CONCURRENCY—PRACTICE AND EXPERIENCE
Concurrency: Pract. Exper. 2000; **00**:1–7 *Prepared using* cpeauth.cls *[Version: 2001/03/05 v2.01]*

The Integrated Simulation Environment TENT

Andreas Schreiber[1,2,*,†]

[1] *Deutsches Zentrum für Luft- und Raumfahrt e.V., Simulation and Software Technology, Linder Höhe, 51147 Cologne, Germany*
[2] *Argonne National Laboratory, Mathematics and Computer Science Division, Argonne, IL 60439, U.S.A.*

SUMMARY

This paper describes recent development efforts on the integrated simulation environment TENT. TENT is a component-based software integration and workflow management system using the capabilities of CORBA and Java. It is used to integrate the applications required to form complex workflows, which are typical of multidisciplinary simulations in engineering, in which different simulation codes have to be coupled.

We present here our work in integrating TENT with the Globus Toolkit to create a Grid computing environment. The Java Commodity Grid Toolkit has been especially useful for this work. Copyright © 2001 John Wiley & Sons, Ltd.

KEY WORDS: Grid computing; CORBA; Component-based software; Problem Solving Environment

1. INTRODUCTION

Many of today's engineering applications require the numerical simulation of the underlying physical processes. For example, fluid mechanics, structural mechanics, thermodynamics, and their coupling. Performing a complex simulation is the travers of a multistage process consisting of the preprocessing of the different simulation codes, the simulations themselves and their coupling, and the appropriate postprocessing. Often the performance requirements for such a complex simulation can be met only by exploiting distributed computing resources. Managing these distributed resources compounds the complexity of handling the different elements of the simulation. This increases the time for performing simulations and forms an upper bound for the complexity of a simulation being manageable.

*Correspondence to: A. Schreiber, DLR, SISTEC, Linder Hoehe, 51147 Cologne, Germany
†E-mail: Andreas.Schreiber@dlr.de

To improve the building and managing of process chains for complex simulations, we have developed the distributed integration and simulation environment TENT. The most important design goals of TENT are as follows:

- Application of component technology in distributed computing.
- Visual composition of process chains (workflows).
- Flexible configuration, online steering, and visualization.
- Easy management of user projects.
- Easy access to interactive simulations from any computer in the net.
- Simple integrability of existing applications and their supporting tools.
- Efficient data exchange between the stages in the process chain.

TENT has been designed with a CORBA [1] based distributed component architecture. Java is used as the implementation language for all essential parts of the system, and C++ is used for wrapping applications to components.

The first version of TENT used proprietary techniques for resource and information management, job submission, and data transfer. Currently this basic computing infrastructure is being replaced with Globus [2, 3] services, which allow one to use TENT as an environment for working in computational Grids [4].

2. ARCHITECTURE

The idea of TENT is to establish a framework for the integration of applications, forming typical engineering workflows, and allowing one to concentrate on solving simulation problems rather than struggling with technical details such as starting applications or passing data through the various stages of the process chain. TENT requires the application developer simply to compose a customized process chain from elementary building blocks and to run the simulation.

The key to an efficient implementation of a distributed integration framework is component technology [5]. For TENT, we have defined a component architecture that is specified using CORBA IDL. All programs that conform to our component architecture are called *TENT components*. Because of their defined interfaces, they can be connected, allowing them to work together in distributed computational environments to form complex workflows.

The TENT system consists of several packages, each containing related components and tools. Figure 1 shows an overview of the existing TENT packages.

- The *Base System* covers all basic functionality of the system needed to use it as an integration environment. This includes a basic component for controlling workflows, factories for starting components and applications in the distributed environment, a name server for managing the names and addresses of all TENT components, and a graphical user interface
- *Facilities* are components that can be used as services by application components but that are not part of the basic functionality. Examples of TENT facilities are a data converter for seamlessly converting data between two components, and a data server for storing data files in databases.

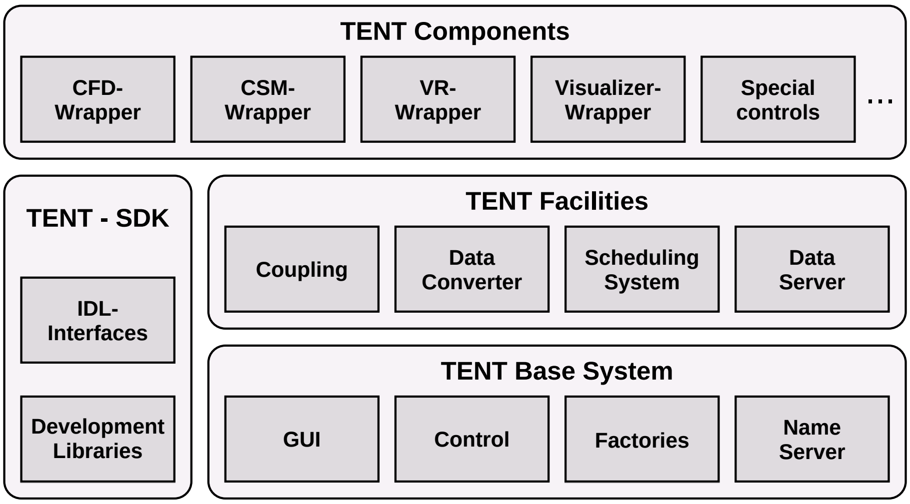

Figure 1. Packages of the TENT system

- The *Components* package contains wrappers for integrated applications. Currently there are wrappers for simulation applications such as the scientific CFD codes TRACE [6] and FLOWer [7], as well as commercial CSM codes such as NASTRAN [8] and ANSYS [9] or visualization tools such as Tecplot [10], AVS [11], and Gnuplot. This package also contains specialized control components.
- The *Software Development Kit (SDK)* is necessary for developing new TENT components. It contains development libraries for C++ and Java, all IDL interfaces, and JavaBeans [12] for the development of customized GUIs.

In the following sections, we explain the TENT system architecture and the system components in more detail. We also describe of which parts of the system have been changed or extended to use Grid services.

2.1. System Architecture

The TENT system consists of application components, system components, and some basic tools. A component is a piece of software with a clearly defined interface. In TENT, this interface is defined by means of a distributed component model which has definitions for the base interface, the property model, the event model, the life cycle, and persistence. To actually build computational workflows in TENT, one selects needed application components from a repository and connects them to proper process chains.

Two different types of *application components* exist. The first type, an *application wrapper*, represents a specific single application by providing a CORBA interface that conforms to the

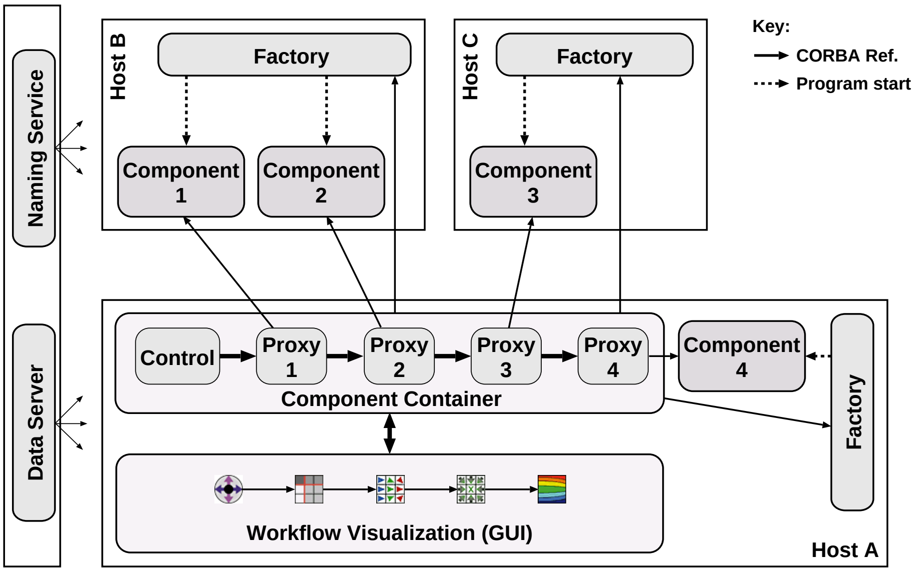

Figure 2. Architectural overview.

TENT component model. The wrapper interface has to provide all necessary functionalities which is needed to work with the particular application. The second component type is a *container* in which other components are grouped and connected. The container's interface includes management functionalities, such as adding and removing components or connecting components by events or direct references, which are used, for example, from a GUI to create workflows. The container implementation usually includes the necessary infrastructure (e.g., an event channel).

Beside the application components, several TENT *system components* and tools are needed to work with the system. A necessary tool is the naming service. This is basically an ordinary CORBA naming service at which every component registers after startup to provide its CORBA reference to the rest of the system. Currently, the naming service is the central information service of TENT. Another tool, the graphical user interface, allows users to create and manage workflows graphically, to manipulate properties of application and system components, to control the simulation, and to view status information of a running system.

The startup of components and applications and their process control in a distributed environment is done by *factorie*. At least one factory must run on every system on which TENT components or their application has to be started. The configuration of a factory defines what component it is able to start. The configurations of all factories (i.e., the information about all startable components) is also provided to a component repository, which is used to select available components for creating workflows.

Prepared using cpeauth.cls

Furthermore, workflows can contain auxiliary components, called TENT *facilities*, that can be used by application components. For example, a data converter can be integrated into the data stream between two components, or a data server can be used for persistent storage of simulation data.

The connection of components is generally done by using events, which are either statically specified in the components IDL interface or dynamically generated for sending them through an event channel. It is also possible to perform direct CORBA method calls between components, a useful strategy for using other components as a service or for integrating special control components such as a script control component.

2.2. System Components and Tools

This section describes the essential system components and tools, as well as their extensions for the Globus integration. All of them are implemented in Java, which makes them platform independent.

2.2.1. Control Components

Basic control component. The basic control component is a simple event sender component that can be connected to certain components in the workflow. Like any other TENT component, it has its own GUI representation. In this case some buttons represent the event actions and input fields for entering event parameters. The event sender component is currently used to start the whole workflow by connecting it to the actual starting point of the workflow. This event control is usually automatically instantiated in the root container of every workflow.

Special control components. For specialized tasks, more sophisticated control components exist. Depending on their task, they are connected to one or more other components, controlling them via events, direct CORBA method calls, or property modifications. They usually also provide other components with certain input data. Examples are control components for doing parameter studies, optimizations, or trajectory computations. Some of these examples are complex, with sophisticated user interfaces and management functions.

Script control. To build more complex process chains, a script control mechanism is integrated, for example, for creating loops or creating flexible reaction on different states of application components. For this, the modern object-oriented scripting language Python [13, 14] is used. The Python control component can be included into workflows like any other component. Its essential property is a Python script that is started at component startup and that reacts properly to incoming events, sends events, and evaluates properties of other components. The script control component is implemented in Java and uses the JPython implementation as its script interpreter.

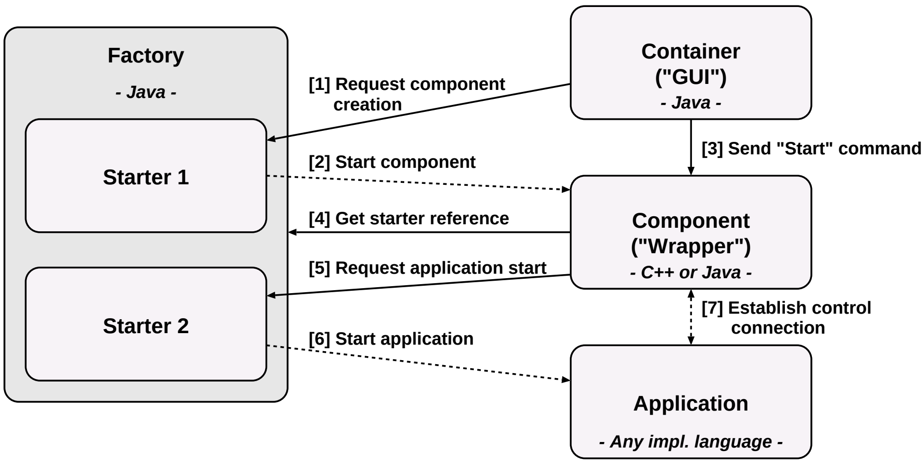

Figure 3. Starting components and application with factories.

2.2.2. Factories

The TENT factory component provides a highlevel interface for creating new stand-alone components and for starting applications out of already running components. Specific *Starter* objects in the factory actually do the process management.

If a client wants to start a program, it requests the factory for a reference to a proper starter, which will then be instantiated by the factory and can be used by the client. This realization allows one to use existing starter implementations from clients written in any language. Currently the factory and the starters are entirely written in Java and are used from Java clients, like the GUI or control processes, and from C++ clients, like the wrapper components (Figure 3).

The most trivial starter does a simple program execution using the Java *Runtime* class. More complex examples of starters are

- an *MPIStarter*, which is able to start parallelized programs and which abstracts from different MPI implementations;
- a starter for submitting jobs to batch and scheduling systems like LSF [15], GridEngine [16], or NQS;
- Grid starter, for submitting applications into computational Grid's built with Globus [2] or Legion [17]; and
- a *JavaInstantiationStarter*, which instantiates components or applications written in Java within the virtual machine of the factory in place of starting them as external processes.

The interfaces of these special starters have specific properties and methods to provide other parts of the TENT system the full functionality of the underlying system, for example, for

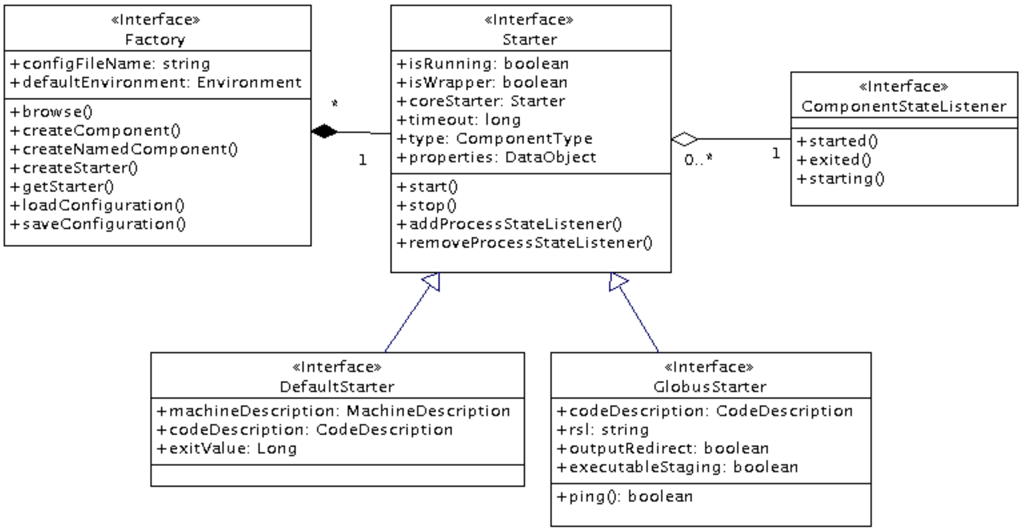

Figure 4. Interface relation between factory and starter (only a part of the interface hierarchy is shown).

getting status information or for accessing special job management features. Figure 4 shows a part of the entire interface hierarchy.

For each starter implementation a related GUI interface exists. This interface is integrated into the TENT GUI, and provides an easy way for setting specific parameters before the start and allows one to monitor the running jobs conveniently.

To work in Grid environments based on the Globus Toolkit, we developed the *GlobusStarter*. It schedules jobs on remote computing resources using the Globus Resource Allocation Manager (GRAM) service.

The implementation uses the Java Commodity Grid Toolkit [18], which allows an easy integration of all needed Grid services. In particular, the GlobusStarter does the following tasks:

1. It initializes the Grid proxy: it loads the user's certificate, asks for the user's Grid pass phrase, loads the CA certificate, and creates the proxy.
2. It creates a GRAM RSL based on all available information about the job: the name and location of the executable, the arguments, the environment, URLs for redirecting standard output and standard error, and futher job requirements such as the number of required processors or the required memory.
3. It starts the job using the GRAM remote job submission. If necessary, it stages the executable to the remote machine before commencing. The implementation uses the Java CoG GRAM client (package *org.globus.gram*).

Prepared using cpeauth.cls

4. The running job is monitored by registering itself as a *GramJobListener*. The status of the running job is displayed in the starter's GUI panel.
5. Optionally, the GlobusStarter fetches the output (stdout and stderr) of the running job and displays it in its GUI panel. This feature is realized by instantiating a Java GASS server.
6. It kills the component or application if the work is finished.

2.2.3. Graphical User Interface

The TENT GUI allows the visual setup of workflows in a block-oriented way, the manipulation of properties or parameters of components, and the control and online steering of simulations. The main parts of the GUI are as follows:

- The component repository for selecting components.
- The wire panel for connecting (or wiring) components.
- The property panel for setting up component parameters.
- The control panel for starting, holding, and stopping simulations.

Each part of the GUI is implemented as an independent JavaBean component. To create a complete, customized GUI, these JavaBeans can be put together using a rapid application development (RAD) tool such as JBuilder or VisualAge.

In addition to the main GUI panels, many other panels exist, for example, a Python interpreter console, a plot panel for 2D plots, and a panel for selecting certain parameters in application input files. Also, components can have their own specialized control panel, which is inserted when the component is started.

A useful feature in distributed environments is the *Remote File Selector* for choosing files on remote machines. It is essentially a regular Java file selection dialog but with a special *FileSystemView*, which queries a CORBA interface (*TentFileNavigator*) that all TENT components implement. In the future this technique will be extended to browse file systems and servers that are accessible by Globus GSI-FTP.

For the Globus integration we developed some basic, low-level GUI panels were developed (see Figures 5 and 6):

- An MDS server selection panel,
- an MDS search table,
- a panel for editing job parameters, and
- a panel for monitoring job status and job output.

Actually, the MDS search table was inspired by the existing search table of the Java CoG Kit (see [18]).

All these panels are JavaBean components and can be used independently from TENT. For example, new graphical Grid applications can be designed using these JavaBeans and any RAD tool.

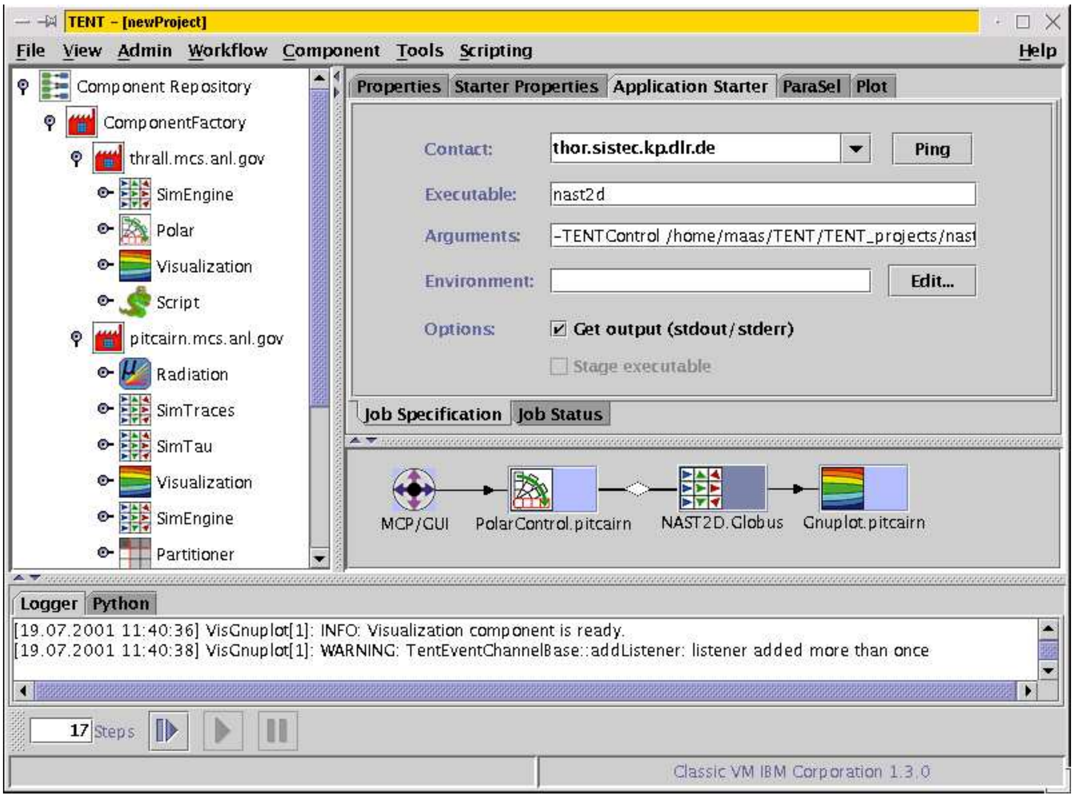

Figure 5. GUI showing the Globus job specification panel.

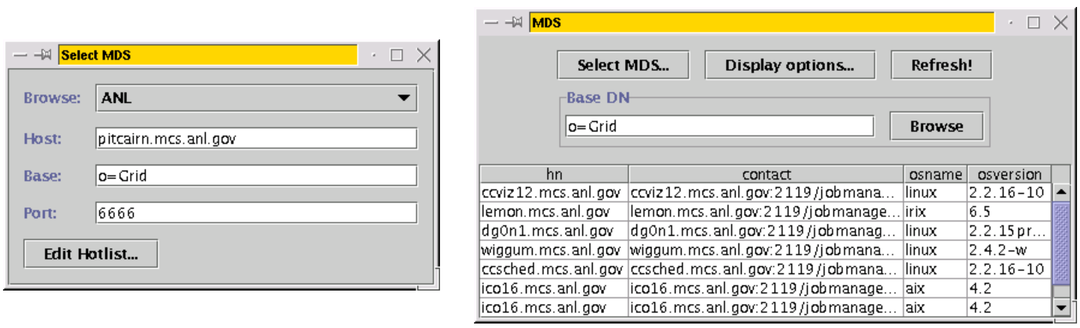

Figure 6. MDS server selection and MDS search table.

2.2.4. *Logger*

The Logger is a system component that logs messages from all TENT components. It can filter different types of messages such as errors, warnings, or informational messages. The Logger is available in two implementations, C++ implementation and Java, the latter being a JavaBean and usually part of the GUI.

2.2.5. *Coupling*

A special feature is the support for coupled simulations, realized by using the MpCCI library [19], a coupling library on top of MPI. The two or more coupled simulation codes usually have to be started simultaneously, a process currently done by a special control component. In the future this might be done by using the MPICH-G2 [20] implementation of MPI to take advantage of the Globus resource management features and especially of the General-purpose Architecture for Reservation and Allocation (GARA) to guarantee the simultaneous job launch.

2.3. Data Transfer

Up to now data transfer between two components is done by the TENT data exchange object. This object is the implementation of a CORBA interface (*TentDataExchange*) which is actually a management interface for the data transfer. It provides methods for registering data sources such as files or memory blocks and methods for querying the registered data as to certain data types, and it allows clients to initiate the data transfer on request. The data exchange object can be configured to use file exchange, ftp, sockets, CORBA, or a parallel data transfer protocol for transfering the data.

If an application component (wrapper) and its application are running on different hosts that do not share a common file system, the data files for the application are transfered by URL copy using the cURL library [21].

In the Globus version of TENT, the data transfer between components and applications is done using Globus services. TENT uses the Globus services Global Access to Secondary Storage (GASS) and GSI-FTP to transfer data between the different distributed hosts.

The data transfer between two running components or between a component and its wrapped application is realized using GASS. For the two different cases the following data has to be transferred (see Figure 7):

1. Between two components in a workflow, the transferred data contains all output files from the wrapped application of the first component that are needed by the application of the second component. The transfer of this data is done on demand by the second component.
2. Between a component and its wrapped application all neccessary input and output files are transferred. These include parameter or steering files for the application, the input data files from previous stages in the workflow, and output files needed by following stages.

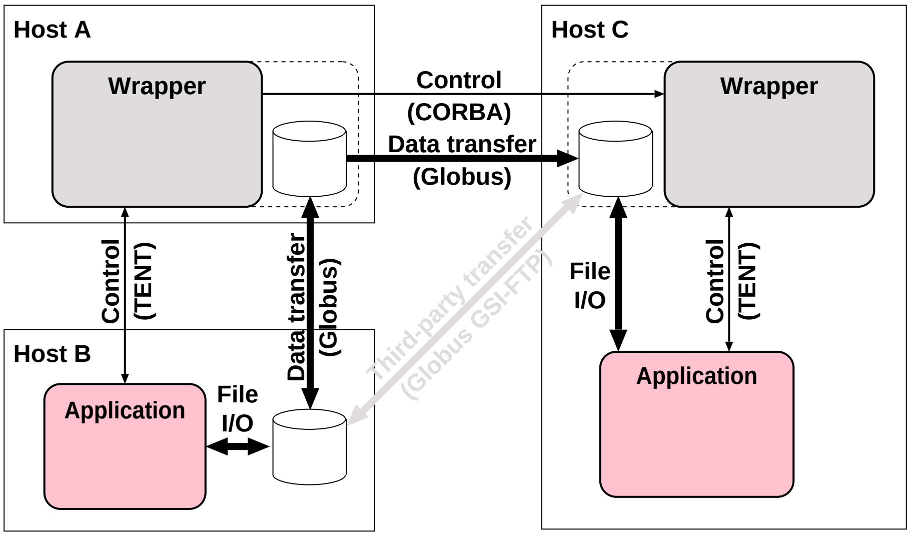

Figure 7. Use of Globus for data transfer.

> This applies if the component and its application are running on different hosts whithout a shared file system. To avoid unneccessary transfers of files, only those files that were changed or created by the application are transferred back to the wrapper component.

The implementation of this file transfer is done using the C libraries of the Globus Toolkit, because the wrapper components are implemented in C++.

In the future, GSI-FTP will also be used, especially for controlling third-party transfers of certain files directly between two applications to avoid the detour through the file systems of the wrapper components.

2.4. Security

Security is based on the Grid Security Infrastructure (GSI) from Globus. The TENT factories and the GUI, in particular, are GSI clients and use the *org.globus.security* package of the Java CoG Kit for authentification to acquire access to GSI-enabled resources.

Future developments will include support for multiple user credentials and the delegation of credentials to MyProxy [22] server.

2.5. Information services

In the current TENT design, a naming service is used to provide information about running components and about components that can be started by running factories. The information is used in clients, such as the component repository panel of the GUI. The component repository queries the naming service, and if a naming service entry is a factory, the component repository browses that factory for its startable components and adds this information to the repository tree.

The naming service of TENT is actually a regular CORBA naming service, conforming to the OMG's specification. But because the CORBA naming service specification does not provide any functionality to notify other CORBA objects about changes, we added a notification event to the naming service interface. This is important for an efficient implementation of graphical user interfaces, because otherwise every naming service browser has to pull the contents of the naming service.

For querying information about the Grid, the Metacomputing Directory Service (MDS) is used. To get information from the MDS, the Java client classes (*org.globus.mds*) are used to connect to an MDS server, to query the server, to retrieve the results, and to disconnect from the server. The results are presented in a GUI panel (Figure 6), which allows the user to select the desired resource.

3. IMPLEMENTATION

The following is a survey of the software technologies used for implementing TENT. Some of these technologies were already mentioned in the preceding section, but here some more details are provided.

Java. For many well-known reasons (see, e.g., [18]), Java is used as the implementation language for all essential parts of the TENT system. Almost all subpanels of the TENT GUI and all provided Globus panels are JavaBean components. The required Java version is Java 2 SDK, v 1.3.

Python. The powerful object-oriented language Python is used as an integrated scripting language. A Python console is integrated in the TENT GUI and can be used for general scripting tasks (e.g. for creating, starting, and controlling workflows), as an alternative to the graphical methods. A control component with an embedded Python interpreter can be integrated into workflows for more specific tasks. In both cases, JPython, a Java implementation of the Python interpreter, is used.

CORBA. TENT uses CORBA as middleware for controlling workflows and IDL for specifying all interfaces. The C++ components of TENT can be compiled and used with either ORBacus [23], omniORB [24], or VisiBroker [25]. The Java parts currently use the ORB implementation shipped with the JDK 1.3, but in the future either the ORBacus Java ORB or the VisiBroker Java ORB will be used.

XML. TENT uses XML for the syntax of configuration files and the main project file. The implementation uses Java APIs for XML processing (JAXP) [26] from SUN.

Servlets. A Web portal version of the TENT user interface using Java servlets is currently under development. This is being done with the Jakarta Tomcat [27] server.

Database. The distributed object database management system (ODBMS) Objectivity [28] is integrated as the database back-end for the data server of TENT.

Prepared using cpeauth.cls

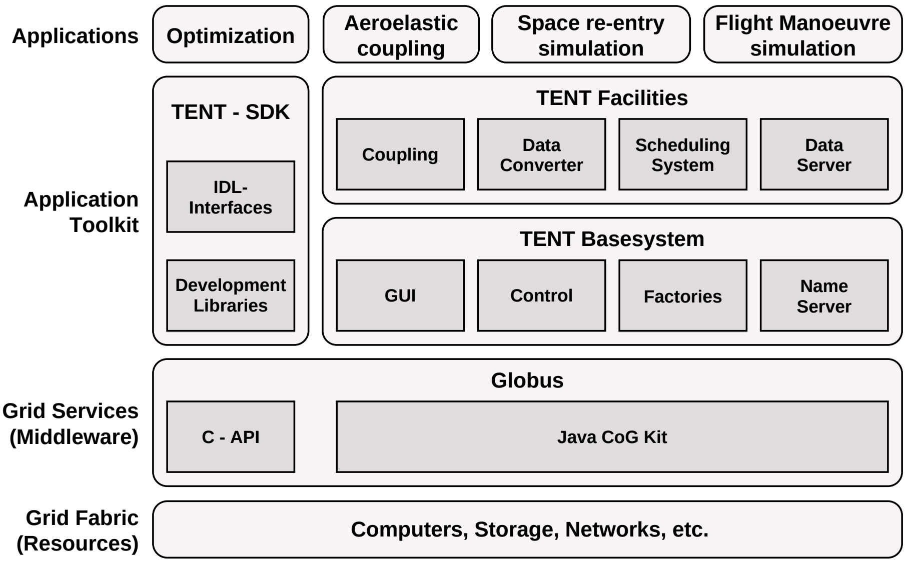

Figure 8. The main layers of the TENT/Globus system.

Globus. TENT uses the Java CoG Kit implementation of the Globus Grid services as well as the Globus C implementation (see Figure 8).

4. APPLICATIONS

The TENT system is used in several research projects and in productive industrial environments for integration of engineering workflows:

- Originally TENT was developed in the project SUPEA [29] for integrating simulation codes to build a numerical testbed for gas turbines.
- In the project AMANDA [30], TENT is used to integrate all software in order to perform a multidisciplinary aeroelastic simulation of a trimmed, freely flying, elastic airplane and to simulate of the airflow around a turbine blade, including the internal heat flow inside the blade.
- TENT is used in the project IMENS which targets the optimization of the heat protection system for the reentry phase of a reusable space transportation system. That includes the integration of several multidisciplinary fluid-structure coupling workflows.
- The AUTOBENCH [31] project uses TENT as an integrated development environment for virtual automotive prototypes. The final environment will integrate several CAE tools as well as virtual reality tools simulation (e.g., crash simulation) of new car bodies.

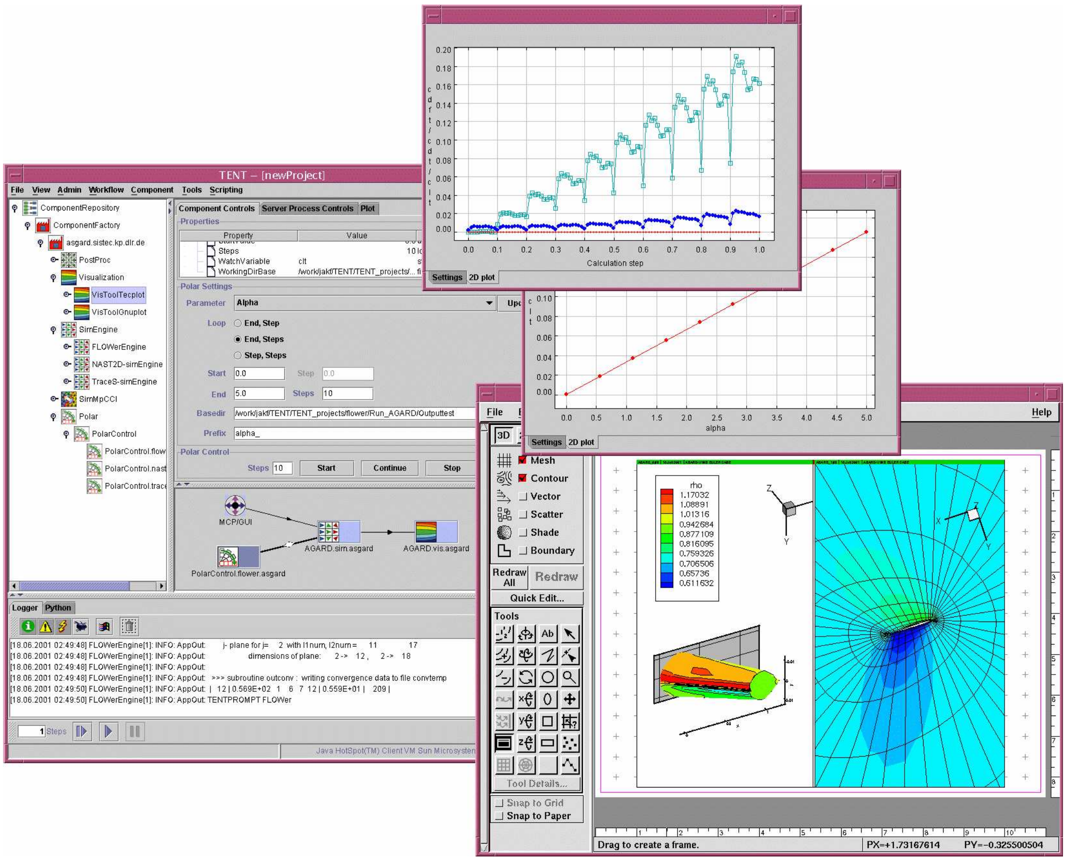

Figure 9. Application: Computation of polar for civil aircraft.

- A system based on TENT for the numerical simulation of manoevring combat aircrafts is developed in the AeroSUM project.
- Currently TENT is used at EADS Airbus for the aerodynamic simulation of wing-fuselage configurations for civil aircrafts, especially for parameter studies such as polar computations (airfoil lift as a function of the angle of attack; see Figure 9 for an example screenshot).

Prepared using cpeauth.cls

5. CURRENT AND FUTURE WORK

Feedback from applications using the TENT system has suggested need for improvements and further development. In particular, advancements need to be made within the following areas:

- A more sophisticated *data management* system needs to be developed. This includes the storage of configuration and simulation data in databases and product data management systems, as well as the integration of results from the DataGrid [32] project.
- The *user management* and *component access control* systems need to be redesigned to allow different users to work together on the same workflow in collaborative way.
- Up to now, TENT is a Grid computing *application*, but additionaly a *portal* version of TENT is currently under development. That will allow one to work with workflows through a Web interface, at least at a basic level.

The deployment of the Grid version of TENT and the underlying Globus infrastructure at industrial and scientific sites is a challenging and important task. Initially both TENT versions will be supported because not all project partners and TENT users have Grid infrastructure installed on their systems.

6. CONCLUSIONS

TENT started as a research project for exploring the usefulness of component software for distributed high-performance computing. It has evolved into a simulation and software integration environment that now can be used for production simulations. Feedback from several scientific and industrial applications has led to significant improvement of the original TENT design [33].

The integration with Globus results in a Grid computing environment that is applicable for scientific and industrial applications. The availability of the Java Commodity Grid Toolkit was a great benefit for doing this integration with the Java parts of TENT.

ACKNOWLEDGEMENTS

This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-ENG-38.

REFERENCES

1. OMG. *CORBA Architecture and Specification.* OMG, 1998.
2. Foster I and Kesselman C. *Globus: A Metacomputing Infrastructure Toolkit.* Intl J. Supercomputer Applications, 11(2):115-128, 1997.
3. Globus Homepage. http://www.globus.org.
4. Foster I and Kesselman C (eds.). *The Grid: Blueprint of a New Computing Infrastructure.* Morgan Kaufmann Publishers, Inc., San Francisco, 1998.

5. Szyperski C. *Component Software - Beyond Object-Oriented Programming.* Addison-Wesley/ACM Press, 1998.
6. Vogel D T and Kuegeler E. *The generation of artificial counter rotating vortices and the application for fan-shaped film-cooling holes.* In Proceedings of the 14th ISABE, ISABE-Paper 99-7144, 1999.
7. Aumann P, Barnewitz H, Schwarten H, Becker K, Heinrich R, Roll B, Galle M, Kroll N, Gerhold T, Schwamborn D, and Franke M. *MEGAFLOW: Parallel complete aircraft CFD.* Parallel Computing 27(4): 415-440, 2001.
8. NASTRAN. http://www.macsch.com.
9. ANSYS. http://www.ansys.com.
10. Tecplot. http://www.tecplot.com.
11. AVS. http://www.avs.com.
12. Hamilton G (ed.). *JavaBeans API specification.* Sun Microsystems, 1997, http://java.sun.com/beans.
13. Python. http://www.python.org.
14. Dubois P F. *Ten good practices in scientific programming.* Computing in Science & Engineering, Jan/Feb 1999, pages 7-11.
15. LSF. http://www.platform.com.
16. GridEngine. http://www.sun.com/gridware.
17. Grimshaw A, Wulf W, and the Legion team. *The Legion vision of a worldwide virtual computer.* Communications of ACM, 40(1):39-45, 1997.
18. von Laszewski G, Foster I, Gawor J, and Lane P. *A Java Commodity Grid Kit.* Concurrency and Computation: Practice and Experience, pages 643-662, Volume 13, Issue 8-9, 2001.
19. Ahrem R, Post P, and Wolf K. *A Communication Library to Couple Simulation Codes on Distributed Systems for Multi-Physics Computations.* In ParCo99 Conference Proceedings, pages 47-55. Imperial College Press, London, 2000.
20. MPICH-G2. http://www.hpclab.niu.edu/mpi.
21. cURL Homepage. http://curl.haxx.se/.
22. MyProxy Homepage. http://dast.nlanr.net/Features/MyProxy/.
23. ORBacus Homepage. http://www.ooc.com/ob/.
24. omniORB Homepage. http://www.uk.research.att.com/omniORB/.
25. VisiBroker Hompage. http://www.inprise.com/visibroker/.
26. Java APIs for XML Processing (JAXP). http://java.sun.com/xml/xml_jaxp.html.
27. Jakarta Tomcat. http://jakarta.apache.org/tomcat/.
28. Objectivity Homepage. http://www.objectivity.com.
29. SUPEA Homepage. http://www.sistec.dlr.de/en/projects/supea/.
30. Kersken H-P, Schreiber A, Strietzel M, Faden M, Ahrem R, Post P, Wolf K, Beckert A, Gerhold T, Heinrich R, and Kuegeler E. *AMANDA - A Distributed System for Aircraft Design.* Proceedings of EuroPar 2000, LNCS 1900, pages 1315-1322.
31. Thole C-A, Kolibal A, and Wolf K. *AUTOBENCH – Virtual Prototyping for Automotive Industry.* Proceedings of the 16th IMACS World Congress, 2000.
32. DataGrid. http://www.globus.org/datagrid.
33. Breitfeld T, Kolibal S, Schreiber A, and Wagner M. *Java for Controlling and Configuring a Distributed Turbine Simulation System.* Workshop Java for High Performance Network Computing, EuroPar 1998, http://www.cs.cf.ac.uk/hpjworkshop/.