

May 6-10, 2007

San Jose Convention Center

San Jose, California, USA

L08

How to call Informix 4gl code from J2EE



IDUG® 2007

North America

Sergio Ferreira
MoreData

Tuesday, May 8, 2007 • 16:20 a.m. – 17:00 a.m.

Platform: Informix



GoFurther

Agenda

- The need for calling 4gl
- J2EE Quick introduction
- Alternatives to solve the problem
- The solution we choose

The problem of software

- Software evolves along with business changes.
- The software is very well adapted to the businesses.
- There are new channels that should be addressed (web applications, web services, integration with new systems).
- The normal temptation is redevelop everything.
- SOA gurus are telling us:
 - Keep with the software that works.
 - Integrate using agnostic technology.

Why call 4gl from Java 2 EE

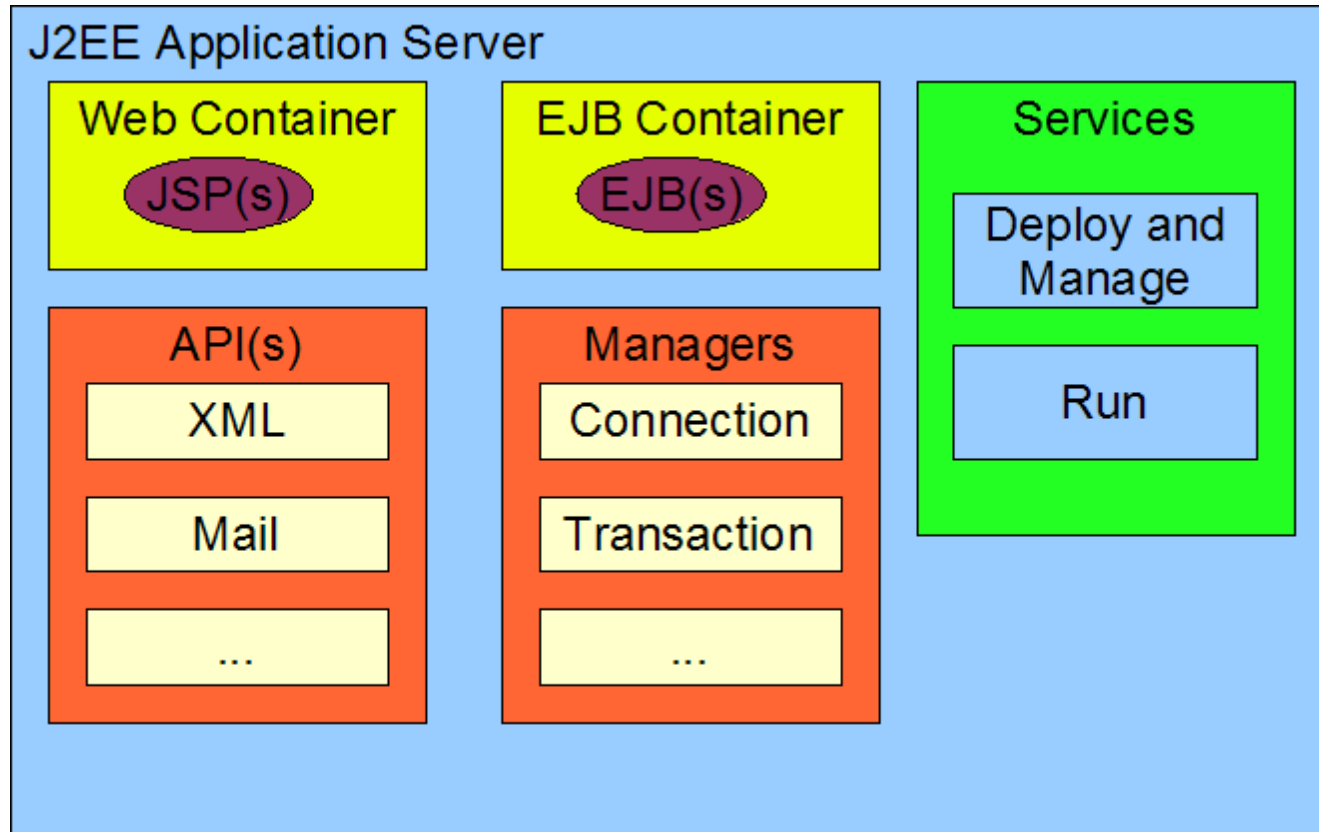
- 4gl supports the business needs.
- It is a tool with a huge productivity and a quick training cycle.
- It's fast ! It's easy ! Why should we change ?
- J2EE is a multi-vendor standard way to create enterprise applications.
- Its good to create web applications, web services, etc
- But J2EE is very complicated. Poor productivity and need for a lot of people training.
- The answer is: **Integrate 4gl with J2EE**

What is J2EE

- A platform to create applications in Java
- Defined by a standard specification
- It is based on the existence of an application server
- Includes specifications for a lot of enterprise requirements:
 - User interface (JSP, portlets, JSF, servlets, swing)
 - Business Rules – EJB(s)
 - Database access – JDBC
 - “Legacy” integration – JCA
 - XML
 - Web services
 - Directory services
 - Transaction management
 - Authentication and Authorization
- The compliant AS implement the specifications

The Application Server

- A Java server that offers : Containers, api(s), managers, admin tools



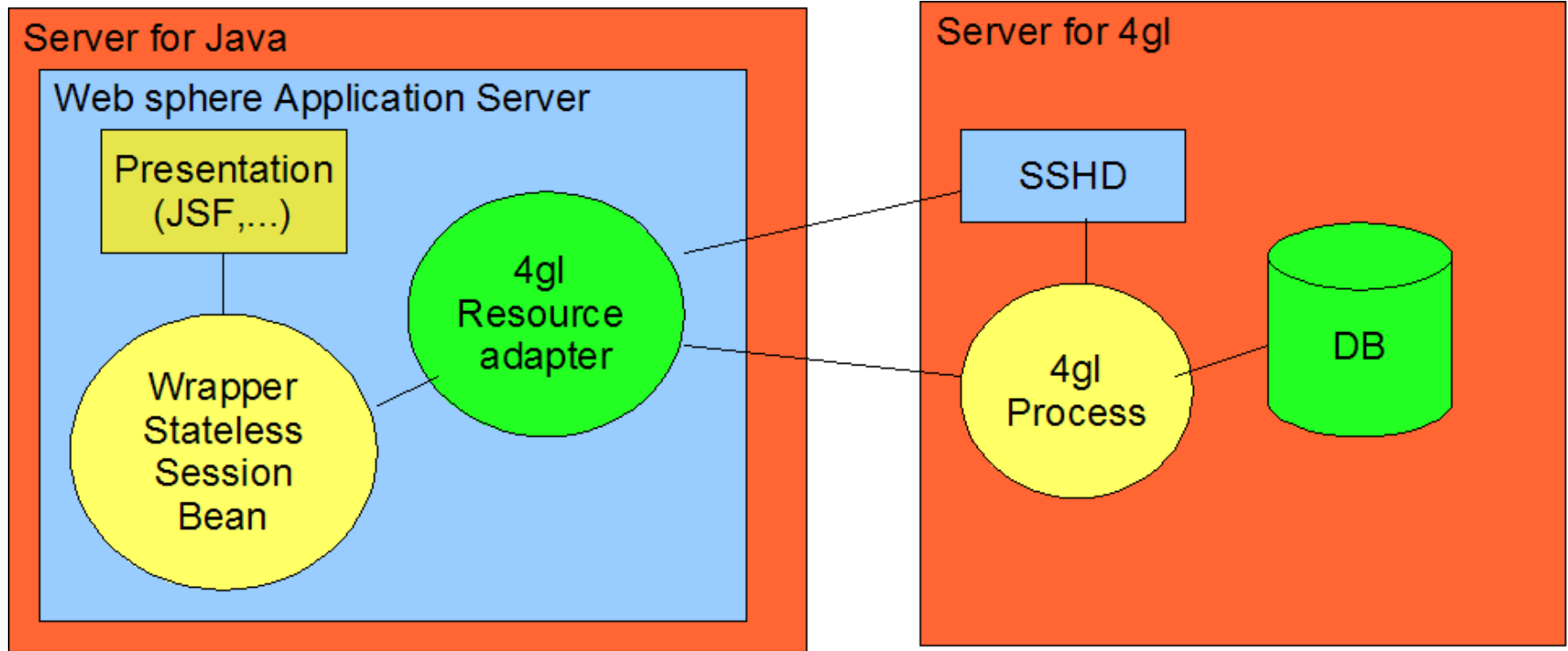
Why not a native call to 4gl

- It can cause problems in the Application Server
- Even when the native code is working properly it is always suspected when problems occur.
- 4gl does not work on a multi-threaded environment
 - The function call stack is static
 - The connection mechanism is based on ESQL/C that works on MT but not on generated from 4gl.

The solution

- Use JCA
- Create a way to spawn 4gl processes
- Communicate between the Resource adapter and the 4gl process

How it works

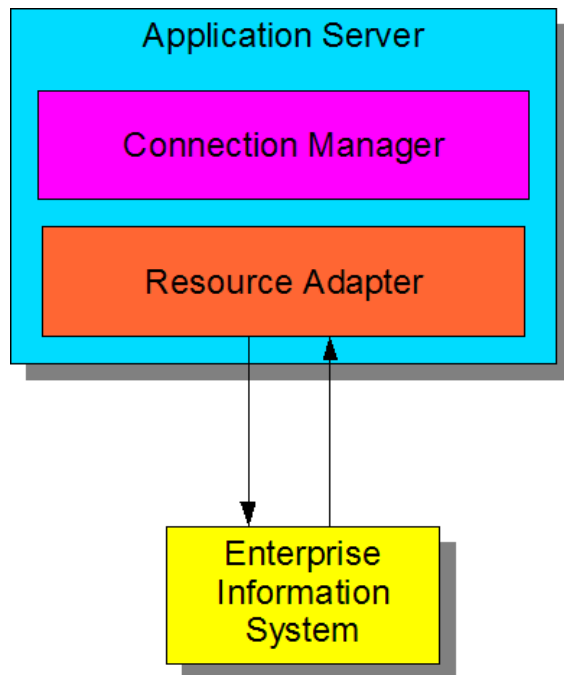


What is JCA

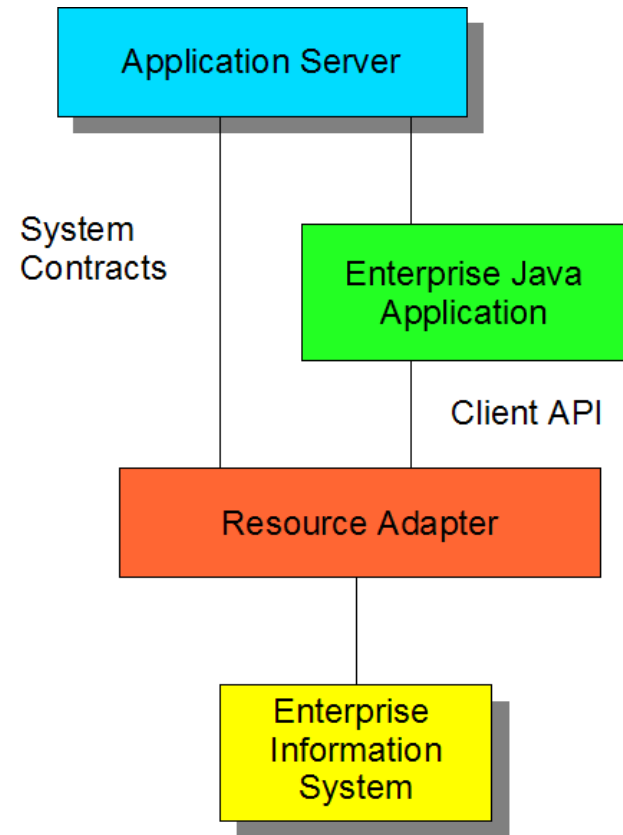
- JCA stands for Java Connection Architecture
- JSR 112
- It is part of J2EE
- It is a standard for connecting Java Application Servers and Enterprise Information Systems (EIS)
- Defines a piece called Resource Adapter that works according a contract
- JDBC could be (and some times is) seen as a Relational Resource Adapter

JCA

- Connections



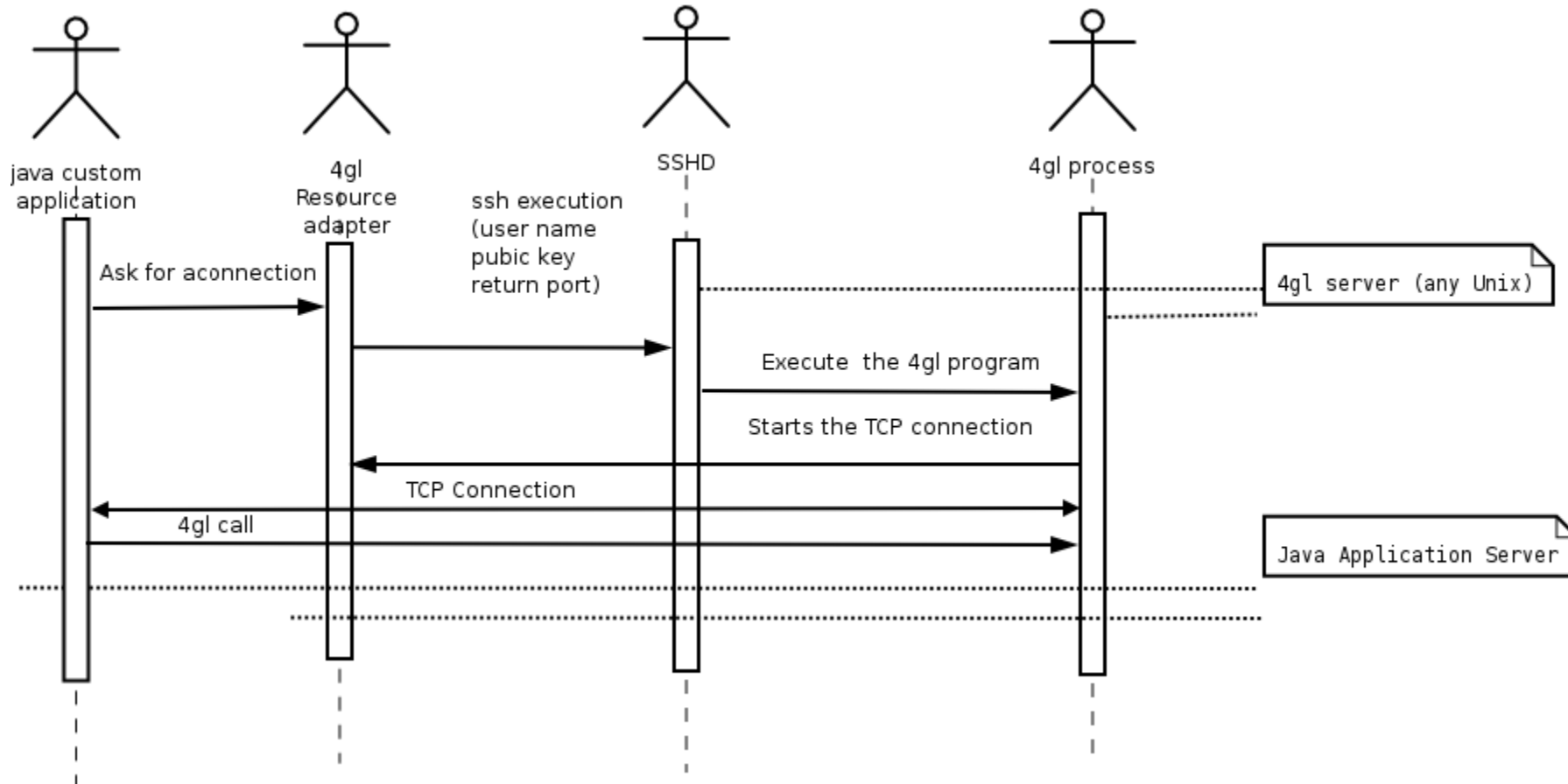
- Contracts



What do we have implemented

- A JCA resource adapter
- A communication mechanism between the 4gl process and the Resource Adapter
- A C library to handle all the 4gl calls and the communication between the AS and the specific function in the 4gl process
- Execute the calls to the remote functions from the java side using the JCA interface (javax.resource.*)
- A way to generate the Java wrappers to 4gl functions

How it works



Work to be done

- This approach is usable to business rules without user interface.
- You can connect to Interactive 4gl programs using a ssh/telnet java applet inside the browser.
- Screen scrapping work can be done redirecting the standard output and input of the 4gl process to the AS
- Use solano to export 4gl functions as stored procedures 😊

What do we gain

- Reuse of “legacy” business rules without shutting down the current 4gl applications.
- Less problems.
- Much lower cost than redeveloping.
- Much more faster productivity by creating the business rules in much higher level language
- 4gl is far more easy to use on this task than java

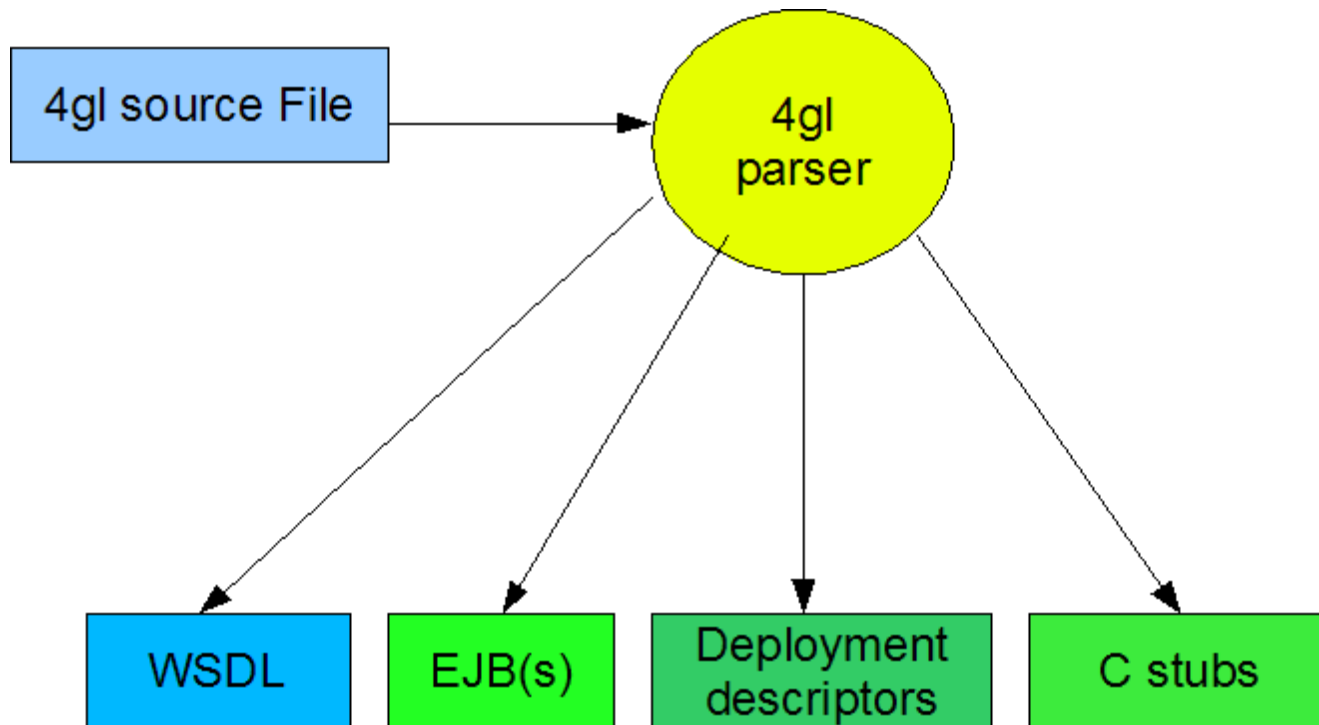
Overhead and scalability

- We measured a medium call overhead of 40 ms
- Scalability:
 - The connection pooling gives a very good way to call functions without forks and new db connections
 - We tested with a medium load of 100 function call(s) in a second and it worked during a whole week.

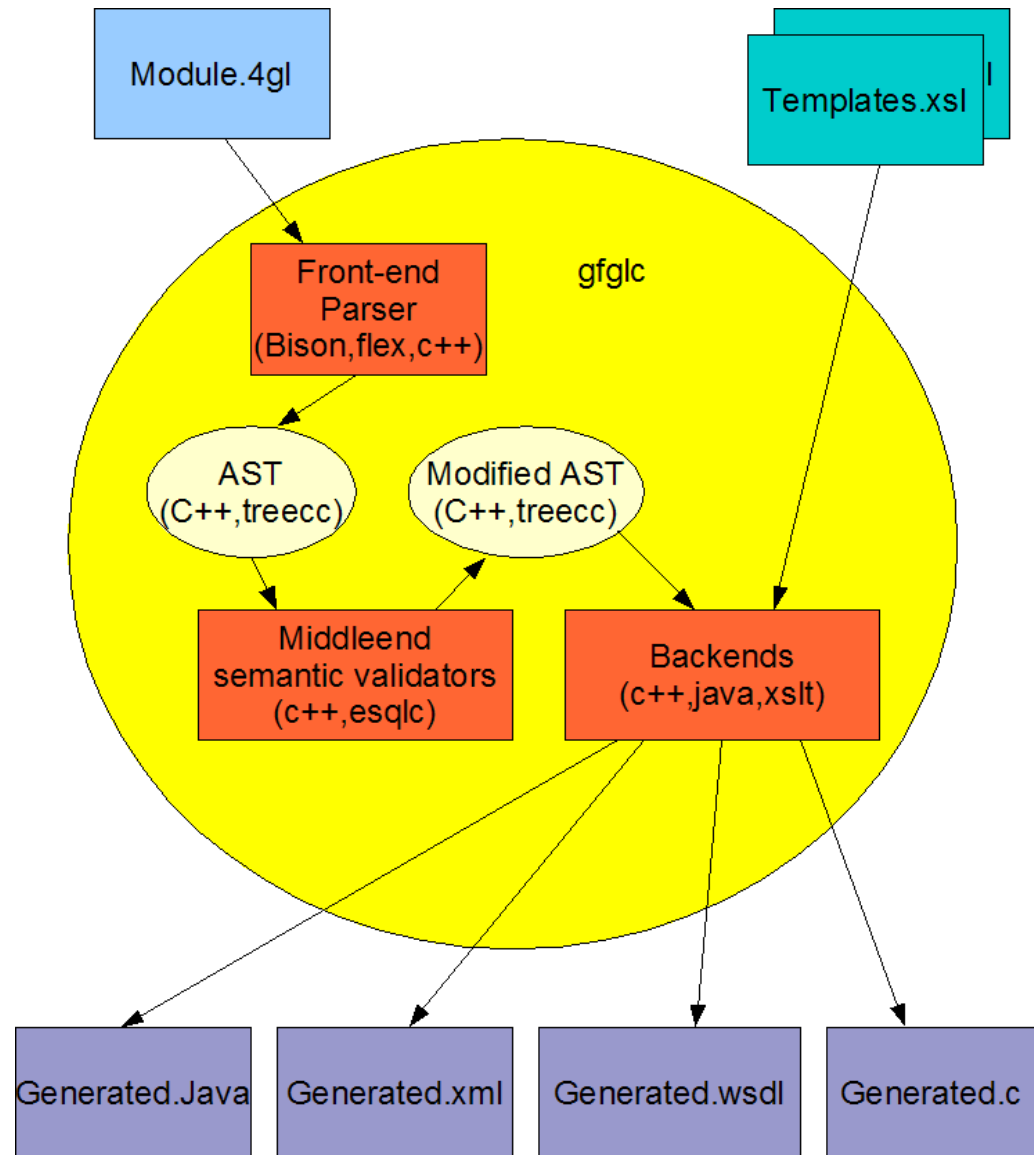
To gain productivity

- It must be easy to export functions to java without needing to write code.
- Automate the exporting process to expose 4gl functions as Stateless Session Beans.
- It is very important because it avoids high cost in exporting the functions.
- It could be made in severall ways from a simple script to a complete 4gl parsing process.
- We created with a parser and we got the 4gl functions in Java and web services in 30 s

Automatic generation



Code generation



Some features

- 4gl executed as the user that was authenticated on AS
- Array(s) as parameters and return values.
- “javadoc” version of Comments(known as fgldoc).
- Special tag(s) on comments control the code generation (ex : @soamethod defines that the function will be exported).
- Authorization to access functions defined in the fgldoc (@authorization_id) and executed on Java side.
- Web services generation

Some features (2)

- 4gl “whenever” handler that returns all 4gl error information to a Java exception
- Hot deploy of new version of the 4gl applications.
- Administrative console in JSF.
- Template based code generation (with XSLT).
- Ssh Key pair based security spawning processes.
- Automatic initialization functions
- Its portable
 - Tested on AS : Websphere ; Geronimo ; JBoss
 - Tested with : Tradicional 4gl ; Aubit 4gl
 - It should work with : 4js Genero ; Querix ; Supra 4gl
 - Tested in Linux and Tru64

Next steps

- Call Java and web services from 4gl in a productive way..

Conclusion

- If you need to call Informix 4GL programs from Java use a JCA resource adapter and code generation.
- It works ! It is a tested solution ! It's in production on the biggest Portuguese telco !
- Its an alternative to everyone that needs to modernize 4gl applications without compromise current applications.
- It enables SOA in your 4gl applications at low cost.

Questions



Session L08

How to call Informix 4gl code from J2EE

Sergio Ferreira

IIUG

sergio@iiug.org