THINK
C Implementation of Fractal and its ADL tool-chain

Fractal Workshop

July 2006

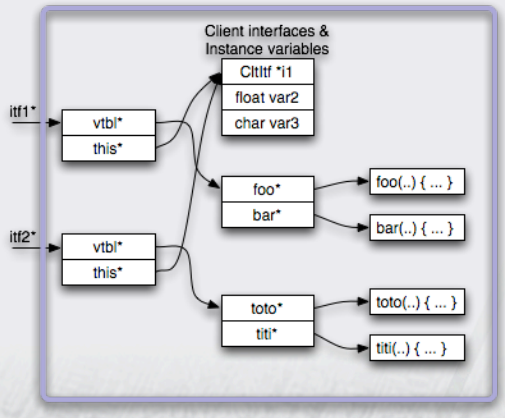
Matthieu Leclercq (ST), Olivier Lobry (France Telecom),
Erdem Özcan (ST, INRIA), Juraj Polakovic (France Telecom),
Jean-Bernard Stefani (INRIA)

Outline

- Think
  - Fractal implementation in C
  - ADL tool-chain for code generation
  - Kortex: A component library for OS construction
- Fractal ADL Compiler for Think
  - Front-end
  - Back-end
  - Plug-in framework
- Conclusions

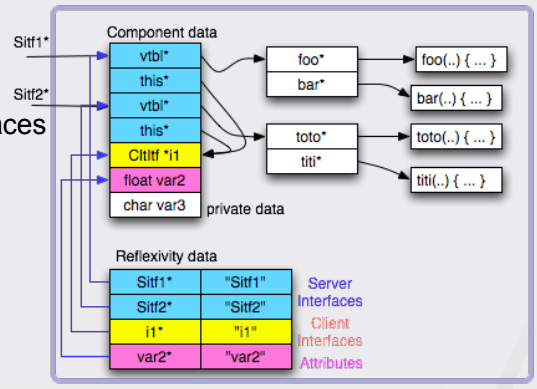
Component implementation

- Binary component model
  - Similar to Microsoft COM



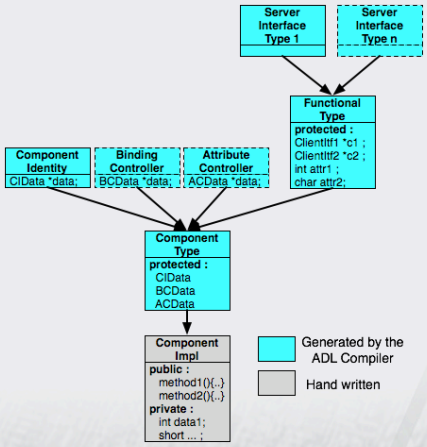
Component implementation

- Binary component model
  - Similar to Microsoft COM
- Support for Fractal interfaces
  - Control interface code
  - Control data

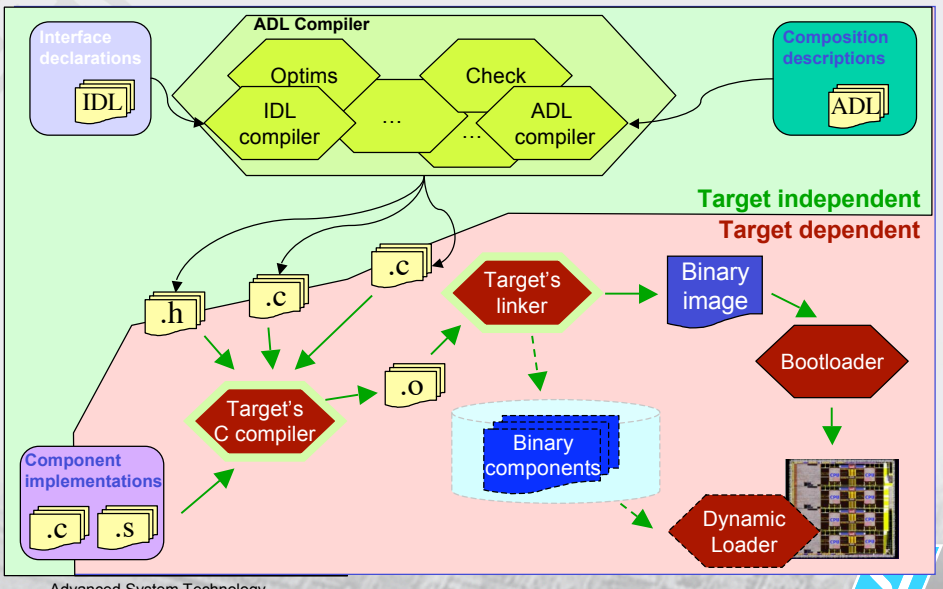


Component implementation

- Binary component model
  - Similar to Microsoft COM
- Support for Fractal interfaces
  - Control interface code
  - Control data
- Code generation & Compilation
  - Help the programming of primitive components
  - Generate code for composite components
  - Bootstrap code generation



Fractal/Think Compilation chain



Kortex: a component library for OS construction

- Components for
  - Classical OS services:
    - ✓ schedulers, memory management, MMU, exception handling, file systems, …
  - Communication services:
    - ✓ TCP/IP, PPP, GPRS, Bluetooth, radio, …
  - Device drivers:
    - ✓ framebuffer, touchpanel, keyboard, serial port, disk, ethernet, …
- Supported Platforms
  - ARM (iPAQ, Apple iPod, ixp425), AVR, PowerPC, ST200

Outline

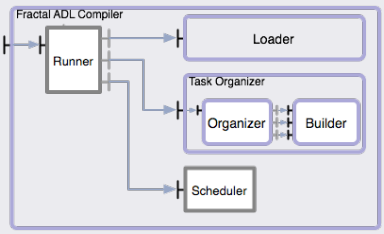
- Think
  - Fractal implementation in C
  - ADL tool-chain
  - Kortex: A component library for OS construction
- Fractal ADL Compiler for Think
  - Front-end
  - Back-end
  - Plug-in framework
- Conclusions

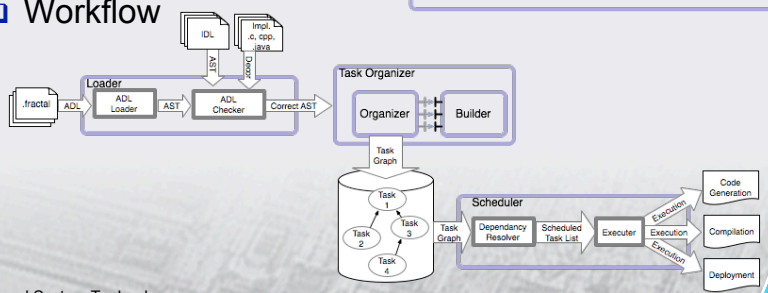
Code generation tool-chain for Fractal ADL

- Starting point : Fractal ADL Factory
  - Deployment from ADL for Fractal/Java
    - ✓ Implemented in Fractal/Java
    - ✓ Component-based architecture described in Fractal ADL
  - Modular
    - ✓ Clear, accessible and modifiable architecture
- Contribution
  - Support for code generation & compilation
    - ✓ New and multiple input languages (Think ADL, Think IDL, join-patterns, etc.)
    - ✓ Support for new and multiple implementation languages
  - Extensibility improvement
    - ✓ Easy support for new ADL features
    - ✓ Allow for third-party developer extensions with plug-ins

Overview

- High-level Architecture
  - Front-end : Loader
  - Back-end :
    - ✓ Organizer
    - ✓ Task framework
    - ✓ Builder
- Workflow

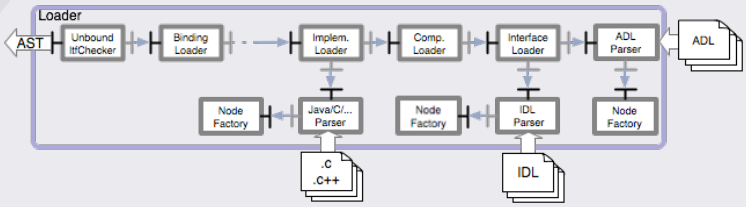




Front-end
Translation of input files to an AST

- Specification
  - Input: ADL, IDL and component implementation files (C, C++, Java, etc.).
  - Output: Unified Abstract Syntax Tree
- Main features
  - Extensible for supporting new input languages.
  - Fine-grain components each responsible for a specific analysis.
  - Robust and extensible AST implementation.
    - ✓ Dynamically generated implementation of AST nodes.
    - ✓ Programmable node factory based on DTD.
    - ✓ Extension transparency for modules that are not involved by thanks to multi-facet nodes.

Internal architecture



- The loader is designed as a component-chain
  - Very modular and extensible
  - Allows for multiple parsers at different stages
  - Simple programming pattern

```
Node load(String name){
  Node myAST = client->load(name) ;
  myCheckOperation(myAST);
  return myAST ;
}
```

Outline

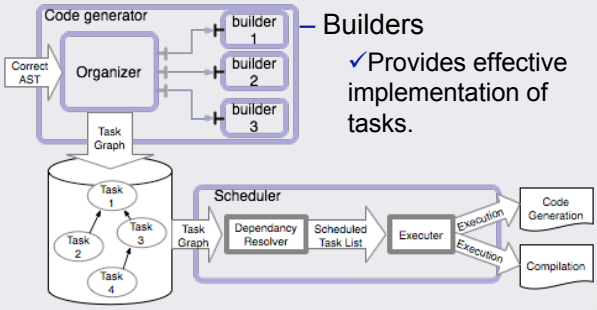
- Think
  - Fractal implementation in C
  - ADL tool-chain
  - Kortex: A component library for OS construction
- Fractal ADL Compiler for Think
  - Front-end
  - Back-end
  - Plug-in framework
- Conclusions

Code generator & Compiler

- Specification
  - Input : A correct AST
  - Output :
    - ✓ Generated code
    - ✓ Compilation
- Main features
  - **Modular:** code generation is split into fine-grain components
  - **Hierarchical** and collaborative: components may aggregate code pieces that are generated by others
  - **Extensible:** components can be added, removed or modified without impact on the rest of modules.
  - **Retargetable :** supports multiple backends for different general purpose programming languages.

Internal Architecture

– Organizers
  ✓Creates code generation and compilation tasks

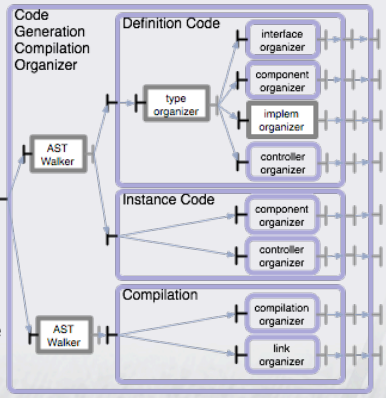


– Builders
  ✓Provides effective implementation of tasks.

- Task framework
  - ✓ Tasks reifies code generation and compilation operations
  - ✓ Dependency declarations
  - ✓ Execution of tasks in a correct order

Organizers

- Maestro of the code generation
  - Input : Correct AST
  - Output : A task graph
- Component based visitor pattern
  - AST walkers
    - ✓ Walks recursively in the AST
    - ✓ Invoke connected visitors for each component node
  - Component visitors :
    - ✓ Responsible for a specific purpose
    - ✓ Create the build tasks

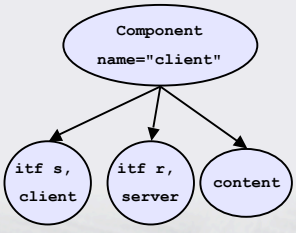


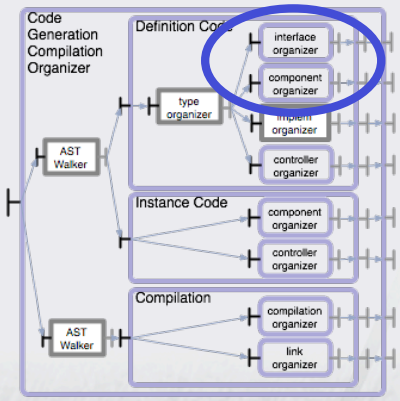
Builders

- Features
  - Builders implement effective code generation/compilation operations
  - They are specific to a given backend (C, C++, etc.)
  - Builders are passive components. They are executed by tasks
  - Fine-grain components, each specific for a given purpose
    - ✓ Interface Attribute /definition, Component definition, instantiation, etc.
- Input
  - Parameters coming from the AST
    - ✓ Interface properties, Attribute value, etc.
  - Results of other builders
    - ✓ Source code, code pieces, files, backend AST, etc.
- Output
  - ✓ Source code, code pieces, files, backend AST, etc.

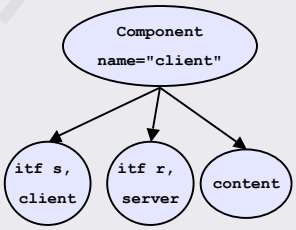
Hierarchical Code Generation

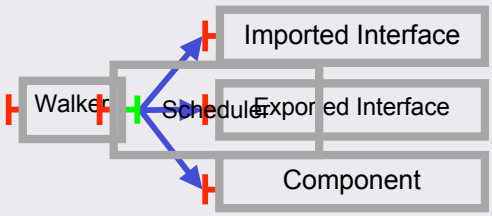
```
<component name="client">
  <interface name="r" role="server"
             signature="Main"/>
  <interface name="s" role="client"
             signature="PrinterService"/>
  <content class="ClientImpl"/>
</component>
```

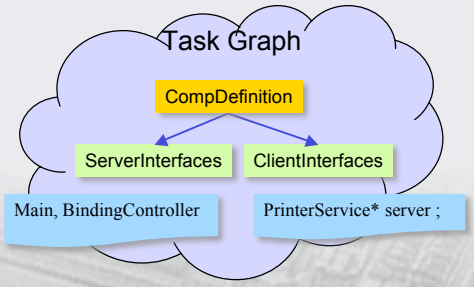




Hierarchical Code Generation







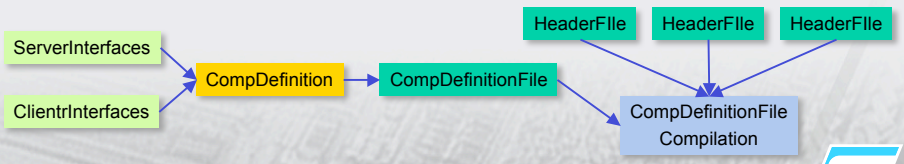

```
class ClientType : Main, BindingController{
  PrinterService* server ;
}
```


Task Framework

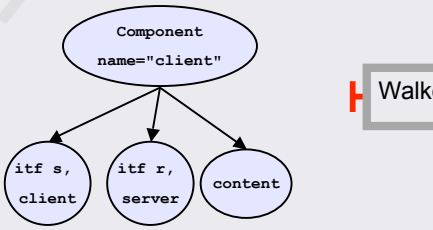
Task types for code generation & Compilation

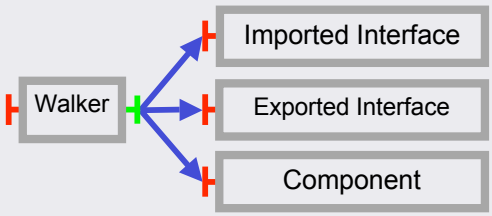
	type name	instance name	source code	file
TypeProvider	↑	-	-	-
InstanceProvider	↓	↑	-	-
SourceCodeProvider	↓	↓	↑	-
SourceCodeConsumerProvider	↓	↓	↓ ↑	-
SourceFileProvider	↓	↓	↓	↑
FileProvider	↓	↓	-	↑
FileConsumerProvider	↓	↓	-	↓ ↑

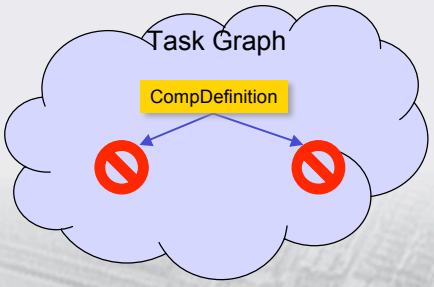
- A complete task graph



Organizer execution order issue



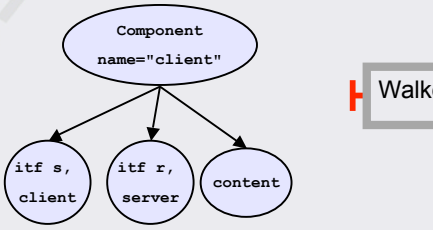


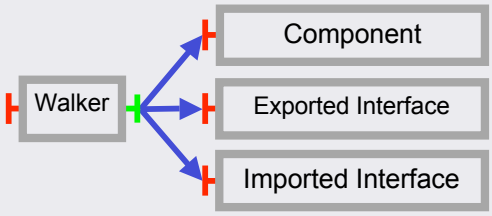


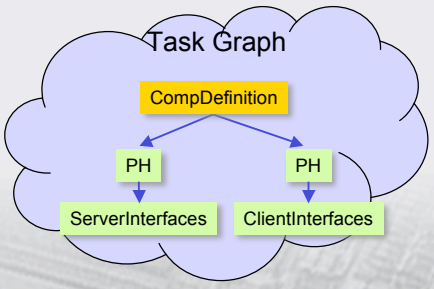
Advanced System Technology

21

Task place holders



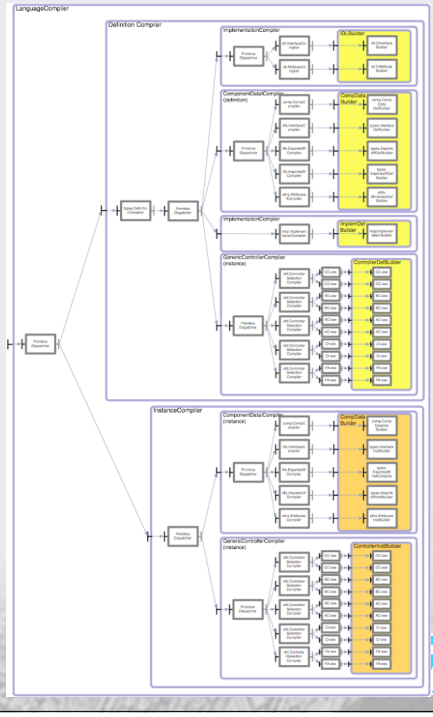




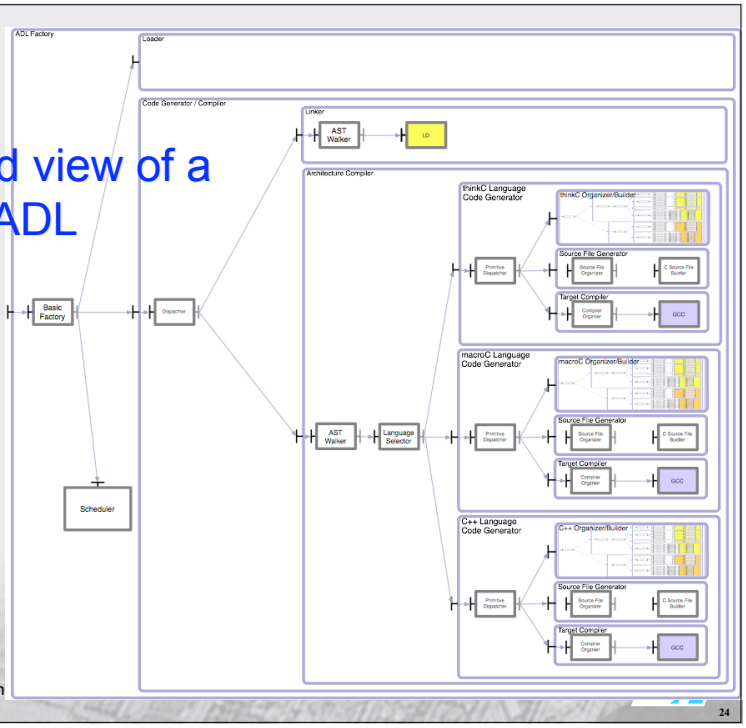
Advanced System Technology

22

View of a realistic organizer/builder



Simplified view of a realistic ADL Factory

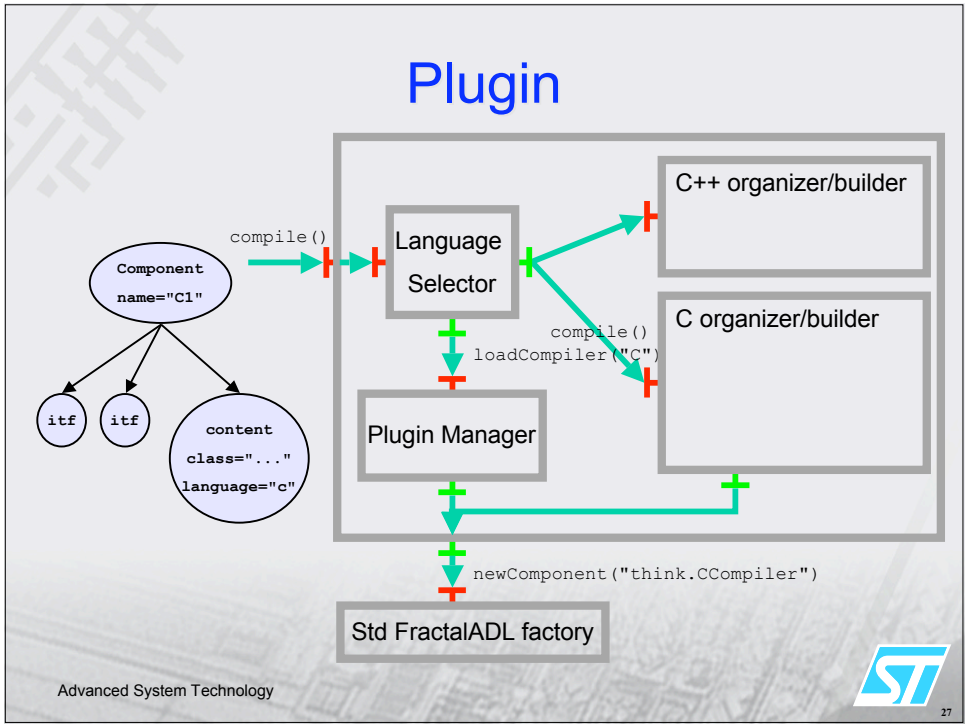


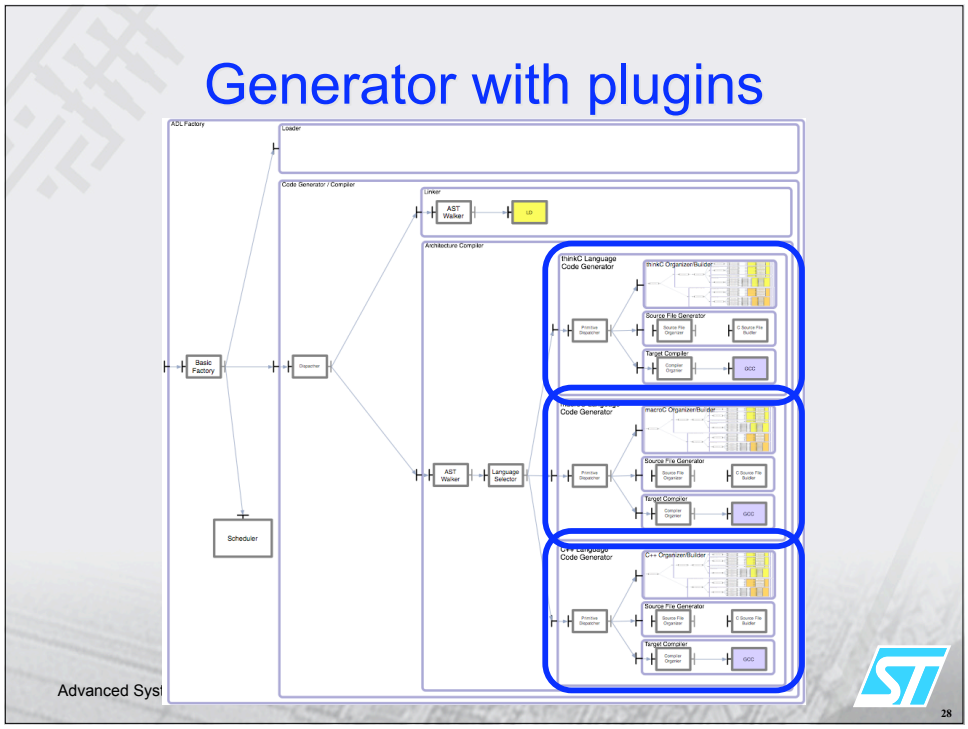
Outline

- Think
  - Fractal implementation in C
  - ADL tool-chain
  - Kortex: A component library for OS construction
- Fractal ADL Compiler for Think
  - Front-end
  - Back-end
  - Plug-in framework
- Conclusions

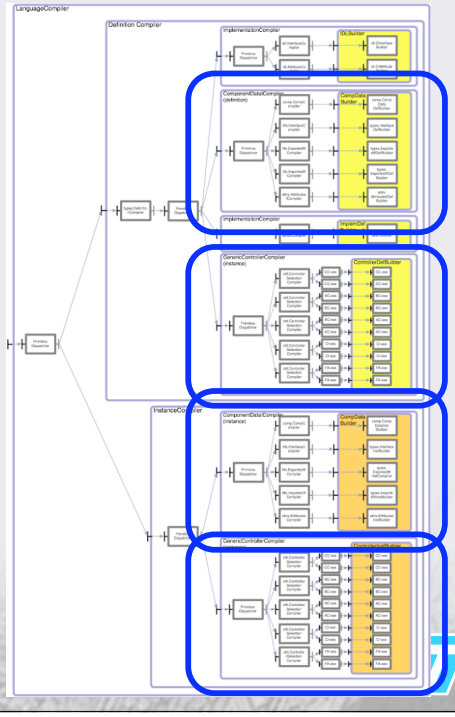
Plug-in framework

- Assessment
  - Very regular but almost huge architecture
    - ✓ Multiplicity of organizers with different builders
  - Presence of many optional modules
    - ✓ Multiplicity of supported languages
    - ✓ Different controller implementations
  - Lots of ADL files
    - ✓ Difficult to maintains
    - ✓ Limit the extensibility by third-party developers
- Motivation for plug-in based design
  - Implement a sort of "exo-factory"
    - ✓ Only the strict minimum is hard-wired
  - On-the-fly adaptation of the compiler for the compiled ADL
    - ✓ Optional modules are loaded at run-time when needed


Plugin
compile()
Component
name="C1"
itf
itf
content
class="..."
language="c"
Language
Selector
C++ organizer/builder
C organizer/builder
compile()
loadCompiler("C")
Plugin Manager
newComponent("think.CCompiler")
Std FractalADL factory
Advanced System Technology
27


Generator with plugins
Advanced Sys
28

organizer/builder with plugins



Conclusion
Assessments

- THINK
  - Fractal component implementation in C/C++
  - ADL based tool-chain
  - Kortex component library for OS construction
- Fractal ADL tool-chain
  - Front-End
    - ✓ Supports multiple input languages
    - ✓ Builds a unified XML-based AST
  - Back-end
    - ✓ Novel approach with fine grain components
    - ✓ Dynamic organization on top of the task framework
    - ✓ Very extensible and multi-target.
    - ✓ Place order to avoid the organizer execution order issue.
  - Plug-in framework
    - ✓ Simplifies the core architecture
    - ✓ Makes it extensible by third party programmers
    - ✓ Can be applied at any stage of decision

Current activities

- Component Model
  - Dynamic reconfiguration support in OS (**FT R&D, Inria**)
  - Isolation and access control for secure systems (**FT R&D**)
  - Behavioral analysis **(FT R&D, VERIMAG)**
  - QoS support **(FT R&D, INSA Lyon)**
- Component library
  - Support for multi-processor platforms (**STM**)
  - Customizable multimedia applications (**STM**)
  - Middleware for dynamic configuration management **(STM)**
- Tool-chain
  - Optimization of Think components **(FT R&D, STM)**
  - Multi-target programming support: C, C++, Java **(STM)**
- Source code and documentation

  http://think.objectweb.org