OMG’s Workshop on Distributed Object Computing for Real-Time and Embedded Systems

July 14-17, 2003 - Arlington, VA

Performance Testing of the TENA Middleware

Gregory L. Schultz
Speaker Introduction

- Member of the Test & Training Enabling Architecture (TENA) Architecture Management Team (AMT)
- Project Lead for one of the FL2010 Test Cases
- Development member on FL2010 Test Team
Presentation Goals

- Briefly describe the TENA middleware
- Discuss results of performance testing
- Enumerate benefits of using the TENA middleware
What is TENA?

- Means “Test & Training Enabling Architecture”
- Defines a common architecture for the test and training range community
- Provides a common software framework called the TENA middleware
- Uses the concept of a Logical Range - “a suite of TENA resources, sharing a common object model, that work together for a given range event”
- Defines a meta-model for the definition of Logical Range Object Models (LROMs)
What is TENA? (Continued)

- Will eventually define common tools, concept of operations, and standard object models to facilitate multi-range exercises
Stateful Distributed Object (SDO)

- SDO classes specified using the TENA Definition Language (TDL)
  - Based on the OMG’s Interface Definition Language (IDL) but with extensions to support the SDO concept
- The creator and maintainer (i.e. the “owner”) of an SDO instance is called a **servant**.
  - There is only one servant for any particular SDO instance
- The remote instance of the SDO is called a **proxy**.
  - There can be one or more proxies for any SDO servant
Stateful Distributed Objects (Continued)

- Provides remotely-invocable methods
  - Implemented by servant in the server application
  - Invoked in client application by using proxy

- Provides publication state
  - Attributes automatically disseminated to all client applications that subscribe to a particular SDO
  - Can be considered a form of distributed shared memory

- Support inheritance and composition
  - can extend ("inherit from") another SDO class
  - can contain (or be contained in) another SDO class
Test Case Description

- Used Release 3.0 of the TENA middleware
- Developed Test Case Matrix which
  - published at the various levels of class inheritance
  - subscribed at the full published class level or at some base-class level
- Each test case varied publication state update rate and number of published SDO instances
  - at 1 Hz varied the published SDO instances from 1 - 256
  - at 10 Hz varied the published SDO instances from 1 - 64
  - at 100 Hz published only one SDO instance
- Compared TENA Middleware performance verses that of TAO Real-Time Event Service
## Test Case Matrix

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Test ID</th>
<th>Aircraft SDO Publisher</th>
<th>Dynamic Object SDO Publisher</th>
<th>Static Object SDO Publisher</th>
<th>Aircraft SDO Subscriber</th>
<th>Dynamic Object SDO Subscriber</th>
<th>Static Object SDO Subscriber</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A-SO</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>A-DO</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>A-A</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>DO-SO</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>5</td>
<td>DO-DO</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>6</td>
<td>SO-SO</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
## Test Platform Description

<table>
<thead>
<tr>
<th>Computer System Characteristic</th>
<th>System 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>localhost</td>
</tr>
<tr>
<td>System Manufacturer</td>
<td>IBM PC compatible</td>
</tr>
<tr>
<td>System Model</td>
<td>n/a</td>
</tr>
<tr>
<td>CPU Type</td>
<td>1000MHz</td>
</tr>
<tr>
<td>Number of CPUs</td>
<td>2</td>
</tr>
<tr>
<td>CPU Cache</td>
<td>16K L1, 256K L2</td>
</tr>
<tr>
<td>RAM</td>
<td>512 MB</td>
</tr>
<tr>
<td>RAM Speed</td>
<td>unknown</td>
</tr>
<tr>
<td>System Bus Speed</td>
<td>133MHz FSB</td>
</tr>
<tr>
<td>Network Interface Type (ATM, Ethernet, Scramnet, etc.)</td>
<td>Ethernet</td>
</tr>
<tr>
<td>NIC Speed (Bits/s)</td>
<td>100 MB</td>
</tr>
<tr>
<td>NIC Manufacturer</td>
<td>3Com</td>
</tr>
<tr>
<td>NIC Model/Type</td>
<td>3c905C-TX</td>
</tr>
<tr>
<td>Operating System</td>
<td>Linux (RedHat 7.1)</td>
</tr>
<tr>
<td>OS Version/Release</td>
<td>Linux 2.4.2-2smp (SMP kernel)</td>
</tr>
</tbody>
</table>
Overall 10Hz - Average Latency
Non-Distributed Execution

Latency (usec)

Number of SDO Instances

Legend:
- a-a
- a-do
- a-so
- da-da
- da-so
- do-so
- so-so
Overall Comparison - 1 SDO
Non-Distributed-Execution

Graph showing latency (usec) vs. test case (so-so, do-so, do-do, a-so, a-do, a-a) for different frequencies (1 Hz, 10 Hz, 100 Hz).
## TENA Object Model
### SDO Publication State Packet Sizes

<table>
<thead>
<tr>
<th>SDO</th>
<th>Packet Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>StaticObject</td>
<td>748 bytes</td>
</tr>
<tr>
<td>DynamicObject</td>
<td>1104 bytes</td>
</tr>
<tr>
<td>Aircraft</td>
<td>1272 bytes</td>
</tr>
</tbody>
</table>
TENA Middleware vs RT Event Service Performance
(StaticObject SDO/748 bytes)
TENA Middleware vs RT Event Service Performance
(DynamicObject SDO/1104 bytes)
TENA Middleware vs RT Event Service Performance
(Aircraft SDO/1272 bytes)
#include <TENA/IKE2/IKE2.h>
#include "TP_Task.h"

int main(int argc, char * argv[] )
{
    ...

    TENA::Middleware::Configuration config( argc, argv );

    TENA::Middleware::RuntimePtr pRuntime = TENA::Middleware::init( config );

    TENA::Middleware::ExecutionPtr pExecution = pRuntime->joinExecution( executionName );
    ...

    for (int i=0; i<(numberInstances+perThread-1)/perThread; i++)
    {
        TP_Task *updaterThread = new TP_Task(
            pExecution, sessionName,
            (commTypeStr == "BestEffort") ? 1 : 0, theDuration, i*perThread,
            (i*perThread+(perThread-1) < numberInstances) ? i*perThread+(perThread-1) : numberInstances-1 );

        if (updaterThread != NULL)
            updaterThread->activate( THR_NEW_LWP, 1, 0, ACE_DEFAULT_THREAD_PRIORITY );
    ...

    return 0;
}
Publisher Example Source-Code (Continued)

```cpp
#include <time.h>
#include <map>
#include <memory>

#include "ace/Task.h"
#include "tao/corba.h"

#include <TENA/IKE2/IKE2.h>
#include <OMstd/StaticObject/StaticObjectPublisher.h>
#include "StaticObjectRemoteServantMethodsFactoryImpl.h"
#include <strstream>

#include "Timer.h"

class TP_Task : public ACE_Task_Base
{
private:
    TENA::Middleware::SessionPtr pSession_; 
    std::auto_ptr< OMstd::StaticObject::ServantFactory_t > servantFactory_; 
    unsigned long long duration_; 
    int done_; 
    int startInstance_; 
    int endInstance_; 
    std::map< std::string, OMstd::StaticObject::ServantPtr > objectMap_; 

public:
```
TP_Task(TENA::Middleware::ExecutionPtr pExecution, std::string sessionName, int commType,
    unsigned long long duration, int startInstance, int endInstance) :
    duration_(duration), done_(0),
    startInstance_(startInstance), endInstance_(endInstance)
{
    pSession_ = pExecution->createSession( sessionName );

    OMstd::StaticObject::RemoteServantMethodsFactoryPtr
        pDefaultStaticObjectRemoteServantMethodsFactory(
            new OMimpl::StaticObject::StaticObjectRemoteServantMethodsFactoryImpl() );

    servantFactory_ =
        IKE2::publish< OMstd::StaticObject::ServantTraits >(
            pSession_,
            pDefaultStaticObjectRemoteServantMethodsFactory );

    for (int i=startInstance; i <= endInstance_; i++)
    {
        std::ostream ostr;
        ostr << "SlamDunk" << i << ends;

        objectMap_[ostr.str( )] = servantFactory_->createServantUsingDefaultFactory(
            (commType == 1) ? TENA::Middleware::BestEffort : TENA::Middleware::Reliable );

        std::auto_ptr< OMstd::StaticObject::PublicationStateUpdater > pUpdater(
            objectMap_[ostr.str( )]->createPublicationStateUpdater() );
OMstd::Range::Time timestamp;

pUpdater->setName( ostr.str() );

ACE_Time_Value tv = Timer::updateTime( );
timestamp.seconds( tv.sec() );
timestamp.nanoseconds( tv.usec() );
pUpdater->setTimestamp( timestamp );

objectMap_[ostr.str( )]->modifyPublicationState( pUpdater );
}
virtual int svc()
{
    struct timespec tm;
    ::ACE_hrtime_t shrtime, ehrtime, nhrtime;
    unsigned long long deltaTime;

    for (; done_ == 0;)
    {
        shrtime = ACE_OS::gethrtime();
        for (int i=startInstance_; i <= endInstance_; i++)
        {
            std::ostringstream ostr;
            ostr << "SlamDunk" << i << ends;

            std::auto_ptr<OMstd::StaticObject::PublicationStateUpdater> pUpdater(
                objectMap_[ostr.str()] -> createPublicationStateUpdater() );

            OMstd::Range::Time timestamp;

            pUpdater->setName( ostr.str() );

            ACE_Time_Value tv = Timer::updateTime();
            timestamp.seconds( tv.sec() );
            timestamp.nanoseconds( tv.usec() );
            pUpdater->setTimestamp( timestamp );

            objectMap_[ostr.str()] -> modifyPublicationState( pUpdater );
        }
    }
}
if (duration_ < 10000000)
{
    nhrtme = shrtime + duration_;  
    do {
        ehrtime = ACE_OS::gethrtime();  
    } while (ehrtime < nhrtme);  
}
else
{
    nhrtme = ACE_OS::gethrtime();  
    deltaTime = nhrtme - shrtime;  
    if (deltaTime > duration_) continue;  
    deltaTime = duration_ - deltaTime;  
    tm.tv_sec  = deltaTime / 1000000000UL;  
    tm.tv_nsec = deltaTime - tm.tv_sec * 1000000000UL;  
    ACE_OS::nanosleep( &tm, NULL );  
}
}

return 0;
}
Conclusion
Benefits of The TENA Middleware

- Abstracts the real-world domain into objects
  - Promotes object-oriented analysis, design and programming practices

- Programmer does not have to deal with the low-level socket API
  - TENA Middleware (via TAO CORBA ORB) manages the connections
  - Programmer can specify type of connection per published object - reliable (TCP/IP) or best-effort (multicast UDP)

- Increases programmer productivity and reduces errors
  - TENA uses UML-based model-driven automated code generation
  - TENA API provides compile-time type-safety
Benefits of The TENA Middleware (Continued)

- Implements a publish/subscribe communication paradigm
  - Similar to HLA (High Level Architecture)

- Implements remote method invocation
  - Similar to CORBA

- Promotes fault tolerant software systems
  - Publishers and subscribers can exit exercise at any time without effecting each other

- Middleware distributions for UNIX (Linux/Solaris) and Windows (NT/2000/XP) platforms
  - IRIX and VxWorks distributions possible in near future
Questions or Comments?

- Phone: (850) 682-1184
- E-Mail: gregory.schultz@warpcomputing.com or gregory.schultz@cox.net