script tags are used to specify the method to invoke in another component, when the given event occurs.

EXPERT ADVISOR USING JESS (Java Expert System Shell)

Objectives:

1. Provides assistance to users in selecting components. Bases criteria on questions asked from the user, and on component properties expressed in C-XML.

2. Each component has constraints on the types of data it can handle, the internal data distribution that is supported, on licensing and other restrictions. The EA is used to check these constraints before the the data flow graph is constructed and handed to the resource manager

3. A database of known facts is used to infer possible components that may be suitable within a particular application or library

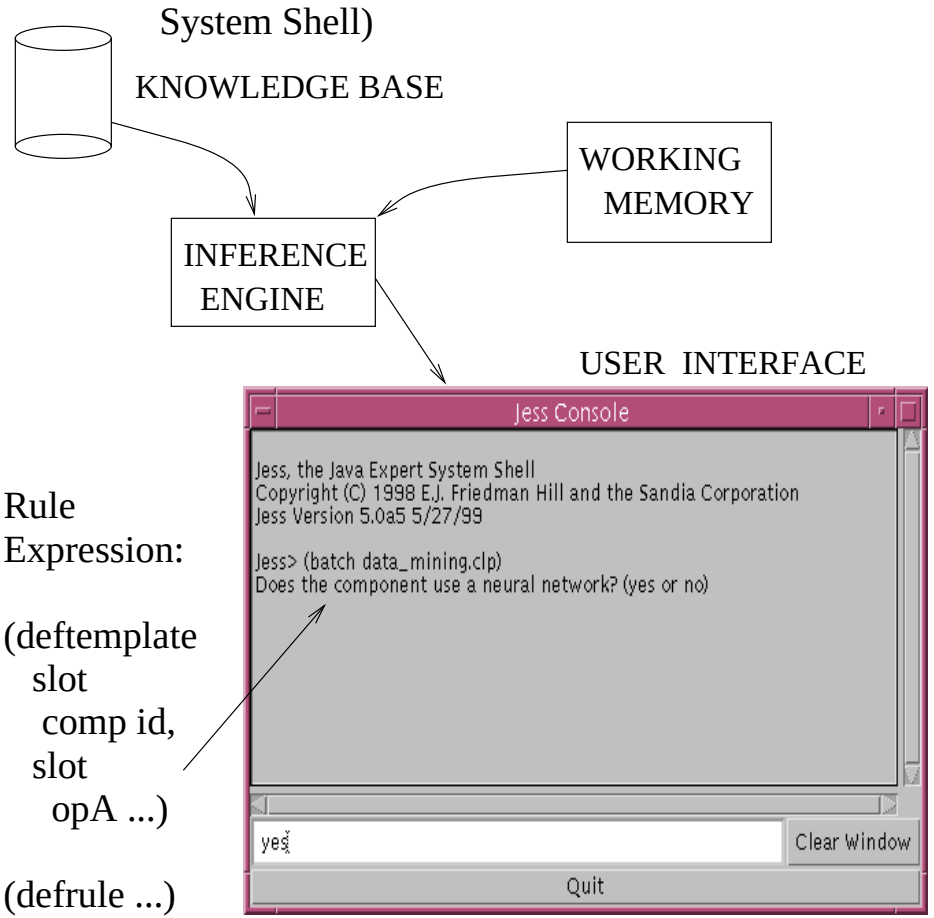
A user is required to provide rules that could be added to the EA's knowledge base, usually maintained locally to the VPCE.

Knowledge is expressed using a frame based representation.

Constraints related to a component can be expressed either as a collection of facts, or through specialised rules, using RULE TEMPLATES supported within the VPCE.

Rule Integrity has to be checked manually -- we do not provide any support for maintaining knowledge integrity at present

Updating and Maintaining Rules



Rule Expression:

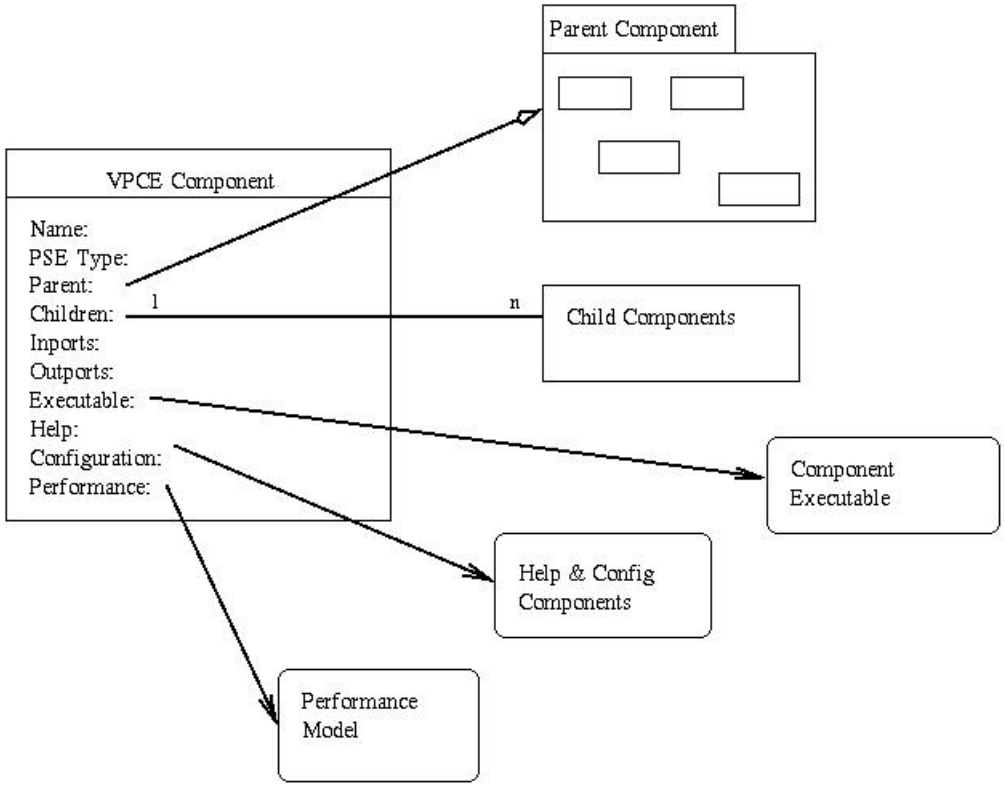
```
(deftemplate
   slot
    comp id,
   slot
     opA ...)

(defrule ...)
(defrule ...)
```

Using Operating Type for Categorisation:

1. Choose Operator:
   a) Laplace b) Helmholtz c) Possion d) Self Adjoint e) Mixed Derivs f) General

2. Equation?
   a) Analytic b) Entire c) Const_Coeff d) Oscillatory e) Singular f) Peaked

3. B-conditions:
   a) Homogeneous b) Dirichlet c) Neumann d) Mixed e) Const_Coeff



Compound Components

Components can be combined within the VPCE to form new, more complex, components that can be stored and then reused as a “black box”. To facilitate this a component has information, about the hierarchy it is part of, encoded in its interface. Each component has a tag that identifies the single parent component and any child components. The hierarchy naming convention is similar to the Java package definition. A component with no children indicates the bottom of the hierarchy.

VPCE Component

Every VPCE Component is represented in the system as a JavaBean, with a well defined interface specified in XML. The Java component provides the visual representation, and means of manipulation, to an underlying component.
The interface for a component describes :-

- the identifying name of the component
- the types of PSE that the component may be used in.
- In the case of components in a hierarchy, the interface identifies the parent component and any children.
- The type and number of input and output parameters.
- The executable underlying component, if applicable.
- A help file or documentation, components will be self documenting within the system.
- An optional configuration file or utility that can be used to customise a component, or preserve a component state.
- A performance model. Each component has an associated cost which can be simple numerical cost to a complex analytical model.