

An Introduction to Intelligent Services

Thomas Steiner
Institute for Business Information Systems
University of Applied Sciences Valais
Sierre, Switzerland
thomas.steiner@hevs.ch

Abstract

This paper introduces the programming of intelligent services for the World Wide Web. It presents the shortcomings of sequential programming with languages like for example Java, C++, Pascal and Basic for some cases of translating business processes into computer programs. Declarative programming is presented as an alternative where the developer experiences the limits with sequential programming. The declarative language Prolog is very briefly introduced. Then, the shortcomings of traditional Prolog environments are also discussed. One of the major weaknesses of these environments being the integration of Prolog into mainstream languages (like for example Java), this point is addressed specifically. The paper continues by demonstrating how Prolog can be extended with host languages to elegantly overcome some of its limitations. Then, Web services are briefly introduced. The remainder of the paper addresses the goal of deploying a Prolog logic server (Amzi!Prolog) as a Web service in a mainstream Web application container (Tomcat / Axis). Finally, a demo client is presented, which invokes these logic services. The paper ends with a discussion of application perspectives and conclusions.

Key words: Declarative programming, Prolog, logic server, embedded predicates, extended predicates, Web services, Tomcat, Axis, WSDD, WSDL, Java, Intelligent Agents, AI.

Table of Contents

Introduction.....	2
The shortcomings of sequential programming.....	3
Introducing declarative programming.....	4
The shortcomings of traditional Prolog engines.....	8
Embedding Prolog in host languages.....	9
Extending Prolog with host languages.....	14
Installing the Web application container (Tomcat).....	17
Installing the Web service extension (Axis).....	20
Deploying the Logic Server as a Web service.....	23
Deriving the client stubs.....	28
A client for logic services.....	30
The family service.....	32
Conclusions.....	33
Exercises.....	34
References and further reading.....	35

Introduction

Ten years ago – in 1995 – intelligent agents had known an astonishing increase in popularity. After higher programming languages, human-computer interfaces, networking and object oriented programming, the software industry seemed to have found a new paradigm, and AI people claimed to be at the roots of significant progress in complex software development.

Since the early 1990s already, the production of research papers addressing the subject of agents increased significantly. Following this trend, an impressive number of implemented systems appeared and was documented in the research community. Hence, despite all the promises for imminent release and availability to every human being by means of putting intelligent agents on the World Wide Web, not many systems really made it to reach the masses.

Hence, the term intelligent agent has become widely used today. There are in fact many systems that claim to be agent-based. The software industry has turned its attention to distributed component technology and has refined the notion of objects as units of software design that encapsulate as well as expose their state and services through interfaces on the Internet. The equation “a more or less smart Web component is an agent” has quickly made the round. It seems that the Internet business has forgotten some of the criteria that qualified the AI agents.

It is not easy to summarize the myriad of agent definitions. But there seems to be a consensus that an **intelligent agent** must be *situated* in an environment, *autonomous*, *adaptive* and *sociable*. Most Web components probably satisfy **situatedness**: they are embedded in Web application containers and can be accessed from distant nodes on the Internet. On an another hand, most AI agents might miss this qualification, because the programs run in an isolated environment and can not directly be made available on the Web.

Autonomy might be satisfied by Web components at a first glance: they sit in Web application containers and wait for method invocations. However, if something unexpected reaches the component, its autonomy to react is limited and in most of the cases the component dies. As in evolution theory, survival is linked to adaptation: if a being is to survive in a hostile environment (and the Web *is* a hostile environment!), it must be able to react to events, learn from experience and **adapt** its future behavior respectively. Most Web components fall through these criteria. Most AI agents might fulfill them.

The last qualification (**sociable**) places the objectives even higher: to be sociable can be interpreted in two ways: if sociable means interaction in a network of peers, Web components brilliantly satisfy the criteria: the Web of objects is per se a nest of remote method invocations (RPC/RMI) and message exchanges (MOM). If sociable were to qualify behavior according to human, social theory, however, all Web components and most of the AI agents still fall through. Sociability then means reacting by deliberating on *believes*, *desires* and *intentions* (the so-called **BDI** architectures in agent theory). Recent developments also associate emotions (Goleman, 1995; Wollheim, 1999) and wisdom to sociability.

What can be concluded from all that for mere human software developers? Is there a way to implement intelligent services that are smarter than most existing Web components but not too smart to loose their situatedness like most of the AI agents? There is, and recent developments with **logic servers** like (Amzi, 2006) – our favorite one - will help us build intelligent services.

But first, let us focus on why we should combine our software process with another paradigm.

The shortcomings of sequential programming

There are business processes that simply escape from the efficient solution space of sequential programming. Under sequential programming we understand the line-by-line interpretation of a program. The mostly cited object-oriented languages like Java, C++, Pascal and Basic all fall under this paradigm. The following problem description shows the shortcomings of sequential programming from a very practical point of view:

You organize a family dinner. As a good restaurant keeper, you try to memorize the relationships that hold among the present persons. Somebody tells you:

Problem 1: The family dinner (natural language version)

Tom, Abraham, Hank, Leo, Ian and Alex are the men at the party table. Melanie, Nancy, Jane and Sarah are the women. Tom is the father of Alex, Nancy and Hank. Alex is the father of Ian. Hank is the father of Leo, Melanie and Jane. Sarah is the mother of Ian and Jane is the mother of Sarah.

You try to memorize all this and answer to a couple of questions that might arise during the dinner party:

- 1.1 Who is the father of Ian?
- 1.2 Is Alex the father of Melanie?
- 1.3 Who are the parents of Ian?
- 1.4 Who is the grandfather of Ian?
- 1.5 Who is here with the grand-parents?

Try to answer these questions with your brain “only”. Then try to develop this application with your preferred tools. If you have never seen an alternative to sequential programming, this problem might occupy you quite some time.

Solution 1: The family dinner (natural language version)

- 1.1 Alex is the father of Ian.
- 1.2 Alex is not the father of Melanie.
- 1.3 Alex and Sarah are the parents of Ian.
- 1.4 Tom is the grandfather of Ian.
- 1.5 Ian is here with his grandparents (Tom and Jane) and he is the only one.

Have you found a solution? OK, now how do you deal with the fact that Theo is arriving with his parents Thomas and Myriam and somebody tells you that Thomas is the uncle of Ian. Who is now here with the grandparents? Who is Thomas' father? Do you still want to program this with the same tools?

Introducing declarative programming

If you have chosen **SQL** for the problem above, you have made a first good move! SQL queries are not sequential in the sense of Java, C++, Pascal and Basic. SQL queries *declare* the data that should be assembled in a resulting table. Whatever the implementation behind an SQL engine is (it might be a sequential one!), the developer can declare the desired behavior without worrying about what's going on behind the scene. You probably know that SQL queries run through selected tables to verify whether the records fulfill some conditions.

The understanding of SQL is very important, because you learn how to separate an application's **state** (the data) from its **action** (the query execution). This separation is fundamental in software development. Following this paradigm, you have learned to design application state with conceptual and physical data models (PDM) and application action with uses cases and state diagrams. This conceptual split is also called a 2-tiered architecture.

From here: (Hernandez & Viescas, 2001) present in their book GoTo SQL the basis of the language in a pedagogically rich way. Their book is part of the Pearson education directory and presents SQL query techniques with illustrations and five ready-to-run databases. If you read French, (Bourguignon, 1991) presents an excellent introduction to SQL by means of a real business case.

But SQL might still be too limited to address some special business problems like the one exposed above. And SQL is not the whole declarative programming story. There are other languages that also allow for declarative programming and that function in another way. They might help us find a solution.

One of these languages is **Prolog**. SQL works on data bases. Prolog works on facts and rules: we call them **knowledge bases**. SQL queries are trying to extract data. Knowledge bases - in the case of Prolog - try to maintain truth. We basically ask questions to the knowledge base and it answers with yes and no and gives the variables that have been filled to arrive at the conclusion.

The Prolog program to the problem above reads as follows:

Problem 2: the family dinner (Prolog)

```
% ----- FACTS -----  
  
male(tom).  
male(abraham).  
male(hank).  
male(leo).  
male(ian).  
male(alex).  
  
female(melanie).  
female(nancy).  
female(jane).  
female(sarah).  
  
father(tom,alex).  
father(tom,nancy).  
father(tom,hank).  
father(alex,ian).  
father(hank,leo).  
father(hank,melanie).  
father(hank,jane).  
  
mother(sarah,ian).  
mother(jane,sarah).
```

```

% ----- RULES -----
parents(X,Y,Z) :- father(X,Z),
                 mother(Y,Z).

son(X,Y) :- male(X),
            father(Y,X).
son(X,Y) :- male(X),
            mother(Y,X).

daughter(X,Y) :- female(X),
                father(Y,X);
                female(X),
                mother(Y,X).

grandfather(X,Y) :- father(X,Z),
                    father(Z,Y).

grandfather(X,Y) :- father(X,Z),
                    mother(Z,Y).

grandmother(X,Y) :- mother(X,Z),
                    mother(Z,Y).

grandmother(X,Y) :- mother(X,Z),
                    father(Z,Y).

grandparents(X,Y,Z) :- grandmother(Y,Z),
                       grandfather(X,Y).

```

Wait a minute, you might say: ok for the facts and the rules, but there is no solution displayed. You are right! We have only stated the problem above. And we have taken the freedom to also state some **rules** that hold for the **facts** that we have stored.

Now we ask the *questions* to the knowledge base (in the Prolog listener).

```

Solution 2.1: Who is the father of Ian? (Prolog)
?- father(X,ian).

X = alex .
yes
?-

```

What have we done here? At the prompt – the knowledge base tells with a ?- that it waits for a question – we have written father(X,ian). Note that everything starting with a lower case is considered to be non variable (**constant**), and everything with an uppercase a **variable**. X is therefore open in our question. To satisfy our goal, Prolog will try to unify X with something constant. The first case for which it succeeds is X = alex. The dot (.) has been entered to say that we are satisfied. So, Prolog was able to certify our goal and answered yes.

What if we were hungry to know whether ian had more than one father?

```

Solution 2.1: Who are the fathers of Ian? (Prolog)
?- father(X,ian).

X = alex ;
no

```

```
?-
```

Here we have entered a semicolon ; after the first answer from Prolog. This means “give me another”. As there is no other father for Ian, Prolog can not satisfy our question and answers no. This is the mechanism of truth maintenance in Prolog.

Solution 2.2: Is Alex the father of Melanie?

```
?- father(alex,melanie) .
no
?-
```

There is no variable space in the question this time. The fact father(alex,melanie) is tested for existence in the knowledge base. There is no such fact. So the answer is **no** – and we have no alternative, so the writing of a semi-colon ; is even not possible, Prolog directly concludes on no.

Is there a way to state after the startup of the knowledge base that alex is also the father of Melanie? Yes, there is:

Adding and retracting facts

```
?- assert(father(alex,melanie)) .

yes
?- retract(father(alex,melanie)) .

yes
?-
```

assert/1 and **retract/1** are Prolog system predicates that allow one to add and remove facts to and from the knowledge base. The /1 means that they are of **arity** 1 – they take one argument. Likewise it is possible to remove all father/2 predicates from the knowledge base with a killer rule (don't do it!):

Bad solution: Removing all father predicates with an additional rule

```
killer :- retract(father(_, _),
               killer).
killer.
```

Although this is not a good idea if you want to continue with the family knowledge base, there are some interesting elements here. killer/0 is a rule that satisfies truth if retract finds a father/2 predicate with any content. The “any” variable is represented with **_** for the first and the second argument. **And** there is a second condition in the rule body (on the right side of the :- which means **If**): you see the and in the form of a comma , - if there were a semi-colon ; between retract(father(,_)) and killer, this would have been an **Or** combination.

Finally, you have certainly recognized that killer/1 is **recursive** by its second condition in the rule body, which is again ... killer. This predicate goes on killing any father – what a perspective ;-(- until there is no one left! This predicate ends with a yes in any case, because the second killer/1 on the third line has no condition.

Solution 2.3: Who are the parents of Ian?

```
?- parents(X,Y,ian) .

X = alex
Y = sarah ;
no
?-
```

This solution illustrates Prolog's **execution model**, which can be stated as: “Prolog's execution mechanism is obtained from the abstract interpreter by choosing the leftmost goal of an arbitrary one and replacing the

non-deterministic choice of a clause by sequential search for a unifiable clause and backtracking.

In other words, Prolog adopts a stack scheduling policy. It maintains the resolvent as a stack: pops the top goal for reduction, and pushes the derived goals onto the resolvent stack.

In addition to the stack policy, Prolog simulates the nondeterministic choice of reducing clause by sequential **search and backtracking**. When attempting to reduce a goal, the first clause whose head **unifies** with the goal is chosen. If no unifiable clause is found for the popped goal, the computation is unwound to the last choice made, and the next unifiable clause is chosen.”

From here: Three Prolog books might explain everything you might ever need to know about Prolog: (Clocksin, 1997) explains on less than 150 A5 pages the essence of the language – you might want to start with this one. The citation above is from (Sterling & Shapiro, 1997) which presents the Art of Prolog. And the third book is (O’Keefe, 1994).

Now back to our family dinner problem:

Solution 2.4: Who is the grandfather of Ian?

```
?- grandfather(X,ian) .
```

```
X = tom ;
```

```
no
```

```
?-
```

Quick exercise: Try to find the matching rule headers in the knowledge base and draw a picture of how the variables unify with the constants.

Solution 2.5: Who is here with the grandparents?

```
?- grandparents(X,Y,Z) .
```

```
X = tom
```

```
Y = jane
```

```
Z = ian ;
```

```
no
```

```
?-
```

From here: Motivated to install a Prolog IDE? Take a tour at (Amzi, 2006). You’ll find a Prolog plug-in for Eclipse and an excellent starting place to the Prolog world on this Website. Installation is feasible for everyone but *please read the installation instructions!*

The shortcomings of traditional Prolog engines

If you have installed Amzi!Prolog, you have already overcome the major shortcomings of traditional Prolog engines:

- The **price** you had to pay for effective Prolog engines.
- No **embedding** of Prolog into real world, daily application development processes. Which means no API to Prolog for other languages.
- No **extension** possibility for Prolog with other languages. Which means no customization.

These limitations conferred to this beautiful language for a long time a touch of dusty Artificial Intelligence theories with **no imminent use for the working programmer**. Times have changed, as you will see now. Starting with the price, Amzi!Prolog is free for evaluation and academic use. The prices for commercial licenses are competitive.

Embedding Prolog in host languages

Regarding the embedding of Prolog in host languages, **Amzi!Prolog has been the pioneer**. The Logic Server – the Prolog engine – easily integrates into a whole palette of existing languages, like Java, C++ and many more. Recently, we have even seen a Prolog for Excel extension on the Website!

For Java – our course language – there exists a very comfortable **API to Amzi!Prolog**.

Suppose you have copied the family.pro program to any directory (e.g. c:\TEMP\family_dinner). The first thing to do is to compile the Prolog program:

```
Step 1 - Compile the family.pro program

C:\TEMP\family_dinner>set AMZI_DIR=c:\environment\amzi_new

C:\TEMP\family_dinner>set PATH=%PATH%;%AMZI_DIR%\bin

C:\TEMP\family_dinner>acmp

Amzi! Prolog Compiler 7.4.0 Windows  Nov  7 2005 10:12:25
Copyright (c)1992-2003 Amzi! inc. All Rights Reserved.

Source Code [.PRO]: family
Object Code [family.PLM]:
Listing File [NULL.PAL]:
1 |-- male/1
2 |-- male/1
3 |-- male/1
4 |-- male/1
5 |-- male/1
6 |-- male/1
1 |-- female/1
2 |-- female/1
3 |-- female/1
4 |-- female/1
1 |-- father/2
2 |-- father/2
3 |-- father/2
4 |-- father/2
5 |-- father/2
6 |-- father/2
7 |-- father/2
1 |-- mother/2
2 |-- mother/2
1 |-- parents/3
1 |-- son/2
2 |-- son/2
1 |-- daughter/2
1 |-- grandfather/2
2 |-- grandfather/2
1 |-- grandmother/2
2 |-- grandmother/2
1 |-- grandparents/3

[CodeSize 2427 Bytes, Compile time 0.0699997 seconds.]

*****
```

```

This Free Edition is licensed for academic,
personal or evaluation use only. (See License
Agreement in documentation for details).
To purchase Student, Developer or Professional
License, visit www.amzi.com or e-mail sales@amzi.com.
*****

```

```
main/0 succeeded
```

```
C:\TEMP\family_dinner>
```

Then you have to link the compiler result into a loadable code file:

Step 2 - Link the family.plm file to family.xpl

```
C:\TEMP\family_dinner>set AMZI_DIR=c:\environment\amzi_new
```

```
C:\TEMP\family_dinner>set PATH=%PATH%;%AMZI_DIR%\bin
```

```
C:\TEMP\family_dinner>alnk
```

```

Amzi! Prolog Linker 7.4.0
Linked Module [.xpl]: family
Compiled Code Module [.plm]: family
Compiled Code Module [.plm]:
Linking: family.xpl
Opening: 'alib.plm'
Opening: 'family.plm'
Link Done

```

```
C:\TEMP\family_dinner>
```

The following Java program works with this knowledge base:

Step 3 - The Java program that hosts the Prolog engine

```

import amzi.ls.*;

class Family
{
    LogicServer ls;

    public static void main(String args[])
    {
        System.out.println("Family dinner Prolog / Java demo");
        try
        {
            Family family = new Family() ;
            family.dinner() ;
        }
        catch (LSEException e)
        {
            System.out.println(e.GetMsg());
        }
    }

    public Family() throws LSEException
    {

```

```
ls = new LogicServer();
ls.Init("");
ls.Load("family");
}

public void dinner() throws LSException
{
    long term;
    String pred = "parents(X,Y,ian)" ;
    term = ls.ExecStr(pred) ;
    String father = ls.GetStrArg(term,1) ;
    String mother = ls.GetStrArg(term,2) ;
    System.out.println("Father: " + father) ;
    System.out.println("Mother: " + mother) ;
    ls.Close();
}
}
```

The most interesting lines are the loading of the family.xpl file with the compiled and linked Prolog program, the execution of a string on the logic server and the extraction of the respective arguments that Prolog has unified. Note here that **term** would be 0 if the predicate can not be verified (no), and any integer when the question succeeds (yes). We have not checked here `term!=0` because we know that there is a answer.

From here: The full Logic Server API is documented at (Amzi, 2006). If you have installed it on your PC, the documentation can be found at `<amzi_dir>docs/amzidoc.htm`

To make compilation and recompilation easy, we write a compile.bat batch file:

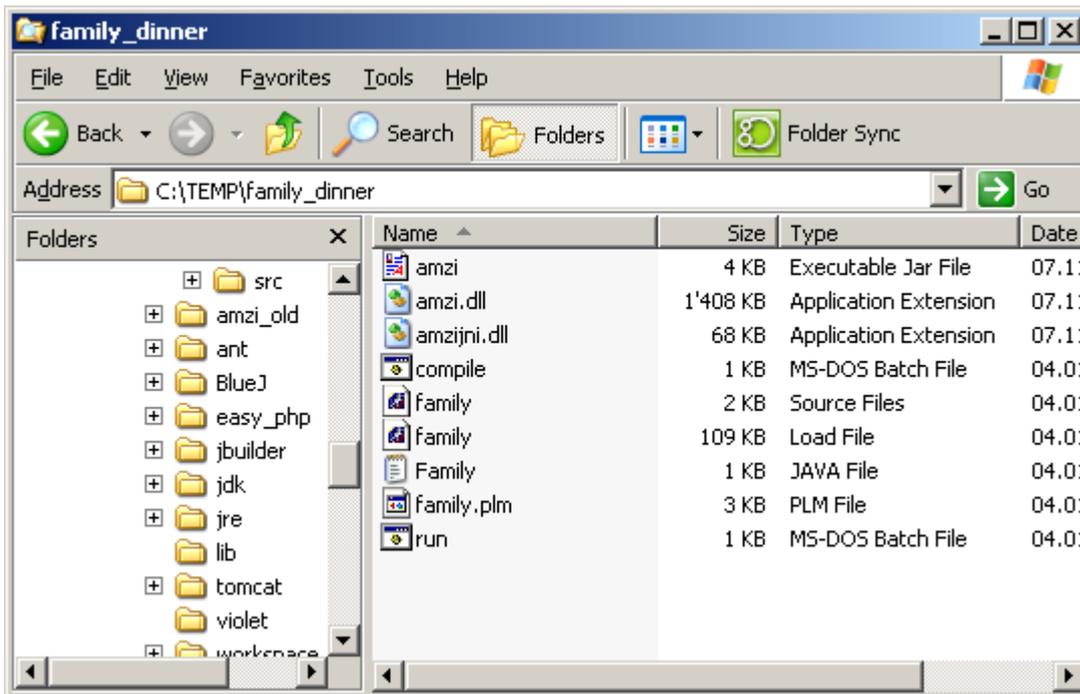
Step 4 - compile.bat

```
SET JAVA_HOME=c:\environment\jdk
SET AMZI_DIR=.

CALL %JAVA_HOME%\bin\javac -classpath ./;./amzi.jar Sudoku.java

pause
```

Here we first point `JAVA_HOME` to a Java Development Kit (1.5.0_05). `AMZI_DIR` is set to a directory containing the Amzi!Prolog files. In this case it is one and the same directory for everything. It looks like this:



If everything goes well, a double-click on compile.bat results in:

```

C:\WINDOWS\System32\cmd.exe

C:\TEMP\family_dinner>SET JAVA_HOME=c:\environment\jdk
C:\TEMP\family_dinner>SET AMZI_DIR=.
C:\TEMP\family_dinner>CALL c:\environment\jdk\bin\javac -classpath ./;./amzi.jar
Family.java
C:\TEMP\family_dinner>pause
Press any key to continue . . . _

```

Then, we write a run.bat batch file:

```

Step 5 - run.bat

SET JAVA_HOME=c:\environment\jre
SET AMZI_DIR=.
SET PATH=%PATH%;./amzijni.dll

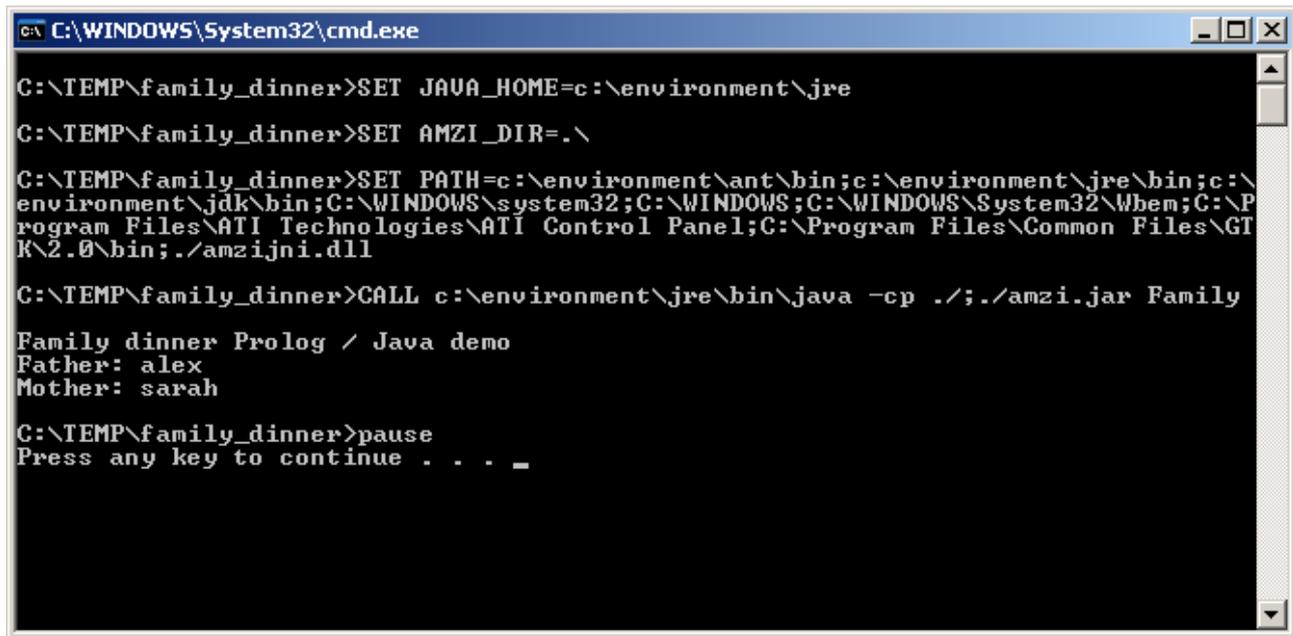
CALL %JAVA_HOME%\bin\java -cp ./;./amzi.jar Family

pause

```

Here JAVA_HOME points to a Java Runtime Environment. AMZI_DIR still points to the present directory. The PATH must contain the file amzijni.dll and the classpath during the program launching the present directory ./ and amzi.jar.

A double-click on run.bat results in:

A screenshot of a Windows command prompt window titled "C:\WINDOWS\System32\cmd.exe". The window shows a series of commands and their output. The commands are: 1. SET JAVA_HOME=c:\environment\jre 2. SET AMZI_DIR=.\ 3. SET PATH=c:\environment\ant\bin;c:\environment\jre\bin;c:\environment\jdk\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\Program Files\ATI Technologies\ATI Control Panel;C:\Program Files\Common Files\GTK\2.0\bin;./amzijni.dll 4. CALL c:\environment\jre\bin\java -cp ./;./amzi.jar Family The output shows: Family dinner Prolog / Java demo Father: alex Mother: sarah The prompt then shows "C:\TEMP\family_dinner>pause" and "Press any key to continue . . . _".

```
C:\WINDOWS\System32\cmd.exe
C:\TEMP\family_dinner>SET JAVA_HOME=c:\environment\jre
C:\TEMP\family_dinner>SET AMZI_DIR=.\
C:\TEMP\family_dinner>SET PATH=c:\environment\ant\bin;c:\environment\jre\bin;c:\environment\jdk\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\Program Files\ATI Technologies\ATI Control Panel;C:\Program Files\Common Files\GTK\2.0\bin;./amzijni.dll
C:\TEMP\family_dinner>CALL c:\environment\jre\bin\java -cp ./;./amzi.jar Family
Family dinner Prolog / Java demo
Father: alex
Mother: sarah
C:\TEMP\family_dinner>pause
Press any key to continue . . . _
```

If you have understood how this example works, you are ready for the development of hybrid Java / Prolog applications.

Extending Prolog with host languages

Let us focus on the extension of Prolog with host languages – the third shortcoming of traditional Prolog IDEs. Suppose you have forgotten to write a Prolog rule

```
brother(X,Y,Z) :- ...
```

which succeeds if X is a man and either the father Z of both X and Y is the same. Do we need to restart things from the beginning? Not with Amzi!Prolog! The Amzi API allows an extension of a Prolog program with predicates that are written in host languages. In such a case, Prolog calls back Java and invokes a method as if it were a predicate in the knowledge base. This very powerful mechanism is called **extended predicates**.

Step 6 - Family dinner, the callback version

```
import amzi.ls.*;

class Family_Callback
{
    LogicServer ls;

    public static void main(String args[])
    {
        System.out.println("Family dinner Prolog / Java demo with Callbacks");
        try
        {
            Family_Callback family = new Family_Callback() ;
            family.dinner() ;
        }
        catch (LSEException e)
        {
            System.out.println(e.GetMsg());
        }
    }

    public Family_Callback() throws LSEException
    {
        ls = new LogicServer();
        ls.Init("");
        ls.AddPred("brother", 2, "Family_Callback", "brother", this);
        ls.Load("family");
    }

    public boolean brother()
    {
        try
        {
            String X = ls.GetStrParm(1) ;
            String Y = ls.GetStrParm(2) ;
            String pred = "male(X)" ;
            long term0 = ls.ExecStr(pred) ;
            if (term0==0)
            {
                return false ;
            }
            else
            {
                pred = "father(Z,"+X+)" " ;
            }
        }
    }
}
```

```
        long term1 = ls.ExecStr(pred) ;
        if (term1==0)
        {
            return false ;
        }
        else
        {
            String Z1 = ls.GetStrParm(1) ;
            pred = pred = "father(Z,"+Y+)" " ;
            long term2 = ls.ExecStr(pred) ;
            if (term2==0)
            {
                return false ;
            }
            else
            {
                String Z2 = ls.GetStrParm(1) ;
                if (Z2.equalsIgnoreCase(Z1))
                {
                    return true ;
                }
                else
                {
                    return false ;
                }
            }
        }
    }
}
catch (LSEException e)
{
    System.out.println(e.GetMsg());
    return false;
}
}

public void dinner() throws LSEException
{
    long term;
    String pred = "parents(X,Y,ian)" ;
    term = ls.ExecStr(pred) ;
    String father = ls.GetStrArg(term,1) ;
    String mother = ls.GetStrArg(term,2) ;
    System.out.println("Father of ian: " + father) ;
    System.out.println("Mother of ian: " + mother) ;
    pred = "brother(leo,melanie)" ;
    term = ls.ExecStr(pred) ;
    if (term!=0)
    {
        System.out.println("leo is the brother of melanie");
    }
    else
    {
        System.out.println("leo is not the brother of melanie");
    }
    ls.Close();
}
}
```

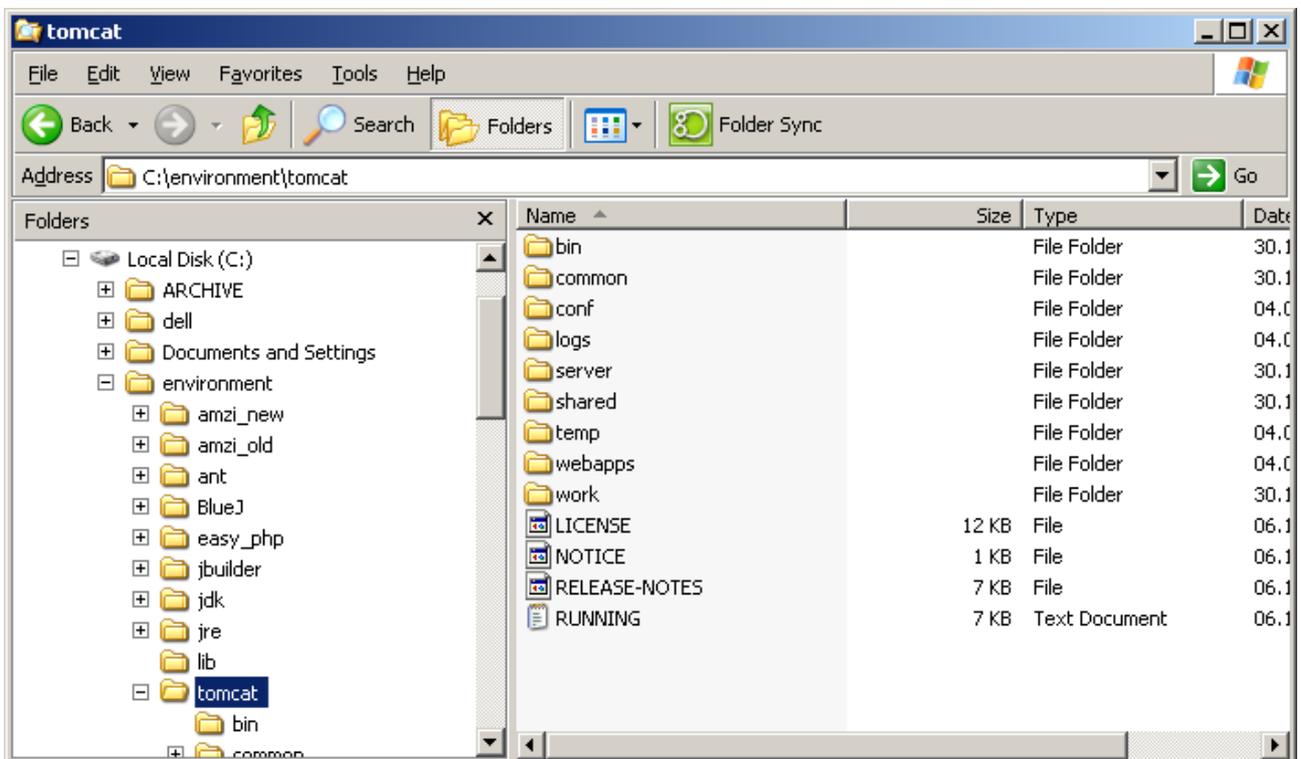
Worth mentioning is the line where the Java method brother (without any arguments and of return type boolean – keep this in memory for all your extended predicates!) is uploaded to the Logic Server. This must be done between the Logic Server initialization and the loading of compiled and linked Prolog programs.

The example is of course not complete: X and Y can also have the same mother Z. This is left as an exercise (see at the end of the paper). Compiling and running this code results in:

```
C:\TEMP\family_dinner>SET JAVA_HOME=c:\environment\jre
C:\TEMP\family_dinner>SET AMZI_DIR=.
C:\TEMP\family_dinner>SET
PATH=c:\environment\ant\bin;c:\environment\jre\bin;c:\
environment\jdk\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\
Program Files\ATI Technologies\ATI Control Panel;C:\Program Files\Common
Files\GT
K\2.0\bin;./amzijni.dll
C:\TEMP\family_dinner>CALL c:\environment\jre\bin\java -cp ./;./amzi.jar
Family_
Callback
Family dinner Prolog / Java demo with Callbacks
Father of ian: alex
Mother of ian: sarah
leo is the brother of melanie
C:\TEMP\family_dinner>pause
Press any key to continue . . .
```

Installing the Web application container (Tomcat)

To install the Web application container, download the latest binary from (Tomcat, 2006). Take the binary .zip version. Unzip the file to any directory on your future Web application server (e.g. c:\environment\tomcat). Here is a typical directory structure:



The next step consists of giving manager rights to a Tomcat user. This is done in the file `./conf/tomcat-users.xml`: Here we add a user “student” with password “sierre” and role “manager”:

Step 7 - Add a Tomcat user

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <role rolename="manager"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="role1" password="tomcat" roles="role1"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
  <user username="student" password="sierre" roles="manager"/>
</tomcat-users>
```

To complete the Tomcat Manager installation, we need to enable it in the file `./conf/server.xml`:

Step 8 - Enable the Tomcat Manager

```
...

<Context path="/manager" debug="0" privileged="true"
  docBase="c:/environment/tomcat/server/webapps/manager">
</Context>
```

```
</Host>
```

Then we write a START_TOMCAT.bat batch file:

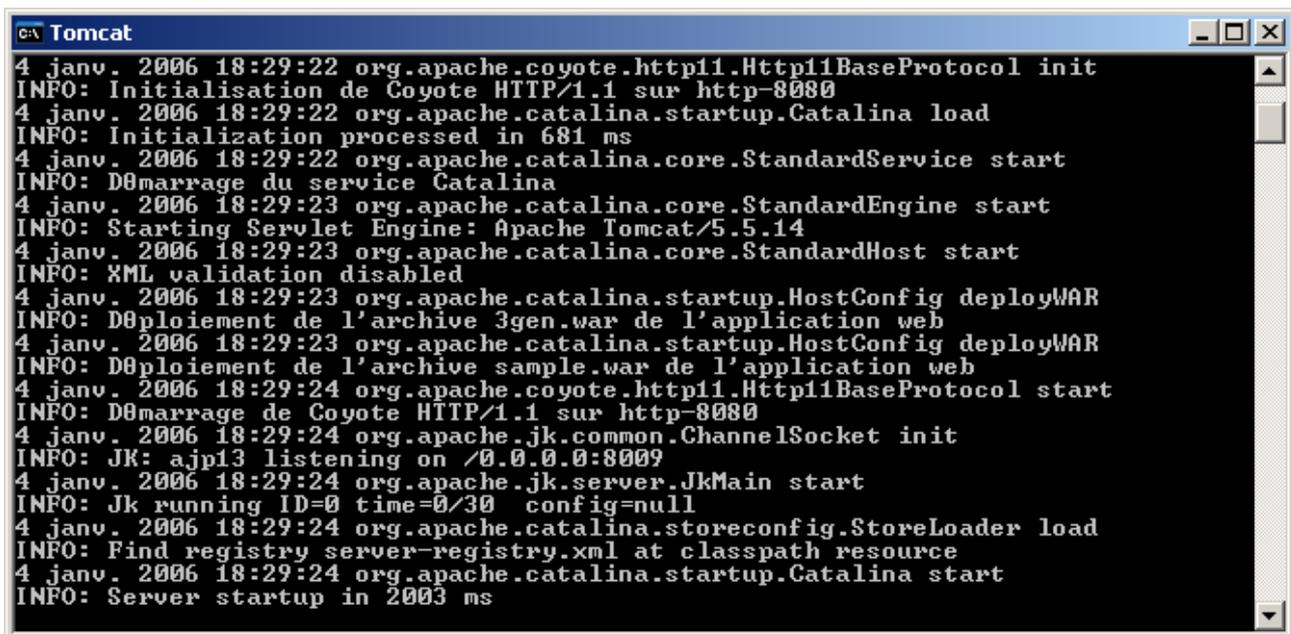
Step 9 - START_TOMCAT.bat

```
SET JAVA_HOME=c:\environment\jdk
SET ANT_HOME=c:\environment\ant
SET CATALINA_HOME=c:\environment\tomcat
SET AMZI_DIR=c:\environment\amzi_new
SET PATH=%PATH%;%AMZI_DIR%\bin\amzi.dll;%AMZI_DIR%\bin\amzijni.dll
SET CLASSPATH=%CLASSPATH%;%AMZI_DIR%\lsapis\java20\amzi.jar

CALL %CATALINA_HOME%\bin\startup.bat
```

Note that we have followed the installation instructions of Amzi: There is a AMZI_DIR pointing to a local Prolog installation, amzi.dll and amzijni.dll are on the PATH and amzi.jar on the CLASSPATH.

Double-clicking this batch should result in something like this:

A screenshot of a Windows console window titled "Tomcat". The window displays the output of the START_TOMCAT.bat batch file. The logs show the initialization of the Coyote HTTP/1.1 protocol, the loading of the Catalina startup class, the processing of initialization in 681 ms, the start of the StandardService, the start of the StandardEngine, the start of the StandardHost, the deployment of the 3gen.war and sample.war archives, the start of the Coyote HTTP/1.1 protocol, the initialization of the ChannelSocket, the listening on port 8009, the start of the JkMain, the loading of the StoreLoader, the finding of the registry server-registry.xml at classpath resource, and the final startup of the Catalina server in 2003 ms.

```
C:\ Tomcat
4 janv. 2006 18:29:22 org.apache.coyote.http11.Http11BaseProtocol init
INFO: Initialisation de Coyote HTTP/1.1 sur http-8080
4 janv. 2006 18:29:22 org.apache.catalina.startup.Catalina load
INFO: Initialization processed in 681 ms
4 janv. 2006 18:29:22 org.apache.catalina.core.StandardService start
INFO: Démarrage du service Catalina
4 janv. 2006 18:29:23 org.apache.catalina.core.StandardEngine start
INFO: Starting Servlet Engine: Apache Tomcat/5.5.14
4 janv. 2006 18:29:23 org.apache.catalina.core.StandardHost start
INFO: XML validation disabled
4 janv. 2006 18:29:23 org.apache.catalina.startup.HostConfig deployWAR
INFO: Déploiement de l'archive 3gen.war de l'application web
4 janv. 2006 18:29:23 org.apache.catalina.startup.HostConfig deployWAR
INFO: Déploiement de l'archive sample.war de l'application web
4 janv. 2006 18:29:24 org.apache.coyote.http11.Http11BaseProtocol start
INFO: Démarrage de Coyote HTTP/1.1 sur http-8080
4 janv. 2006 18:29:24 org.apache.jk.common.ChannelSocket init
INFO: JK: ajp13 listening on /0.0.0.0:8009
4 janv. 2006 18:29:24 org.apache.jk.server.JkMain start
INFO: Jk running ID=0 time=0/30 config=null
4 janv. 2006 18:29:24 org.apache.catalina.storeconfig.StoreLoader load
INFO: Find registry server-registry.xml at classpath resource
4 janv. 2006 18:29:24 org.apache.catalina.startup.Catalina start
INFO: Server startup in 2003 ms
```

To stop Tomcat, we can not simply type control-c in the console! Here is the batch STOP_TOMCAT.bat:

Step 10 - STOP_TOMCAT.bat

```
SET JAVA_HOME=c:\environment\jdk
SET ANT_HOME=c:\environment\ant
SET CATALINA_HOME=c:\environment\tomcat

CALL %CATALINA_HOME%\bin\shutdown.bat
```

A double-click of this batch closes the Tomcat console window (after some time) and itself.

You can check the correct running of Tomcat by opening <http://localhost:8080> in your browser:

Apache Tomcat/5.5.14 - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://localhost:8080/

Amazon.com Free M... Amazon.com Free M... Amazon.com Free M... Amazon.com Free M...

 Apache Tomcat/5.5.14

 The Apache Jakarta Project
http://jakarta.apache.org/

Administration

- [Status](#)
- [Tomcat Administration](#)
- [Tomcat Manager](#)

Documentation

- [Release Notes](#)
- [Change Log](#)
- [Tomcat Documentation](#)

Tomcat Online

- [Home Page](#)
- [FAQ](#)
- [Bug Database](#)
- [Open Bugs](#)
- [Users Mailing List](#)
- [Developers Mailing List](#)
- [IRC](#)

If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

As you may have guessed by now, this is the default Tomcat home page. It can be found on the local filesystem at:

```
$CATALINA_HOME/webapps/ROOT/index.jsp
```

where "\$CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't think you should be, then either you're either a user who has arrived at new installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Providing the latter is the case, please refer to the [Tomcat Documentation](#) for more detailed setup and administration information than is found in the INSTALL file.

NOTE: This page is precompiled. If you change it, this page will not change since it was compiled into a servlet at build time. (See `$CATALINA_HOME/webapps/ROOT/WEB-INF/web.xml` as to how it was mapped.)

NOTE: For security reasons, using the administration webapp is restricted to users with role "admin". The manager webapp is restricted to users with role "manager". Users are defined in `$CATALINA_HOME/conf/tomcat-users.xml`.

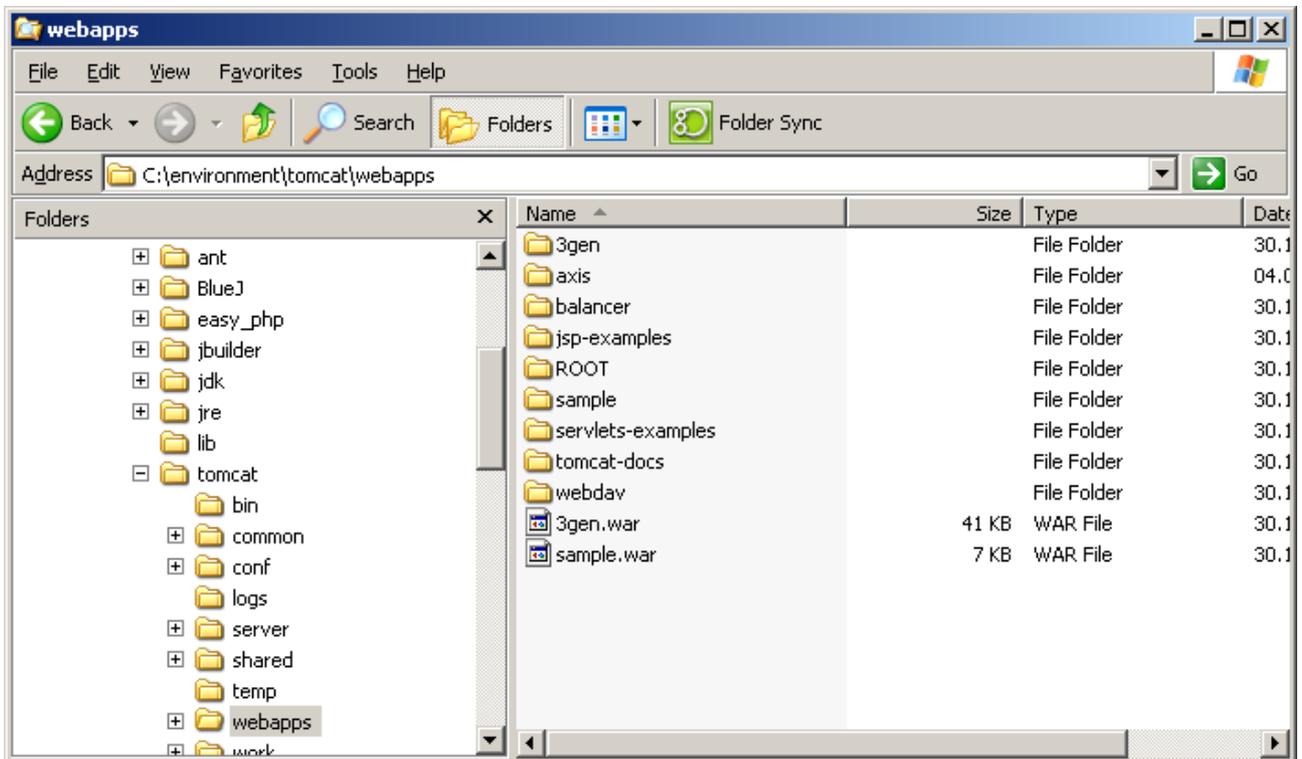
Included with this release are a host of sample Servlets and JSPs (with associated source code), extensive documentation (including the Servlet 2.4 and JSP 2.0 API JavaDoc), and an introductory guide to developing web applications.

Done

By clicking on "Manager", you can log in with the above created user "student" and the password "sierre". So far for the Web application container.

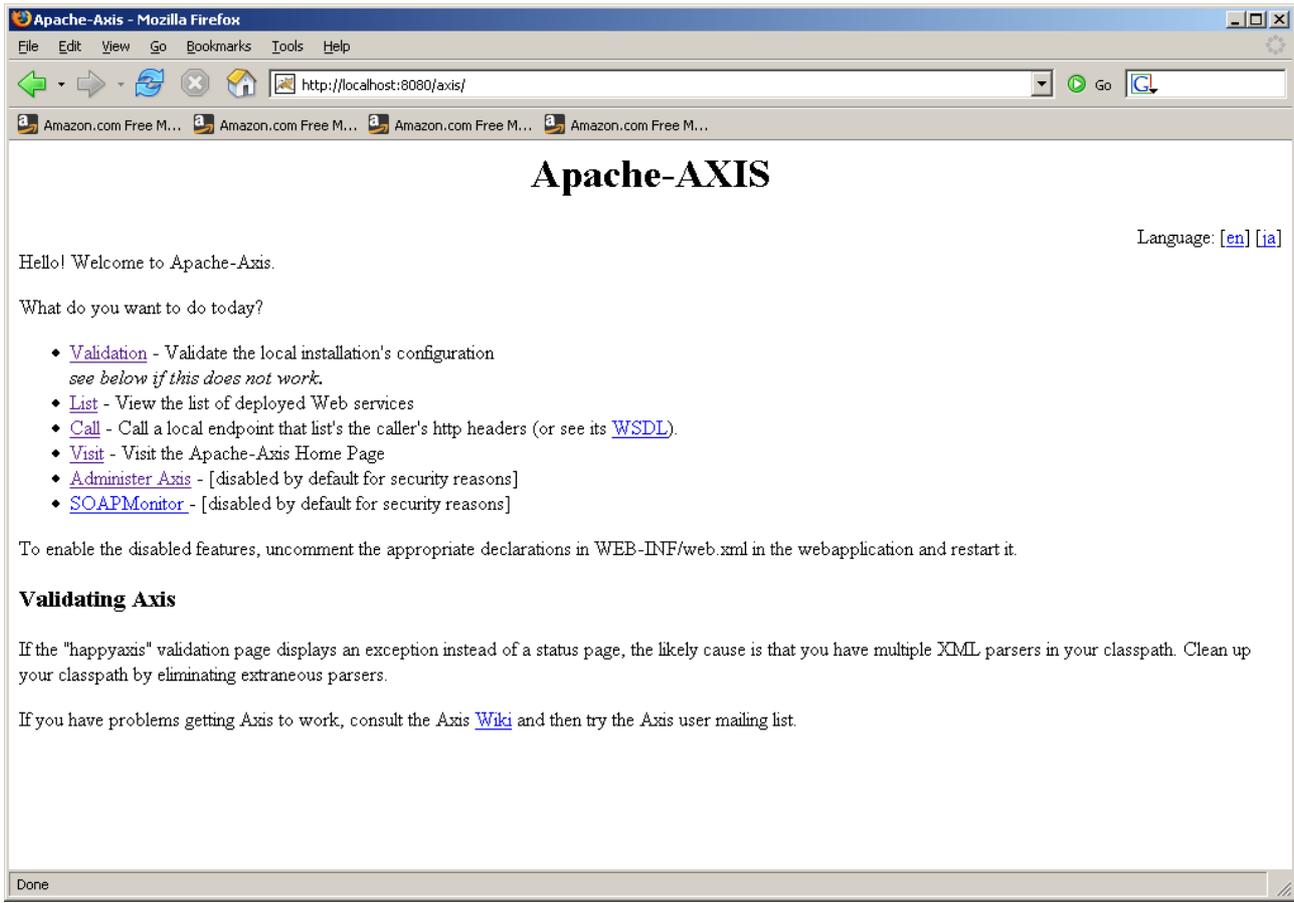
Installing the Web service extension (Axis)

To install *the* Webservice facility on Tomcat, download the latest binary .zip from (Axis, 2006). Unzip the file somewhere on your future Webservice application server. Open the webapps directory. Copy the axis subdirectory to your <tomcat>webapps directory. Here is a typical directory structure:

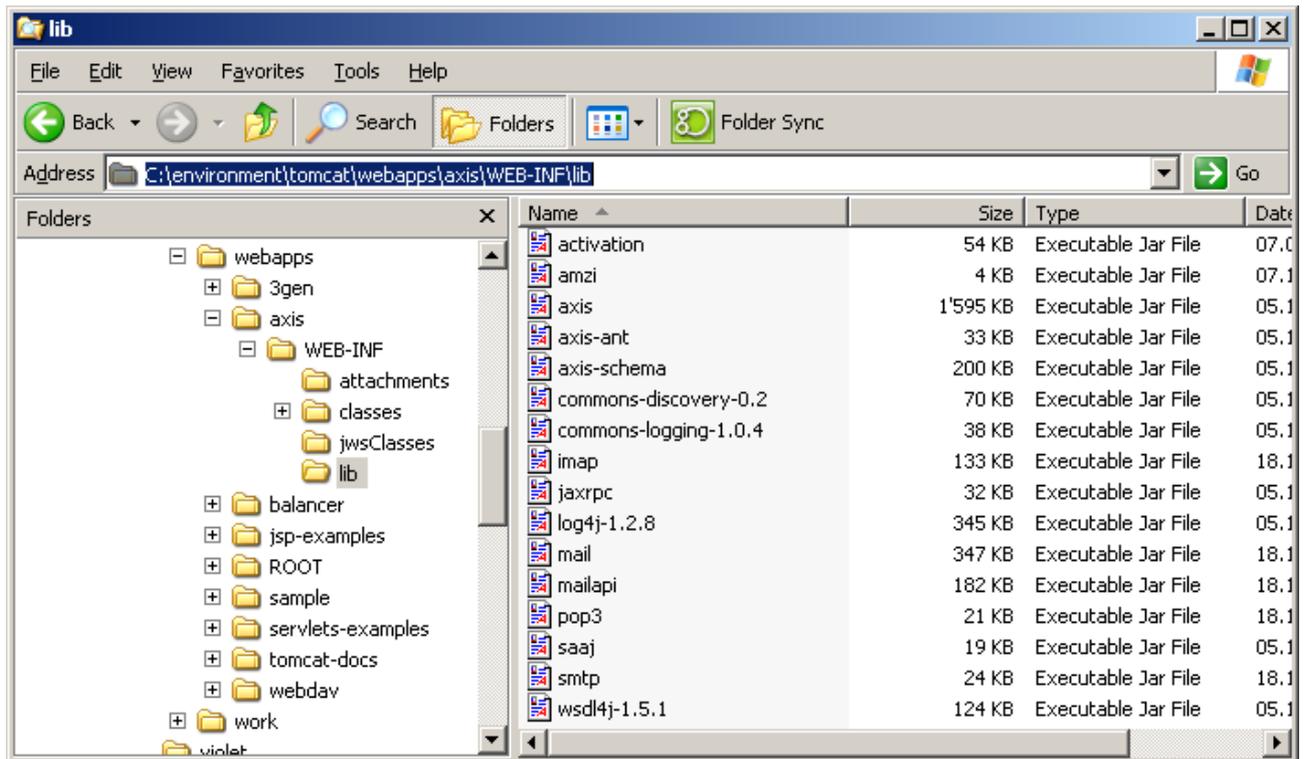


Don't care about 3Gen – this is our middleware architecture generator (3Gen, 2006). You certainly do not have this subdirectory, unless you have deployed an application that has been generated during the course at the University of Applied Sciences Valais. The rest should pretty look the same on your machine. Note that the directory axis must reside here.

You can now open your browser at <http://localhost:8080/axis> and you should see the Axis welcome page:



The first thing to do is validate your Axis installation. Click on “Validation”. You will most probably see a page that displays one major error preventing Axis from running: activation.jar is missing. This is not a bug. You have to choose an activation API implementation and install it into Tomcat / Axis – take for example Sun's (JAF, 2006). Download the .zip, extract everything and localize the file activation.jar This file must be copied to the subdirectory WEB-INF\lib of your axis installation. Here is a typical directory structure:



Restart Tomcat and when done, refresh the Axis validation page. Everything should be set up now.

Deploying the Logic Server as a Web service

We have encountered problems when trying to directly expose the LogicServer as a Web service. The only way to get this working seems to be to write a wrapper class around the LogicServer. This class has the same method names as the LogicServer (we have not taken all of them) and simply maps a service request to the corresponding method:

STEP 11 - LogicService.java

```
import amzi.ls.* ;

public class LogicService
{
    public static LogicServer ls ;

    public LogicService()
    {
    }

    public void Load(String pFile) throws LSEException
    {
        ls = new LogicServer() ;
        ls.Init("") ;
        ls.Load(pFile) ;
    }

    public long ExecStr(String pString) throws LSEException
    {
        long result = ls.ExecStr(pString) ;
        return result ;
    }

    public String GetStrArg(long pTerm, int pArgument) throws LSEException
    {
        String result = ls.GetStrArg(pTerm,pArgument) ;
        return result ;
    }
}
```

A few words about this solution: the LogicServer attribute must be declared as *static*. And you can not instantiate the LogicServer, nor call the Init method in the class constructor. The Load method creates a new instance of the LogicServer and initializes it. This might not be the most elegant way, but it works.

Now we make use of (Ant, 2006). Get the latest binary distribution and unzip it into the directory that has been specified in the START_TOMCAT.bat batch (for example c:\environment\ant). Now we write a build.bat batch that controls the ANT execution – you can place the following files anywhere on your machine:

STEP 12 - build.bat

```
SET JAVA_HOME=c:\environment\jdk
SET ANT_HOME=c:\environment\ant

"%ANT_HOME%\bin\ant"

pause
```

The corresponding build.xml file that contains the deployment tasks looks like this:

Step 13 - build.xml

```

<!-- Webservice-specific ANT build script -->

<project name="LogicService" default="all" basedir=".">

  <!-- Initializing -->
  <target name="init">
    <property name="dir" value="."/>
    <property name="axis.dir" value="c:\environment\tomcat\webapps\axis">
    <property name="axis.host" value="localhost">
    <property name="axis.port" value="8080">

    <path id="axis.classpath">
      <fileset dir="${axis.dir}/WEB-INF/lib">
        <include name="**/*.jar" />
      </fileset>
    </path>

    <taskdef resource="axis-tasks.properties"
      classpathref="axis.classpath" />
  </target>

  <!-- Expose the LogicServer as a Web service -->
  <target name="all" depends="init,undeploy,deploy"/>

  <target name="undeploy" depends="init">
    <axis-admin
      port="${axis.port}"
      hostname="${axis.host}"
      failonerror="false"
      servletpath="axis/services/AdminService"
      debug="true"
      xmlfile="${dir}/undeploy.wsdd"
    />
  </target>

  <target name="deploy" depends="init">
    <copy todir="${axis.dir}/WEB-INF/classes">
      <fileset dir="${dir}">
        <include name="**/*.class"/>
      </fileset>
    </copy>
    <axis-admin
      port="${axis.port}"
      hostname="${axis.host}"
      failonerror="true"
      servletpath="axis/services/AdminService"
      debug="true"
      xmlfile="${dir}/deploy.wsdd"
    />
  </target>

</project>

```

You can leave this file nearly as is. But, please be sure that the path to Axis (bold) is correctly set for your machine!

This file references two subsequent files: `undeploy.wsdd` and `deploy.wsdd`. These are the Webservice deployment description files. Here is `deploy.wsdd`:

Step 14 - `deploy.wsdd`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE deployment PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
<service provider="java:RPC" name="LogicService">
<parameter value="LogicService" name="className"/>
<parameter value="ExecStr,GetStrArg,Load" name="allowedMethods"/>
</service>
</deployment>
```

Three lines are of interest here: the Web service name (LogicServer), the class name (`amzi.ls.LogicServer`) and the allowed methods. We use only three of them for the family service: `Load`, `ExecStr` and `GetStrArg`. You can expose here all the public `LogicServer` methods, of course. This `wsdd` is used by the Ant script to deploy the `LogicServer` as a Web Service on Tomcat / Axis. The other one, `undeploy.wsdd`, does – as its name says – the opposite:

Step 15 - `undeploy.wsdd`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE undeployment PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<undeployment xmlns="http://xml.apache.org/axis/wsdd/">
<service name="LogicServer"/>
</undeployment>
```

Double-click on your `build.bat` batch and Ant will do the job for you.

The Tomcat server must be restarted each time this `LogicService` is deployed! The files `amzi.dll` and `amzijni.dll` must be placed into the same directory where `START_TOMCAT.bat` and `STOP_TOMCAT.bat` are located.

Congratulations! If you have followed the paper up to here, you have a working Logic Service on your application server. Let's display the functional specification by pointing the browser to the Webservice Description Language (WSDL) file: <http://localhost:8080/axis/services/LogicService?wsdl>

```
<wsdl:definitions
targetNamespace="http://localhost:8080/axis/services/LogicService">
<!--
WSDL created by Apache Axis version: 1.3
Built on Oct 05, 2005 (05:23:37 EDT)
-->
  <wsdl:types>
    <schema targetNamespace="http://ls.amzi">
<import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
      <complexType name="LSEException">
        <sequence>
<element name="message" nillable="true" type="xsd:string" />
        </sequence>
      </complexType>
    </schema>
  </wsdl:types>
  <wsdl:message name="ExecStrRequest">
```

```

<wsdl:part name="pString" type="xsd:string"/>
</wsdl:message>
  <wsdl:message name="GetStrArgResponse">
<wsdl:part name="GetStrArgReturn" type="xsd:string"/>
</wsdl:message>
  <wsdl:message name="ExecStrResponse">
<wsdl:part name="ExecStrReturn" type="xsd:long"/>
</wsdl:message>
  <wsdl:message name="GetStrArgRequest">
<wsdl:part name="pTerm" type="xsd:long"/>
<wsdl:part name="pArgument" type="xsd:int"/>
</wsdl:message>
<wsdl:message name="LoadResponse">
  </wsdl:message>
  <wsdl:message name="LSEException">
<wsdl:part name="fault" type="tnsl:LSEException"/>
</wsdl:message>
  <wsdl:message name="LoadRequest">
<wsdl:part name="pFile" type="xsd:string"/>
</wsdl:message>
  <wsdl:portType name="LogicService">
    <wsdl:operation name="Load" parameterOrder="pFile">
<wsdl:input message="impl:LoadRequest" name="LoadRequest"/>
<wsdl:output message="impl:LoadResponse" name="LoadResponse"/>
<wsdl:fault message="impl:LSEException" name="LSEException"/>
</wsdl:operation>
    <wsdl:operation name="ExecStr" parameterOrder="pString">
<wsdl:input message="impl:ExecStrRequest" name="ExecStrRequest"/>
<wsdl:output message="impl:ExecStrResponse" name="ExecStrResponse"/>
<wsdl:fault message="impl:LSEException" name="LSEException"/>
</wsdl:operation>
    <wsdl:operation name="GetStrArg" parameterOrder="pTerm pArgument">
<wsdl:input message="impl:GetStrArgRequest" name="GetStrArgRequest"/>
<wsdl:output message="impl:GetStrArgResponse" name="GetStrArgResponse"/>
<wsdl:fault message="impl:LSEException" name="LSEException"/>
</wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="LogicServiceSoapBinding" type="impl:LogicService">
<wsdlsoap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="Load">
<wsdlsoap:operation soapAction=""/>
  <wsdl:input name="LoadRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded"/>
</wsdl:input>
  <wsdl:output name="LoadResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/axis/services/LogicService" use="encoded"/>
</wsdl:output>
  <wsdl:fault name="LSEException">
<wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
name="LSEException" namespace="http://localhost:8080/axis/services/LogicService"
use="encoded"/>
</wsdl:fault>
</wsdl:operation>
  <wsdl:operation name="ExecStr">
<wsdlsoap:operation soapAction=""/>

```

```
<wsdl:input name="ExecStrRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded"/>
</wsdl:input>
  <wsdl:output name="ExecStrResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/axis/services/LogicService" use="encoded"/>
</wsdl:output>
    <wsdl:fault name="LSEException">
<wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
name="LSEException" namespace="http://localhost:8080/axis/services/LogicService"
use="encoded"/>
</wsdl:fault>
</wsdl:operation>
  <wsdl:operation name="GetStrArg">
<wsdlsoap:operation soapAction=""/>
  <wsdl:input name="GetStrArgRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded"/>
</wsdl:input>
    <wsdl:output name="GetStrArgResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/axis/services/LogicService" use="encoded"/>
</wsdl:output>
      <wsdl:fault name="LSEException">
<wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
name="LSEException" namespace="http://localhost:8080/axis/services/LogicService"
use="encoded"/>
</wsdl:fault>
</wsdl:operation>
</wsdl:binding>
  <wsdl:service name="LogicServiceService">
    <wsdl:port binding="impl:LogicServiceSoapBinding" name="LogicService">
<wsdlsoap:address location="http://localhost:8080/axis/services/LogicService"/>
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

Deriving the client stubs

We now switch to the development of the client. In order to communicate with the logic service – and question the knowledge base remotely – we must first know how to marshal our queries. The Web Service specification knows different protocols for invoking a Web Service. We will use RPC / SOAP without attachments for the following.

Axis makes the marshalling of Web service invocations easy: there is a tool to derive the client stubs directly from the deployed service. For this, we write a `generate_stubs.bat` batch (do this in a directory different from what has been done so far):

Step 16 – generate_stubs.bat

```
SET JAVA_HOME=c:\environment\jdk
SET ANT_HOME=c:\environment\ant

"%ANT_HOME%\bin\ant" stubs

pause
```

Then we need a `build.xml` that contains the tasks to perform on the Axis administration service:

Step 17 – build.xml (for the client, this time)

```
<!-- Webservice-Client-specific ANT build script -->

<project name="LogicServiceClient" default="stubs" basedir=".">

  <!-- Initializing -->
  <target name="init">
    <property name="dir" value="."/>
    <property name="axis.dir" value="c:\environment\tomcat\webapps\axis"/>
    <property name="axis.host" value="localhost"/>
    <property name="axis.port" value="8080"/>

    <path id="axis.classpath">
      <fileset dir="${axis.dir}/WEB-INF/lib">
        <include name="**/*.jar" />
      </fileset>
    </path>

    <taskdef resource="axis-tasks.properties"
      classpathref="axis.classpath" />
  </target>

  <target name="stubs" depends="init">
    <axis-wsd12java
      output="${dir}"
      testcase="true"
      verbose="true"
      url="http://${axis.host}:${axis.port}/axis/services/LogicService?wsdl" >
      <mapping
        namespace="http://${axis.host}:${axis.port}/axis/services/LogicService"
        package="marshalling" />
    </axis-wsd12java>
  </target>
```

```
</project>
```

Double-clicking `generate_stubs.bat` generates a package marshalling with all the stubs we will need to program a client that remotely invokes the LogicServer methods.

Note that if we were to attach to a remote logic service on a server other than localhost, all we had to change was the `axis.host` and `axis.port` properties above. The stubs would have been generated accordingly. This mechanism completely abstracts over networking issues. Even the host name is masked to the client programmer!

A client for logic services

Finally, we need to write a client program that makes use of the LogicService service:

Step 18 - FamilyService

```
class FamilyService
{
    marshalling.LogicServiceSoapBindingStub ls;

    public static void main(String args[])
    {
        System.out.println("Family dinner Prolog / Java demo");
        FamilyService family = new FamilyService() ;
        family.dinner() ;
    }

    public FamilyService()
    {
        try
        {
            marshalling.LogicService binding =
                new marshalling.LogicServiceServiceLocator().getLogicService() ;
            ls = (marshalling.LogicServiceSoapBindingStub) binding ;
            ls.setMaintainSession(true) ;
        }
        catch(Exception e)
        {
            System.out.println("Error in constructor: ") ;
            e.printStackTrace() ;
            System.exit(1) ;
        }
    }

    public void dinner()
    {
        try
        {
            ls.load("family");
            long term;
            String pred = "parents(X,Y,ian)" ;
            term = ls.execStr(pred) ;
            String father = ls.getStrArg(term,1) ;
            String mother = ls.getStrArg(term,2) ;
            System.out.println("Father: " + father) ;
            System.out.println("Mother: " + mother) ;
        }
        catch(Exception e)
        {
            System.out.println("Error during dinner: ") ;
            e.printStackTrace() ;
            System.exit(1) ;
        }
    }
}
```

To ease things, we write a compile.bat batch:

Step 19 - compile.bat

```

SET JAVA_HOME=c:\environment\jdk
SET AXIS_LIB=c:/environment/tomcat/webapps/axis/WEB-INF/lib

"%JAVA_HOME%\bin\javac"                                -classpath
.\;%AXIS_LIB%\xalan.jar;%AXIS_LIB%\xercesimpl.jar;%AXIS_LIB%\axis.jar;%AXIS_LIB
%\jaxrpc.jar;%AXIS_LIB%\activation.jar;%AXIS_LIB%\commons-discovery-
0.2.jar;%AXIS_LIB%\axis-schema.jar;%AXIS_LIB%\axis-ant.jar;%AXIS_LIB%\commons-
logging-1.0.4.jar;%AXIS_LIB%\imap.jar;%AXIS_LIB%\log4j-
1.2.8.jar;%AXIS_LIB%\mail.jar;%AXIS_LIB%\mailapi.jar;%AXIS_LIB%\pop3.jar;%AXIS_
LIB%\saaj.jar;%AXIS_LIB%\smtp.jar;%AXIS_LIB%\wsdl4j-1.5.1.jar *.java

pause

```

This file is a little bit tricky: You must put all the .jar of your Axis installations META-INF\lib directory on the CLASSPATH. In this directory must also be located what the Axis manual calls a XML parser. We have used (Xalan,2006) and copied the files xalan.jar and xercesimpl.jar to the lib directory.

Finally we also use a script to run the FamilyService:

Step 20 - run.bat

```

SET JAVA_HOME=c:\environment\jre
SET AMZI_DIR=. \
SET PATH=%PATH%;./amzijni.dll

CALL                                %JAVA_HOME%\bin\java                                -cp
.\;%AXIS_LIB%\xalan.jar;%AXIS_LIB%\xercesimpl.jar;%AXIS_LIB%\axis.jar;%AXIS_LIB
%\jaxrpc.jar;%AXIS_LIB%\activation.jar;%AXIS_LIB%\commons-discovery-
0.2.jar;%AXIS_LIB%\axis-schema.jar;%AXIS_LIB%\axis-ant.jar;%AXIS_LIB%\commons-
logging-1.0.4.jar;%AXIS_LIB%\imap.jar;%AXIS_LIB%\log4j-
1.2.8.jar;%AXIS_LIB%\mail.jar;%AXIS_LIB%\mailapi.jar;%AXIS_LIB%\pop3.jar;%AXIS_
LIB%\saaj.jar;%AXIS_LIB%\smtp.jar;%AXIS_LIB%\wsdl4j-1.5.1.jar FamilyService

pause

```

FamilyService remotely invites the LogicService to load the family.xpl file into the logic base.

The file family.xpl must be placed into the same directory as START_TOMCAT.bat and STOP_TOMCAT.bat. If you use a free, personal or academic version of Amzi!Prolog, you must compile and link the file on the machine that runs Tomcat / Axis.

The family service

And here we proudly present the result:

```
C:\TEMP\logic_service_client>run.bat

C:\TEMP\logic_service_client>CALL          c:\environment\jre\bin\java          -cp
.\;c:/environm
ent/tomcat/webapps/axis/WEB-
INF/lib\xalan.jar;c:/environment/tomcat/webapps/axis
/WEB-INF/lib\xercesimpl.jar;c:/environment/tomcat/webapps/axis/WEB-
INF/lib\axis.
jar;c:/environment/tomcat/webapps/axis/WEB-
INF/lib\jaxrpc.jar;c:/environment/tom
cat/webapps/axis/WEB-
INF/lib\activation.jar;c:/environment/tomcat/webapps/axis/W
EB-INF/lib\commons-discovery-0.2.jar;c:/environment/tomcat/webapps/axis/WEB-
INF/
lib\axis-schema.jar;c:/environment/tomcat/webapps/axis/WEB-INF/lib\axis-
ant.jar;
c:/environment/tomcat/webapps/axis/WEB-INF/lib\commons-logging-
1.0.4.jar;c:/envi
ronment/tomcat/webapps/axis/WEB-
INF/lib\imap.jar;c:/environment/tomcat/webapps/a
xis/WEB-INF/lib\log4j-1.2.8.jar;c:/environment/tomcat/webapps/axis/WEB-
INF/lib\m
ail.jar;c:/environment/tomcat/webapps/axis/WEB-
INF/lib\mailapi.jar;c:/environmen
t/tomcat/webapps/axis/WEB-
INF/lib\pop3.jar;c:/environment/tomcat/webapps/axis/WE
B-INF/lib\saaj.jar;c:/environment/tomcat/webapps/axis/WEB-
INF/lib\smtp.jar;c:/en
vironment/tomcat/webapps/axis/WEB-INF/lib\wsdl4j-1.5.1.jar FamilyService
Family dinner Prolog / Java demo
Father: alex
Mother: sarah

C:\TEMP\logic_service_client>pause
Press any key to continue . . .
```

Conclusions

The intelligent service paradigm offers a new promise for building complex software because of the abstraction and embeddability it provides. But definitely: despite the promising combination of a declarative language (Prolog) with a sequential one (Java), the Family service has not much of an intelligent agent. There is much more to intelligent agents than hybrid Prolog / Java components. The latin source of the word agent is to drive, lead, act or do. How can what has been demonstrated in this paper be used to implement a component that intelligently drives, leads, acts or does? The idea behind obviously is to build computer surrogates for human behavior in fields that are too hostile or too complex for us. The approach to reduce problem complexity with loosely coupled, interconnected intelligent services is very promising here. Regarding that, the paper shows the **potential to overcome traditional barriers to “intelligent” interoperability** and to efficient incorporation of heterogeneous Web resources.

It is clear that probably no sequentially programmed intelligent service will ever see the light in the Web of services. To address criteria for intelligence with believes, desires, intentions, emotions and wisdom, we need a combination of traditional software engineering approaches with languages that provide facilities for expressing complex deliberation with a kind of abstraction over lower-level, communication protocol-close preoccupations. Both software engineering (with the Web service specifications) and applied AI engineering (with embeddable and extensible logic servers) have done a major step to facilitate the task of implementing **intelligent services**. If truly intelligent services are to be made available to the masses on the Internet one day, it will probably be by a combination of both approaches.

Finally, the question of what to place in the intelligent services' minds (in the knowledge base in the declarative kernel) relates to meta-physics. Developments in semantic Web, ontologies and domains are once again opening the door between the software developer's and the philosopher's offices. For us, it is clear that declarative languages allow “the spark to spring out of our brain and jump into a computer program”. And this is much easier than with traditional, sequential languages. With these tools in our hands, we feel ready to **focus on real life business processes** with an intrinsic logic that goes further than the classic HelloWorld, PetShopStore and Customer File application demos for mainstream Web components.

Exercises

1. Take the code from step 6 and program a second brother/2 predicate in Java that results in true if both arguments X and Y have the same mother Z.

2. Write the corresponding extended predicates in Java for sister/2.

* 3. Write an extended predicate brother/3 that functions like brother/2 but where the common father Z is unified back to Prolog after successful proof that Z is the father of X and Y. For this, look up the syntax

```
ls.UnifyStrParm(3, Z);
```

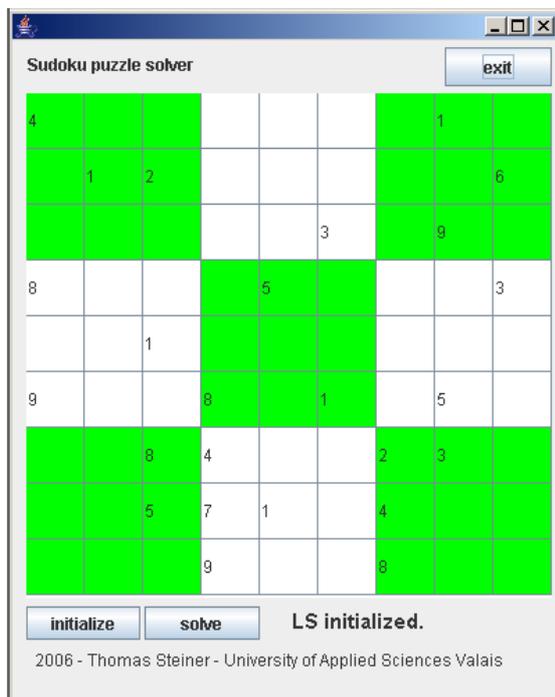
in the Amzi documentation. This results in true if the third argument of the previously received term can be unified with Z. Z is a variable in Java here that will hold the name of the father. The evident condition is that the Prolog predicate brother/3 is called with a variable as argument 3.

**4. Program a Sudoku service!

The Sudoku game is played on a 9X9 grid. Each figure from 1 to 9 can once and only once appear on each column, on each line and in each quadrant.

Program and compile a Sudoku solver in Prolog.

Then take the program and upload it to the LogicService as developed in this paper. Write a client that accesses the remote Sudoku service. Such a client could look like this:



References and further reading

- Amzi. <http://www.amzi.com> [01/06].
- Axis. <http://ws.apache.org/axis/> [01/06].
- Bourguignon, J.-M. *SQL - Concevoir et programmer les bases de données relationnelles*. Dunod informatique, 1991.
- Clocksinn, W.F. *Clause and Effect – Prolog Programming for the Working Programmer*. Springer, 1997.
- Denet, D.C. *Kinds of Minds*. Weidenfeld & Nicholson, 1996.
- Fagin, R, Halpern, J.Y., Moses, Y. & M.Y. Vardi: *Reasoning about knowledge*. MIT press 1995.
- Gardner, H. *Frames of Mind*. Basic Books, 1983.
- Goleman, D. *Emotional Intelligence*. Bantam Books, 1995.
- Hernandez, M.J. & J.L. Viescas. *GoTo SQL*. Addison-Wesley, 2001. (Exists in English, German and French).
- JAF. Sun's JavaBeans Activation Framework. <http://java.sun.com/products/javabeans/glasgow/jaf.html> [01/06]
- Maes, P. *Designing autonomous agents*. MIT press 1990.
- O'Keefe, R. *The Craft of Prolog*. MIT press, 1994.
- Russell, S. & P. Norvig. *Artificial Intelligence – A Modern Approach*. Prentice-Hall, 1995.
- Sterling, L. & E. Shapiro. *The Art of Prolog*. MIT press, 1997.
- Steiner, T. *Distributed Software Agents for World-Wide-Web based Destination Systems*. PhD thesis, University of Lausanne, 1999.
- Steiner, T. *Prolog goes Middleware - Java-based Embedding of Logic Servers*. PC Artificial Intelligence - July / August 2001.
- Steiner, T. *Prolog, Java und Middleware*. Java Spektrum 5, 101 SIGS-Datacom 2001.
- Tomcat. <Http://tomcat.apache.org> [01/2006].
- Wollheim, R. *On the Emotions – The Ernst Cassirer Lectures, 1991*. Yale University Press, 1999.
- Xalan. <http://xml.apache.org/xalan-j/> [01/06]