TNI Europe
« the HOOD company »

2000
Created near Manchester (UK)

2001
Acquisition of CP-HOOD from Critical Path
(40 sites, 500 licenses)

2004
Acquisition of Stood from TNI-Valiosys
(20 sites, 150 licenses)
Office in Brest (F)

Release of Stood 5.0
Stood

• An industrial software design tool
• Already deployed & supported on many critical projects (DO-178B, ECSS-E40, MIL-STD-498)
• UML 2.0 front end & AADL plug-in
Background

- Ada 83
- metaH (1993)
- Ada 95
- HRT-HOOD (1995)
- UML 1.1 (1997)
- UML 1.4 (2001)
- UML 2.0 (2004?)

- HOOD 1.0 (1987)
- HOOD 3.0 (1989)
- HOOD 3.1 (1992)
- HOOD 4.0 (1995)
- OMT (1991)
- Stood
- Stood 5.0

- Ada 0y
- AADL 1.0 (Sept 2004)
- Cotre
In line with current trends

• promotes Model Driven Engineering: « designing before coding »
  - advanced modeling solution
  - model transformations

• promotes Component Based Architectures to ease:
  - team development
  - reuse
  - testing
  - maintenance

• promotes flexible Software Design practices:
  - incremental documentation
  - incremental coding and round-trip engineering
  - incremental requirements traceability
  - extensive tool customization capabilities
Tool overview

- **GUI**
- **DataBase**

**input**
- SW Requirements
- Ada legacy code
- C legacy code

**interchange**
- AADL
- XML/SIF

**output**
- Conf. Management

- **model transformation plugins**
- **kernel**

**Req. Traceability Verification reports**
- Source files:
  - Ada 95
  - Ada Ravenscar
  - C/C++

- Documentation:
  - PostScript
  - PDF
  - Word
  - FrameMaker
  - HTML
Model transformations

- Transformation engine
- Stood Plugins
- Generation Rules
- Stood
- Generation Predicates
- Reverse Rules
- Reverse Predicates
- SIF file
- Other language
- Other Components repository
- Analyser
Formal transformation rules
example: AADL generator

• AADL definition:

```plaintext
component_type_extension ::= 
  component_category defining_component_type_identifier 
  extends unique_component_type_identifier 
  [ features ( { feature | feature_refinement }+ | none_statement ) ] 
  [ flows ( { flow_spec | flow_spec_refinement }+ | none_statement ) ] 
  [ properties ( { component_type_property_association }+ | none_statement ) ] 
  { annex_subclause }* 
end defining_component_type_identifier ;
```

• Corresponding code generation rule in prolog:

```prolog
genComponentType(X,C,I,P) :-
  indent(I), write(C), sp, write(X),
  opt_EXTENSION(X,C), nl,
  optFEATURES(X,I,P),
  opt_FLOWSPEC(X,I),
  opt_TYPPROPERTIES(X,I),
  opt Annexes(X,I),
  indent(I), write('END '), write(X), sc, nl, nl.
```
What is a Component?

• **UML 2.0 (final adopted specification)**
  « A component can always be considered an autonomous unit within a system or subsystem. It has one or more **provided** and **required** interfaces (...), and its **internals** are hidden and inaccessible other than as provided by its interfaces. Although it may be dependent on other elements in terms of interfaces that are required, a component is **encapsulated** and its dependencies are designed such that it can be treated as independently as possible. »

• **AADL 1.0 (AS5506)**
  « A **component** represents some hardware or software entity that is part of a system being modeled in AADL. A component has a **component type**, which defines a **functional interface**. The component type acts as the specification of a component that other components can operate against. (...) A component has zero or more **component implementations**. A component implementation specifies an **internal structure** for a component as an assembly of subcomponents. »

• **HOOD (HRM 4)**
  « A HOOD object is thus a software module specification, being primarily an encapsulation of services provided to other client software. (...) An object has a visible part (the **interface**) , and a hidden part (the **internals**) which cannot be accessed directly by external objects. (..) The interface part defines the services (...) **provided** by the object, as well as the services **required** from other objects. »
## Mapping 1/2

<table>
<thead>
<tr>
<th>AADL</th>
<th>UML 2.0</th>
<th>HOOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>component</td>
<td>component</td>
<td>(parent) module</td>
</tr>
<tr>
<td>subcomponent</td>
<td>part</td>
<td>(child) module</td>
</tr>
<tr>
<td>features</td>
<td>provided interface</td>
<td>provided interface</td>
</tr>
<tr>
<td></td>
<td>required interface</td>
<td>required interface</td>
</tr>
<tr>
<td>containment connection</td>
<td>delegate (provided)</td>
<td>implemented_by</td>
</tr>
<tr>
<td></td>
<td>delegate (required)</td>
<td>use (uncle)</td>
</tr>
<tr>
<td>components connection</td>
<td>assembly</td>
<td>use (sibling)</td>
</tr>
</tbody>
</table>
## Mapping 2/2

### predefined Components

<table>
<thead>
<tr>
<th>AADL</th>
<th>UML 2.0</th>
<th>HOOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>System instance</td>
<td></td>
<td>System configuration</td>
</tr>
<tr>
<td>System</td>
<td></td>
<td>Non terminal object</td>
</tr>
<tr>
<td>Process</td>
<td></td>
<td>Active root object</td>
</tr>
<tr>
<td>Thread group</td>
<td></td>
<td>Active non terminal object</td>
</tr>
<tr>
<td>Thread (aperiodic)</td>
<td></td>
<td>Active terminal object</td>
</tr>
<tr>
<td>Thread (periodic)</td>
<td></td>
<td>Cyclic object</td>
</tr>
<tr>
<td>Thread (sporadic)</td>
<td></td>
<td>Sporadic object</td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td>Protected object</td>
</tr>
<tr>
<td>Package</td>
<td></td>
<td>Passive terminal objects</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Class</td>
</tr>
</tbody>
</table>
why the AADL?

- The AADL is System oriented and can be used in the early phases of a project.
  - It complements and easily interacts with the UML 2.0 / HOOD Software modeling approach
  - It may become an efficient communication media all along the project lifecycle.

- It brings a default predefined behavioural semantics to real-time components.
  - It can be used at System level for simulation
  - It can be used at Software level for advanced real-time code generation

- It offers wide extension mecanisms
  - Property_sets and Annexes
  - Already used by the COTRE (ending) and ASSERT (starting) projects

- It is already supported by the industry of critical systems in the USA and in Europe.
Graphical notations

Note: an annex of the AADL standard also defines a specific graphical notation
Summary

UML gives the general background:

What is a component?

+ AADL brings precise semantics for real-time components:

What is the behaviour of a periodic thread?

+ HOOD offers a well structured process to build the system:

How do I define and assemble my components?

=

Stood provides the appropriate framework to support all that in the context of real industrial projects:

- productivity: distributed development, reuse of legacy data, code generation
- quality: verifications, documentation, certification issues
Use case 1
AADL Modeller

• UML 2.0 structure diagrams front end
• HOOD design rules:
  – visibility rules
  – information hiding (i.e. for ports)
  – immediate C, Ada, ... and doc generation
• AADL 1.0 generator
• AADL 1.0 semantics checker (under dev.)
• AADL 1.0 code generation rules (under dev.)
• future possible improvements:
  – AADL graphical notation,
  – XML output, ...
Use case 2
"bridging the gap"

• using AADL as a System to Software bridger
• importing AADL 1.0 specifications
  – to be developed with other AADL compliant tools
  – preserving the System architecture
• standard Software development process
  – SW architectural design refinement
  – SW detailed design and documentation
  – SW coding and round-trip engineering
• using the AADL output again for V&V
  – checking System to Software compliancy
  – connecting to external Verification tools (i.e. Cheddar)
  – implementing the COTRE annex
Use case 3
reusing legacy systems

• a three steps process:
  – Ada or C legacy code reverse engineering
  – architecture adjustments at SW design level
  – AADL generator

• benefits:
  – let existing source code components be made visible for
    new systems at high level
  – building non proprietary format component libraries
  – facilitating reuse of specialized building blocks
Features summary 1/2
Support of the Software Design activities

Architectural Design
- components based approach with black-box and white-box views
- UML 2.0 graphical notation
- AADL import/export
- support of HOOD and HRT-HOOD methodology
- built-in real-time model

Detailed Design & Coding
- customizable structured detailed design framework
- incremental documentation
- incremental coding and round-trip engineering
- incremental requirements coverage
- legacy Ada and C code reverse engineering

Verifications
- cross references table
- automatic calculation of the required interfaces
- automatic generation of call trees and dataflow graphs
- real-time schedulability analysis
- requirements traceability matrix
- design rules checker
- design metrics
Features summary 2/2
Workflow Integration

Project management
- full Windows-Unix interoperability
- network distributed project bases
- integrated interface to remote Configuration Management Systems
- multi user management at system and subsystem level
- SIF and XML design model interchange

Requirements traceability
- import of high level requirements
- incremental requirements coverage
- management of the derived requirements
- bidirectional interface with Reqtify™

Compliancy to Standards
- DO-178B for embedded avionics
- ECSS-E40 for space systems
- EN-50128 for railways
- MIL-STD-498 for military

Code & Doc generators
- Ada95
- C/C++
- HTML
- PostScript/PDF
- RTF (Word™)
- MIF (FrameMaker™)
Try it...

stood@tni-world.com
www.tni-world.com