Developing, Simulating and Managing Distributed COM Applications with XJ DOME

Andrei V. Borshchev, Alex E. Filippoff and Yuri G. Karpov
Experimental Object Technologies (XJ) & St.Petersburg Technical University, Russia
dome@xjtek.com   http://www.xjtek.com

Abstract

XJ DOME is a set of tools and technologies for those who wish to speed up development of distributed COM applications and improve their quality and manageability. DOME supports several development phases (graphical modeling, code generation, simulation and deployment), as well as run-time (monitoring and management). DOME is tuned to work with MS Visual C++.

1 Introduction

While distributed computing platforms such as CORBA and DCOM are now used industry-wise, there still are no corresponding high-level support tools. Most commonly used development environments (e.g. Microsoft Visual Studio) provide only low-level support for distributed programming leaving a lot of routine work to be done manually.

The main idea behind XJ DOME project is to let the developer concentrate on application-specific design by automating tasks common for most distributed COM applications. XJ DOME enables the developer to:

- Rapidly create and launch distributed application prototypes using visual design environment with complete code generation
- Debug timing and synchronization and estimate performance of a distributed application by simulating it in virtual time on the developer's machine
- Deploy the application components over the target network
- Monitor the distributed application: collect and display statistics, inspect threads and synchronization objects, view logs, etc.
- Manage the distributed application via COM interfaces using standard and custom controls

The main DOME components are *Engine* and *Application Viewer*, see Figure 1. DOME Engine is responsible for simulation and deployment, while DOME Application Viewer enables the user to monitor and manage the distributed application. Engine and Viewer communicate with the application using pure COM.

Actually, DOME does not care how exactly was the application developed as long as its components support `IDomeObject` interface. However, DOME offers specific support for MS Visual C++. This includes *DOME Application Editor* and a set of *DOME Wizards*. Those who want to rapidly set up COM project and perform first development cycles with minimal efforts would use graphical DOME Application Editor, whereas advanced users of MS Visual C++ (especially those who are in the middle of the development) can use DOME Wizards.

2 DOME Application Editor

For rapid prototyping of distributed COM applications XJ DOME offers *Application Editor*, see Figure 2. It includes graphical COM Object Diagram and Statecharts editors and form-based COM Interface Designer. DOME Application Editor generates the complete application code including IDL, C++, resources, and MS Visual C++ project. It builds the application components using MS Visual C++ command-line compiler and starts DOME Engine and Application Viewer. DOME Application Editor drastically saves developer's time on the early design stages. Later on, when "COM skeleton" of the application becomes more stable, the developer can continue with MS Visual C++ environment using built-in DOME Wizards.

3 Wizards for MS Visual C++

DOME can be used with new projects as well as with the legacy software. In the latter case DOME functionality can be added to the application gradually.

Proceedings of the Workshop on Computer Science and Information Technologies CSIT'99
Moscow, Russia, 1999

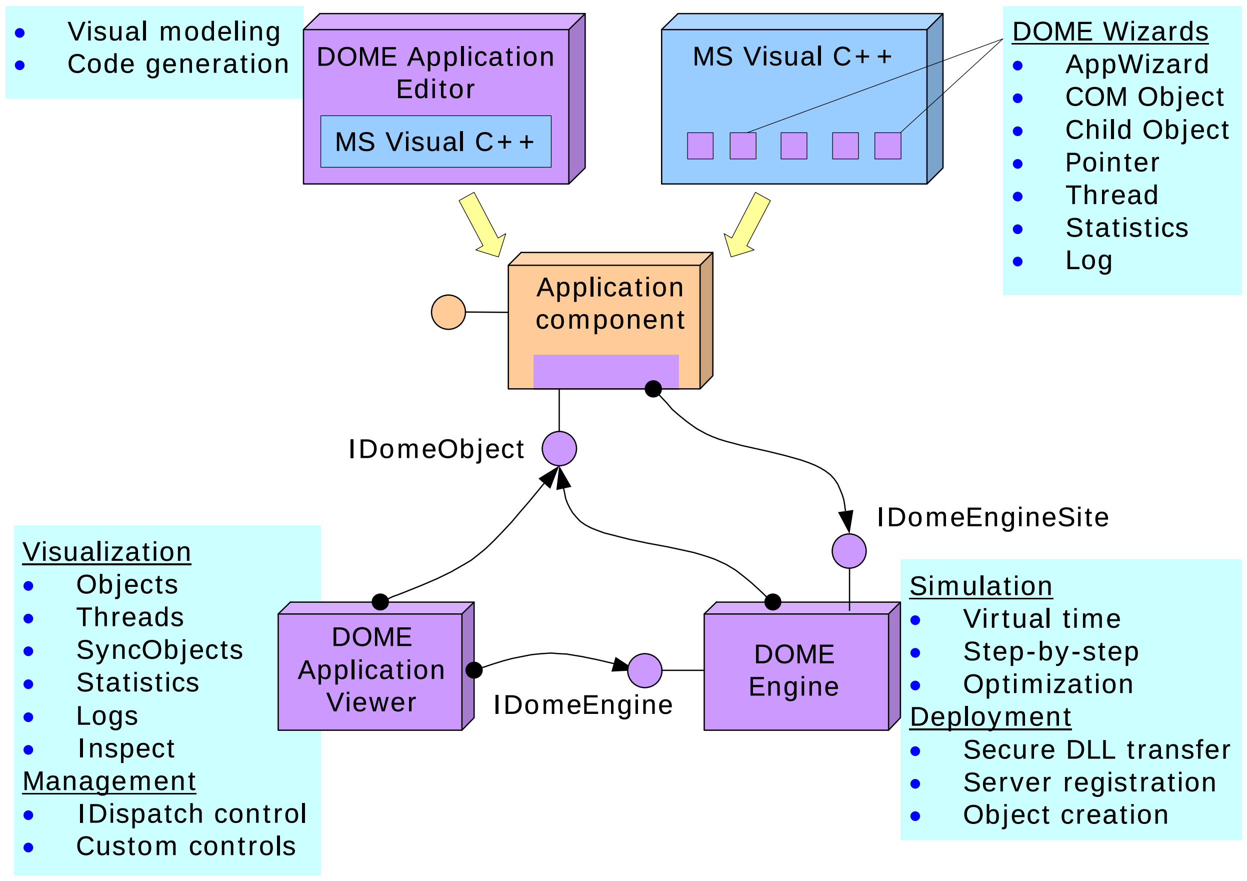


Figure 1 XJ DOME components and their functionality

By following a set of simple rules one can easily tune COM objects under development to use DOME simulation, deployment, visualization and management facilities. To automate this work DOME adds to MS Visual C++ development environment a set of *wizards* covering all related tasks:

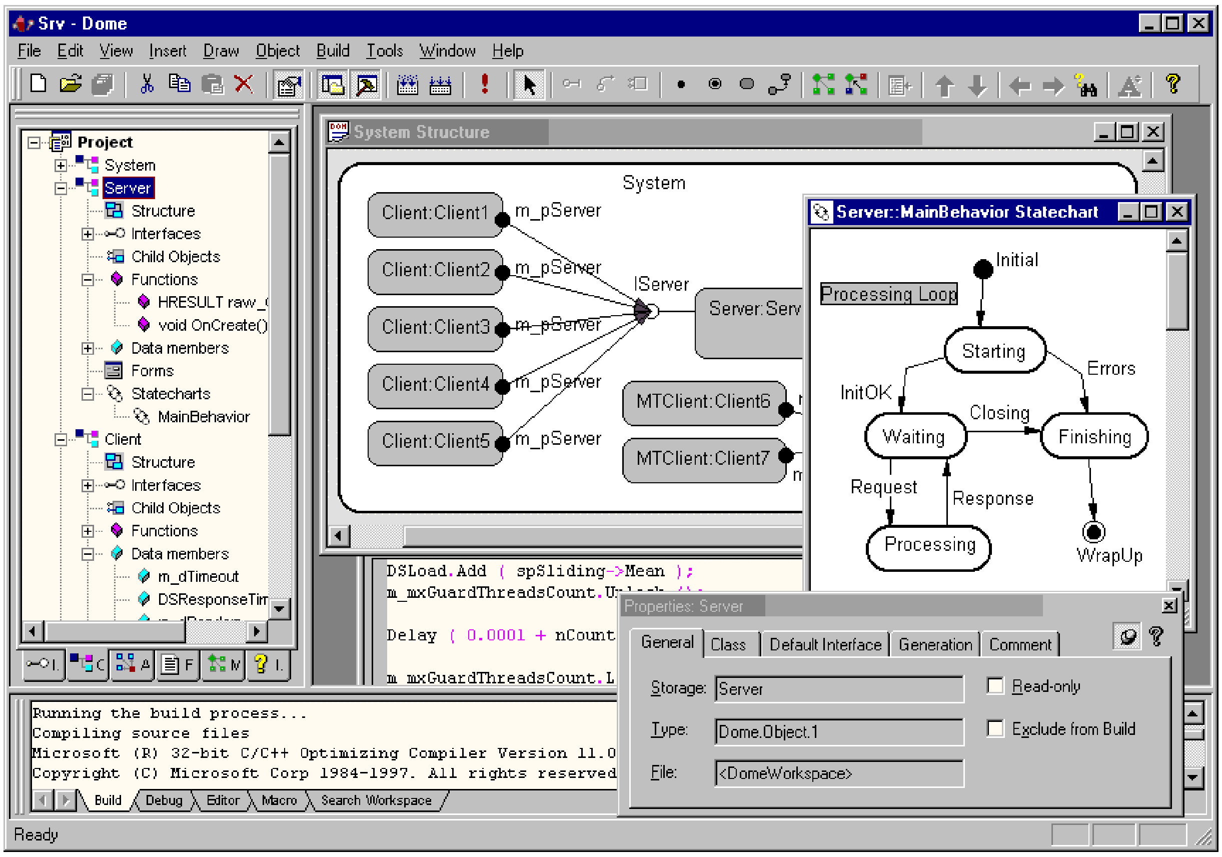
- DOME App Wizard
- Add DOME Object
- Add Child Object
- Add COM pointer
- Add DOME Thread
- Add DOME Statistics
- Add DOME Log

4 Simulation

Simulation is used for preliminary analysis of the application correctness and estimation of its performance. In the simulation mode the most detailed information on threads and synchronization objects is available online in DOME Application Viewer. The user can run the application step-by-step, stop upon a certain condition, e.g. when enough statistics is collected, etc. Using automation the developer can program DOME to run multiple simulation sessions to find optimal parameter values or to test the application scalability. And all that is performed on a single developer's machine.

DOME simulates the application not in real, but in virtual time, thus making arbitrary complex experiments possible on a single workstation. For example, if you provide the deployment information and characterize the target network, the simulated DCOM calls will be correspondingly delayed. Thus, DOME will advise you on the performance of the distributed application before it is actually deployed.

Simulation is supported by DOME Engine that implements `IDomeEngineSite` interface. The application objects developed according to DOME technology invoke the functions creating new objects, threads, synchronization objects, delaying thread execution, waiting for events, etc. through `IDomeEngineSite`. In the normal execution mode such call is immediately passed to the local operating system (in fact, the call does not even leave the local machine, as the engine is represented there by small DOME Engine Proxy object, see Figure 4). In the simulation mode, however, the engine takes care of thread scheduling, synchronization and time.

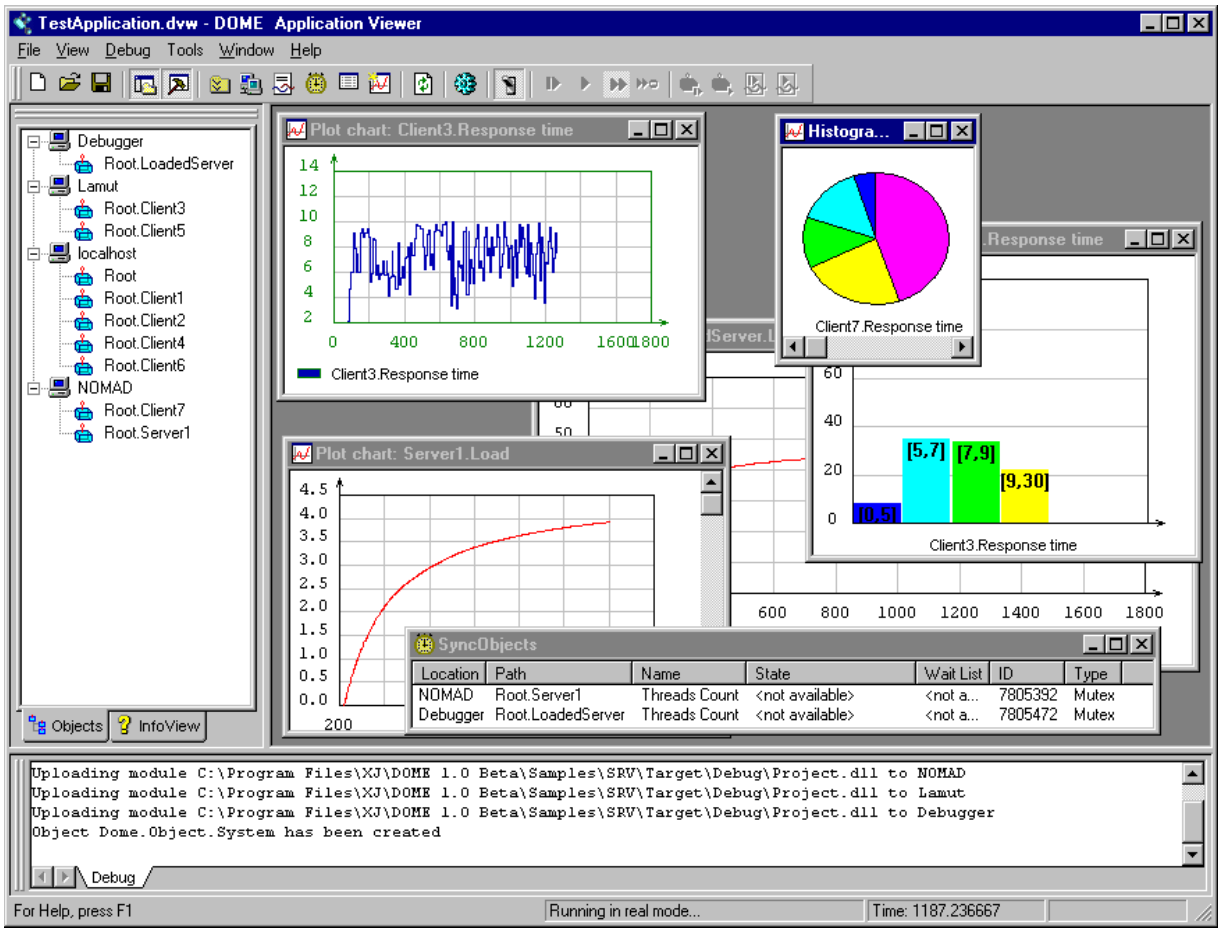


```
DSLoad.Add ( spSliding->Mean );
m_mxGuardThreadsCount.Unlock ();

Delay ( 0.0001 + nCount

m_mxGuardThreadsCount.L
```

```
Running the build process...
Compiling source files
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 11.0
Copyright (C) Microsoft Corp 1984-1997. All rights reserved
```

Storage: Server
Type: Dome.Object.1
File: <DomeWorkspace>
☐ Read-only
☐ Exclude from Build

Figure 2 XJ DOME Application Editor



Location	Path	Name	State	Wait List	ID	Type
NOMAD	Root.Server1	Threads Count	<not available>	<not a...	7805392	Mutex
Debugger	Root.LoadedServer	Threads Count	<not available>	<not a...	7805472	Mutex

```
Uploading module C:\Program Files\XJ\DOME 1.0 Beta\Samples\SRV\Target\Debug\Project.dll to NOMAD
Uploading module C:\Program Files\XJ\DOME 1.0 Beta\Samples\SRV\Target\Debug\Project.dll to Lamut
Uploading module C:\Program Files\XJ\DOME 1.0 Beta\Samples\SRV\Target\Debug\Project.dll to Debugger
Object Dome.Object.System has been created
```

Figure 3 XJ DOME Application Viewer

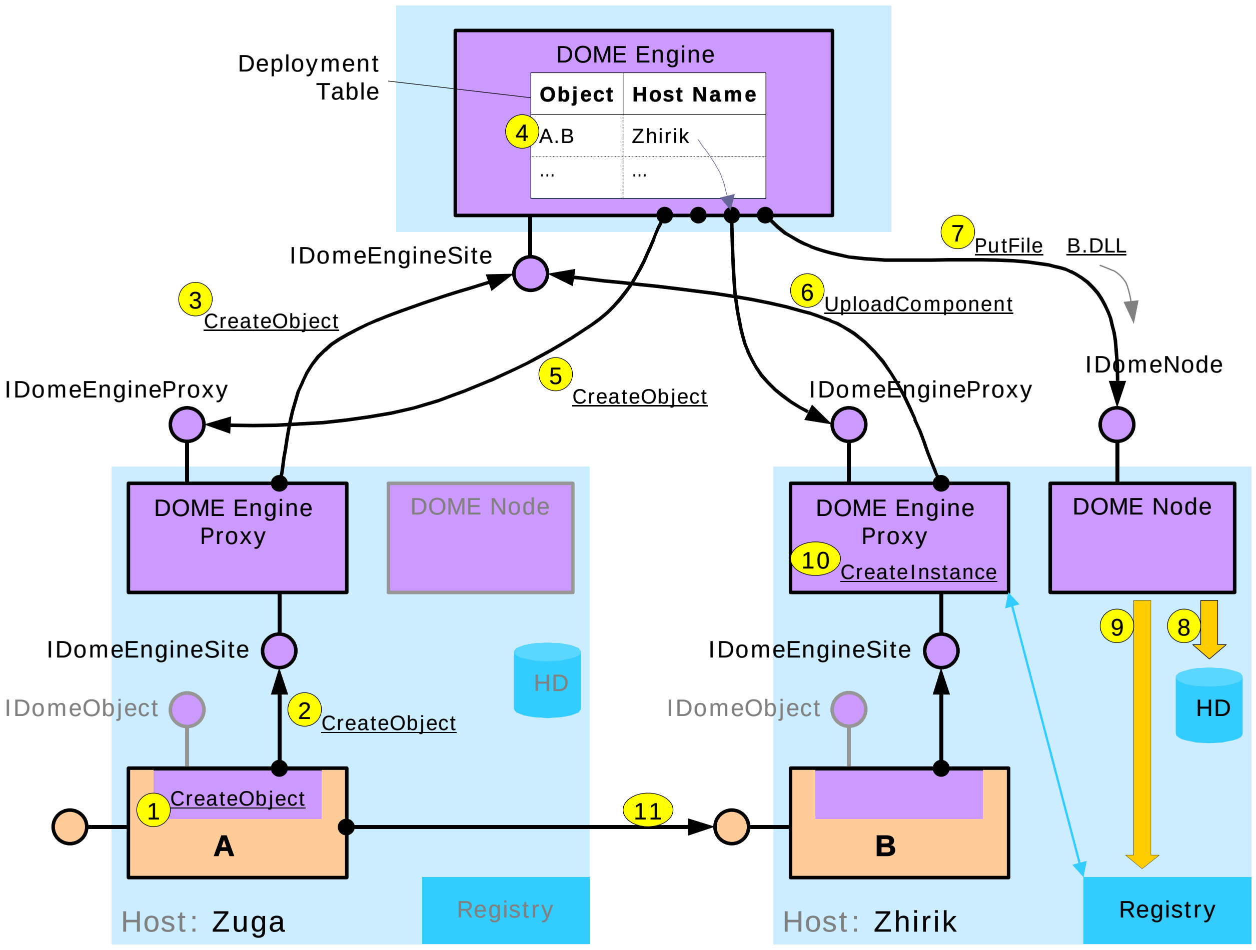


Figure 4 XJ DOME deployment protocol

5 Deployment

DOME automates deployment of the application components. Suppose the application object A running on machine Zuga wants to create object B. It simply calls DOME function `CreateObject` which returns the pointer to `IUnknown` interface of B. All other work including locating host for B, transferring the corresponding DLL file, registering it on the target machine and creating the object instance is done by DOME Engine as shown in Figure 4. The deployment table may be formed either directly from the user instructions or on the basis of a given algorithm, e.g. load balancing.

DOME Node runs on each machine where you wish to deploy the application components. It is responsible for uploading the component code and registering it locally. As long as the application can run across a non-secure network, e.g. the Internet, DOME uses DCOM security. A list of users that may upload, register and run DOME components can be specified for DOME Nodes.

6 Visualization

The user can monitor the running application with *DOME Application Viewer*, see Figure 3. The viewer retrieves the information via `IDomeObject` interfaces implemented by DOME-compatible application components and displays the global picture of the application, including:

- Application objects and their hierarchy
- Threads
- Synchronization objects
- Statistics
- Logs
- Inspection views
- IDispatch interfaces of objects

The particularity of the displayed information depends on the execution mode. Namely, in the simulation mode the user can watch the current states of the synchronization objects and threads, and wait queues, whereas in the normal mode these details are not available.

7 Management

The user can manage COM application with *DOME Application Viewer*. Before the application starts the user chooses the execution mode (simulation or normal), root objects and gives deployment instructions.

When the application is running, commands available in the viewer depend on the execution mode. In simulation mode the user has full control over the application execution. Since DOME Engine manages time and synchronization, the user can run the application step-by-step, stop, watch the activity of the selected object, etc. In the normal execution mode time and synchronization are managed by the operating system.

In any mode the user can change the properties of any application object with DOME IDispatch Browser. Moreover, the user can create custom controls using MS Forms technology and build them into DOME objects. These controls will be displayed in DOME Application Viewer that acts as the *application management console*.

8 Future work

We consider DOME as a framework technology that can be used as a basis for building distributed applications with predictable quality of service. At the moment we are working in the three directions:

- Incorporating generic model-based management facilities into DOME objects [3]
- Implementing general-purpose distributed algorithms in DOME objects, such as distributed snapshot, distributed termination, distributed deadlock detection
- Developing DOME object-compatible simulation models of communication media (networks and protocols) for better prediction of the application performance

References

1. D. Krieger and R.M. Adler. *The Emergence of Distributed Component Platforms*. IEEE Computer, March 1998, pp. 43-53.
2. Andrei V. Borshchev, Yuri G. Karpov and Victor V. Roudakov. *Systems Modeling, Simulation and Analysis Using COVERS Active Objects*. Proceedings of the IEEE International Conference and Workshop on Engineering of Computer Based Systems, March 24-28, Monterey, California, 1997.
3. Maxim Aleksandrov and Vlad Voinov. *Designing and Implementing QoS Management of the Web*. Proceedings of IBM CASCON'98.