Adapting a Legacy Code for Ordinary Differential Equations to Novel Software and Hardware Architectures

DANA PETCU, ANDREI ECKSTEIN, CLAUDIU GIURGIU
Institute e-Austria Timisoara
and Computer Science Department, Western University of Timisoara
B-dul Vasile Parvan 4, 300223 Timisoara
ROMANIA
petcu@info.uvt.ro http://web.info.uvt.ro/~petcu

Abstract: - Modern software engineering concepts, like software as a service, allow the extension of the legacy code lifetime and the reduction of software maintenance costs. The transformation of a legacy code into a service is not straightforward task, especially when the initial code was designed with a rich user interface. A special case is presented in this paper, that of a software code for solving ordinary differential equations. Initially designed to use parallel computing techniques in the solving process, the code is now modified to take advantages of the current multi-core architectures. The transformation paths are general and can be followed by other similar legacy codes.

Key -Words: - Wrapper, Web service, Parallel methods, Multicore architectures, Ordinary differential equations

1 Introduction

The service oriented architecture (SOA) is a current paradigm for organizing and utilizing distributed capabilities that may be under control of different ownership domains. It provides uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations. Its attraction is due to the fact that it builds on concepts of reusable software components, while emphasizing the service's abstraction. This means that the services are interoperable, reusable, independent, stateless and autonomous. To enable interoperability, services should be composable, loosely coupled, and standards compliant. The resources available across a network are made available as independent services that can be accessed without knowledge of their underlying platform implementation. The primary focus of the SOA latest developments is on dynamic reconfiguration of services and on developing business services.

Service-oriented technologies as current solution for large distributed systems are addressing the computing and storage needs arising from many scientific and industrial application areas. Developing new codes, instrumenting applications with middleware specific interfaces, or designing applications to explicitly take advantage of distributed resources is a significant burden for the developers who are often reluctant to allocate sufficient effort on non application specific problems. The middleware is therefore expected to ease legacy codes migration to service-oriented infrastructures by proposing a non-intrusive interface to existing legacy codes, and optimizing the execution of the application on the available resources. In this context, enabling legacy code execution on service-oriented infrastructures is a high priority challenge.

The migration of a legacy system towards a service-oriented architecture is, unfortunately, not a straightforward task. At least two problems can be straightforward identified:

- establishing which part of the legacy system can be exposed as service;
- establishing how the transformation will be done technically.

The most appropriate legacy systems for the migration towards Web services are those which are conceived as black-boxes that are callable through a command line and having fixed input and output formats. The full functionality of the system can be available through wrapping. Two issues should be still treated:

- if the number of the legacy code functions are very high (of thousands order) the available tools for handling services are not facing the requirements;

- the public expose of all the legacy code functions can be a danger for the system hosting the wrapped software if the exposed function list includes functions that modify the host environment.

In both cases the solution is to have a restricted list of functions that are exposed through the wrapper. The wrapper translates the incoming requests from the XML-like formats into the format understand by the legacy code and the code outputs into XML-like format. Recently we have analyzed several cases that conform to these characteristics and details are given in [16].

The problem of which part of the legacy code can be exposed is not easy to handle in the more complex case, that of migrating a legacy system with a rich user interface. In this paper we discuss such a case.

The second problem, establishing how the transformation will be done, can be approached via several techniques. We review them briefly. A detailed analysis is presented in [5].

A first class of techniques comprises the black-box reengineering techniques which integrate systems via adaptors that wrap legacy code as a service (as mentioned above).

A second class comprises white-box methods which require code analysis and modification in order to obtain the code components of the system to be presented as services. Both approaches are valid in different circumstances, depending on factors such as the granularity of the code, the assumed users and application area.

The first class is mainly applied in the case when the code is not available. Recent papers on this subject are [1] and [9]. A solution for the particular case of interactive legacy systems is described in [4].

Java wrapping can be used to generate the service interfaces automatically as outlined in [13]. Prominent examples in this direction are SWIG, JAVAW or MEDLI [10].

The most remarkable non-invasive solution is represented by GEMLCA, the Grid Execution Management for Legacy Code [8]. The deployment process of a legacy code as GEMLCA service requires only a user-level understanding of the legacy application (e.g. parameters of the legacy code, kind of environment needed to run). GEMLCA provides the capability to convert legacy codes into Grid services by describing the legacy parameters and environment values in the XML-like file. A drawback is that it supposes that the legacy code is activated in a command-line style and does not exploit the possible successive interactions. The same comment is valid also for O'SOAP [18] that also allows legacy command-line oriented applications to be deployed as Web services without any modification, as well as for OPAL [12].

Recently, we proposed in [6] some technical solutions for the migration of the well-known interactive software tools used in the particular field of symbolic computations.

The second class mentioned above is based on invasive procedures on the legacy codes that usually improve the efficiency of legacy code. In this invasive approach, it is typically assumed that the application programmer has some programming background and would like to build services using specific software libraries.

In this paper we make use of a third possible class, mentioned in [5], the class of the grey-box techniques, that combine wrapping and white-box approaches for integrating those parts of the system that are more valuable.

We present a case study on an interactive legacy system that was designed ten year ago to provide numerical solutions for initial value problems for systems of ordinary differential equations and incorporates an expert system. The part of the legacy system that is the most computationally intensive is migrated as a Web service, while the user interface and the expert part are recoded in Java for portability reasons. Following this approach, the computational service can be accessed by any client that sends a message in a specific XML-like format containing the problem description and the method to be applied. Furthermore, the module that implements the parallel numerical methods and differentiate the code from others available at its designing time, as one important component of the part wrapped as a Web service, was extended to allow the efficient use of the multicore architectures.

Taking into consideration the current trends to increase the number of processors on a chip, the extent to which software can be multithreaded to take advantage of the multicore chips is likely to be the main constraint on software performance in the future. Numerical computations requiring both CPU power and large memory are well suited candidates for deriving advantages from the current multicore architectures. In this context, it is necessary to design and implement new libraries and tools for parallel numeric computations, or to re-engineer the old ones, especially for the new parallel computing environments using multicore processors.

One can notice that several parallel numeric computation packages were designed at the beginning of the previous decade assuming a

shared-memory parallel computing environment. The subsequent evolution of the hardware towards distributed-memory parallel computers and clusters of workstations has lead to the impossibility to use previously developed shared-memory parallel codes and to the need of designing and implement new versions that are well suited for distributed memory. In particular, for the case of computing the numerical solutions of large systems of ordinary differential equations, this architectural change had a tremendous effect: the class of techniques well suited for implementation on parallel computing environments has been changed from the ones applying parallelism across the method towards those applying parallelism across the steps. The techniques that are mentioned above were revised in [3]. By switching to multicore architectures, the question that raises naturally is that of re-imposing the status of the parallelism across the method. We prove in this paper that there is a positive answer: one can consider again the parallelism across method as an efficient technique for improving the response time of the numerical software codes for ordinary differential equations when a multicore architecture is used.

The paper is organized as follows. Section 2 describes shortly the system that is used as case study, while Section 3 presents the system's computational component that is wrapped as a Web service. The benefits of adding multithreaded functionality is discussed in Section 4. Finally, some conclusions are drawn in Section 5. This paper is an extended version of the recent paper [17].

2 EpODE's Characteristics, Components, and Current Limitations

The ExPert system for Ordinary Differential Equations, EpODE, was designed as a tool for solving by numerical procedures initial value problems for large systems of ordinary differential equations (ODEs). It is also an expert system since it provides:

- an automated identification of problem properties that is defined by the system user, e.g. linearity, sparsity, stiffness, degree of parallelism across the problem;
- an automated identification of the properties of the solving method, e.g. explicit or implicit, onestep or multistep, onederivative or multiderivative, onstage or multistage, method order, error constant, stability characteristics, degree of parallelism across the method;
- an automated selection of the adequate method according the problem properties;
- an automated estimation of the computation time for a specific problem and a specific method using the host computer;
- parallel computing facilities in a cluster or a parallel computer in the case when the estimated time for solving the problem is too high.

2.1 Characteristics

EpODE can be used as a tool for describing, analyzing and testing new types of iterative methods for ODEs, mainly due to the method properties detector, as well as the immediate possibility to apply them on a large class of problems. In particular, it allows also to study the methods that are proposed for parallel or distributed implementation using real or simulated parallel computing environments.

After defining the problem, the solving method and the computation parameters can be given by an human expert, or can be the task of the automatic selector. In the human-exert mode, the tool can be used to underline the effects of over-passing the step size restrictions imposed by accuracy or stability of the numerical process. The method automatic selection is based on a simple decision tree, and depends on the type of problem that will be solved, the admitted global error, and a maximum for the computation time. The approximate values of the solution can be visualized using some graphic facilities in two- and three- dimensional space or using some tabular form. The numerical results can also be saved in order to be interpreted within other tools. Classical performance measurements, like computation time, number of function evaluations, or the estimated error, are provided after the end of the solving process. These measurements can be used for comparing distinct methods applied to the same problem.

It is important to notice that EpODE is freely distributed with a rich database of problems (at least one hundred real and test problems, including those classical ones that are used in tesing new methods) and a rich database of solving methods (almost one hundred too, including Runge-Kutta methods, multistep methods, multi-derivative multistep methods, block methods, hybrid methods, nonlinear multistep methods, general linear methods). These databases can be extended by the tool user with its own defined problems or methods, allowing a very easy and comprehensive comparison with classical problem and methods that are already in the database.

The main characteristics of EpODE which distinguish it from other ODE solving environments are the followings:

- the friendly interface for describing new problems and solving methods;
- the method recommender system specially designed for stiff large systems;
- the extensible database of methods and problems;
- the extensive problem and method properties detector;
- the dynamic memory allocation scheme avoid the constraint on the dimension of the problem to be solved;
- the unique problem solving procedure for all the methods that allows all solvers to behave in a coherent way;
- the independence from other software packages with one exception, that of Parallel Virtual Machine (PVM) used for parallel or distributed computations.

Details about EpODE's design are given in the early paper [14]. Several experiments on parallel computers and cluster environments were reported later in [15].

2.2 Components

EpODE has five major components:

1. a user interface, the front end of which permits the description of an initial value problem for the ODEs or an iterative method, the control of the solution computation process, and the interpretation of the results of the computation; help facilities are provided in order to assist the user in using the software;
2. a properties detection mechanism containing the procedures for establishing some properties of ODEs or those of an iterative method;
3. a mechanism for selecting the solving procedure, implementing the decision tree for the selection of the class of iterative methods according to the properties of the initial value problem for ODEs and for the selection of one method from this class according to the solution accuracy requirements and time restrictions;
4. a sequential computing procedure, a generic solving procedure whose parameters corresponds to the current problem and the selected method.
5. a parallel computing procedure, a generic solving procedure that is similar with the sequential procedure, but includes also the splitting of the computational effort to more than one process as well as they coordination.

At the time of its design EpODE was the unique tool that allowed the above mentioned facilities. Only a recently developed tool reported in [2] has similar facilities (without the ones for applying parallelism techniques).

2.3 Limitations

EpODE was written ten years ago in C++ and two graphical interfaces were provided, for Windows'95 and X Windows. In other words, the first component mentioned above was not designed to be portable and this fact lead to usage problems when new operating systems have appear. In order to solve this issue, a rewritten of this component is needed in a portable version, e.g. as Java code. The next section presents the new interface exposed as a Web service.

The other complex components can be conserved as they are. In Section 3 we describe how the last two components are wrapped and presented as Web service. The other two are not included yet in the service, but will be the subject of further development of the Web service or of another specific Web service.

Concerning the efficiency of the parallel techniques in solving ordinary differential equations, that were also implemented by EpODE, one should note that the rapid development of the hardware in the last ten years have affected the notion of the most adequate technique. Indeed, a rerun of the experiments reported in [15] revealed that the current hardware improvements led to a response time of the computational procedures hundreds of times shorter. In these conditions the problem dimension for which the parallel computing techniques are efficient, in the sense that the computational time dominates the communication time, is increasing by at least ten times.

One should remember that the three classes of techniques applied to achieve parallelism in solving ODEs are: parallelism across the system (across space), parallelism across the method, and parallelism across the steps (across time). More details about this subject can be found in [3].

According to the technique of parallelism across the system various components of the system of ODEs are distributed amongst available processors. This technique is especially effective in explicit solving methods and when the system can be split into a number of independent systems, that is a uncommon case. EpODE detects the sparsity of the

system and allows to apply the technique of parallelism across system. The efficiency results are not considerable affected by the hardware changes since the computations are almost independent.

According to the technique of parallelism across method, each processor executes a different part of a method. This approach has the triple advantage of being application-independent (it does not require user intervention or special properties of the given systems of ODE), of avoiding load balancing problems, and of using a small number of processors. The main disadvantage is the limited speed-up. EpODE detects the degree of parallelism across the method and allows to apply the technique in the solving process. The efficiency results are strongly affected by the kind of memory that is used in the parallel computing environment, as well as the ratio between the communication and computation times.

The parallelism across steps is the only possibility for using large-scale parallelism on small problems. Contrary to the step-by-step idea, several steps are performed simultaneously, yielding numerical approximations in many points of the independent variable axis (the time). Some continuous time iteration methods are used to decouple the ODE system, and henceforth to discretize the resulting subsystems, by solving them concurrently. The number of discrete points handled simultaneously is the degree of parallelism of the method. The main weakness of this approach is that the iteration process may suffer from slow convergence or even divergence. Despite the fact that EpODE implements also this technique, we have not perform yet efficiency tests to see how the new hardware architecture affects the efficiency results – this is a subject for further developments.

3 EpODE's Computational Kernel as a Web Service

The most intensive computational part of EpODE consists in the generic numerical solving procedure for sequential or parallel iterative methods applied to initial value problems for ODEs. The procedure is generic in the sense that it does not depend on the specific problem or the particular method – the concrete problem and methods are given as parameters. Since there is no need of user intervention in the computational process, and, at the same time, there is a need for a fast response, this part of EpODE is well suited for transformation into a computational service lying on a remote high-performance server.

3.1 Technologies and operations

The component that implements the computational procedure in C++ and PVM is wrapped as a statefull Web service (WSRF implementation using Globus Toolkit 4 – see other wrapping examples in black-box style reported in [16]).

The container of the Web service is based on Tomcat technologies. Axis is used as implementation of the SOAP specification. The WSDL file of the service was generated with the Java2WSDL tool of Axis.

The Web service has four operations:

1. *setmethod* to set the solving method;
2. *setproblem* to set the problem to be solve;
3. *setcompute* to set the computation parameters;
4. *compute* to start a computation;
5. *getstatus* to retrieve the computation status;
6. *getresults* to retrieve the computation results.

We describe in the subsection 3.3 their actions.

3.2 Data structures

The unique solving procedure for any type of iterative method for ODEs takes into account the variety of mathematical forms that a solving method can have (one- or multi-stage, one- or multi-step, one- or multi-derivative, explicit or implicit methods).

A specification of the method in EpODE includes: the iterative formula, the starting procedure, the implicit equation solver, the error control procedure. In order to define a new iterative formula, the user must specify the variables, the right side of the formula, the variables whose values will be stored, and the link between the old and the new values of the variables. In order to define a starting procedure for a multistep method, the user must describe or a onestep method (the right side of an iterative formula). In order to define an implicit equation solver, e.g. simple iterations or Newton like iterations and some starting values (using, for example, onestep method). In order to define the error control procedure, the user must specify the variables which must be checked.

Uniformity in defining the difference methods allows in EpODE to construct a unique procedure for interpreting the data about an arbitrary method. The parameters of such a procedure are the outputs of the method interpreter provided in a condensed form, like evaluation trees of some arithmetic expression.

The Web service receives the method definition in the form of an XML-like structure that is described it its WSDL as follows:

```
<xsd:element name=”setmethod”>
 <xsd:complexType>
 <xsd:sequence>
   <xsd:element name="Implicit" type="xsd:Boolean"/>
   <xsd:element name="MStep" type="xsd:Boolean"/>
   <xsd:element name="MStage" type="xsd:Boolean"/>
   <xsd:element name="MDeriv" type="xsd:Boolean"/>
   <xsd:element name="Newton" type="xsd:Boolean"/>
   <xsd:element name="Nsta" type="xsd:int"/>
   <xsd:element name="Nfin" type="xsd:int"/>
   <xsd:element name="Nplu" type="xsd:int"/>
   <xsd:element name="Nimp" type="xsd:int"/>
   <xsd:element name="Mpas" type="xsd:int"/>
   <xsd:element name="Care" type="xsd:int"/>
   <xsd:element name="VarMet" type="xsd1:ArrayStr"/>
.  <xsd:element name=”SupEqs” type="xsd1:ArrayStr"/>
.  <xsd:element name=”StaEqs” type="xsd1:ArrayStr"/>
   <xsd:element name="FinEqs" type="xsd1:ArrayStr"/>
   <xsd:element name="PluEqs" type="xsd1:ArrayStr"/>
  </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

The data structure fields above are referring to some method properties: implicit or explicit, multistep or not, multistage or not, multiderivative or not, etc. Then the method equations follow. ArrayStr is described in the <types> part of the WSDL:

```
 <complexType name="ArrayStr">
  <complexContent>
   <restriction base="soapenc:Array">
     <attribute ref="soapenc:arrayType"
          wsdl:arrayType="string[]"/>
   </restriction>
  </complexContent>
 </complexType>
```

A Java client that will access the service to compute the solution of a problem with the DIRK4 method [3], described by the iterative process:

$$y_{n+1}=y_n+h(11k_1+25k_2+11k_3+25k_4)/72,\ n=0,1,...$$
$$k_1=f(y_n+hk_1)$$
$$k_2=f(y_n+3hk_2/5)$$
$$k_3=f(y_n+h(171k_1-225k_2+44k_3)/44)$$
$$k_4=f(y_n+h(39k_2-43k_1+12k_4)/20)$$

where f is the system function and y the unknown vector function from the $y'(t)=f(t,y(t)),\ y(0)=y_0$, can have a piece of code similar to the following sequence:

```
Epode_service.setmethod(
      true,   //implicit  method
      false,  //onestep method
      true,   //multistage method
      false,  //onederivative method
      true,   //use Newton iteration to solve implicit eqs.
      1,      //no. of stages
      1,      //no. of variables stored at the end of the step
      4,      //no. of intermediate variables
      4,      //no. of implicit equations
      0,      //steps skipped at next iteration (block case)
      0,      //index of the method variables to be saved
      {“h”,”y”,”k1”,”k2”,”k3”,”k4”,”x”}, //method vars
      {}      //supplementary eqs. for implicit solving proc.
      {}      //start eqs. for multistep methods
      {“x+h*(11*k1+25*k2+11*k3+25*k4)/72)”} //fin.eq.
      {“fct(x+h*k1)”,”fct(x+3*h*k2/5)”,
        ”fct(x+h*(171*k1-215*k2+44*k3)/44)”,
        “fct(x+h*(39*k2-43*k1+12*k4)/20)”} //plus eqs.
   );
```

The data structure describing the problem to be solved is present in the WSDL file:

```
<xsd:element name=”setproblem”>
 <xsd:complexType>
 <xsd:sequence>
  <xsd:element name="Dim" type="xsd:int"/>
  <xsd:element name="Vars" type="xsd1:ArrayStr"/>
  <xsd:element name="Eqs" type="xsd1:ArrayStr"/>
  <xsd:element name="T0" type="xsd:double"/>
  <xsd:element name="InitV" type="xsd1:ArrayStr"/>
 </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

where Dim is the problem dimension, Vars is a vector with the problem variables, Eqs are the differential equations, BJacob is a Boolean matrix indicating the non-zero positions in the Jacobian matrix of the system, Jacob are the non-zero elements of the Jacobian matrix of the system, T0 is the initial value of the independent variable, and InitV is the vector of the initial values.

A Java client that access the Web service to compute the solution of a simple problem like the following one:

$$u'(t)=v(t),\ v'(t)=5(1-u(t)^2)v(t)-u(t),\ u(0)=2,\ v(0)=0$$

will have a piece of code similar to the following:

```
Epode_service.setproblem(
      2,                          //problem dimension
      {“t”,”u”,”v”},              //problem variables
      {“v”,”5*(1-u*u)*v-u”}, //problem equations
       0.0,                       //Start value of the indep.var.
       {“2”,”0”}                  //Initial values
   )
```

The data structure describing the options for the computations includes:

```
<xsd:element name=”setcompute”>
 <xsd:complexType>
 <xsd:sequence>
  <xsd:element name="Step" type="xsd:double"/>
  <xsd:element name="T1" type="xsd:double"/>
  <xsd:element name="WhichV" type="xsd1:ArrayStr"/>
  <xsd:element name="PVM" type="xsd:Boolean"/>
 </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

A Java client that access the Web service to compute the solution of a the above described problem will have a piece of code similar to:

```
Epode_service.setcompute(
    0.0001,     //method step
    1,          //maxim value of  the independent var.
    {"u","v"}, //variables to be stored
    true        //apply parallelism across method
)
```

Due to the fact that the method and problem properties detector was currently not included yet in the Web service has the disadvantage that the user must provide at this moment a more complex structure that is described above: the equations should be given not only in they explicit form, but also in the Polish form, and, moreover, in the case of the problem, the Jacobian matrix should be provided too. This inconvenient should be removed soon. Moreover, for the sake of the initial testing, only the option for parallelism across the method is activated through the interface. Further developments will include into the computational kernel the EpODE's component that transforms any expression in its Polish form, the EpODE's facilities for parallelism across steps, as well as for parallelism across the problem.

3.3 Actions

The Java code that implements the wrapper translates the setting requests as copying actions of the incoming complex data structures into its private variables. A request for *compute* has the following consequences:

1. write the problem and method descriptions (implicit values are used if set* requests were not used), as well as supplementary information requested to proceed with the computation, into a file with a specific format;
2. transfer at the server site where the computational procedure will be launch the previously written file with the problem, method, and computation parameters;
3. call the computational procedure through a line command that specifies the file.

A pipe is establish between the wrapper and the computational kernel. The client receives an acknowledgement if the computational procedure has start. Later it can ask for the status of the computa-tion through the *getstatus* operation of the Web service (the wrapper). If the status is "done" than the client can ask for the results through the *getresults* operation of the Web service that will transfer the file with the numerical solution.

To allow the late retrieval of the result, for each client a new instance of the service will be created. The client knows the address of a register of services and queries the registry about the computational service. The service register sends back the address of the service factory that matches the query. The client contacts the service factory and requests a service instance. The service factory creates an instance of a service interface (the wrapper). Then the client sends to the interface the method, problem and computing parameters, as well as the compute request. The service interface will launch the code of interest, and:

- if the code will run on the server, then the code will be launched by a thread of the service interface;
- if a cluster scheduler is installed on the server, then the code will be launched on the cluster by the scheduler that is called by a thread of the service interface;
- if no scheduler is installed and the server is not part of the cluster, but lies in the same security domain as the cluster, the remote code can be invoked through classical rsh/ssh commands by a thread of the service interface.

3.4 Usage scenarios

We can imagine three kinds of clients of the Web service:

1. EpODE GUI interface
2. another tool that needs a fast numeric computation of the solution of an initial value problem for ordinary differential equations;
3. a workflow execution engine.

EpODE's graphical used interface will be redesigned soon to allow remote computations when the estimated computation time is too high and there is a need for a faster response by using the Web service that is lying on a high-performance server;

The role of the client of the Web service can also be played by another tool that sends the input data in the requested format. For example, through the tools described in [6] symbolic computing systems can access and consume Web services – it is known that such systems are slow when they work with their internal numerical procedures and they can really benefit from faster external codes.

We can imagine also the case when the Web service is called by another numerical software code, that solves partial differential equations and during its solving procedure it transforms the problem into a large system of ordinary differential

equations. Note that the largest ODE systems that are usually used in testing ODE software tools are provided by a such discretization processes [3]. Moreover, the Jacobian of the system in its symbolic form currently requested by EpODE computational procedures can be easily generated with a computer algebra system – for example an older software tool, ODEXPERT [11] uses Maple for this task.

We can further imagine a more complex scenario in which several Web services are composed: one for generating the ODE system, another for computing the Jacobian, both wrapped as Web services, are sending the necessary information for the expert system (also a Web service) that picks an appropriate method from a rich database (can be the EpODE component) and asks the above described Web service to perform the computation, and finally sends the numeric computation results to a visualization tool that is also wrapped as a Web service. The entire scenario is described into the graphical interface of a workflow editor and engine. This scenario will be the basis of our future development.

Compared to the wrapping examples that were described in the recent paper [16], the current one has a higher degree of potential integration in a service-oriented architecture. For example, in [16] is described the wrapping of a CFD code and a evolutionary computing: a specific simulation is transferred from the client side to the service side – the client should know how this file should look like. In the case described in this section, the input data are described in an XML-like format that can be discovered by any client of the service.

4 Adding a Multi-threaded Facility for Multicore Architectures

The ability of multi-core processors to increase application performance depends on the use of multiple threads within applications. Numerical computations are well-suited candidates for deriving advantages from multi-core parallel architectures. This is possible only if the specific libraries and tools are designed to allow multi-threading and multi-processes.

EpODE was designed to allow the experimentation of parallel methods when solving initial value problems for ODEs. As mentioned in Section 2, there are three classical approaches:

1. parallelism across the problem that depends on the degree on the sparsity of the system's Jacobian matrix;
2. parallelism across the method that depends on the number of the method's variables that can be computed simultaneously, and
3. parallelism across the steps that allows a higher degree of parallelism with the drawback of heavy control of the convergence of the numerical solutions towards the exact one.

The parallelism across the method was a viable solution ten years ago in the case of large systems and the availability of a small network of computers or a parallel computer. With computation power increasing faster than communication speed, parallel computations based on parallelism across the method are justified only in the case of systems with hundreds of equations. Indeed, we have re-run the experiments reported in [15] dealing with systems of almost one hundred equations on a new generation cluster (7 HP ProLiant DL-385 with 2 x CPU AMD Opteron 2.4 GHz, dual core, 1 MB L2 cache per core, 4 GB DDRAM, 2 network cards 1 Gb/s) and the results show that the parallel variant is no longer efficient at the same scale of problem dimensions.

The question is if we can improve the efficiency of the basic non-parallel computational procedures by implementing parallelism across the method through multithreading when running on multicore architectures. To be able to answer this question, we have rewritten some parts of the C++ code for the computational procedures of EpODE dealing with the parallelism across method. The multithreading implementation is close to the one based on the PVM library – instead of PVM processes, threads are used, and instead message passing, threads are communicating through a common matrix.

The answer to the above question is positive: the response time of the computational procedure is clearly improved using the multi-threaded version compared with the non-thread or parallel version at the problem dimensions for which the old parallel code is no more efficient.

In order to prove the above statement, Table 1 shows the response times of the code in the case of two classical problems of 81, and 140 equations, respectively, solved by some representative methods from different classes of parallel methods. DIRK4 is the 4-stage 4th order Diagonally Implicit Runge-Kutta method that was described in Section 3, PC1 is the predictor-corrector scheme based on the implicit trapezoidal rule, PC6 is another predictor-corrector scheme, while BL1 and BL2 are one-stage block methods, all of them available through the rich database of methods provided by EpODE. ME140 is a discretization of the Medical Akzo

Nobel problem, using the method of lines [7]. Plate81 is obtained by following the same procedure starting from a diffusion problem. Please refer to [3, 15] for the description of these problems and methods.

Table 1: Response times of the computational procedure with or without threads

Problem	Method		No. steps	Time (ms)	
	Acronym	Parall. degree		No threads	With threads
Plate81	DIRK4	2	5	2031	1519
	FR2	2	50	3658	2068
ME140	PC1	2	50	5165	3624
	PC6	2	50	10022	4992
	BL1	2	50	3428	2768
	BL2	3	10	3119	1431

In order to incorporate the multithreading facility into the Web service described in the previous section, the following field from the computing parameters description should change from a Boolean to an integer value:

```
<xsd:element name="PVM" type="xsd:int"/>
```

This variable should be used as follows:

1. PVM is set to the value 1 with the meaning that parallelism across method through PVM should be used when the problem dimension is of hundreds order of equations;
2. PVM is set to the value 2 with the meaning that parallelism across method through multithreadding should be used when the problem dimension is of ten order of equations;
3. PVM is set to the value 0 when parallelism facilities are not used due to the fact that the system to deal with is small

As stated in Section 3 there are different ways to launch the legacy code: on the same server as the service interface or at a remote location (specified or decided by a scheduler). In the case of the current Web service that wrap the legacy code, the service container and the service itself run on the interface node of the above-mentioned multicore cluster, so each option specified by the PVM parameter can be properly exploited when EpODE's computational kernel is launch by the service interface.

5 Conclusions and Future Work

In order to prolong the lifetime of a legacy code we have used a partial-invasive technique for migrating it towards a service-oriented architecture. One of its unique components, the one that can profit from the computational power of remote high-performance servers, was wrapped as a Web service. This new service can be accessed by any client code that respects the format of the input data describing the problem to be solved, the iterative method and the computation parameters. The migration opens new possibilities to exploit the facilities provided by the legacy code by combining it with other services to offer complex computational scientific services.

Taking into consideration the current trends to increase the number of processors on a chip, the extent to which software can be multi-threaded to take advantage of the multi-core chips is likely to be the main constraint on software performance in the future. Therefore the computational kernel of the legacy code was modified to allow multithreading. Initial tests are proving that the transformation has clear advantages when solving medium size initial value problems for ordinary differential equations.

While the computational kernel was successfully adapted to make efficient use of multicore architectures, several other components are still left to be translated into the new user interface. The improvement and the integration of the expert system into the same Web service or another Web service is one of the next steps to be taken soon. Complex usage scenarios, as the one described in Section 3, should be the context of the future intensive tests of the Web service.

Moreover, the transformations that were performed on the expert system for ordinary differential equations have a high degree of generality and can be easily apply to other tools in order to be incorporated as valuable components in the new service-oriented or multi-core architectures.

Acknowledgements: The reported work was performed in the frame of the national projects NanoSim CEEX-I-PC-D03-PT04-439 (first author, from Institute e-Austria Timisoara) and GRAI CEEX-II-12/2006 (second author, from Western University of Timisoara) funded by the Romanian Ministry of Research.

References:

[1] B. Balis, M. Bubak and M. Wegiel, A Solution for Adapting Legacy Code as Web Services, *Component Models and Systems for Grid Applications*, V. Getov and T. Kiellmann (eds.), Springer, 2005, pp. 57–75.

[2] B. Bunus, A Simulation and Decision Framework for Selection of Numerical Solvers in Scientific Computing, *Procs. Annual*

Simulation Symposium 2006, vol. 39, IEEE Computer Press, 2006, pp. 178–187.

[3] K. Burrage, *Parallel and Sequential Methods for Ordinary Differential Equations*, Numerical Mathematics and Scientific Computation, Oxford University Press, 1995.

[4] G. Canfora, A.R. Fasolino, G. Frattolillo, P. Tramontana, Migrating Interactive Legacy System to Web Services, *Procs. 10th European Conference on Software Maintenance and Reengineering*, IEEE Computer Press, 2006, pp. 23-32.

[5] G. Canfora, A.R. Fasolino, G. Frattolillo, P. Tramontana, A Wrapping Approach for Migrating Legacy System Interactive Functionalities to Service Oriented Architectures, *J. Syst. Software*, 81, issue 4, 2008, pp. 463-480.

[6] A. Carstea, M. Frincu, G. Macariu, D. Petcu, K. Hammond, Generic Access to Web an Grid-based Symbolic Computing Services, *Procs. ISPDC 2007*, Hagenberg, IEEE Computer Press, 2007, pp. 143–150.

[7] M.T. Darvishi and M. Javidi, Method of Lines for Solving Systems of Time-dependent Partial Differential Equations, *WSEAS Transactions on Mathematics*, vol. 1 (4), 2002, pp. 218– 222.

[8] T. Glatard, D. Emsellem, J. Montagnat, Generic Web Service Wrapper for Efficient Embedding of Legacy Codes in Service-based Workflows, *Procs. GELA 2006*, pp. 44—53.

[9] D. Gannon, S. Krishnan, A. Slominski, G. Kandaswamy, L. Fang, Building Applications from a Web Service based Component Architecture, *Component Models and Systems for Grid Applications*, V. Getov and T. Kiellmann (eds), Springer, 2005, pp. 3-17.

[10] Y. Huang, I. Taylor, D. W. Walker, Wrapping Legacy Codes for Grid-based Applications, *Procs. IPDPS'03*, IEEE Computer Press, 2003, pp. 139.

[11] M.S. Kamel, K.S. Ma and W.H. Enright, ODEXPERT - An Expert System to Select Numerical Solvers for Initial Value ODE Systems, *ACM Transactions on Mathematical Software (TOMS)*, vol. 19: 1, 1993, pp. 44– 62.

[12] S. Krishnan, B. Stearn, K. Bhatia, K. Baldridge, W. Li and P. Arzberger,, Opal: Simple Web Services Wrappers for Scientific Applications, *Procs. ICWS'06*, IEEE Computer Press, 2006, pp. 823-832.

[13] D. Kuebler, W. Eibach, Adapting Legacy Applications as Web services", IBM Developer Works, http://www-106.ibm.com/developerworks/webservices/library/ws-legacy/

[14] D. Petcu, M. Dragan, Designing an ODE Solving Environment, *Lectures Notes in Computational Science and Engineering* 10: *Advances in Software Tools for Scientific Computing*, H.P. Langtangen, A.M. Bruaset and E. Quak (eds.), Springer-Verlag, Berlin, 2000, pp. 319–338.

[15] D. Petcu, Experiments with an ODE Solver on a Multiprocessor System, *Computers & Mathematics with Applications* 42, no. 8-9, Pergamon-Elsevier Science, 2001, pp. 1189–1199.

[16] D. Petcu, A. Eckstein and C. Giurgiu, Using Statefull Web Services to Expose the Functionality of Legacy Software Codes, *Procs. SACCS 2007*, Iasi, 2007, pp. 257–263.

[17] D.Petcu, A. Eckstein, C. Giurgiu, Reengineering a Software System Implementing Parallel Methods for Differential Equations, *Procs. SEPADS'08*, 2008, pp. 95-100.

[18] K. Pingali, P. Stodghill, A distributed System based on Web Services for Computational Science Simulations, *Procs. of the 20th International Conference on Supercomputing*, 2006, pp. 297–306.