



Dr. Dobb's DIGEST

The Art and Business of Software Development

June 2010

Editor's Note by Jonathan Erickson	2
Techno-News Gesture-Based Computing on the Cheap With a single piece of inexpensive hardware — a multicolored glove — MIT researchers are making gesture-based interfaces more accessible.	3
Features The NoSQL Alternative by Ken North Low-cost, high-performance database options make gains.	5
Enforcing Correct Mutex Usage with Synchronized Values by Anthony Williams Ensuring that the correct mutex is locked when accessing data.	8
Java: Better Interfaces via <i>JFormattedTextField</i> by Mark O. Pendergast Creating forms that provide guided input for users and validated data for applications.	12
The Road Ahead for UML by Ivar Jacobson and Steve Cook UML comes of age — but now what?	18
Columns Book Review by Mike Riley Examining <i>Head First C#, Second Edition</i> by Andrew Stellman and Jennifer Greene.	22
Q&A: CodePlex and Open Source Software by Jonathan Erickson CodePlex executive director Paula Hunter takes a few minutes to talk.	23
Blog of the Month by Avo Reid Apples in the Walled Garden.	25
Other Voices by Jake Sorofman Automation, economics, and the birth of market-driven IT.	27
Effective Concurrency by Herb Sutter How to automate best practices for threads and raise the semantic level of our code	28

Good On Ya Atlassian: A Job Well Done



By Jonathan Erickson,
Editor In Chief

Let's hear it for Atlassian (www.atlassian.com), the Australian vendor of software development and collaboration tools. No, not for the quality of their tools (which is first rate; Atlassian has won more than one Jolt Award). Nor for the cool, but unusual, names of their products (Clover, Fisheye, Bamboo, Greenhopper, and Jira, among others). Rather, three cheers for the company's humanitarian efforts, as exemplified by its recently announced Community Award.

The recipient of Atlassian's first ever Community Award is Mercy Ships (www.mercyships.org), the organization that uses hospital ships to deliver free health care to the developing world. Since 1978, Mercy Ships have visited more than 70 countries, providing services valued at more than \$808 million, directly impacting more than 2.5 million beneficiaries. More than 1200 crew worldwide, representing more than 40 nations are joined each year by 2000 short-term volunteers. Professionals including surgeons, dentists, nurses, health care trainers, teachers, cooks, seamen, engineers, and agriculturalists donate their time and skills to the effort.

Atlassian's \$10,000 Community Award will benefit the people of Togo, West Africa, through the charity's program of plastic reconstructive surgeries onboard the hospital ship Africa Mercy. In truth, Atlassian has been supporting Mercy Ships for several years by donating some of those fine products with odd names. Since October 2005, Atlassian donated software via an Atlassian Community License to Mercy Ships, a gift estimated at \$30,000 according to the charity. With this cost savings, Mercy Ships surgeons can remove 60 facial tumors from patients ostracized by their communities, do 120 cataract surgeries to restore sight to young and old, correct 120 cleft lip/palates in children seen as cursed, correct 60 obstetric fistulas for women usually abandoned by their husbands and families, or offer 600 free dental procedures in areas where there is no dentist.

According to Mercy Ships CIO Chris Gregg, Mercy Ships Information Services began using Atlassian Confluence as a wiki to enable information between the charity's 15 offices and a ship in Africa to be easily shared, commented upon, and edited, regardless of different geographical locations and time zones. They use Atlassian JIRA to manage issues for IT projects, systems, and services. Procurement offices in the U.S., Holland, Germany, and the UK use the resource to collaborate on restocking items for the ship's hospital and community. Additionally, the ship's Marine Operations is able to publish business management documents to a standard that satisfies certification conditions to operate the Africa Mercy at sea and in port.

"The partnership with Atlassian, and Confluence in particular, has helped to increase the efficiency of our organization and help us bring hope to more people," says Sam Smith, Mercy Ships CEO. Daily, more than 400 Mercy Ships crew members from 40 nations onboard ship and hundreds of land-based support crew working internationally in remote locations benefit from Atlassian's wiki program through interaction with the charity's intranet.

Smith went on to say that "Mercy Ships has been fortunate to increase our capacity to serve the poor with the addition of our new flagship, Africa Mercy. This ship more than doubled the capacity of all of the ships in our history."

You can see how Mercy Ships is transforming lives in West Africa in a video at www.youtube.com/v/Cr6zB1qlwx0&hl=en_US&fs=1&.

And Atlassian? Beauty, mates!

A handwritten signature in blue ink that reads "Jonathan Erickson". The signature is stylized with a large, flowing 'J' and 'E'.

[Return to Table of Contents](#)

Gesture-Based Computing on the Cheap

With a single piece of inexpensive hardware — a multicolored glove — MIT researchers are making gesture-based interfaces more accessible

By Larry Hardesty MIT News Office

Ever since Steven Spielberg's 2002 sci-fi movie *Minority Report*, in which a black-clad Tom Cruise stands in front of a transparent screen manipulating a host of video images simply by waving his hands, the idea of gesture-based computer interfaces has captured the imagination of technophiles. Academic and industry labs have developed a host of prototype gesture interfaces, ranging from room-sized systems with multiple cameras to detectors built into laptops' screens. But MIT researchers have developed a system that could make gestural interfaces much more practical. Aside from a standard webcam, like those found in many new computers, the system uses only a single piece of hardware: a multicolored Lycra glove that could be manufactured for about a dollar.

Other prototypes of low-cost gestural interfaces have used reflective or colored tape attached to the fingertips, but "that's 2D information," says Robert Wang, a graduate student in the Computer Science and Artificial Intelligence Laboratory (CSAIL; <http://www.csail.mit.edu/>) who developed the new system together with Jovan Popovic, an associate professor of electrical engineering and computer science. "You're only getting the fingertips; you don't even know which fingertip [the tape] is corresponding to." Wang and Popovic's system, by contrast, can translate gestures made with a gloved hand into the corresponding gestures of a 3D model of the hand on screen, with almost no lag time. "This actually gets the 3D configuration of your hand and your fingers," Wang says. "We get how your fingers are flexing."

The most obvious application of the technology, Wang says, would be in video games: Gamers navigating a virtual world could pick up and wield objects simply by using hand gestures. But Wang also imagines that engineers and designers could use the system to more easily and intuitively manipulate 3D models of commercial products or large civic structures.

The glove went through a series of designs, with dots and patches of different shapes and colors, but the current version is covered with 20 irregularly shaped patches that use 10 different colors. The number of colors had to be restricted so that the system could reliably distinguish the colors from each other, and from those of background objects, under a range of different lighting conditions. The arrangement and shapes of the patches was chosen so that the front and back of the hand would be distinct but also so that collisions of similar-colored patches would be rare. For instance, Wang explains, the colors on the tips of the fingers could be repeated on the back of the hand, but not on the front, since the fingers would frequently be flexing and closing in front of the palm.

Technically, the other key to the system is a new algorithm for rapidly looking up visual data in a database, which Wang says was inspired by the recent work of Antonio Torralba, the Esther and Harold E. Edgerton Associate Professor of Electrical Engineering and Computer Science in MIT's Department of Electrical Engineering and Computer Science and a member of CSAIL. Once a webcam has captured an image of the glove, Wang's software crops out the background, so that the glove alone is superimposed upon a white background. Then the software drastically reduces the resolution of the cropped image, to only 40 pixels by 40 pixels. Finally, it searches through a database containing myriad 40x40 digital models of a hand, clad in the distinctive glove, in a range of different positions. Once it has found a match, it simply looks up the corresponding hand position. Since the system doesn't have to calculate the relative positions of the fingers, palm, and back of the hand on the fly, it's able to provide an answer in a fraction of a second.

EDITOR-IN-CHIEF
Jonathan Erickson

EDITORIAL
MANAGING EDITOR
Deirdre Blake
COPY EDITOR
Amy Stephens
CONTRIBUTING EDITORS
Mike Riley, Herb Sutter
WEBMASTER
Sean Coady

VICE PRESIDENT, GROUP PUBLISHER
Brandon Friesen
VICE PRESIDENT GROUP SALES
Martha Schwartz

AUDIENCE DEVELOPMENT
CIRCULATION DIRECTOR
Karen McAleer
MANAGER
John Slesinski

DR. DOBB'S
600 Harrison Street, 6th Floor, San Francisco, CA, 94107. 415-947-6000.
www.drdoobs.com

UBM LLC

Pat Nohilly Senior Vice President,
Strategic Development and Business
Administration
Marie Myers Senior Vice President,
Manufacturing

TechWeb

Tony L. Uphoff Chief Executive Officer
John Dennehy, CFO
David Michael, CIO
John Siefert, Senior Vice President and
Publisher, InformationWeek Business
Technology Network
Bob Evans Senior Vice President and
Content Director, InformationWeek
Global CIO
Joseph Braue Senior Vice President,
Light Reading Communications
Network
Scott Vaughan Vice President,
Marketing Services
John Ecke Vice President, Financial
Technology Network
Beth Rivera Vice President, Human
Resources
Fritz Nelson Executive Producer,
TechWeb TV





The hardware for a new gesture-based computing system consists of nothing more than an ordinary webcam and a pair of brightly colored lycra gloves. Photo: Jason Dorfman/CSAIL

Of course, a database of 40×40 color images takes up a large amount of memory — several hundred megabytes, Wang says. But today, a run-of-the-mill desktop computer has four gigabytes — or 4,000 megabytes — of high-speed RAM memory. And that number, according to Wang, is only going to increase.

“People have tried to do hand tracking in the past,” says Paul Kry, an assistant professor at the McGill University School of Computer Science. “It’s a horribly complex problem. I can’t say that there’s any work in purely vision-based hand tracking that stands out as being successful, although many people have tried. It’s sort of changing the game a bit to say, ‘Hey, okay, I’ll just add a little bit of information’” — the color of the patches — “and I can go a lot farther than these purely vision-based techniques.” Kry particularly likes the ease with which Wang and Popović’s system can be calibrated to new users. Since the glove is made from stretchy Lycra, it can change size significantly from one user to the next; but in order to gauge the glove’s distance from the camera, the system has to have a good sense of its size. To calibrate the system, the user simply places an 8.5×11 -inch piece of paper on a flat surface in front of the webcam, presses his or her hand against it, and in about three seconds, the system is calibrated.

Wang initially presented the glove-tracking system at last year’s Siggraph conference on computer graphics (<http://www.siggraph.org/s2010/>). But at the time, he says, the system took nearly a half-hour to calibrate, and it didn’t work nearly as well in environments with a lot of light. Now that the glove tracking is working well, however, he’s expanding on the idea, with the design of similarly patterned shirts that can be used to capture information about whole-body motion. Such systems are already commonly used to evaluate athletes’ form or to convert actors’ live performances into digital animations, but a system based on Wang and Popovic’s technique could prove dramatically cheaper and easier to use.

[Return to Table of Contents](#)

The NoSQL Alternative

Low-cost, high-performance database options make gains

By Ken North

A new generation of low-cost, high-performance database software is rapidly emerging to challenge SQL's dominance in distributed processing and Big Data applications. Some companies have already traded SQL's rich functionality for these new options that let them create, work with, and manage large data sets.

A big reason for this movement, dubbed NoSQL, is that different implementations of Web, enterprise, and cloud computing applications have different requirements of their databases. Not every app requires rigid data consistency, for example.

Also, when an application uses data distributed across hundreds or even thousands of servers, simple economics points to using no-cost server software as opposed to paying per-processor license fees. Once freed from license fees, you can scale horizontally with commodity hardware or opt for a cloud computing service and avoid the capital expenses altogether. Previous tools didn't always facilitate this.

Challenges to SQL's hegemony are coming from specialized products built from the ground up for large-scale analytics and document storage, as well as for building operational systems that require high availability more than consistency when partitioning data.

Applications such as online transaction processing, business intelligence, customer relationship management, document processing, and social networking don't have identical needs for data, query, or index types, nor do they have equivalent requirements for consistency, scalability, and security.

For example, BI applications run analytical and decision-support queries that can exploit bitmap indexes for operations with gigabyte- or terabyte-sized databases. Web analytics, drug discovery, financial modeling, and similar applications look to distributed systems for efficiently processing gigabyte- to terabyte-sized data sets. OLTP puts a

premium on reliability. And social network applications such as Facebook and Amazon.com have adopted BASE (basically available, soft state, eventually consistent) properties over the more familiar ACID (atomicity, consistency, isolation, durability) ones to serve their massive Web user communities of millions.

These differences are one reason non-relational NoSQL data stores, document-centric databases, and column stores have gained traction. They're more like specialized tools rather than the Swiss Army knife functionality of SQL platforms.

System architects should consider the specialized features and functions an app needs in choosing a database. NoSQL databases can be built specifically for functions such as BI, OLTP, CRM, social networks, and data warehousing, and include features such as scalability, partitioning, security, and elasticity.

Scalability and High Availability

For cloud computing and high-volume Web sites such as eBay, Amazon, Twitter, and Facebook, scalability and high availability are essential. In fact, they're the reason distributed databases have relaxed consistency requirements.

Operational systems in high-availability environments must survive software, server, and network segment failures, and provide scalability despite unpredictable surges in demand for computing resources. One approach to building such systems is to use distributed databases with a shared-nothing architecture and horizontal partitioning. Elasticity and sharding (partitioning) — both NoSQL features — are solutions for scaling out horizontally to provide availability and for processing Big Data.

A variety of data stores are gaining popularity for creating applications for scalable websites and elastic environments such as the private or public cloud. Distributed key-value stores are great when you don't need SQL rule enforcement, strong consistency, complex queries, integrated queuing, or

the ability to operate with operational databases that exceed available RAM.

New low-latency data stores provide scalability for applications that don't require rich query and analytics capabilities. Amazon has developed SimpleDB, and Google developed Bigtable. Other low-latency, open source options include Cassandra, Hypertable, MongoDB, Project Voldemort, Redis, Tokyo Tyrant, and Dynamo, the database used for Amazon S3, which as of March was hosting 102 billion objects.

The Options

Google developed Bigtable to distribute data across thousands of servers and scale to petabyte-sized data sets. A variety of Google applications use it, including Web indexing, Google Earth, Google Maps, Blogger, YouTube, and Gmail. The YouTube collection of 100 million videos requires 600 TB of storage. Bigtable is proprietary, but the data model exists in open source implementations, including Hypertable, Cassandra, and HBase. Bigtable can be used as input or output to MapReduce, which enables distributed processing of files or databases using mapping and reduction functions.

Dynamo was created to provide a key-value data store designed for high availability, permitting updates to survive server failures and network outages. Amazon subsequently built SimpleDB as a key-value store available for Amazon Web Services customers. SimpleDB, which is in beta, is restricted to items having no more than 256 attribute name-value pairs, domains having no more than 10 GB, and databases that are no more than 1 TB. Amazon reports consistency is usually attained across all copies of the data within a second. SimpleDB uses a SQL-like query language.

Project Voldemort, an open source clone of Amazon Dynamo, is a keyvalue data store that supports versioning, eventual consistency (where the database sometimes returns the wrong answer in order to maintain scaling), and automatic partitioning and replication. Keys and values can be complex objects such as maps or lists. Project Voldemort supports offline building of distributed data stores. LinkedIn developers created it, and sites such as Lookery have used it.

Cassandra integrates the Bigtable data model with the Dynamo distributed design. It offers eventual consistency, not the rigid consistency that ecommerce transactions and stock trading require. Instead of data stored in row-major or column-major sequence, Cassandra uses the ColumnFamily order inspired by Bigtable.

Cassandra is geographically distributed across multiple data centers, such as Amazon EC2 availability zones. Bulk loading can be done with Hadoop.

Cassandra provides availability and scalability for a number of well-known sites, including the huge Twitter and Facebook user communities. When Twitter's user numbers took off, it migrated from MySQL to a combination of MySQL/memcached plus 45 nodes running Cassandra. That mixed environment now handles 50 million tweets per day. Facebook adds about 60 million photos per week using Cassandra. For Digg, Cassandra manages about 3 TB of data.

Economics of Scaling

SimpleGeo, a geographic data provider, is using Apache Cassandra, an open source NoSQL offering, to avoid DBMS licensing costs as part of an effort to scale out to a multiserver database architecture.

It's running a 50-node cluster, which spans three data centers on Amazon's EC2 service for about \$10,000 a month, says CTO Joe Stump, who previously used Cassandra at Digg. By contrast, MySQL premium support would cost about \$5,000 per year per node, or \$250,000 per year — more than double the Cassandra setup, Stump says, and Microsoft SQL Server can cost as much as \$55,000 per processor per year.

The \$10,000 is an operational expense as opposed to a capital expense, and that's "a bit nicer on the books," he says.

Digg for one made a highly publicized move from MySQL to Cassandra. Digg's primary motivation for using Cassandra was "the increasing difficulty of building a high-performance, write-intensive application on a data set that is growing quickly, with no end in sight," says John Quinn, Digg's VP of engineering. Growth forced Digg into horizontal and vertical partitioning strategies that eliminated most of the value of a relational database, while still incurring all the overhead, Quinn says.

"Our system grows rapidly and requires us to provide performance and redundancy with multiple data centers and to add capacity or replace failed nodes with no downtime," he adds. As for data consistency, Digg's engineers can implement application-level controls much more efficiently with Cassandra than MySQL does generically, Quinn says.

Tokyo Tyrant is an open source database server, with a companion full-text search engine, that has a following in the NoSQL community. It's a key-value database with a hash and b-tree index structure, capable of inserting 1 million records at 0.4 seconds per record and executing 58,000 queries per second. It supports asynchronous replication and transaction processing with ACID properties and write-ahead logging. There are bindings for multiple programming languages, including Perl, Java, Ruby, and PHP. Production deployments include Scribd and Mixi, the Japanese equivalent of Facebook. LightCloud turns Tokyo Tyrant into a horizontally scalable distributed database with the addition of a universal hashing layer. Social journal Plurk uses the LightCloud Tokyo Tyrant offering.

Document Database Stores

CouchDB and MongoDB are representative of the JSON class of document database, whereas there are a large number of products that store documents encoded as XML. MongoDB is a popular

product based on client-server database architecture with b-tree indexes and communication over TCP/IP networks.

MongoDB manages JSON object collections and provides scalability with sharding and replication. Queries are JSON objects, and MongoDB also provides 2-D geospatial searches. There are driver APIs for various languages, including JavaScript, Java, Perl, PHP, Python, Ruby, and C++. Notable production deployments include Justin.tv, *The New York Times*, Disqus, Electronic Arts, and Business Insider.

CouchDB is a schema-less data store that provides a REST-style API for document-centric CRUD (create, retrieve, update, delete) operations. CouchDB can do retrieval based on key values, and it can operate with Hadoop MapReduce for nontrivial queries. It's also possible to generate views using JavaScript functions. Creating a view can be time-consuming but subsequent retrievals with the view are speedier.

CouchDB supports multimaster replication and distribution of data across multiple instances. It manages documents in JSON format and uses the SpiderMonkey JavaScript engine, making it well-suited for Web applications, such as Erlang, HTTP, JavaScript, PHP, Python, and Ruby.

For applications where XML documents and XQuery are preferred over JSON documents and JavaScript queries, there are a number of open source and commercial products. Besides the XML document stores, there are dozens of XQuery processors. The list includes Apache Xindice, Berkeley DB, eXist-db, IBM DB2, MonetDB, Mark Logic, Sedna, WebMethods Tamino, and TigerLogic XDMS.

Distributed Processing

When it comes to distributed processing of massive data sets, Hadoop MapReduce has become the red-hot technology du jour. Researchers at Yahoo, for instance, used 3,800 nodes with it to sort a petabyte of data in 16.25 hours.

Google developed and recently patented MapReduce. The map function produces a list of key-value pairs that MapReduce turns into a list of values.

The Apache Hadoop Project includes the Hadoop Distributed File System (HDFS), MapReduce, HBase database, Pig analysis language, Hive query and analysis tool, and other software. HBase is a distributed column store, modeled after Google Bigtable that can serve as input or output for MapReduce.

HBase is one of several column stores competing in the analytics and business intelligence market. Storing tables in column-major order provides substantial performance improvements over row-major stores. Benefits such as improved locality and cache performance make for better performance of retrieval-oriented queries, but performance is poor for insertion queries. Other column stores include Sybase IQ, Vertica, and CStore, an open source collaboration among several universities.

Increased interest in semantic searching and Linked Data has brought RDF triples store into the spotlight. These offerings include AllegroGraph, Bigdata, Garlik, Jena, Ontotext Big-OWLIM,

OpenLink Virtuoso, Oracle 11g, and Sesame. Several have been deployed on Amazon EC2 to exploit the distributed processing power of the cloud. Raytheon BBN researchers have also used Hadoop MapReduce to create a distributed RDF store that supports SPARQL query processing.

Restrictions and Best Practices

To ensure durability and data integrity, SQL databases provide logging and data replication. NoSQL offerings need a similar safety net. Cassandra, for example, supports transaction logging and automatic replication. Tokyo Cabinet and HBase support write-ahead logging. Tokyo Cabinet and CouchDB support master-master replication, whereas MongoDB supports master-slave replication and replica pairs.

Architects using document-oriented databases must deal with how to store different document types and whether to have a separate database for each one. Alternatives to separate databases include using an attribute to specify type or using separate collections.

Because the new generation of data stores is intended to address scalability and availability needs, certain restrictions apply for maximum efficiency. With Amazon SimpleDB, for example, the time limit for queries is five seconds. If a query takes longer, SimpleDB returns partial results, and the application must make additional calls to obtain complete results. SimpleDB restricts a query result to a maximum of 250 items, whereas Google recently lifted the AppEngine data store query result limit of a thousand items.

In horizontally partitioned systems, queries that require cross-shard joins are expensive, so the design and algorithms for partitioning require skill and knowledge of data-usage patterns. When complex queries such as aggregation are required, NoSQL operational databases aren't typically a good fit, but they can provide source data for separate solutions for analytics. Organizations using a key-value datastore sometimes need the indexing and query capabilities of SQL. They can turn to other software that supports indexing and queries, such as Apache Lucene. Regardless of whether your organization is using SQL or NoSQL databases, it's still a good idea to use version control and separate databases for testing and production.

For all the areas that NoSQL options address, we're still left with the question of which database software to adopt. The answer depends on fundamental issues: How much and what types of data will you store? Will it be used for complex queries? How many concurrent users are you supporting? And will your database scale as it takes on more users and data? SQL or NoSQL, this is the place to start.

— Ken North is an author, consultant, industry analyst, and database specialist.

[Return to Table of Contents](#)

Enforcing Correct Mutex Usage with Synchronized Values

Ensuring that the correct mutex is locked when accessing data

By Anthony William

In “Associate Mutexes with Data to Prevent Races” (<http://www.drdobbs.com/224701827>), Herb Sutter examined the importance of ensuring that the correct mutex is locked when accessing data, then presented one technique for achieving this. In this article, I present an alternative technique that I have used to achieve the same aims, in the form of the *SynchronizedValue* class template.

The Problem with Mutexes

The key problem with protecting shared data with a mutex is that there is no easy way to associate the mutex with the data. It is thus relatively easy to accidentally write code that fails to lock the right mutex — or even locks the wrong mutex — and the compiler will not help you.

```
std::mutex m1;
int value1;
std::mutex m2;
int value2;

int readValue1()
{
    std::lock_guard<std::mutex> lk(m1);
    return value1;
}
int readValue2()
{
    std::lock_guard<std::mutex> lk(m1); // oops: wrong mutex
    return value2;
}
```

Moreover, managing the mutex lock also clutters the source code, making it harder to see what is really going on.

The use of *SynchronizedValue<T>* solves both these problems — the mutex is intimately tied to the value, so you cannot access it without a lock, and yet access semantics are still straightforward. For simple accesses, *SynchronizedValue<T>* behaves like a pointer-to-*T*; for example:

```
SynchronizedValue<std::string> value3;
std::string readValue3()
{
    return *value3;
}
void setValue3(std::string const& newVal)
{
    *value3=newVal;
}
void appendToValue3(std::string const& extra)
{
    value3->append(extra);
}
```

Both forms of pointer dereference return a proxy object rather than a real reference, to ensure that the lock on the mutex is held across the assignment or method call, but this is transparent to the user.

Beyond Simple Accesses

The pointer-like semantics work very well for simple accesses such as assignment and calls to member functions. However, sometimes you need to perform an operation that requires multiple accesses under protection of the same lock, and that's what the *update()* method provides.

By calling `update()` you obtain an *Updater* object that holds a lock on the mutex protecting the data, and which can be used to access the protected data. The lock is held until the *Updater* object is destroyed, so you can safely perform multi-part operations. The *Updater* object also acts as a pointer-to-*T*, just like *SynchronizedValue* does, but this time the lock is already held. For example, the following function adds a trailing slash to a path held in a *SynchronizedValue*<std::string>. The use of the *Updater* object ensures that the string hasn't changed in between the query and the update.

```
void addTrailingSlashIfMissing(SynchronizedValue<std::string> & path)
{
    SynchronizedValue<std::string>::Updater u=path.update();

    if(u->empty() || (*u->rbegin() != '/'))
    {
        *u+="/";
    }
}
```

Operations Across Multiple Objects

Though *SynchronizedValue*<*T*> works very well for protecting a single object of type *T*, nothing that we've seen so far solves the problem of operations that require atomic access to multiple objects unless those objects can be combined within a single structure protected by a single mutex.

One way to protect access to two *SynchronizedValue*<*T*> objects is to construct a *SynchronizedValue*<*T*>::Updater for each object and use those to access the respective protected values; for instance:

```
SynchronizedValue<std::queue<MessageType>> > q1,q2;
void transferMessage()
{
    SynchronizedValue<std::queue<MessageType>> >::Updater u1=q1.update();
    SynchronizedValue<std::queue<MessageType>> >::Updater u2=q2.update();
```

```
    if(!u1->empty())
    {
        u2->push_back(u1->front());
        u1->pop_front();
    }
}
```

This works well in some scenarios, but not all — if the same two objects are updated together in different sections of code, then you need to take care to ensure that the *Updater* objects are constructed in the same sequence in all cases. Otherwise, you have the potential for deadlock. This is just the same as when acquiring any two mutexes.

Under the Hood

We've seen how to use *SynchronizedValue*<*T*> to ensure that the right mutex is always locked when accessing an object; now it's time to look under the hood and see how it works.

It really is very simple: At its heart, a *SynchronizedValue*<*T*> is just a *T* object and an *std::mutex*. All the work is done by the proxy objects returned from the operators.

```
template<typename T>
class SynchronizedValue
{
    T data;
    std::mutex m;
public:
    DerefValue operator*();
    Updater operator->();
    Updater update();
};
```

Basic Pointer Operations

The basic pointer dereference operator returns an instance of the private nested class *SynchronizedValue*<*T*>::DerefValue:

Selling Pizza in the Cloud: Domino's and Windows Azure:

When you're one of the busiest pizza companies in the world, how do you scale your web site to meet regular and irregular demand? You move it to the Cloud, of course! Click screen to watch a short video explaining these details about Azure on Dr. Dobb's Microsoft Resource Center.



Click screen to link to video

```
DerefValue operator*()
{
    return DerefValue(*this);
}
```

This class holds an `std::unique_lock<std::mutex>`, which manages the lock and a reference to the protected `T` object. The lock is acquired when the object is constructed by the call to `operator*`, and released when the object is destroyed. `DerefValue` has a simple conversion operator to `T`, which allows retrieval of the protected value, and also an assignment operator that allows the value to be set.

```
operator T()
{
    return data;
}

DerefValue& operator=(T const& newVal)
{
    data=newVal;
    return *this;
}
```

The arrow operator is implemented similarly, except this time it returns an instance of `SynchronizedValue<T>::Updater`. Because `SynchronizedValue<T>::Updater` also has an implementation of the arrow operator, the chaining rules for this operator mean that you can access members of the protected object directly using the normal pointer syntax, and the temporary `Updater` object holds the lock for you. Just like `DerefValue`, the `Updater` object acquires the lock in its constructor and releases it in its destructor.

This is the same `Updater` object we get from the `update()` member function, so it has a plain pointer dereference operator that returns a reference to the protected object in addition to the arrow operator that returns the pointer:

```
T* operator->()
{
    return &data;
}

T& operator*()
{
    return data;
}
```

Wrapping It Up

I've found classes like `SynchronizedValue<T>` invaluable in many projects, as it provides a simple way of ensuring that the right mutex is always held when accessing protected data. It's not a panacea, but those cases where it doesn't work likely require more careful thought anyway.

You can download the source code at http://i.cmpnet.com/ddj/images/article/2010/code/synchronized_value.zip or from <http://www.stdthread.co.uk/syncvalue> if you want to try it out. As written, it relies on C++0x facilities, but it should be easy to substitute an alternative mutex implementation if desired.

— Anthony Williams is author of the book *C++ Concurrency in Action* (<http://www.manning.com/williams/>) and of the *just::thread C++0x thread library* (<http://www.stdthread.co.uk/>). He can be contacted at anthony.ajw@gmail.com.

[Return to Table of Contents](#)

How Do I: Deploy My First Azure Application: Step-by-Step?

In this video, Max Adams will deploy a new Windows Azure Web Role Application to the Cloud in Azure Platform. Click screen to watch a short video explaining these details about Azure on Dr. Dobb's Microsoft Resource Center.



Click screen to link to video



I KNOW A PLACE THAT'S DIFFERENT, BUT FAMILIAR

Code in the cloud ... with the development technologies you already know

With **Windows Azure™**, you have a flexible, on-demand infrastructure that taps into all the IT resources you need. The advantages?

You have the freedom to focus.

With the cloud as your development platform, you can focus on what you do best: create applications, without the restrictions on performance, capacity, or deployment that come with using your own hardware rather than the cloud.

You can code in the language you choose.

Windows Azure enables you to build applications in multiple languages, including .NET, PHP, and Java.

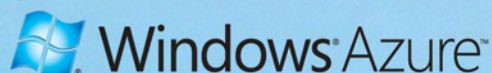
You're already equipped for the cloud.

You can start developing applications for Windows Azure right away, using familiar tools such as Microsoft® Visual Studio® 2008.

Find out more.

See how Windows Azure can help get you coding in the cloud.

Go to **windowsazure.com** now.



Java: Better Interfaces via *JFormattedTextField*

Creating forms that provide guided input for users and validated data for applications

By Mark O. Pendergast

Data quality and accuracy is key to success for every application. Databases full of invalid phone numbers, improperly formatted email addresses, or out of range numeric data are of no use to anyone. Java Swing interface programmers know all too well the work involved in error checking user inputs taken from vanilla *TextFields*. Many probably have been tempted to implement interfaces using *JFormattedTextFields*, but have been put off by the complexity of masks, formatters, verifiers, abstract classes, and interfaces that are required. In this article, I explain the proper use of the *JFormattedTextField* and its related classes to create forms that provide both guided input for users and validated data for the application. The source code for a sample application and a library of reusable classes for common data entry items based on *JFormattedTextField* is available for download at <http://i.cmpnet.com/ddj/images/article/2010/code/JFormattedText.zip>.

JFormattedTextFields are alternatives to *TextFields* that are well worth using. They differ from *TextFields* in several ways:

- First, they maintain a current “value” associated with the field as an *Object*. The *getValue* and *setValue* methods are provided to access it.
- Second, they use formatter objects to control the display of the value. One formatter is used to convert the value object to a *String* for display when the field does not have the focus; a separate formatter is used to convert

the value when it receives the focus. Therefore, a field that holds currency can limit the number of decimal places shown, have a dollar sign, and have commas separating the thousands while it is not being changed, then automatically change to a form with just digits and decimals when the focus is shifted to it. A type of formatter called a *MaskFormatter* can provide a “type-in-the-blank” template to guide the users input. *MaskFormatters* are useful for data types such as phone numbers, dates, and Social Security numbers.

- The third, and most important difference is that when combined with *InputVerifiers* they can handle all the parsing, range checking, and formatting required for processing GUI form interfaces. For example, Figure 1(a) shows a *JFormattedTextField* used to input currency with the focus, Figure 1(b) shows the same *JFormattedTextField* without the focus. Table 1 compares the code required to input, convert, and validate a double value using *TextField* and *JFormattedTextField*.

Implementing a *JFormattedTextField*

According to the Java API:

JFormattedTextField does not do the formatting itself, rather formatting is done through an instance of *JFormattedTextField.AbstractFormatter*, which is obtained from an instance of *JFormattedTextField.AbstractFormatterFactory*.

What this really means is that to make a *JFormattedTextField* operate, you need to create



Figure 1a



Figure 1b

Code using a <i>TextField</i>	Code using a <i>JFormattedTextField</i> and <i>InputVerifier</i>
<pre>Double payment = 0; try { payment = Double.parseDouble(paymentField.getText()); if (payment < 0 payment > 1000000) { JOptionPane.showMessageDialog(this, "Invalid value for payment"); return; } } catch (NumberFormatException ex) { JOptionPane.showMessageDialog(this, "Invalid value for payment"); return; }</pre>	<pre>Double payment = ((Double)paymentField.getValue());</pre>

Table 1: Comparison of code to access *TextField* and *JFormattedTextField* values.

an instance of an *AbstractFormatterFactory*. The easiest way to do this is to use a convenience class called *DefaultFormatterFactory*, then attach it to the *JFormattedTextField*. The primary purpose of the *DefaultFormatterFactory* is to maintain three *AbstractFormatter* classes for the *JFormattedTextField*. One abstract formatter is the default formatter, it is used if a more specific formatter cannot be found; another is the display formatter used to format the value when the *JFormattedTextField* does not have the focus; and the third is the edit formatter, used to format the value when the *JFormattedTextField* has the focus.

```
DefaultFormatterFactory dff =
    new DefaultFormatterFactory(defFormatter,
        displayFormatter, editFormatter);
JFormattedTextField tf = new JFormattedTextField(dff);
```

The formatter classes are primarily responsible for providing the body for two methods, *stringToValue* and *valueToString*. *stringToValue* parses a *String* and creates an object of the proper type (*Long*, *Double*, *Date*, *Integer*, etc). *valueToString* takes the current value and formats it for display. *AbstractFormatter* classes may in turn rely on specific format classes such as *DecimalFormat* and *SimpleDateFormat*. Creating formatter classes from scratch can be a daunting task. Fortunately, the standard Java API provides numerous formatter classes that can be used as is. The standard class you use is determined by the type of data you need to format. Numeric data (currencies, counts, salaries) should use the *NumberFormatter* class, fixed length string data should use the *MaskFormatter* class (social security numbers, credit card numbers, phone numbers, dates), and variable length string data will need to have a custom formatter created based on the *DefaultFormatter* class.

NumberFormatter

A *NumberFormatter* can be used for any numeric field, age, income, annual rainfall, quantity, cost, etc. A *NumberFormatter* object is created using a constructor that accepts an object of type *NumberFormat* as its only parameter. Typically a *DecimalFormat* object (subclass of *NumberFormat*) is used. For example:

```
NumberFormatter currencyFormatter =
    new NumberFormatter(new DecimalFormat("$ #,###.00"));
```

In the above example, a *DecimalFormat* object is specified that can take a numeric value and convert it to a *String* using the specified pattern. The “#” character is replaced with a digit (or absent if 0), the “0” is replaced with a digit. The fact that there are only

two “0”s to the right of the decimal point indicates only two decimal places are to be shown (rounding up/down automatically). The “,” acts as a grouping separator. Since there are three “#”s between the decimal point and the comma, the numbers to the left of the decimal will be grouped in sets of three. You only need to specify the position of the first grouping separator.

The example above adds the grouping separator and dollar sign. Both of these will cause parse errors when converting back to a number. Therefore, you will typically need a separate *NumberFormatter* for the edit and display formats. The one used for the edit format would exclude the dollar sign, grouping separator (comma), and use “#”s to the right of the decimal. For example:

```
NumberFormatter displayCurrencyFormatter =
    new NumberFormatter(new DecimalFormat("$ #,###.00"));
NumberFormatter editCurrencyFormatter =
    new NumberFormatter(new DecimalFormat("#.##"));
```

The *getValue* method of the *JFormattedTextField* calls the parse method within the *DecimalFormat* class, and may return a *Long* or a *Double*, depending on the *String* involved. If the *String* has a decimal in it, a *Double* is returned, if it doesn't, a *Long* is returned. To make the class of the return value more predictable (and avoid class cast exceptions), you should call the *setValueClass* method and specify whether you want a *Long*, *Double*, *Float*, *Short*, *Byte*, or *Integer*; for example:

```
displayCurrencyFormatter.setValueClass(Double.class);
editCurrencyFormatter.setValueClass(Double.class);
```

The following is a complete example for creating a *JFormattedTextField* for entering salary:

```
// create the JFormattedTextField
JFormattedTextField salaryField = new JFormattedTextField();
// create the formatters, default, display, edit
NumberFormatter defaultFormatter = new NumberFormatter(new
DecimalFormat("#.##"));
NumberFormatter displayFormatter =
    new NumberFormatter(new DecimalFormat("$ #,###.00"));
NumberFormatter editFormatter = new NumberFormatter(new
DecimalFormat("#.##"));
// set their value classes
defaultFormatter.setValueClass(Double.class);
displayFormatter.setValueClass(Double.class);
editFormatter.setValueClass(Double.class);
// create and set the DefaultFormatterFactory
DefaultFormatterFactory salaryFactory =
    new
DefaultFormatterFactory(defaultFormatter, displayFormatter, editFor
matters);
salaryField.setFormatterFactory(salaryFactory);
```

MaskFormatter

The *MaskFormatter* is best used to format *Strings* that have a fixed length and set format. Examples include: dates, Social Security numbers, phone numbers, credit-card numbers, ZIP codes, vehicle ID numbers (VINs). *MaskFormatters* are not suitable for *Strings* with variable numbers of characters, e.g. URLs and email addresses. A single instance of the *MaskFormatter* can be used for both the edit and display formats of an *AbstractFormatterFactory*. A *MaskFormatter* object is created using a constructor that accepts a formatting mask as its only parameter. The formatting mask specifies what type of characters can appear in each position and any

Mask Character	Meaning
A	Any character or number
a	Any character, lower case are converted to upper case
L	Any character, upper case are converted to lower case
#	Any valid number (0-9)
?	Any character
H	Any hex character (0-9, A-F)
*	Anything
^	Escape character

Table 2: *MaskFormatter* characters.

literal characters. Literal characters are fixed in place and are not changeable by the users; for example, the dash and parentheses in a phone number would be considered literal characters. Table 2 shows the different mask characters used by the *MaskFormatter*.

For example, the mask "###-##-####" could be used for a Social Security number. The #s allow the user to type in any number, the dashes are literals and fixed in place. Similarly for a ZIP code the mask "#####-####" could be used:

```
MaskFormatter zipMask = new MaskFormatter("#####-####");
MaskFormatter ssnMask = new MaskFormatter("###-##-####");
```

You can restrict what the user types in even more by invoking the *setValidCharacters* or the *setInvalidCharacters* method. For example, if you wanted to restrict the digits in a ZIP code to just 2s and 3s, then you could make the following call:

```
zipMask.setValidCharacters("23"); //only allow 2, 3 in the zip code
```

Alternatively, if you wanted any digit but a 2 or a 3 then you could make the following call:

```
zipMask.setInvalidCharacters("23"); // don't allow 2,3 in the zip code.
```

A final consideration when creating a *MaskFormatter* is setting the placeholder character. This character is inserted into the *String* if the current value has too few characters or when the user presses the delete or backspace keys while editing. It is common to use an underscore as the placeholder, but any character will do. To avoid confusion, select a character that is not used as a literal or valid character in the mask. To set the placeholder, call the *setPlaceholderCharacter* method; for example:

```
zipMask.setPlaceholderCharacter('_'); // use an underscore as a placeholder
```

Table 3 provides a few common masks. You may want to consider using a fixed-width font (such as Courier) for the *JFormattedTextField*. This makes the overall appearance and character spacing more pleasing to the eye.

The example that follows creates a *JFormattedTextField* for entering a date:

```
// create a MaskFormatter for Dates
JFormattedTextField dateField = new JFormattedTextField();
MaskFormatter dateFormatter = new MaskFormatter("##/##/####
##:##:## UM");
dateFormatter.setValidCharacters("0123456789AP");
dateFormatter.setPlaceholderCharacter('_');
dateFormatter.setValueClass(String.class);
DefaultFormatterFactory dateFormatterFactory = new
    DefaultFormatterFactory(dateFormatter);
dateField.setFormatterFactory(dateFormatterFactory);
```

The example that follows creates a *JFormattedTextField* for entering a Social Security number:

```
JFormattedTextField ssnField = new JFormattedTextField();
MaskFormatter ssnFormatter = new MaskFormatter("###-##-####");
ssnFormatter.setValueClass(String.class);
ssnFormatter.setPlaceholderCharacter('_');
DefaultFormatterFactory ssnFormatterFactory = new
    DefaultFormatterFactory(ssnFormatter);
ssnField.setFormatterFactory(ssnFormatterFactory);
```

Description	Mask	Valid Characters
Social Security Number	"###-##-####"	N/A
Zip code	"#####-####"	N/A
Phone number (North American)	"(###) ###-####"	N/A
Credit Card Number	"#### #### #### ####"	N/A
Date	"##/##/#### ##:##:## UM"	"0123456789AP"

Table 3: Masks for common data types.

Custom Formatters

If you have a data type that is variable in length and does not fit well with one of the existing formatter classes, or requires special rules for conversion, then you can create your own formatter class based on the *DefaultFormatter* class or any of the other *Formatter* classes (e.g. *NumberFormatter*). You will need to override the *stringToValue* and *valueToString* methods to provide the behavior you desire. *stringToValue* accepts a *String* as an input, parses it, and returns an *Object*. If the parsing fails, then a *ParseException* should be thrown. The type of *Object* returned should match the results of *getValueClass*. The code necessary to parse the *String* will depend on the data you are parsing. If you are parsing a proper name, then your method may just check to see that the *String* is not empty and force the first letter to be capitalized. If you are parsing a URL or email address, then you'll want to employ a *Pattern* object to test the validity and sequence of the characters.

valueToString accepts an *Object* and converts it to a *String*. If the conversion fails, then a *ParseException* is thrown. The conversion may be as simple as calling the *toString()* method of the *Object* or the conversion may adjust the data before returning it. For example, a field that holds a percentage may want to store 5.3% as .053, but display it as 5.3%. In this case, the *valueToString* method would multiply the value by 100, format it with the desired number of decimal places, then tag on the percent sign.

The *EmailFormatter* class extends the *DefaultFormatter* class and uses a precompiled pattern to check an email address. For more information on the email format and pattern matcher, see "How to Validate Email, SSN, Phone Numbers in Java Using Regular Expressions" by Zaheer Paracha (<http://www.zparacha.com/validate-email-ssn-phone-number-using-java-regular-expression/>). The code that follows creates a *JFormattedTextField* for entering an email address using *EmailFormatter.java* class, which you can download at <http://i.cmpnet.com/ddj/images/article/2010/code/JFormattedText.zip>.

```
JFormattedTextField emailField = new JFormattedTextField();
EmailFormatter emailFormatter = new EmailFormatter();
DefaultFormatterFactory emailFormatterFactory =
    new DefaultFormatterFactory(emailFormatter);
emailField.setFormatterFactory(emailFormatterFactory);
```

If you want a custom formatter that handles numbers, then you should create a class that extends *NumberFormatter*. The *PercentFormatter* class extends *NumberFormatter* to create a class

to handle percentage data. The constructor accepts a *NumberFormat* object to handle formatting and parsing. The *valueToString* method first multiplies the value by 100, then calls super class's *valueToString* method. Finally a percent is added and the *String* returned. The *stringToValue* first strips off any extra whitespace or percent signs, then converts to a double using the super class's *stringToValue* method, divides by 100, then returns a *Double* object. In this way, the value is stored in a form amenable to using in calculations, but displayed in a manner the user expects.

The code that follows creates a *JFormattedTextField* for entering a percent using *PercentFormatter.java*, which you can download at <http://i.cmpnet.com/ddj/images/article/2010/code/JFormattedText.zip>.

```
JFormattedTextField percentField = new JFormattedTextField();
PercentFormatter percentFormatter =
    new PercentFormatter(new DecimalFormat("#.##"));
DefaultFormatterFactory percentFormatterFactory =
    new DefaultFormatterFactory(percentFormatter);
percentField.setFormatterFactory(percentFormatterFactory);
```

InputVerifiers

By default, when the focus is lost on a *JFormattedTextField* the rules for *COMMIT_OR_REVERT* are followed. That is, if the text typed in is legal, the value is updated and the new value is displayed using the display formatter; if the text is not legal, then the text being edited is discarded and the old value is displayed. A *JFormattedTextField* can also be set to use the rules for *COMMIT*. Under those rules, when an invalid string is entered, it is left displayed, but the value is not changed. When this happens the text shown on the screen does not match the value stored in the *JFormattedTextField*. So it seems the choice is to lose the edit or have a mismatch in values. Neither of these options is desirable. To correct this, all you need to do is to attach an *InputVerifier* to the *JFormattedTextField*. An *InputVerifier* provides the capability to force the control to retain the focus if it has an invalid value. Further, the formatters attached to a *JFormattedTextField* usually only check the format of the data; an *InputVerifier* can perform range checks and other validity checks as well.

An *InputVerifier* is a class that inherits the *InputVerifier* class. These classes will contain a constructor and must define the body of an abstract method named "verify." This method is called whenever the focus is about to be lost on a control. If *verify* returns true, this means all is well and the focus can be moved. If *verify* returns false, then the focus remains on the control. The *verify* method accepts a *JComponent* as input (the *JFormattedTextField*) and then can do whatever manipulations it needs before returning true or false. The following example is a reusable *InputVerifier* to handle double values.

The data fields define the minimum and maximum permissible values. The constructor and the included mutator methods can set these. Two *Color* values are also present, one for invalid values, the

other for valid values. The *verify* method first retrieves the text from the *JComponent* and tries to parse it to a double. If an exception is thrown, then the foreground color is set to *INVALID_COLOR* and false is returned. If an exception is not thrown, then the value is tested to see if is in a valid range. If it is, the foreground color is set to the *VALID_COLOR* and true is returned. If the value is out of range then the foreground color is set to *INVALID_COLOR* and false is returned. (Also see *DoubleVerifier.java*, which you can download at <http://i.cmpnet.com/ddj/images/article/2010/code/JFormattedText.zip>.)

To use this *InputVerifier*, first create an instance of the verifier, then attach it to the *JFormattedTextField* using the *setInputVerifier* method.

```
JFormattedTextField salaryField = new JFormattedTextField();
DoubleVerifier salaryVerifier = new DoubleVerifier(0,1000000);
salaryField.setInputVerifier(salaryVerifier); // attach the verifier
```

Putting It All Together

Creating a *JFormattedTextField* that does it all requires you to do the following steps:

1. Create an instance of a formatter for display mode (when the control does not have the focus)
2. Create an instance formatter for edit mode (when the control does have the focus)
3. Create an instance *DefaultFormatterFactory* based on the two formatters
4. Attach the *DefaultFormatterFactory* to the *JFormattedTextField* using the *setFormatterFactory* method
5. Create an instance of an *InputVerifier*
6. Attach the *InputVerifier* to the *JFormattedTextField* using the *setInputVerifier* method

```
// create the JFormattedTextField
JFormattedTextField salaryField = new JFormattedTextField();
// create the formatters, default, display, edit
NumberFormatter defaultFormatter = new NumberFormatter(new
    DecimalFormat("#.##"));
NumberFormatter displayFormatter =
    new NumberFormatter(new DecimalFormat("$ #.###.00"));
NumberFormatter editFormatter = new NumberFormatter(new
    DecimalFormat("#.##"));
// set their value classes
defaultFormatter.setValueClass(Double.class);
displayFormatter.setValueClass(Double.class);
editFormatter.setValueClass(Double.class);
// create and set the DefaultFormatterFactory
DefaultFormatterFactory salaryFactory =
    new DefaultFormatterFactory(defaultFormatter,
        displayFormatter, editFormatter);
salaryField.setFormatterFactory(salaryFactory);
// create and attach an InputVerifier
DoubleVerifier salaryVerifier = new
    DoubleVerifier(0,1000000); // salaries 0 to 1 mil
salaryField.setInputVerifier(salaryVerifier);
// attach the verifier
```

As you can see, this is quite a bit of code to create a fancy equivalent of a *TextField*. Doubtless this is one of the reasons why the *JFormattedTextField* is not covered in many Java texts or used as frequently by programmers. Fortunately, object-oriented program-

ming can come to your aid. Instead of repeating the sequence of code for every field in every program, just create a few reusable classes for common data types. Adding *PropertySupport* to the classes makes them easy to integrate into the Netbeans GUI designer palette, allowing you to add them to your program graphically like any other control.

The code project attached to this article (available for download at <http://i.cmpnet.com/ddj/images/article/2010/code/JFormattedTexttr>) provides examples via the following reusable classes:

- FormattedTextFields
 - o FormattedCurrencyField
 - o FormattedDateField
 - o FormattedDoubleField
 - o FormattedEmailField
 - o FormattedIntegerField
 - o FormattedPercentField
 - o FormattedPhoneField
 - o FormattedSSNField
- Custom verifiers
 - o DateVerifier
 - o DoubleVerifier
 - o EmailVerifier
 - o IntegerVerifier
 - o MaskVerifier
 - (used by JFormattedPhoneField and JFormattedSSNField)
 - o PercentVerifier
- Custom formatters
 - o EmailFormatter
 - o PercentFormatter

Figure 2 shows the use of these classes. I will cover *FormattedDoubleField* and *FormattedDateField* as examples:

FormattedDoubleField extends *JFormattedTextField*. (*FormattedDoubleField.java* can be downloaded at <http://i.cmpnet.com/ddj/images/article/2010/code/JFormattedText.zip>.) It has properties for the number of decimal places to display (*decimalPlaces*), the minimum and maximum valid values

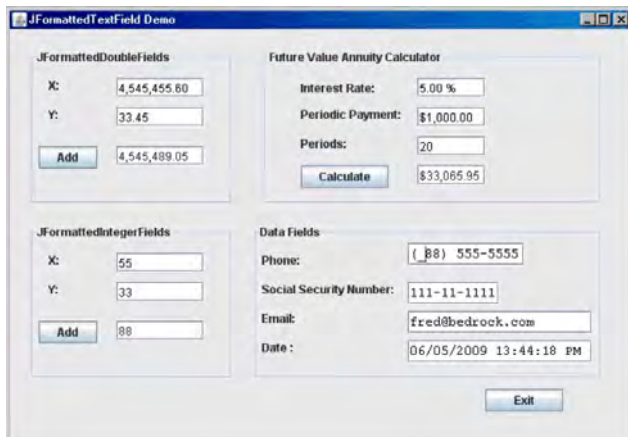


Figure 2: Sample application

(*minValue*, *maxValue*). Four constructors are provided, a no-arg constructor that uses the default values, a constructor that accepts the number of decimal places, a constructor that accepts the *min* and *max* values, and a constructor that accepts the *min* value, *max* value, the current value, and number of decimal places. *Accessor* and *mutator* methods are provided to change the *maxValue*, *minValue*, and *decimalPlaces*.

The constructor first checks the *min* and *max* values, generating an *IllegalArgumentException* if *min* is greater than *max*. A *DoubleVerifier* (discussed previously) is then created and attached using the *setInputVerifier* method. Three *NumberFormatters* are created. The display and edit formatters are created based on the *DecimalFormat* class. *Display* includes the grouping character while the edit does not. These formatters are then used to create a *DefaultFormatterFactory*, which in turn is attached using the *setFormatterFactory* method. *PropertyChange* support is provided with the *propertySupport* variable and implemented using calls to the *firePropertyChange* method in the mutator methods. The *FormattedDoubleField* class can be compiled in NetBeans and added to the GUI palette using the Palette Manager.

Program code that uses the *FormattedDoubleField* just needs to call the *getValue* and *setValue* methods. *getValue* will return an *Object* that can be cast to a *Double*, *setValue* assumes a *Double* object will be given. A simple adder using three *FormattedDoubleFields* — *xField*, *yField*, *zField* — would look like:

```
Double x = (Double)xField.getValue();
Double y = (Double)yField.getValue();
zField.setValue(new Double(x+y));
```

There is no need to invoke *Double.parseDouble* or try/catch *NumberFormatException*. That is all provided along with range checking of the inputs. Implementing similar code using a *JTextField* would look like:

```
try{
    double x = Double.parseDouble(xField.getText());
    if(x < min || x > max) {
        JOptionPane.showMessageDialog(this, "invalid value for x");
        return;
    }
}
catch (NumberFormatException exX) {
    JOptionPane.showMessageDialog(this, "invalid value for x");
    return;
}

try{
    double y = Double.parseDouble(yField.getText());
    if(y < min || y > max) {
        JOptionPane.showMessageDialog(this, "invalid value for y");
        return;
    }
}
catch (NumberFormatException exY) {
    JOptionPane.showMessageDialog(this, "invalid value for y");
    return;
}

DecimalFormat formatter = new DecimalFormat("#,###.##");
double z = x+y;
xField.setText(formatter.format(z));
```

FormattedDateField extends *JFormattedTextField*. (*Formatted-*

DateField.java can be downloaded at <http://i.cmpnet.com/ddj/images/article/2010/code/JFormattedText.zip>.) It provides several constructors: a default one that uses the current date, one that accepts a *GregorianCalendar* object, one that accepts the month, day, year, hour, minute, and second as integers, and one that accepts a *Date* object. The principle constructor first sets the value of the *JFormattedTextField* to the specified date and time using a *SimpleDateFormat* object to convert it to a *String* in the proper format. It then creates a *DateVerifier* and attaches it using a call to *setInputVerifier*. A *MaskFormatter* is created using “##/##/#### ##:##:## UM” as the format string. Recall that the #s are masks for digits, the U for an uppercase character, and the slashes, spaces, colon, and M are literals. This creates a formatter that allows the user to only change the certain characters leaving the literals in place for guidance. The valid characters for the *MaskFormatter* are set to the digits 0 to 9 and the characters A and P. Thus the U placeholder can only be replaced with an A or P. A *DefaultFormatterFactory* is then created with the *MaskFormatter* and attached using the *setFormatterFactory* method. A *DateFormatter* could have been created using the *SimpleDateFormat* class and used in place of the *MaskFormatter*. However, the *DateFormatter* allows editing of the entire text and does not “lock down” literal characters, giving less guidance for the user and leaving a greater chance for error.

Since a *MaskFormatter* is being used, the *getValue* and *setValue* methods associated with the *JFormattedTextField* class will return a *String*. Separate *getDate* and *setDate* methods were added to provide support for *Date* objects. The *getDate* method uses a *SimpleDateFormat* object to parse the text value into a *Date* object; *setDate* uses the same *SimpleDateFormat* object to format a *Date* into a *String*. A *Timer* object was added to allow the field to automatically update itself. A Boolean variable “ticking” can be turned on/off to control the *Timer*.

Program code that accesses a *FormattedDateField* can call the *getDate* or *getValue* to retrieve the contents of the field.

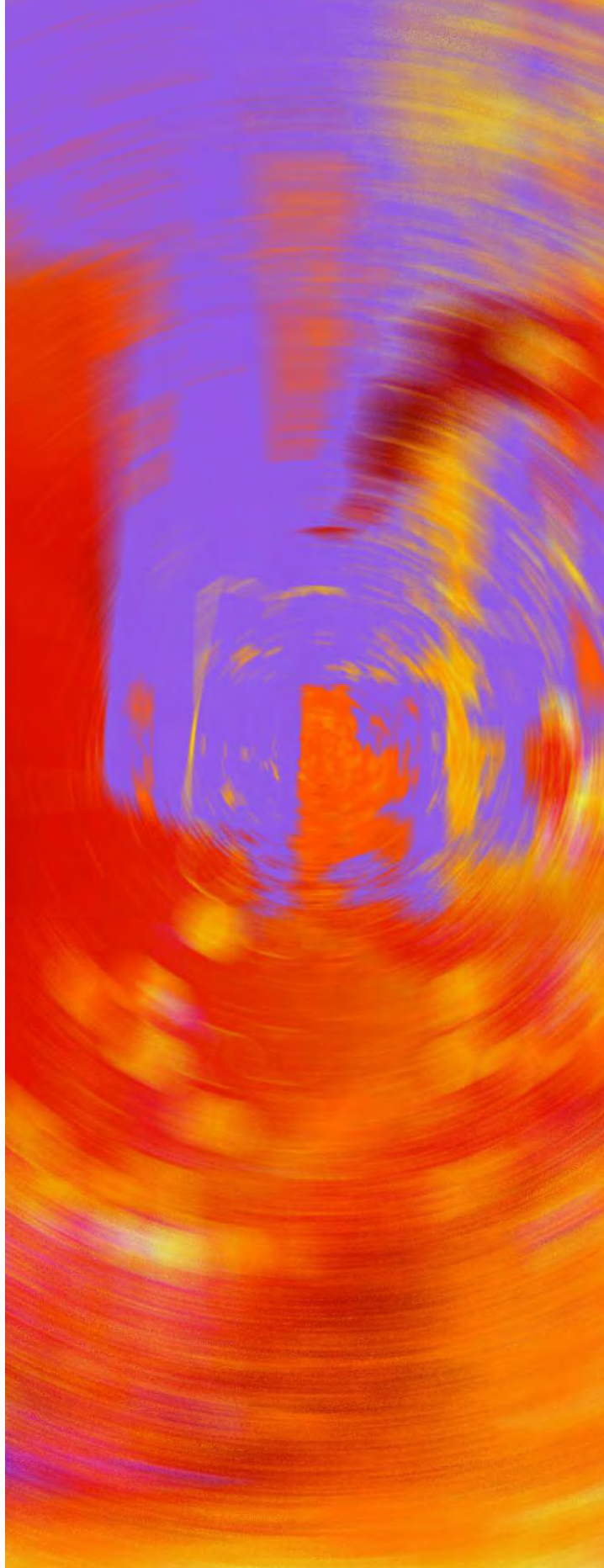
```
Date date1 = dateField.getDate();
String date2 = (String)dateField.getValue();
```

Conclusion

This article has demonstrated how to use *JFormattedTextFields* for numeric data, fixed size string data, and custom data. Using *JFormattedTextFields* in place of *JTextFields* requires extra upfront coding for initializing the interface, but saves code when accessing and processing the data. *JFormattedTextFields* also provide a more fool-proof interface and better guidance to the user. Coding efficiency and reusability is further enhanced if custom classes based on *JFormattedTextField* are created for generic data types.

— Mark Pendergast is an Associate Professor of Computer Information System at Florida Gulf Coast University.

[Return to Table of Contents](#)



The Road Ahead for UML

UML comes of age — but now what?

By Ivar Jacobson and
Steve Cook

More than 12 years have passed since the Unified Modeling Language (UML) became a standard (www.uml.org). During these years, opinions of UML have varied between delight and distaste. In this article, we discuss the deficiencies of the current UML specification and propose how to make it agile, leaner, smarter, and more flexible — in short, how to prepare it for the future so that users can be confident that their investments in UML today will increase in value going forward.

At the beginning of the '90s, there were 26 published methods on object-orientation, most with its own notation with its own set of icons. It was in this environment that UML was born. Although Grady Booch, Jim Rumbaugh, and Ivar Jacobson initiated the design of what became UML, many other contributors (including Steve Cook) quickly joined the effort and the Object Management Group (OMG) launched the result (www.omg.org). UML quickly made most other methods — along with their own notations — obsolete. UML eventually became the standard we had hoped for, and toolbuilders and practitioners rapidly adopted the new approach.

Since UML first had outstanding success, we all knew that the pendulum would swing in the opposite direction some day — and we were right. After a few years the setback arrived, but admittedly for good reasons. For instance, at the outset there weren't many good UML tools. Some were very advanced, but hard to use. That disappointed many users and hindered the wide adoption of UML. The language received criticism from the academic world; for example David Parnas nicknamed it the “Undefined Modeling

Language”. The criticism was exaggerated, but not unfounded. Likewise, the original leaders of the agile movement were negative to modeling. They said “no modeling — just code.” Many people were skeptical about the tools, so they worked more with UML sketches on the whiteboard than formally with the tools themselves.

However, the pendulum is now swinging back. The tools have become better. Criticism from academics has mostly stopped. Agility has come to big companies and modeling is agile if done sensibly (and not as a “silver bullet”). Microsoft, for instance, has implemented UML in Visual Studio 2010, alongside domain-specific languages. Other important standards such as SysML are implemented as extensions to UML.

Thus it seems that today the world looks upon UML with a more balanced view. UML is not the panacea that it was sometimes sold as 10 years ago. Nor is it as bad as some academics, agilistas, and competitors have claimed. It is a practical tool to raise the level of abstraction of software from code to system level. Many organizations claim benefits from their use of UML, ranging from improved communication among developers to productivity gains using code generation from UML models.

UML is a good language to describe software systems at a higher level than code. Properly used UML can improve productivity and quality. After several years of consolidation, the OMG (the owner of the UML) is taking the initiative to improve it, having issued a Request for Information (RIF) on the Future Development of UML in 2009 under the leadership of Steve Cook.

The results of this RFI showed that users and vendors want UML to be leaner, more expressive,

easier to learn, easier to integrate with other modeling and programming languages, and more relevant to today's technologies. For example, some users want to use UML to drive the interactive behavior of a mobile application. Others want to use UML diagrams to automatically visualize the structure and dependencies within a massive distributed service-oriented application. Some would like models to be deeply integrated with modern programming languages without any semantic conflicts. Increasingly, users would like to use UML to model applications deployed in the Cloud. To address these needs, the OMG and UML vendors are working together towards making UML smarter and more agile.

Size and Complexity

One of the biggest complaints about UML is that it is too large and too complex. Typically, a project that uses UML only uses 20% of the specification to help build 80% of the code and other deliverables. This 20% can vary according to the type of project: real-time systems, telecommunications, web applications, business applications, etc. What is considered essential may vary according to the kind of project, but in all cases the unused 80% obscures and complicates the essential.

To address this complaint, UML should be described differently for different user groups. There are ordinary users such as analysts, designers, website builders, database designers, developers, operators, architects, and testers, each bringing a different — but valid — perspective that uses different but overlapping subsets of UML. A particular class of users comprises the designers of UML itself and UML tool builders. It goes without saying that if the language is complex, these designers will have a hard time creating a language that is complete, consistent, extensible, and able to integrate with other languages, and the number of specification defects will become high.

Figure 1 depicts the main components of UML 2 and the dependencies between them. Although there is some layering, the overall structure contains a lot of cyclic dependencies, which makes it difficult to define useful subsets of the language. The UML specification does define formal “compliance points” that supposedly describe legal subsets of UML, but UML tool vendors have taken little or no notice of these, because they do not correspond to important use cases for UML users.

A key point with the current UML is that there is no way in a compliant implementation to use the simple version of a concept without having the complicated version; for example, take *Class*. Most users think of a *Class* as a simple thing that has attributes, operations, inheritance, etc. But a UML 2 class also has *Ports*, *Parts*, *Connectors*, and *Receptions* — concepts only useful in specialized domains. There is no way to have just the simple one, so all users are burdened with the understanding required by advanced users. This can be — and is — mitigated to some extent by good tools. However, we believe that the simple options should be inherent in the language definition itself. Furthermore, the UML *Class* differs in detail from the concept of *Class* found in any

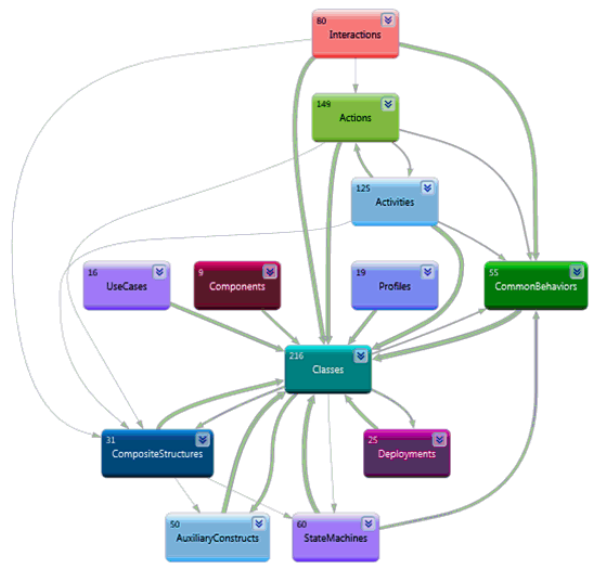


Figure 1: Components and dependencies of UML 2.

particular programming language, which introduces additional conceptual barriers between UML and those of its users who are software developers. Again, flexibility of these definitions should be inherent in the language, so that it can be fine-tuned to match modern programming technologies.

In summary, there are two major challenges to be addressed: complexity of the UML specification itself, and the need to describe UML in coherent subsets that address the actual needs of users in particular domains.

To address the first challenge, as a direct response to the feedback from the RFI, the OMG has embarked on a program of simplification for the UML specification. By the middle of 2011, a much simplified version of the UML specification should be available in which cyclic dependencies and redundancy have been greatly reduced. This specification will be compatible with existing tools and models, but described in a way that makes it much more amenable to further simplification and integration.

Refactoring UML

Once the simplification of the UML specification is complete in 2011, we will be able to move onto the next phase, which will be to refactor UML so that it can more effectively address the changing needs of many different classes of users. This section proposes some techniques that we can apply for this. It will be very important to retain backwards compatibility with the existing UML while doing this: We must not introduce changes that invalidate existing investments in tools, models, or training.

We suggest it is possible to create a very small kernel of no more than 20 elements such as objects, links, types, and actions, so that almost everyone can learn it in a few hours. In these elements, we will only add things that are necessary to understand

the kernel, so that it in a way becomes complete. The kernel by itself is not particularly useful to any developer as it is, although it might suffice for abstract conceptual modeling. It is intended to serve as a basis to describe useful elements.

On top of the kernel we now add more useful concepts. The most essential UML use cases can be defined as extensions to the kernel. An example of an essential use case of UML would be: "Developing from requirements to test;" see Figure 2.

The key idea here is that the kernel should be kept small and only include generic elements and attributes needed by most use cases of UML. Specific elements or attributes to existing elements in the kernel, which are needed to provide support for the use case "Developing from requirements to test", are added along with that use case. Thus they don't belong to the kernel. Everything new is added with the new use case, since it supports the use case. This is the whole point. This is what is called an "aspect-oriented structure", where the kernel can be kept clean and simple without knowledge of how it will be extended. The new additions to the kernel will be added with the new elements coming with the use cases that need them.

Returning to the use case example of "Developing from requirements to test"... There are many specializations of this use case; for example, "Developing a web application from requirements to

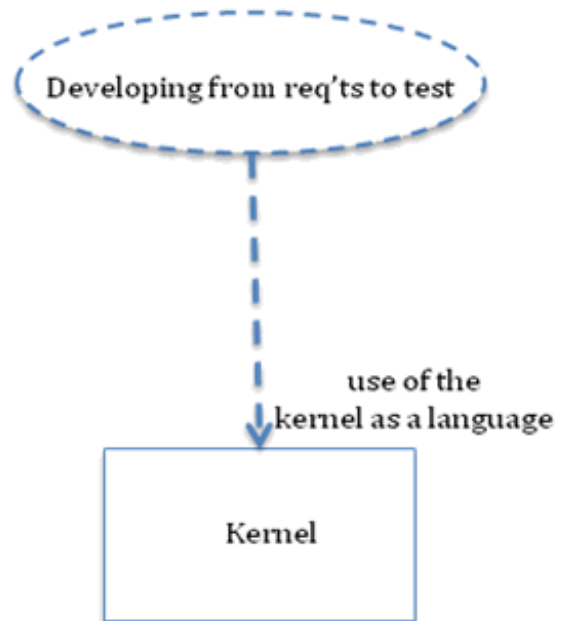
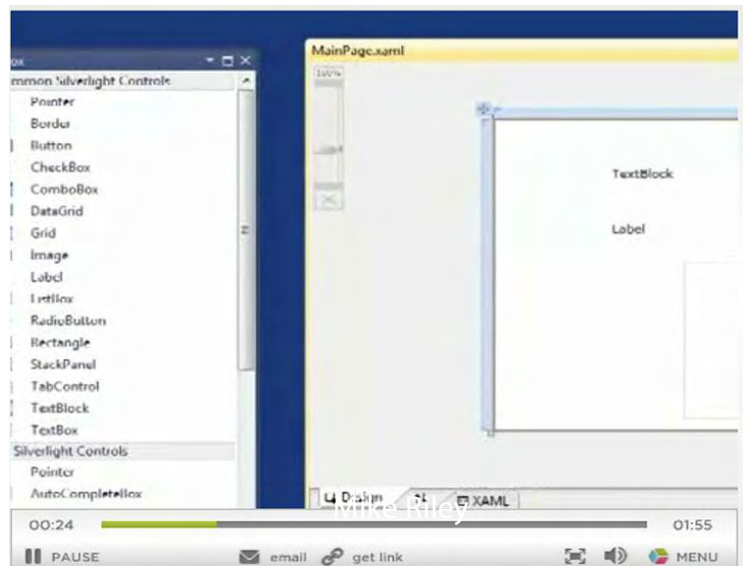


Figure 2: Suggested simplification of UML through separation of concerns.

Learn More About Microsoft's Visual Studio 2010

To learn more about Visual Studio 2010, visit **Dr. Dobb's Visual Studio 2010 Channel** at www.drdobbs.com/visualstudio, where you'll find feature articles, news, blogs, videos, and tweets devoted to all things VS 2010! In the video featured here, Dr. Dobb's contributing editor Mike Riley talks about new VS 2010 features like support for multiple monitors. Click the image to play the video on **Dr. Dobb's Visual Studio 2010 Video Library** at www.drdobbs.com/tv/ddjvisualstudio2010.jhtml.



Click screen to link to video

test”, and variants such as complex system and distributed system, which all wouldn't be essentials, but extensions to the essential use cases; see Figure 3.

Since the use case “Developing from requirements to test” in itself is a large use case, we would need to find its constituent smaller use cases: Use Case Modeling, Designing, Implementing, Testing, Deploying. They are all different aspects or separate concerns of UML (see “Aspect-Oriented Software Development With Use Cases”; <http://www.pearsonhighered.com/educator/product/AspectOriented-Software-Development-with-Use-Cases/>). Note that there is a danger in breaking down the complete use case into constituent use cases, which is that it might create silos and increase the risk of getting waste, contradicting principles of lean development. We mitigate this by using the Framework design principle where “each component should only do what only that component can do”. This forces clean minimal factoring; see Figure 4.

Technically, we need some enhancements to the mechanisms used to extend UML. Today there is a mechanism called “Profiles” (http://www.omg.org/technology/documents/profile_catalog.htm), which provides some of the required capability but is not well integrated with the rest of the UML architecture. A simple new mechanism for extending UML — multiple dynamic classification of model elements — is currently under development at the OMG, and is scheduled to be available as an OMG standard in 2011. This enables, for example, the specification of *Ports* to be completely separated from the specification of *Classes*, so that *Ports* can be added to *Classes* only for those advanced use cases where they are required.

Structuring UML like this will help eliminate today's dilemma of choosing between UML and one or more domain-specific languages (DSLs). When UML is structured as a simple kernel plus extensions, new domains can be addressed by crafting further extensions that can be seamlessly integrated with the existing structures. Another mechanism currently under development at the OMG — Diagram Definition — will enable new standard diagram types to be specified and integrated with existing ones. Diagram Definition is also scheduled to be available as an OMG standard in 2011, and will be applied to UML and to other modeling languages including BPMN (Business Process Model and Notation).

To help users get started we should define a basic subset of UML, here called “Essential UML,” which can be learnt quickly in a few days. This would include the simple aspects of most of the familiar UML diagrams. The rest of the UML can be added as a set of seamlessly interwoven, deeply integrated yet completely independent extensions without altering what already has been described and taught. This is smart!

During the last 10 years, UML has not adapted quickly enough to developments in the software world. Ten years ago, for example, we didn't have so many frameworks. Today we are inundated with frameworks. The work of programming has increasingly moved to become a job of connecting already developed components or to use existing frameworks. The amazing production of apps for mobile phones is a perfect example. Today, we have not found a place for UML in this context. That does not mean that such a

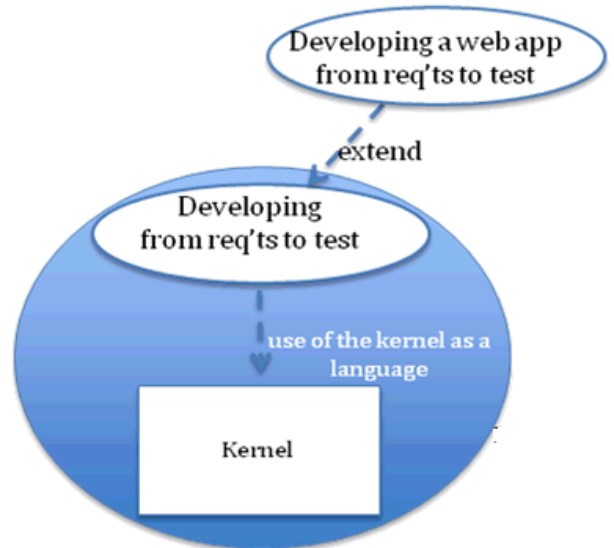


Figure 3: Adding a non-essential use case.

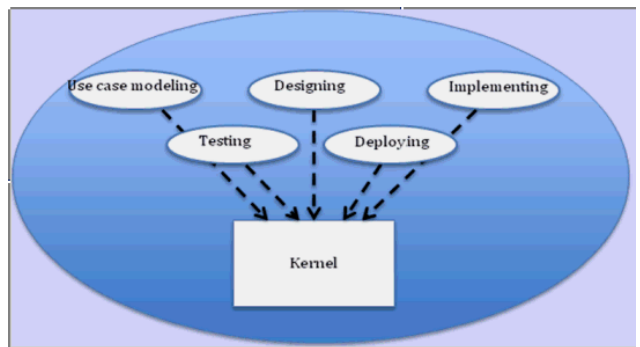


Figure 4: Constituent use cases of the larger use case “Developing from requirements to test”.

place does not exist, just that we have not yet adapted UML to this context or made UML appetizing to use in these contexts.

As we move forward with improving the UML, however, the first step is to simplify the existing UML, so we get it to become what UML should have been. We need to shrink UML first to lay the foundation for radical expansion of UML into many different new and exciting domains. As the basis for this work is being laid, we can begin to enrich UML with the possible new constructs we need today and tomorrow.

As a consequence of these developments, users can carry on investing in UML with confidence that the value of their investment will grow as UML moves into new domains, and becomes easier to apply and to integrate with other technologies.

— Ivar Jacobson is founder and CTO of Ivar Jacobson International and codeveloper of UML. Steve Cook is a software architect at Microsoft and represents Microsoft at the OMG.

[Return to Table of Contents](#)

Head First C#, 2nd Edition

Reviewed by Mike Riley

Head First C#, Second Edition
Andrew Stellman and Jennifer Greene
O'Reilly Media
\$49.99 US

I always look forward to new releases in the “Head First” series of books. The approach used in these titles remind me of high school textbooks presented in a much more energized, engaging way. This updated edition of *Head First C#* features the .NET 4.0 framework and the Visual Studio 2010 IDE. I didn’t read the first edition, so I can’t comment on what has changed, but the material related to the latest .NET and Visual Studio features is relevant and well represented.

Head First C# follows the progression one would expect in an introductory programming text, but not quite in the style “Head First” employs. Building a Windows Form-based application connecting to a contact database has never been presented quite this way before. Copious screenshots, sketches, hand-written notes, and illustrations punctuate the material, especially with code listings. There are even crossword puzzles concluding some of the chapters to help emphasize the concepts being taught! Some may find this style distracting, but I enjoyed seeing this detail presented in this new, information-rich way. Debugging, object-oriented methodologies, types, references, enum, collections, exceptions, events and delegates are each given space to help readers get comfortable with how C# uses these aspects and why they are so powerful. The 15th and final chapter on LINQ is one of the best I’ve seen in terms of helping readers grasp as quickly as possible the power of this remarkable and potent technology.

Three lab exercises (“A Day at the Races,” “The Quest,” and “Invaders”) help readers assemble real-world applications leveraging what was learned in the sections preceding these labs. They do a super job of showing how the foundational aspects of what C# provides can be put into gratifying use.

Coauthors Andrew Stellman and Jennifer Greene effectively use a potpourri of images, callouts, gags, and highlights to maintain readers’ interest and understanding of the topics covered in the book. While these may be too distracting to more serious readers, I found them helpful and, yes, a bit corny at times. But they kept my attention and helped reinforce the less obvious aspects of C# in action. For those seeking a more serious but very effective large trim size book, I recommend holding out for Mike Murach & Associates treatment on the new release, as I found their C# 2008 book an excellent, straight-talking guide to learning the language.

Like most new book titles I review these days, I read the electronic edition of *Head First C#*. I prefer to read EPUB versions

due to the fact that they are based on industry standard, open technologies like XHTML and CSS. As such, they are better optimized for text reflow and presentation on electronic reading devices like Sony’s E-Reader and Apple’s iPad. There is also a more personal reason for my format preference since I worked on the predecessor to the EPUB specification, known as the Open eBook Publication Structure. Unfortunately, for a book like *Head First C#* with its highly graphical presentation and need to maintain fixed positioning of these images for print-centric output, converting such a title to EPUB would take a considerable amount of effort and cost. As such, O’Reilly only offers the title in PDF format. I used GoodReader on the iPad to read the book, and while GoodReader is a super application for rendering PDF and Microsoft Office-formatted documents, it still can’t compare to the less constrained reading experience that EPUB-rendering apps like Apple’s iBooks provides. Even with the iPad’s 768x1024 portrait screen resolution, I had to occasionally resize, scroll, reset, rinse-cycle-repeat, to adequately read the book and its smaller font call-outs. In the larger paper book edition, these page design choices work. However, in the PDF, they are a big pain. Consequently, I would advise those interested in reading this book to obtain the print edition rather than struggle through the electronic formatted version. Another bone of contention regarding the PDF is that O’Reilly resampled the images (except for the first two pages — the cover) for black and white reproduction only. This makes sense for the print version to keep costs low, but the electronic edition could have greatly benefited from the color imagery, especially with color-highlighted code listings, application screen shots, and even the call-out photos. With the cost of the PDF edition only a few dollars less than the print version, there are really no other advantages to the electronic edition of the book.

My electronic publishing rant aside, the content of *Head First C#* is solid and entertaining. Seasoned developers need not apply, as the “Head First” series best caters to the younger, less experienced crowd who are interested in learning new technologies in a fun, approachable way.

[Return to Table of Contents](#)

Q&A: CodePlex and Open Source Software

CodePlex executive director Paula Hunter takes a few minutes to talk

by Jonathan Erickson

As executive director of the CodePlex Foundation (www.codeplex.org/), Paula Hunter has a lot of irons in the fire. She recently took some time to chat with Jonathan Erickson.

Dr. Dobb's: Paula, can you give us a brief overview of the CodePlex Foundation?

Hunter: The CodePlex Foundation is a new, independent open source foundation that enables corporations to innovate with more agility by collaborating with open source communities. Our core mission is to provide governance and legal mechanisms necessary to ensure clarity and certainty for corporations that want to participate in open source. We also develop and promote best practices, processes, and legal structures that meet the needs of open source developers.

We're platform, technology, and license agnostic — which is how we differ from other foundations. We made a conscious effort to differentiate ourselves in this way. This allows us to provide unique support and services that complement the important work of other foundations.

We formed to encourage greater use of OSS by corporations and developers by providing support and services to achieve the following objectives:

- Make it easy to participate in open source
- Provide a standard, but flexible contribution agreement
- Create a legal entity/framework that enables code to be shared easily with reduced risk
- Lower legal risk and costs of bringing open source projects into the commercial world
- Reduce the time, legal complexity, and expense of contributing projects to the open source community

Dr. Dobb's: To date, has the Foundation enjoyed any “big wins?”

Hunter: We launched in September 2009, and I think our biggest win is what we've accomplished in such a short time. One of those first early milestones was the publication of our Project Acceptance and Operations Guidelines where we mapped out our mission and guidelines for submitting a project to the Foundation. We introduced a “Project Gallery” model as an organizational structure for managing Foundation projects. The museum metaphor provides a way to group projects within a taxonomy, similar to how art collections are organized and presented in a museum exhibition. For example, sponsors can host entire galleries of projects with related applications and technologies. Introducing these guidelines has helped us attract interesting projects and project sponsors, and we're picking up momentum.

So far, we have the ASP.NET Open Source Gallery that has four active projects. The Web Forms MVP project is the most recent addition to this gallery. We also have the Systems Infrastructure and Integration Gallery, consisting of two active projects with more on the way.

We'll continue growing these galleries and adding new ones along the way. That's the beauty of it. The community can have a strong influence on the types of projects and applications we support simply by submitting to sponsor a gallery.

Dr. Dobb's: What will the Foundation be like two years from now?

Hunter: More projects, more galleries, more sponsors, more community, more impact, more



independence. I've set a goal for 2010 of adding at least two sponsors, at least two more galleries, and at least one additional board member. In two years I plan to have more sponsors — which will prove our independence for those who are still skeptics — and a tiered sponsorship model. I'm working to attain 501.6c status as a non-profit organization, and looking at different models under which we can accept members at different levels of participation and contribution. The goal is to diversify the organization, increasing our footprint, influence, and credibility in the OSS community, and demonstrating our value to commercial software companies, non-software companies that use OSS and enterprise IT.

Dr. Dobb's: Looking beyond the Foundation, what do you see as opportunities for developers in the next year or so?

Hunter: Open source developers have become mainstream. The big opportunity for them is to take advantage of the status they deserve as major influencers in the software development world. Open source dominates in cloud and mobile and is a large part of the virtualization market. As the focus moves from operating systems to cloud-based, virtualized environments delivered to end users on mobile devices, the opportunity for open source developers just grows. It's a very exciting time to be part of the community.

Dr. Dobb's: As executive director of a not-for-profit entity, what's the toughest part of your job?

Hunter: Managing expectations is the toughest job. I have to answer to the Board, take advice from the Board of Advisors, honor the intent of our founding sponsor Microsoft, and provide an operational and technical environment in which our galleries and projects flourish. It's a balancing act I've done before at nonprofits and open source foundations — I expected it, but it's always a challenge.

Dr. Dobb's: Your biggest surprise?

Hunter: I have been pleasantly surprised at the level of support and readiness to discuss our mission within the open source community. Skeptics often have the loudest voice, and I heard and read their complaints before I accepted this position, but the people that I have worked with over the last decade, who represent some of the best and brightest open source developers and business professionals in the industry, have been encouraging and supportive. That's not to say that they don't have some criticism, but thus far it has been very constructive.

[Return to Table of Contents](#)



Apples in the Walled Garden

By Avo Reid

Apple wants its users to have an optimal experience using its devices. That's what attracted people to the Macintosh and created the cult-like following that defined Mac Users and PC Users.

But it is interesting that not until Apple made the Intel Transition (for details, see http://en.wikipedia.org/wiki/Apple%E2%80%933Intel_transition), changing the CPU of Macintosh computers from PowerPC processors to Intel x86 processors, in 2006–2007, that sales really took off, especially for laptop sales. The gate to the walled garden was opened to all who might wander in to take a gander.

Arguably, many of these new users were like me, intrigued by Apple products, functionality, and usability but hesitant to commit because of the high price to pay in terms of leaving behind the years of investment and familiarity in Windows.

What sealed the deal for me was that the Mac now appeared to be more open, at least a little more open than before. Now I could install VMWare Fusion and have access to Windows and all the software I use for business, but still be able to use the Mac OS X and all of its software. I was a new customer and I have to say it will take a lot to get me to switch back.

So why does Apple act so guarded against anyone entering its Walled Garden? Is it really so that the user will have an optimal experience? I have to say I probably would not have become a user unless they opened the gate the way they did in 2006, intentionally or not.

Recently, Apple has been making the case that Adobe Flash is an inferior, evolutionary 'dead-end' technology that won't be allowed entrance into the Walled Garden; i.e., Flash Videos and Flex/Flash Applications will never run on an iPhone or iPad. As Steve Jobs stated at the 2010 D8 Conference (<http://allthingsd.com/d/>), "We don't think Flash makes a great product, so we're leaving it out...Instead, we're going to focus on

technologies that are in ascendancy. If we succeed, people will buy them and if we don't they won't."

In his now famous "Thoughts on Flash" memo, (<http://www.apple.com/hotnews/thoughts-on-flash/>), Jobs writes, "Flash was created during the PC era — for PCs and mice. Flash is a successful business for Adobe, and we can understand why they want to push it beyond PCs. But the mobile era is about low power devices, touch interfaces and open web standards — all areas where Flash falls short."

But what's most interesting about the memo is the definition of 'openness'. Steve Jobs writes that "Adobe's Flash products are 100% proprietary. They are only available from Adobe, and Adobe has sole authority as to their future enhancement, pricing, etc. While Adobe's Flash products are widely available, this does not mean they are open, since they are controlled entirely by Adobe and available only from Adobe. By almost any definition, Flash is a closed system."

He goes on to admit that "Apple has many proprietary products too. Though the operating system for the iPhone, iPod, and iPad is proprietary, we strongly believe that all standards pertaining to the Web should be open. Rather than use Flash, Apple has adopted HTML5, CSS, and JavaScript — all open standards." We might also point out here that there is currently only one way to develop and publish an iPhone Application today, and that is to use Apple products.

John Warnock and Chuck Geschke in their rebuttal "Our Thoughts on Open Markets" (<http://www.adobe.com/choice/openmarkets.html>) state, "If the web fragments into closed systems, if companies put content and applications behind walls, some indeed may thrive — but their success will come at the expense of the very creativity and innovation that has made the Internet a revolutionary force."

They continue to argue for the consumer, "We believe that consumers should be able to freely

access their favorite content and applications, regardless of what computer they have, what browser they like, or what device suits their needs. No company — no matter how big or how creative — should dictate what you can create, how you create it, or what you can experience on the Web.”

This view of the world is reiterated at the “Open Screen Project” (<http://www.openscreenproject.org/>) an industry-wide initiative, led by Adobe and backed by other industry leaders (http://www.openscreenproject.org/partners/current_partners.html) such as Google, RIM, Intel, and Nokia, who as the site says “all share one clear vision: Enable consumers to engage with rich Internet experiences seamlessly across any device, anywhere.”

The Open Screen Project asserts that “The number and diversity of devices in our lives is exploding. Consumers want and demand the total Internet, with open access to websites, applications, and services using all devices. The challenge is that fragmentation across devices, operating systems, and browsers hinders innovation. The result? Consumer demands are not being met. The Open Screen Project was established to meet these challenges and expectations. It is an industry-wide initiative, led by Adobe with the participation of other industry leaders, to enable the delivery of rich multiscreen experiences built on a consistent runtime environment for open web browsing and standalone applications.”

The Open Screen Project goes on to make an appeal to developers: “With a consistent and broadly adopted runtime, stakeholders can focus more closely on developing the next-generation experiences that will differentiate their devices, software, and services. Consumers ultimately win with richer, more interactive, and universal user experiences across devices.”

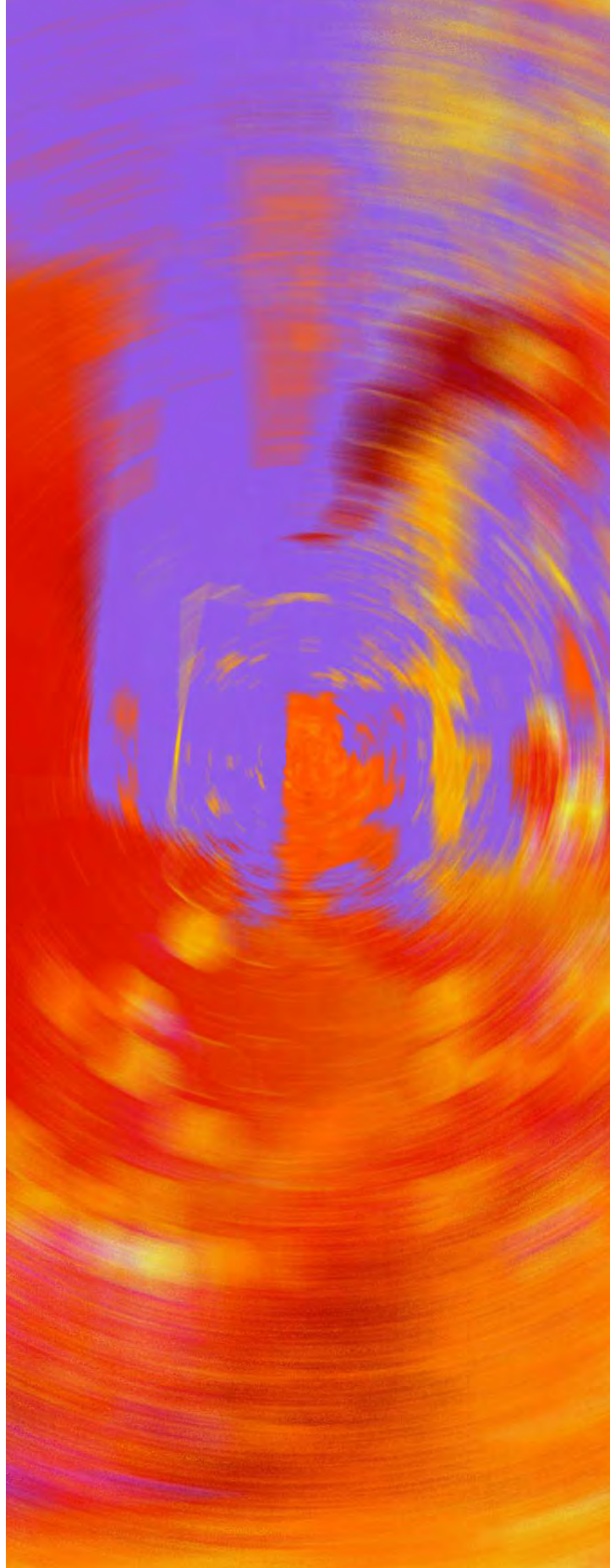
Steve Jobs makes his appeal to developers in the final paragraphs of his memo: “Our motivation is simple — we want to provide the most advanced and innovative platform to our developers, and we want them to stand directly on the shoulders of this platform and create the best apps the world has ever seen. We want to continually enhance the platform so developers can create even more amazing, powerful, fun, and useful applications. Everyone wins — we sell more devices because we have the best apps, developers reach a wider and wider audience and customer base, and users are continually delighted by the best and broadest selection of apps on any platform.”

While it is impossible to say how all this will play out, what is certain is that portability between devices, at least between Apple devices and everything else will be an issue for developers in the coming years, at least until Apple owns the market.

[CLICK HERE TO COMMENT ON THIS POST](#)

http://www.drdobbs.com/blog/archives/2010/06/apple_wants_its.html

[Return to Table of Contents](#)



Automation, Economics, and the Birth of Market-Driven IT

As IT efficiency increases, switching costs decrease, liquidity emerges, and a true market is born

By Jake Sorofman

Recently, I had a conversation with a well-known industry analyst — a very creative guy drawn to big ideas. He got me thinking about a big idea that has been rattling around in my brain to some degree of distraction. In my estimation, the idea is nothing short of transformational to the economics of enterprise computing.

It's a story about taking back control of IT and truly rationalizing costs. The story has two parts:

- Part 1: Returning control and flexibility to IT by making system workloads autonomous, self-healing, and portable across deployment environments.
- Part 2: Allowing IT leadership to make dynamic decisions about where workloads should run based on optimizations for price and service levels. Think of this as the logical evolution of IT financial management.

In combination, these two concepts have the potential to change everything.

IT has long been beholden to outsourced service providers for managing systems. In some cases, the comparative cost of outsourcing system management is lower due to scale economies and other built-in efficiencies. But, more often than not, it's the predictability of a contract relationship that IT leadership covets: Fixed bid contracts are easily forecast and service-level agreements allow IT to share some risk.

But the cost of admission is exceptionally high. For a very large company a contract may exceed \$40 or \$50 million — a year.

Why have CIOs been willing to spend this much?

- Efficiency. Traditionally, it has been more efficient to throw systems “over the wall” due to service providers’ comparative cost advantage through specialization.
- Switching costs. Once your systems are deployed, it has been too expensive to move them to a new provider; you're locked in — and so is your price.

- Liquidity — or the absence thereof. Traditional pricing models have forced enterprises to engage in onerous, long-term commitments, removing the liquidity that drives market-efficient pricing.

But this is changing — and fast:

- Efficiency. Emerging approaches to automation make system scale and complexity far less prohibitive and intimidating for enterprise IT organizations to handle in-house.
- Switching costs. Deep automation also allows you to regenerate systems that are ready to deploy to new environments, setting workloads free to run anywhere.
- Liquidity. At the same time, elastic computing models allow you to pay as you go. Systems can be moved and capacity can be consumed on demand.

IT efficiency increases, switching costs decrease, liquidity emerges, and a true market is born. Then, introduce decision support for driving price and SLA optimizations and CIOs can dramatically improve the efficiency of IT spending.

As context for what is possible, consider the achievements we've seen in modern manufacturing. Supply and capacity are dynamically shifted to the highest-yield demand. Capacity is allocated based on contract values, delivery commitments, and the on-hand availability of raw materials. Raw materials are procured based on competitive bids. Production runs are scheduled against available capacity.

The entire manufacturing process is automated and dynamically optimized. Why shouldn't this be the inspiration for the future of enterprise IT?

— Jake Sorofman works for rPath, a provider of automated application deployment tools. You can contact Jake at jsorofman@rpath.com.

[Return to Table of Contents](#)

Prefer Using Active Objects Instead of Naked Threads

How to automate best practices for threads and raise the semantic level of our code

By Herb Sutter

As described in “Use Threads Correctly = Isolation + Asynchronous Messages” (www.drdobbs.com/high-performance-computing/215900465 [1]), to use threads well we want to follow these best practices:

- Keep data isolated, private to a thread where possible. Note that this doesn't mean using a special facility like thread local storage; it just means not sharing the thread's private data by exposing pointers or references to it.
- Communicate among threads via asynchronous messages. Using asynchronous messages lets the threads keep running independently by default unless they really must wait for a result.
- Organize each thread's work around a message pump. Most threads should spend their lifetime responding to incoming messages, so their mainline should consist of a message pump that dispatches those messages to the message handlers.

Using raw threads directly is trouble for a number of reasons, particularly because raw threads let us do anything and offer no help or automation for these best practices. So how can we automate them?

A good answer is to apply and automate the Active Object pattern [2]. Active objects dramatically improve our ability to reason about our thread's code and operation by giving us higher-level abstractions and idioms that raise the semantic level of our program and let us express our intent more directly. As with all good patterns, we also get better vocabulary to talk about our design. Note that active objects aren't a novelty: UML and various libraries have provided support for active classes. Some actor-based languages already have variations of this pattern baked into the language itself; but fortunately, we aren't limited to using only such languages to get the benefits of active objects.

This article will show how to implement the pattern, including a reusable helper to automate the common parts, in any of the popular mainstream languages and threading environments, including C++, C#/NET, Java, and C/Pthreads. We will go beyond the basic pattern and also tie the active object's private thread lifetime directly to the lifetime of the active object itself, which will let us leverage rich object lifetime semantics already built into our various languages. Most of the sample code will be shown in C++, but can be directly translated into any of the other languages (for example, destructors in C++ would just be expressed as disposers in C# and Java, local object lifetime in C++ would be expressed with the *using* statement in C# or the “finally dispose” coding idiom in Java).

Active Objects: The Caller's View

First, let's look at the basics of how an active object is designed to work.

An active object owns its own private thread, and runs all of its work on that private thread. Like any object, an active object has private data and methods that can be invoked by callers. The difference is that when a caller invokes a method, the method call just enqueues a message to the active object and returns to the caller immediately; method calls on an active object are always nonblocking asynchronous messages. The active object's private thread mainline is a message pump that dequeues and executes the messages one at a time on the private thread. Because the messages are processed sequentially, they are atomic with respect to each other. And because the object's private data is only accessed from the private thread, we don't need to take a mutex lock or perform other synchronization on the private shared state.

Here is an example of the concurrency semantics we want to achieve for calling code:

```
Active a = new Active();
a.Func();           // call is nonblocking
... more work ...   // this code can execute in parallel with a.Func()
```

Next, we want to tie the lifetime of the private thread to the lifetime of the active object: The active object's constructor starts the thread and the message pump. The destructor (or “disposer” in C# and Java) enqueues a sentinel message behind any messages already waiting in the queue, then joins with the private thread to wait for it to drain the queue and end. Note that the destructor/disposer is the only call on the active object that is a blocking call from the viewpoint of the caller.

Expressing thread/task lifetimes as object lifetimes in this way lets us exploit existing rich language semantics to express bounded nested work. For example, in C# or Java, it lets us exploit the usual language support and/or programming idiom for stack-based local lifetimes by writing a using block or the Dispose pattern:

```
// C# example
using( Active a = new Active() ) {    // creates private thread
    ...
    a.SomeWork();                     // enqueues work
    ...
    a.MoreWork();                     // enqueues work
    ...
} // waits for work to complete and joins with private thread
```

As another example, in C++ we can also express that a class data member is held by value rather than by pointer or reference, and the language guarantees that the class member's lifetime is tied to the enclosing object's lifetime: The member is constructed when the enclosing object is constructed, and destroyed when the enclosing object is destroyed. (The same can be done by hand in C# or Java by manually wiring up the outer object's dispose function to call the inner one's.) For example:

```
// C++ example
class Active1 {           // outer object
    Active2 inner;         // embedded member, lifetime implicitly
                           // bound to that of its enclosing object
};

// Calling code
void SomeFunction() {
    Active1 a;             // starts both a's and a.inner's private threads
    ...
}                           // waits for both a and a.inner to complete
```

That's what we want to enable. Next, let's consider how to actually write a class with objects that are active and behave in this way.

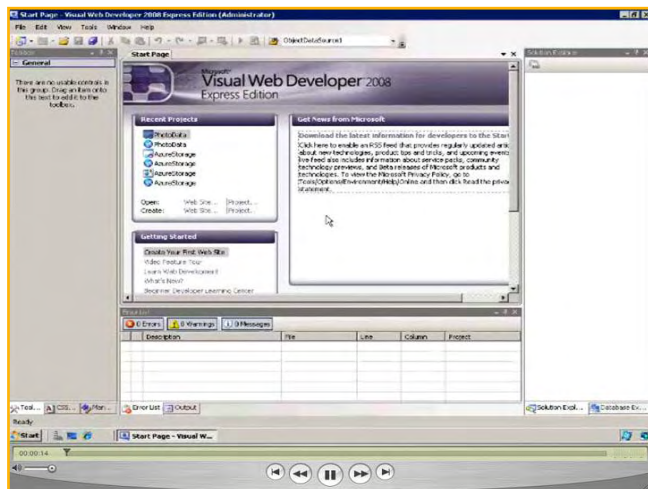
An Active Helper

The first thing we want to do is to automate the common parts and write them in one reusable place, so that we don't have to write them by hand every time for each new active class. To do that, let's introduce a helper class *Active* that encapsulates a thread behind a messaging interface and provides the basic message-sending and automatic pumping facilities. This section shows a straightforward object-oriented implementation that can be implemented as shown in any of our major languages (even in C, where the class would be written as a *struct* and the object's methods would be written as plain functions with an explicit opaque “this” pointer). After that, we'll narrow our focus to C++ specifically and look at another version we can write and use even more simply in that language.

The *Active* helper below provides the private thread and a message queue, as well as a sentinel *done* message that it will use to signal the private thread to finish. It also defines a nested *Message* class that will serve as the base of all messages that can be enqueued:

How Do I: Use Windows Azure Storage With My Applications

Windows Azure Storage provides table, blob, and queue storage for your application. In this video Scott Golightly will walk through a sample application that will take advantage of Windows Azure Storage to communicate between different application roles and complete a business task. Click screen to watch a video explaining these details about Azure on Dr. Dobb's Microsoft Resource Center.



Click screen to link to video



I KNOW A PLACE THAT'S DIFFERENT, BUT FAMILIAR

Code in the cloud ... with the development technologies you already know

With **Windows Azure™**, you have a flexible, on-demand infrastructure that taps into all the IT resources you need. The advantages?

You have the freedom to focus.

With the cloud as your development platform, you can focus on what you do best: create applications, without the restrictions on performance, capacity, or deployment that come with using your own hardware rather than the cloud.

You can code in the language you choose.

Windows Azure enables you to build applications in multiple languages, including .NET, PHP, and Java.

You're already equipped for the cloud.

You can start developing applications for Windows Azure right away, using familiar tools such as Microsoft® Visual Studio® 2008.

Find out more.

See how Windows Azure can help get you coding in the cloud.
Go to **windowsazure.com** now.

```
// Example 1: Active helper, the general OO way
//
class Active {
public:

    class Message {          // base of all message types
    public:
        virtual ~Message() { }
        virtual void Execute() { }
    };

private:
    // (suppress copying if in C++)

    // private data
    unique_ptr<Message> done;           // 1e sentinel
    message_queue< unique_ptr<Message> > mq; // 1e queue
    unique_ptr<thread> thd;             // 1e thread
```

Note that I'll assume *message_queue* is a thread-safe internally synchronized message queue that doesn't need external synchronization by its caller. If you don't have such a queue type handy, you can use an ordinary queue type and protect it with a mutex.

With that much in place, *Active* can go on to provide the common machinery, starting with the message pump itself:

```
private:
    // The dispatch loop: pump messages until done
    void Run() {
        unique_ptr<Message> msg;
        while( (msg = mq.receive()) != done ) {
            msg->Execute();
        }
    }

public:
    // Start everything up, using Run as the thread mainline
    Active() : done( new Message ) {
        thd = unique_ptr<thread>(
            new thread( bind(&Active::Run, this) ) );
    }

    // Shut down: send sentinel and wait for queue to drain
    ~Active() {
        Send( done );
        thd->join();
    }

    // Enqueue a message
    void Send( unique_ptr<Message> m ) {
        mq.send( m );
    }
};
```

The Active Cookbook: A Background Worker

Now let's use the helper and actually write an active class. To write a class with objects that are active, we must add an *Active* helper and then for each public method:

- Have the public method capture its parameters as a message and call *Send* to send the call for later execution on the private thread. Note that these functions should not touch any member variables. They should use and store their parameters and locals only.
- Provide a corresponding nonpublic class derived from *Message* with constructor and data members that will capture the parameters and *this* pointer, and with an *Execute* method that will perform the actual work (and can manipulate member variables; these *Execute* methods should be the only code in the class that does so).

For example, say we want to have an agent that does background work that we want to get off a responsive thread, including the performance of background save and print functions. Here's how we can write it as an active object that can be launched from the responsive thread using asynchronous method calls like *b.Save("filename.txt")* and *b.Print(myData)*:

```
class Backgrounder {
public:
    void Save( string filename ) {
        a.Send( new MSave( this, filename ) );
    }

    void Print( Data& data ) {
        a.Send( new MPrint( this, data ) );
    }

private:
    class MSave : public Active::Message {
        Backgrounder* this_;    string filename;
    public:
        MSave( Backgrounder* b, string f ) : this_(b), filename(f) {
```

Cloud Test Solution With Azure:

SOASTA developed a cloud-based testing application that it now deploys on the Windows Azure platform. With the Windows Azure platform, SOASTA has easy access to servers all over the world under a pay-as-you-go pricing model. Click screen to watch a short video explaining these details about Azure on Dr. Dobb's Microsoft Resource Center.



Click screen to link to video

```

}
    void Execute() { ... }    // do the actual work
};

class MPrint : public Active::Message {
    Backgrounder* this_;    Data& data;
public:
    MPrint( Backgrounder* b, Data& d ) : this_(b), data(d) { }
    void Execute() { ... }    // do the actual work
};

// Private state if desired
PrivateData somePrivateStateAcrossCalls;

// Helper goes last, for ordered destruction
Active a;
};

```

Now we can simply enqueue work for the background thread as messages. If the worker's private thread is idle, it can wake up and start processing the message right away. If the worker is already busy, the message waits behind any other waiting messages and all get processed one after the other in FIFO order on the private thread. (Teaser: What if the background thread wants to report progress or output to the caller, or a side effect such as reporting that a *Print* message is done using the shared *data*? We'll consider that next month.)

A C++0x Version

The aforementioned OO-style helper works well in any mainstream language, but in any given language we can often make it even easier by using local language features and idioms. For example, if we were writing it in C++, we could observe that *Message* and its derived classes are simply applying the usual “OO way” of rolling your own function objects (or functors), and *Execute* could as well be spelled *operator()*, the function call operator. The only reason for the *Message* base class is to provide a way to hold and later invoke an arbitrary message, whereas in C++0x we already have *std::function<>* as a handy way to hold and later invoke any suitable callable function or functor.

So let's leverage the convenience of C++ function objects. We'll avoid a lot of the “OO Message hierarchy” boilerplate. *Active* will be a simpler class. Derived classes will be easier to write. What's not to like?

```

// Example 2: Active helper, in idiomatic C++(0x)
//
class Active {
public:
    typedef function<void()> Message;

private:
    Active( const Active& );           // no copying
    void operator=( const Active& );  // no copying

    bool done;                        // 1e flag
    message_queue<Message> mq;         // 1e queue
    unique_ptr<thread> thd;            // 1e thread

    void Run() {
        while( !done ) {
            Message msg = mq.receive();
            msg();                      // execute message
        } // note: last message sets done to true
    }

public:
    Active() : done(false) {
        thd = unique_ptr<thread>(

```

```

        new thread( [=]{ this->Run(); } ) );
    }

    ~Active() {
        Send( [&]{ done = true; } ); ;
        thd->join();
    }

    void Send( Message m ) {
        mq.send( m );
    }
};

```

You may never need or want to write a
naked thread again

Next, we can use the lambda functions language feature to make an implementing class like *Backgrounder* even simpler to write, because we don't even have to write the external “launcher” method and the actual body in different places...we can simply write each method the same way we write it naturally, and send the body of the message as a lambda function:

```

class Backgrounder {
public:
    void Save( string filename ) { a.Send( [=] {
        ...
    } ); }

    void Print( Data& data ) { a.Send( [=, &data] {
        ...
    } ); }

private:
    PrivateData somePrivateStateAcrossCalls;
    Active a;
};

```

This isn't just a C++ trick, by the way. If you're using C#, which also has generalized delegates and lambda functions, you can do likewise. Here's a sketch of how the simplified *Backgrounder* code would look in C#:

```

class Backgrounder : IDisposable {
    public Backgrounder() { /*...*/ a = new Active(); }
    public Dispose() { a.Dispose(); }

    public void Save( String filename ) { a.Send( () => {
        ...
    } ); }

    public void Print( Data data ) { a.Send( () => {
        ...
    } ); }

private:
    PrivateData somePrivateStateAcrossCalls;
    Active a;
};

```

Summary

Unlike with free threading, which lets us randomly do anything at all, active objects make it easier to express what we mean by automatically organizing the private thread's work around an event-driven message pump, naturally expressing isolated private data as simple member data, and offering strong lifetime guarantees by letting us express thread and task lifetime in terms of object lifetime (and therefore directly leverage the rich support for object lifetime semantics already built into our programming languages). All of this raises the semantic level of our program code, and makes our program easier to write, read, and reason about. Major uses for active objects include the same motivating cases as for any threads: to express long-running services (for example, a physics thread or a GUI thread); to decouple independent work (for example, background save, or pipeline stages); to encapsulate resources (for example, I/O streams or certain shared objects). You may never need or want to write a naked thread again.

Coming Up

So far, we've looked only at two of the basics of active objects, namely their lifetimes and asynchronous method call semantics. Next month, we'll complete the overview by considering important

remaining details — how to handle a methods' return values and/or *out* parameters, and other kinds of communication back to the caller. Stay tuned.

References

- [1] H. Sutter. "Use Threads Correctly = Isolation + Asynchronous Messages" (*Dr. Dobb's Digest*, April 2009: <http://www.drdobbs.com/high-performance-computing/215900465>).
- [2] R. Lavender and D. Schmidt. "Active Object: An Object Behavioral Pattern for Concurrent Programming" (update of paper published in *Pattern Languages of Program Design 2*, Addison-Wesley, 1996: <http://www.cs.wustl.edu/~schmidt/PDF/Act-Obj.pdf>).
- [3] The ADAPTIVE Communication Environment (ACE): <http://www.cs.wustl.edu/~schmidt/ACE.html>.

— Herb Sutter is a bestselling author and consultant on software development topics, and a software architect at Microsoft. He can be contacted at www.gotw.ca.

[Return to Table of Contents](#)

