RECORD, Volume 27, No. 3*

New Orleans Annual Meeting
October 21-24, 2001

Session 52SM/OF/L
Introduction to J/ACORD XML Standards

Track: Computer Science

Chairperson: MARK D. J. EVANS
Lecturers: CHRISTOPHER D. BURKE
ROBERT J. MARONE†

Summary: This session provides an opportunity for Computer Science Section members to learn about current section activities and to provide input to the Section Council. A brief introduction to the J computer language is provided. The session also provides an open forum to discuss other new technologies and their impact on he actuarial profession.

MR. CHARLIE LINN: We have two speakers this morning. The first speaker this morning will be Chris Burke from J Software, who is going to give us a presentation on the J computer language and its evolution and uses. The second speaker is Rob Marone, who is the standards evangelist from the Association for Cooperative Organization Research & Development (ACORD). He will be telling us about what ACORD does with a concentration on the XML standards specifically. As we mentioned, we are hoping to work with ACORD to establish data standards that we have been working on. Hopefully in the near future we will be setting up a joint working group with ACORD and the Society of Actuaries to work toward that.

MR. CHRISTOPHER D. BURKE: I'm here to talk about the J programming language. I should also explain that I currently wear two hats. I am partner in the firm of J Software, which produces the J language, but also for the last year or so, I have worked as a full-time consultant for a company called Luen Thai, based in Hong Kong. You probably have not heard of them, but you almost certainly have one or more of their products. Luen Thai is one of the largest garment

*Copyright © 2002, Society of Actuaries
†Mr. Robert J. Marone, not a member of the sponsoring organizations, is standards evangelist from the Association for Cooperative Organization Research & Development (ACORD) in Pearl River, NY

manufacturers in the world, selling quality garments primarily to the USA.

The History of J
In talking about J, I want to try to answer two fundamental questions. First of all, why would actuaries be interested in J? Well, it is really because of our focus for the language. In developing J, we focused on doing numerical computations, expressing complex algorithms, analyzing large amounts of data, and doing so in a very rapid development framework.

Of course, people who produce other software can also make similar claims. But this is our focus, and we don't really worry about anything else. We think that we've done a good job solving these kinds of problems. These are typical actuarial problems. When you do modeling or valuations, then you do numerical computations, or you're analyzing a lot of data. So we do have actuaries using J, and we are also encouraging more.

Second, why develop another programming language? There are lots of programming languages. Well, there's actually an interesting story here.

Fun. The original reason was that it was a lot of fun. It was neat to work on a new language that did what we thought was the right thing in programming languages. We didn't much like C or VB or FORTRAN, and we felt we had a better idea.

Applicable. After a couple of years, better reasons for developing the language came along. In particular, we found applications in which the language works well. These applications were typically in the financial services business, but there are applications in many other businesses as well. However, we found that banking, investment, and insurance had the kind of problems that our language is really good at.

The developers all have a lot of experience with interpreted array base languages. We all have similar backgrounds, and I also should say that all of us have programmed in many different languages, as well; so we have wide experience. But in particular, we were very enthusiastic about this type of approach.

Clients. And then the third reason is clients.

Clients are always surprising. If you have ever worked as a consultant, your first clients always come as a surprise. Why would anyone call me up and say, "We'd like to bring you over and pay for your services?" But we started getting clients for J, and they were very enthusiastic clients and were willing to spend a lot of money, because they found that the particular language we developed was ideally suited for the particular problems they were solving.

In many cases, the clients we had were people who had spent a lot of time and a lot of money trying to develop applications in standard languages such as C and completely failed. They would spend a *long* time doing C. Then they came to us and said, "Can you do this in J?" And very quickly, we would say, "Yes we can. Here's the solution."

This point is actually fairly typical: People tend to use oddball languages or use consultants when all else fails. They've tried everything, and it doesn't work, so they come to use something unusual. But we have developed a good client base, and that's really kept the language going and kept the language expanding.

Complementary
How does J compare with things like Excel, C++, and VB—all standard languages? Well, we really don't compete with them. And we don't want to compete with them. We think that we complement them, and we do things that these languages do not do particularly well.

Almost all our serious clients do not use pure J. Instead they use one of these languages—Excel or VB—and they call J to do their calculations.

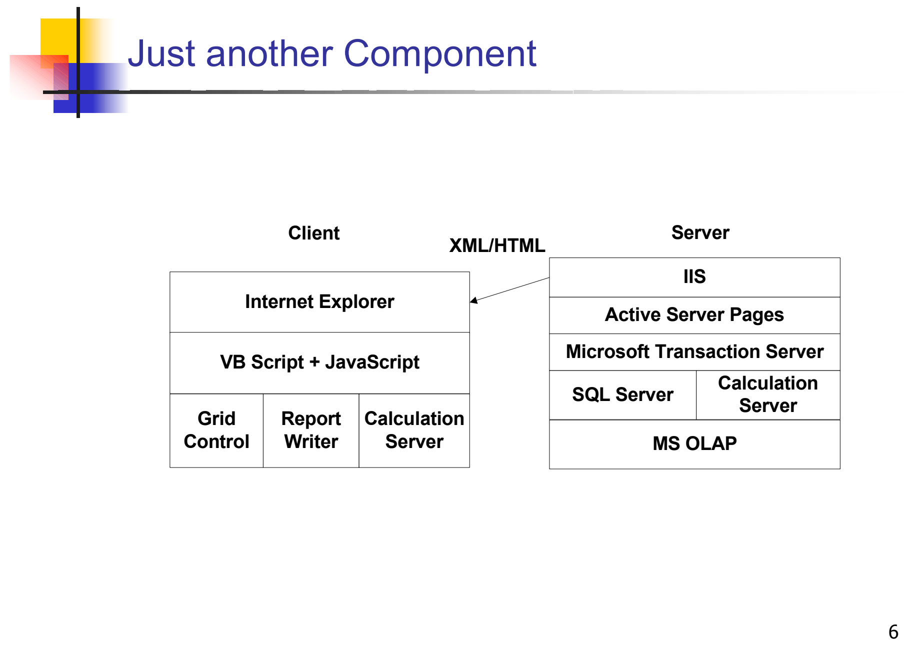
For example, one of our investment clients uses Excel for almost all its work, and the people using their applications load up an Excel worksheet, and they type in Excel, click buttons in Excel, and so forth. But when clicking those buttons, a J interpreter is sitting underneath Excel invisibly, doing all the calculations. So we're able to add calculational functionality to Excel. The reason they want to use J is that Excel does not work very well with large data sets. So they find that they if they use Excel for processing, the task takes forever; or Excel just falls over. However, they can use J and get essentially instantaneous results, but with the same interface they would use normally.

So we feel we complement these languages; we don't compete against them.

Core Language
We have focused almost all our efforts on the core language interpreter. There is a database and graphic user interface (GUI) in J, and they work quite well for simple applications; but they're not our focus. We occasionally have people come to us and ask, "Well can you add a little functionality to the GUI?" Typically our answer is, "No, we are not interested. If you want to do that, then use VB for the GUI".

Just Another Component
J is not intended to be a complete programming system. We're not reinventing the wheel—instead we are just another component in an application. This component business (Table 1) is quite interesting. Consider, for example, maps of applications that our clients develop. Most maps are much more complex than this, and have a page full of little diagrams; this is just a simplified map. But the interesting thing about this particular map is that although J is part of the system, you never would notice it. You can see a calculation server, and that would be the J application. The point is that it's just one component of the system.

Table 1



We have spent a lot of time making J work well with other applications, so it just slots in there. You can see in this example that J is running on both client and server.

Implementation

J is written in pure C; not C++. It's very easily ported to various machines and operating systems. One way of describing J, in fact, is simply to say it's a collection of C classes that has been optimized for numerical computations. It's pure C and easy to port. We also have a 64-bit version as well.

The J system runs equally well on the client and server, and it's small enough to be downloaded to a client from a Web application. Now this is something we've done quite a bit, and something that I do myself. The J engine is about 700K, and that's actually quite easy to download; on the Web you get such downloads in a few minutes. This is a one-time download, and the Web application can then do computations locally. We use that in scheduling, but, as you know, a typical application in an actuarial business would be a Web-based life insurance illustration system. In such a system, you could download a copy of J and have J run locally on the client. The user would never know it was J, but you would have the benefit and do the computations locally.

J is an absolutely standard Windows object. It's also a standard Windows 32-bit DLL. There is nothing special about J at all. And if you don't want to use COM or DLLs, but you want to run under Linux or UNIX, we support sockets very efficiently.

Interpreter
We make a very clear distinction between the language interpreter and the development environment. What I mean is that these two things are completely different programs, completely independent programs.

The development environment is disposable. If you don't want it, you can get rid of it and the language will still work fine. The particular benefit here is that an external program has complete control over a J application. For example, you can do anything in J by calling it from VB.

Just to emphasize the point, suppose we weren't talking about J but about VB. You could not imagine another program such as Excel driving the VB development environment—it's just not possible. How could you create a form in VB from Excel or from another language? You can't. VB, like almost all languages, is a monolithic system, where the development environment is a part of the program. But in J, we make it completely separate. The development environment can be discarded, and you can use another program to drive it. So, VB client can drive J in a way that J could not drive VB or anything else could drive VB. We found this an extremely useful facility, and it's one of the reasons why J fits in so well with other applications.

Key Ideas
Let's talk about some of the key ideas of J. We call it a mathematical calculation engine that is good at, well, mathematical-type things. It's like a mini-version of Mathematica where we have optimized pure numerical calculations.

It's an interpreter, so you get immediate response when you're using it, and it's a very nice interactive development environment. The functions are optimized for large data sets. We typically don't mind in J whether you're adding two numbers or adding a million numbers. It doesn't matter. Of course, a million numbers will take a little bit longer, but as far as the interpreter or writing the language is concerned, it doesn't matter. J is very scalable.

Built-in Functions. There are also many built-in functions to manipulate data. These are the kind of functions you would have to write yourself if you were using C or VB. Interestingly, there are many functions to create and manipulate functions—this is a feature you see fairly rarely in languages. Again, with VB or C, how can you create new functions as part of the language? You can't. And of course, you write your own programs. But you're not writing built-in functions. In J, you can essentially write your own built-in functions. It's very easy to extend the language.

Syntax. The syntax is very simple and consistent. It's independent of data type and how the data is stored. And it's independent of the underlying machine and the operating system.

For example, you can write a program under Windows and take exactly the same program and run it under UNIX. It's exactly the same code, unless you happen to have used something specific to Windows. If your code makes a call to Windows API, it's not going to run under UNIX. If it doesn't make any such call, and it just does calculations—which is very typical of actuarial work—then you can use exactly the same code under any platform, and you can develop under Windows and run under UNIX if you want better performance.

Special Features
There are actually a lot of neat things in J, and it has many things that are very useful.

For example, we have a Unicode data type, which was requested by clients overseas. If you do business in Japan, Korea, or China, you have to support their character sets. Most programmers do it in a very crude way; for example, a C programmer treats Unicode characters as a 2-byte character string. But in J, it is built it into the language, because it is so useful.

Another data type is the symbol data type. What is a symbol? Let me try and explain this with a problem. Suppose you're doing investment analysis and getting a real-time data feed on stock prices. Let's say you're tracking 10,000 stocks. You're going to have 10,000 names in memory, and you're going to get a real-time data feed that has the IBM price or AOL price or whatever. Each time you get a data feed, you're going to look up the ticker in your list of 10,000 names.

Now In J, such things are highly optimized. It's really a fast look up, but the fact of the matter is if you have to look up a name in 10,000 names and do this several times per second because you're getting a real-time feed, it can take time. So, the idea of symbols is that we pre-hash the name list so as to allow look-ups that are almost two orders of magnitude faster than a plain look-up. Essentially, you can have real-time data feed with instantaneous look up.

Sparse Data Sets. One of the case studies I'll discuss is of a client who has a very large amount of data that wouldn't, in fact, fit on the machine, but most of the data is empty. This occurs quite often in practice. In J we can represent such data as a sparse data set. It provides very efficient calculations and storage on such data sets.

Memory Mapped Files. Let's say you're running your machine with 256 megabytes of memory. You have a file of data that is 500 megabytes. Clearly you cannot read that file until your active memory is increased. So if you have to process that file all at once, you'd have to do something clumsy, like reading a bit of the file and doing the processing, and then reading a bit more of the file and doing some more processing, and so on.

In J we can memory map the file. We can assign the entire file to a variable in J and process it, so you could add up the contents of a file as a single statement, even though you've only got 256 megabytes of RAM. It's an extremely useful feature for people who deal with very large data sets.

Object Orientation. J is a fully object-oriented system. I'm aware that people mean different things by object orientation. But J is object-oriented in this sense; I, and the people who use J, use classes and objects throughout our programs, and we find it extremely useful. It is a different type of object orientation from JAVA or C++. It's customized for J, but in fact, it works very well, and we find it extremely useful in development.

Case Studies

I wanted to look at three case studies.

Budgeting. The first case study is a budgeting company. This particular company, I think, is the largest company in the world that specializes in budgeting.

Budgeting is an interesting problem. Before my first look at budgeting a few years ago, I thought it was very simple. It is, in fact, simple if you have a Mom and Pop store—you can load Excel, key in the months and your expenses and do it that way.

But budgeting is a seriously difficult problem for big multinationals. A classic case that this company handles has about 150 companies around the world. It's very difficult for them to handle the different data feeds and different levels of organization. It's a seriously difficult problem to acquire and process their data.

As well as that, they use very sophisticated algorithms to do the budgeting, because budgeting is not simply a matter of doing accumulations. These guys want to budget downwards. They want to be able to say things such as, "Well, the expenses for this particular office add up to $5 million, but what I would really like to see is, what would happen if I cut the expenses to $4.5 million and break back the calculations?" They have very complex algorithms to do this.

They also need both client and server calculations. They spent a lot of effort trying to write their code in C and failing. They came to us and asked if we could do it. One of the problems they had encountered is a combinatorial explosion. In budgeting, you typically want to budget by many different dimensions—by many different types of things, such as the customer, the account, and so on. Now just imagine: suppose you had 10 different factors you wanted to put into budgeting, and each factor had 100 different possibilities. Then the possible budget items would be 100 to the power of 10, which is a big number. And that certainly couldn't fit on the computer.

But if you had that 100 to the power of 10 possibility, but only a relatively small number of real data items—which is really the case—then what you have is a sparse data set. In fact, this particular company came to us and said, "Can we do sparse data?" And we said, "Yes." And that's the reason why they use J.

Underneath all their budgeting calculations is a copy of J doing sparse data calculations.

Financial Analysis. The second company I want to look at does financial data mining.

Most of its clients are insurance companies, typically general insurance companies that need to analyze their data. Rather than describe their products, I've just taken a couple of quotes from their own promotional materials, so let me just read them out loud.

"Today's data analysis market is a nightmare. In order to deliver timely tailored information to users, companies must stitch together a complex, unmanageable system of data marts, online analytical processing (OLAP) cubes, ad hoc query tools, report engines, and data mining tools."

"The software tools in use today were not defined with these goals in mind. Few can handle sufficient volumes of soft data, and few, if any, can process, model, and aggregate fast enough to deliver information in real time."

This company uses J for their calculations. They use Java for the GUI, and J for the calculations. And J, in fact, solves these problems for them.

Scheduling. The last company is my own. What I do is scheduling, and my company has extremely complex scheduling requirements.

Simple scheduling is easy. If your schedule has only processes that follow each other linearly, then you just add up the times in each process, and you've got the total time of the combined operation.

But for the kind of work that my company does, scheduling is extremely difficult. You have all sorts of complications. For example, throughput varies by product line and by the particular product you're manufacturing. You have to look at the operating hours, machine availability, time-shared work, and line-shared work. Then people come along and ask, "Can this particular product start from this state and work on this particular line? Let's look at your raw materials and shipping resources."

I am with a small team at Leun Thai that started working on scheduling about 18 months ago. We now have the scheduling working fine, both Web-based and server-based. But even though they work well, we still have work to do because you always can refine it. It's an endless process.

J has worked really well for this. The thing about J that is really beneficial here is that it is so quick to write programs. Users come along to say, "We need to modify the schedule by taking into account this particular factor." We usually can modify the entire scheduling system within a day or so. We are continually refining the way the scheduling works.

We also use J for several other things—capacity planning, quota management, electronic data interchange (EDI), and reporting.

Quotes
Here are some quotes from some of our users.

"J makes for very quick writing, but more importantly, quick rewriting. So even large programs can stay young forever."

"Well-written J programs run very fast. They picked the best algorithm for your data and use tricks you wouldn't think of."

A lot of work has been done on optimizing the algorithms and also in doing some neat tricks. So J runs very fast. It would be hard for a good C programmer to keep up with J and do the things that we're doing.

APL
Now, while I'm talking about this, you're probably saying, "Well, this sounds like A Programming Language (APL)." And it does. Let me explain that. Many of the people on our team are long-time APL programmers. We liked the language, but we didn't like a lot of the features. We didn't like the monolithic development environment. We felt a lot of things that were fine 20 to 30 years ago, when APL was first invented, just didn't work well.

J is essentially a complete reworking and complete re-implementation of APL. It definitely isn't APL. One of the things that we found difficult when we first started thinking about J was that someone would ask, "Well, how did they do it in APL?" To which the invariable answer was, "That's not relevant. It's not important." We basically squashed any APL component. And that was the right thing to do, because we didn't really want to be held back with baggage from an old program.

J is much simpler than APL. It's much more consistent. It's a lot more powerful. It has a lot more features. The performance is much better than APL. It depends on what you look at, but overall, you're going to get several times better performance—some things we can do in J that APL is far too slow for doing.

J is a plain scripting language. We use plain text files. There's nothing special about a J program. You can use Notepad to create it or to edit it. You can use a standard programming environment. You can use any version control system. If you want to use a Microsoft SourceSafe, or any other version control system, you can use that for J. We made it pretty easy to access some other software. It's just a plain, ordinary, COM object. All these things are problems with APL.

The way that I think about it, is that if you ever used APL and you like it, then you'll like J, because we do "APL-ish" type things in a much better way. But if you know about APL and you hate APL, which a lot of people do, then the things that you hate APL for have vanished.

MR EVANS: Thank you very much, Chris. We appreciate the presentation. As I mentioned before, Rob Marone is from ACORD, and he'll be talking about the ACORD XML standards.

MR. ROBERT J. MARONE: I'm a standards evangelist for ACORD, which means my full-time job is to go around and educate people about the ACORD standards, encourage them to implement the ACORD standards, and assist in any way in that implementation activity.

I'm going to give you an overview this morning about ACORD as an organization and talk a little bit about our XML standards and why they may be of interest to members of the Society of Actuaries.

ACORD actually is an acronym. It stands for Association for Cooperative Organization Research & Development. We don't say that very often, though. It's been around since 1970. It's a not-for-profit corporation, and it was formed originally to standardize paper forms to the property and casualty insurance industry. But since then, we've gotten involved in the EDI standards for property and casualty, a variety of standards for life insurance, and for reinsurance as well.

Business Drivers for Standards
There's a lot of interest today in standards. What are the business drivers behind the interesting standards in the insurance industry? Well, they are a lot of the drivers that are changing the industry, such as globalization—entering the markets in other parts of the world. With merger and acquisitions activity, when organizations come together, the integration of those organizations—if they're based on standards to begin with—can be much simpler.

The entire e-business revolution, the search for straight-through processing, and the yearning for driving down costs through automating the business processes of this industry all are drivers that have made people much more interested in standards today.

History Lesson
I'll give you a quick overview of the history of ACORD standards. Like I said, in 1970, we started with the forms initiative. That's still a big part of what ACORD does today. We do all of the filing with the states for the property and casualty paper forms and that sort of thing. In the 1980s, ACORD got involved in EDI standards for automating the transmission of information from the forms, mostly between agencies and carriers. And we have a standard that's called AL3, which is our EDI standard for property and casualty insurance. That is still supported very

widely today. We have more than 60,000 implementations of this standard between agencies and carriers.

Throughout the '90s, we were focusing mostly on implementation of those EDI standards. How many people are familiar with the term EDI? It stands for electronic data interchange. During the 1980s and '90s, that was the way that trading partner communication was implemented, for the most part.

It wasn't until 1996 that ACORD got involved in life standards. What happened at that time was, Microsoft had created a little working group of a couple of vendors to develop a standard for integrating a life insurance agent's applications in a Windows desktop environment using its object technology, which at that time was called SLIEC (Solutions for Life Insurance Enterprise Computing). At first they thought they would be able to just work with these three vendors, and release the standard and that people would cheer and adopt and implement the standard. But they were a little naïve politically, because what really happened was everyone else that wasn't one of those three vendors got extremely upset that they were working on this without being involved.

Microsoft didn't want to be a standards organization; they just wanted a standard that was based on their "OLE" (object linking and embedding) technology, and they didn't really care who managed it and who participated in developing it. So they looked for an organization to turn that standard over to, and they found ACORD. ACORD expanded the scope of its operations to start working on life standards. At that time, it still was pretty much based on integrating applications in a desktop environment. And that standard was called "OLifE".

It wasn't until 1998 that ACORD got involved in standardizing XML standards, and the scope of our initiatives included applications throughout the entire enterprise, as well as trading partners communications—any time one enterprise needs to communicate with another—using XML.

Today we have a number of initiatives, and I'm going to talk about each one individually.

Convergence
Our convergence initiative is called "emerge." We've started a number of global initiatives, and we have reinsurance initiatives for the first time in both P&C and life.

Supported Standards
ACORD supports three distinct standards. On the life insurance side, we support all the major lines of business that are considered to be life insurance. We support an underlying object model for the life insurance processing, we have XML, and we have a brand-new forms initiative to standardize paper forms for life insurance. That's a separate issue.

On the P&C side, we have support for the major lines: personal lines, large and small commercial lines, surety bonds, and we have the EDI standards for P&C, which are called the AL3. We also have XML for P&C, which is a set of XML messages defined for property and casualty insurance.

Third are the reinsurance standards on the P&C side, which we acquired this year from an organization called WISE. I don't know if any of you are familiar with that. It was a European standards organization, and they had a set of standards called the JV standards, which stand for the joint venture. They're used in the reinsurance market and large commercial market in Europe mostly, but also in the U.S.

Those are our three distinct lines; they all support XML, and we support EDI on the P&C and the reinsurance side.

ACORD Members

Who are the members of ACORD? We have carriers—life insurance companies—who are the full members of ACORD. They're the only ones that are eligible to be on our board of directors and certain steering committees. The associate members comprise other associations, solution providers, or software vendors, and this list of other organizations that are involved in some way or another in the insurance business. This is where ACORD's revenue comes from. Members pay us a fee to be a part of the process of setting the standards, and that's really the only source of income we have.

As of August of this year, we were up to 405 active members—that's counting only carriers and solution providers. That represents a healthy growth over the past five years.

We have a number of international efforts that have taken place. We are working with other standards organizations. With local associations in areas such as South Africa and Australia, what happens is that an organization in that particular region will act as what we call a regional management association. They represent their members at ACORD and essentially speak with one voice and make sure that whatever standards we're developing address the needs of that particular region. In addition, we have another couple of initiatives that we're working on in Southeast Asia and Japan, as well as one in Latin America.

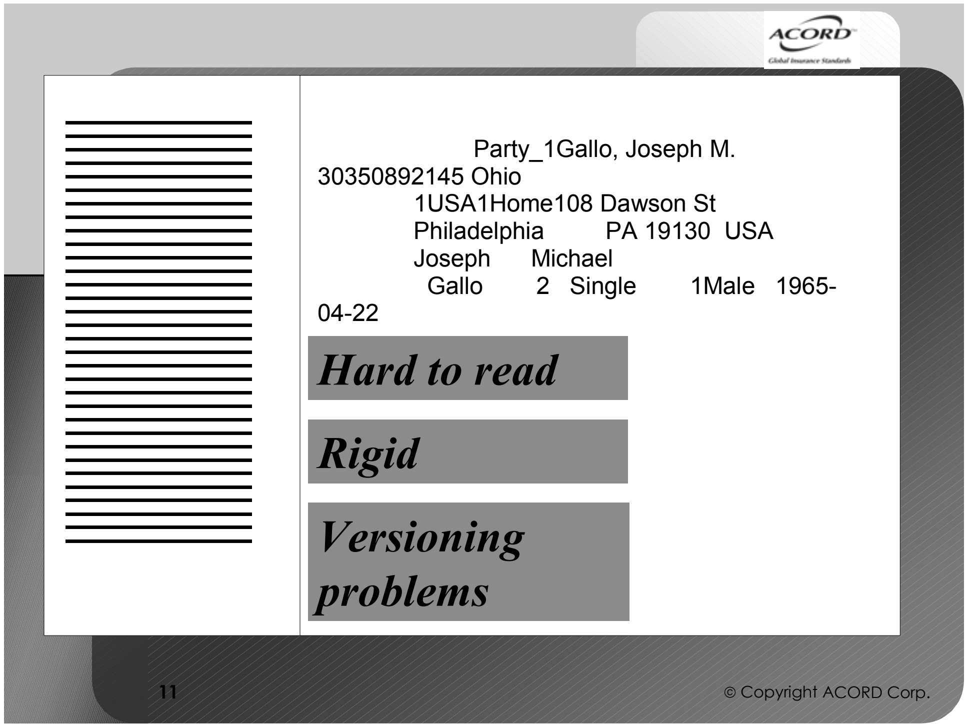
XML

Now for XML: Most of the growth in ACORD's membership, and interest in our standards really have taken off since 1998 when we got involved in XML standards.

Why all the interest in XML? How many people here are familiar with XML itself as a technology? Everybody knows what it is today. It's a technology that's promulgated by the W3C, which is the same organization that develops the standards for the World Wide Web, such as HTML. ACORD is a member of the W3C, and we actually vote on things such as XSLT, which is the style sheet on language that was adopted last month. We got to cast a vote in favor of making that an official W3C standard and that sort of thing.

XML is a big improvement over EDI. EDI was the way that companies would communicate between one another in the 1980s and 1990s. This is an example from our AL3 standard for a small snippet of EDI that is representing information about a person (Table 2). It's based on a fixed-length, binary data stream. And because of that, it's not easy to see where one field begins and the next one ends, because the data set itself does not describe that. You need to go to external documentation to know how to interpret this. For some of the information, you can sort of guess.

Table 2



```
                 Party_1Gallo, Joseph M.
30350892145 Ohio
          1USA1Home108 Dawson St
          Philadelphia        PA 19130  USA
          Joseph     Michael
            Gallo      2   Single       1Male   1965-
04-22
```

Hard to read

Rigid

Versioning problems

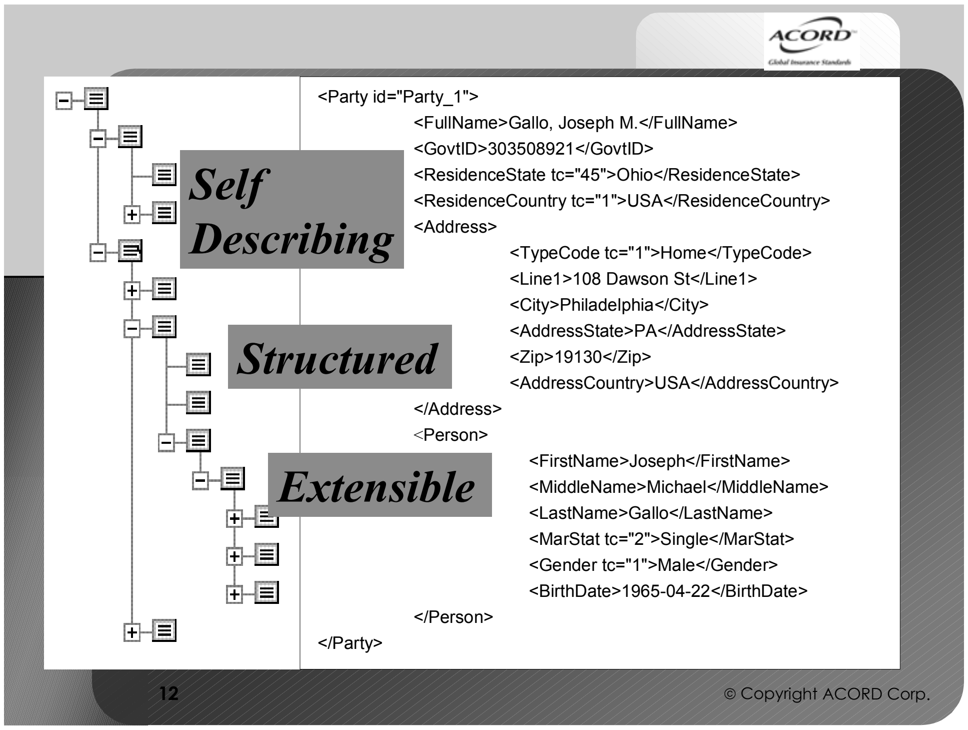
11 © Copyright ACORD Corp.

EDI is rigid. For example, since each of these fields is a fixed length, if I decide, "OK, I've allocated 20 characters per name, and tomorrow I want to do business in Thailand, and the last names average 25 characters, I've got a problem." And I can't easily just say that this message is now going to have 25 characters or 40 characters for the name, because doing that has an effect on the rest of the side of the message. That will break existing programs that are relying on this too, being exactly 112 characters from the start. So you get into versioning problems with the EDI because, again, it's based on these fixed records.

Now, let's take a look at the same snippet of information. This is an XML format (Table 3). This actually is from the ACORD standard called XML Life, which is our

XML standard for life insurance. And this is a snippet of information from an element of the file that we call the party, which represents either a person or an organization.

Table 3



```
<Party id="Party_1">
        <FullName>Gallo, Joseph M.</FullName>
        <GovtID>303508921</GovtID>
        <ResidenceState tc="45">Ohio</ResidenceState>
        <ResidenceCountry tc="1">USA</ResidenceCountry>
        <Address>
                <TypeCode tc="1">Home</TypeCode>
                <Line1>108 Dawson St</Line1>
                <City>Philadelphia</City>
                <AddressState>PA</AddressState>
                <Zip>19130</Zip>
                <AddressCountry>USA</AddressCountry>
        </Address>
        <Person>
                  <FirstName>Joseph</FirstName>
                  <MiddleName>Michael</MiddleName>
                  <LastName>Gallo</LastName>
                  <MarStat tc="2">Single</MarStat>
                  <Gender tc="1">Male</Gender>
                  <BirthDate>1965-04-22</BirthDate>
        </Person>
</Party>
```

First of all, the data is self-describing. Every piece of data has a tag that has both a start and an end, the ending being indicated with a slash at the beginning of the name. If the tags are selected wisely, you will be able to interpret that this data gives the full name of the person. There is structure to it.

Encoding

One of the things that we've done at ACORD is decide how this information should be encoded for each of the different data types that are commonly found in a message like this—things such as strings and integers and currency amounts and dates and times and coded sets of information such as the states.

This is a good example of that. You can see that there's a tag called resident state and it has two parts to it. The description of the state itself is the English word for the state, Ohio; but then there's an attribute in the tag—this part here, which is called TC, which stands for type code, which means that there is an official code and set of information.

ACORD publishes all of the codes that go along with this. The code for the state of Ohio is 45. All of our coded sets work this way. You can see it down here again in the marital status. You can see a TC of two and a gender male, TC of one. It's actually that integer value that programs will work against. So if you're writing a program, and say you're in JAVA or J, you can go against this integer, and it's the same no matter what. If I were writing a file that was for an application in a foreign language—say this is a message going to a Spanish insurance company—I could change the description to say "*soltero*" instead of "single"; but this code 2 would be the same. That's so the application could continue to work, and yet we can support multiple languages with the same XML message.

This gives you an idea of what XML looks like. It is an improvement over EDI because it is self-describing, it gives structure to the data, and it is easy to extend. None of the individual tags has an intrinsic length associated with it. So if I wanted to have 40 characters for the name instead of 20, that's not a problem because, again, the name doesn't end until it finds the end tag. There's no intrinsic size associated with each element.

In addition, I can add tags that current programs will simply ignore because of the way XML programs process this information. They parse the data, and they look for the tags that they're aware of; but anything that's additional is just noise for them. They don't see it. They don't care. And so that gives us a way to add to the standard in a backward, compatible way as we go from version to version. So all of those are big improvements.

The W3C provided the language called XML and the syntax for this data stream. But it's ACORD that's providing the language for the insurance industry. Of course, it's our members that are providing that. I'll talk a little bit more about the specific standards that we support.

On the life insurance side, I mentioned that we have an underlying object model. The object model defines the entities that we deal with—the parties, which are people. It defines things such as policies and coverages and riders and that sort of thing—very insurance-specific. It also has support from investments and other financial instruments that are related to the insurance industry. We currently support the physical implementations on three different technologies. One is the original OLifE, which is based on Microsoft's COM technology. Another implementation of that is called JLife, which is based on JAVA and, more specifically, the EJB JAVA objects.

Then we have the XML version, which is "XMLife" and "TXLife"—just so you're aware that the actual XML standard has two different parts to it. The one standard is called XMLife, which is an XML encoding of ACORD's abstract object model. The TXLife is the "Transactions for XMLife", in which the actual message performs specific business functions.

All of this information is available in great detail on the ACORD Web site. A large amount of it is available in the public domain; anyone, including nonmembers, can get the information about specifications.

The model itself supports our transactions. There is an extensive list of the transactions that have been standardized so far. These are messages that businesses can use to communicate within themselves one system to another, using XML, or between enterprises to go from one company to another—for example, between an agency and a carrier or a lab company doing underwriting or testing.

Possible Transactions

We've broken these down into a number of different types of transactions. The new business submission, for example, is one of the most popular ones; this is the way we can send in a case for underwriting using XML. And it has supports for all the major lines of business in life, annuities, disability, and investments. On the P&C side, we have the equivalent transaction, as well, for all the lines of business and P&C. Even on the reinsurance side, we have the same basic message for a new business submission.

One of the other ones is the illustration calculation message, which was developed by our Illustration Working Group. It's a full-featured interface for getting back the results of calculation for a projection of life insurance, including all of the supporting calculations that you'd need to do an NAIC-compliant sales illustration, and all of that.

We have product profile information such as policy product inquiry and investment product inquiry; that's the way that you can get information about a plan design that can be used by an illustration client to customize the interface for requesting an illustration. It could be used in an electronic app submission to tailor the application for the product and that sort of thing.

Then there are administrative functions, such as address changes and name changes and inquiry transactions for things such as a holding inquiry, which is the way you find out about a policy that either is going through underwriting or that's in force to support giving status to a producer or letting policy owners see what their current policy values are. And then there are some financial transactions for things such as fund transfers, changing fund allocations for variable products, and making withdrawal requests. It's actually a fairly comprehensive list of functionality that we support today.

We have another project that's new this year, which is to do what we did back in the '70s for P&C on the life side, which is standardizing paper forms for life insurance. It's a little bit easier now, because we have the XML transactions behind the new business submission. So we have a much easier go of doing this, rather than doing it the other way around, which we did on the P&C side. If anybody is interested in that, I can hook you up with the project team that's working on that project.

Important Associations
ACORD maintains an extensive array of relationships with other associations and standards organizations. There are many other standards organizations that are doing work that touches the insurance industry in some way.
I hope that after our meeting in November that we will be able to add Society of Actuaries to this list, because we're talking about working on the data for the mortality table standard in XML in association with the Society. That's something that we'll be talking about in our steering committee meeting in November.

eMerge
I already mentioned that we have three distinct lines of standards—one for reinsurance, one for life, and one for P&C. eMerge is the effort to take those three and converge them into a single unified standard for all lines of business of insurance.

What is the driver behind that? Well, a lot of the same drivers that are behind standards in general are in force here, but especially the consolidation of companies and companies going into multiple line of business. They want to have a single set of standards that they can use throughout their enterprise if they're working with life, reinsurance, or P&C.

Our members are telling us that they want that, and so we're trying to deliver it. We kicked off this project in June 2001. It has an 18-month schedule. So by our November meeting in 2002, we expect to have the first release of the Emerge standard available. In fact, we're having a meeting in London right now where people are working on that.

The idea behind it is, first of all, to come up with a single standard for insurance. That includes the unified data dictionary of all the common terms between life, P&C, and reinsurance, an underlying object model that is technology neutral and models the different entities of the insurance industry. Upon that, we're going to build the messages, just as we have in the current XML standards that are distinct for P&C and life and reinsurance.

As I mentioned also, we are merging these three existing standards of XML for P&C and XML for life and the JV reinsurance XML standards into a single unified model.

One of the important parts of that is that it's going to be built upon a technology-neutral object model. We actually use UML, which is unified modeling language. How many people are familiar with that? OK, it's a technology for doing abstract object modeling that's not tied to any particular implementation, and it's actually developed by a group called the OMG (Object Management Group). That's how we will publish this technology-neutral model; and from that, we will then be able to drive specific implementation, such as an XML schema, to support XML messages, IDL for Java, or IDL for COM objects or for common object request broker architecture (CORBA) objects, or that sort of thing. So you can go from this UML, which is an abstract representation of the model, and then deliver different platform-specific implementation.

Global Framework
And then finally, when we're talking about the XML implementation of this, we're talking about building this model on a global framework. The global framework that ACORD has been working with is called the ebXML. How many people have heard of that? It's an XML framework that has been developed by the United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT), which is a standards body that's supported by the United Nations, as well as the Organization for the Advancement of Structured Information Standards (OASIS), which is an organization for structured information exchange or.

They have been developing this framework called ebXML, which is Electronic Business XML, which includes things such as the standard naming convention for XML identifiers and the rules for transporting XML from one system to another using various technologies. These are rules for putting an envelope around the XML so that it can be routed as needed to go from one organization to another, or maybe through a portal or through intermediaries. ACORD has been an active member of that framework, and now what we expect to do is build the insurance layer on top of what this ebXML framework already has developed as a generic way to do XML messaging in any industry.

Open Participation
The process that we follow to develop standards has open participation. Anyone is allowed to join ACORD and be a member; only the active members can vote, while others are allowed to participate in the meetings. We have strict antitrust guidelines like the Society of Actuaries has to follow whenever you convene meetings, where competitors are getting in a room to develop something. We have to be very careful about the way we conduct the meeting, and we should start off by reading our antitrust statements and that sort of thing.

Collaboration
We have an elaborate committee structure that is used to set the direction of the group, to provide a route for appeals, and that sort of thing. That's what it looks like. Today representatives from our carrier members are on the board of directors—only carriers are allowed to be on the board. We have a global standards committee, which is dealing with things like eMerge, which is our effort to convert the lines of business into one. And then each of the individual lines of business—life, P&C, and reinsurance—each have their own steering committees that actually set the practical objectives for each group within each six month period of time. Again, only carriers are allowed to be on the steering committees.

And then we have the subcommittees where the actual voting takes place for each line of business. For any standard to be passed, we need to have a super majority of 75 percent for it to carry. So there's no such thing as a small group of people getting together and railroading something through without broad-based support.

Then there are the working groups. The working groups are where the real work gets done, and that's open for participation by anybody, including our associate members and nonmembers and that sort of thing. Working groups reach decision by consensus—there's no actual voting—and they in turn send their proposals on to the various subcommittees, where they're voted on. The actual process itself looks something like this.

A proposal for a new standard or a proposal for a maintenance request on an existing standard is made by somebody, usually a member. And then the ACORD staff reviews it and makes sure that it gets directed to the appropriate committee. The committees typically send it to a working group, where it'll be developed and the details will be flushed out. The subcommittee votes on the standard, and assuming a 75 percent super-majority is obtained, then it's a candidate standard. At that point, it goes to the steering committee, which will make a decision on whether there needs to be an implementation pilot.

If it's a complex proposal or a new standard, it oftentimes requires an implementation pilot, with which the information will be released and companies can begin to implement based on the candidate standard and give us feedback as to whether it's workable or not. And that's where we flush out the fine details of anything that may not have been thought over when we were debating the standard originally.

Once the implementation pilot is successful, there may be modifications to the standard during that time. Then the steering committee itself passes the final vote, which makes it finally an official ACORD standard. There's an appeals process as well, if somebody is not happy with the standard, even after it's been approved by the steering committee. We go to the global strategy committee; I think there's a provision for it going to the board of directors as well.

All of our standards are open. Our open standards are available in the public domain—anybody can use the standards without paying any royalties to ACORD. Anyone can get information about the standards without being a member of ACORD. If you are a member, however, you get additional benefits, such as consulting services that are available only to our members for doing things such as mapping the ACORD standards to your own internal data structures.

We offer training courses to educate you about the ACORD standards and how to implement them. We offer a certification process through which solution providers typically submit their applications to ACORD for testing to see that they actually have implemented the standard correctly. In the case of an XML, we look at the actual XML, validate that it parses correctly against the document type definitions (DTD) or the schema and actually check to make sure that all of the required components of the message are there and that the interpretation of the standard has been consistent with our understanding of it. That gives our members some assurance that if they're going to buy an application from a software vendor, they have some assurance that it implements the standard, and they'll be able to integrate it easily into their enterprise.

And of course, ACORD does process facilitation; basically, we run this whole process of debating the standards, voting on them, and publishing the information. If you are interested in the standards themselves, www.acord.org is the place to go to find a lot of additional information.

MR. EVANS: Thank you very much. Thank you Rob, it looks like ACORD has been doing lots of good work, and we look forward to working with you on hopefully many projects.

I have a question for Rob. As far as implementation of the standard, has a timeframe been set, or are companies that want to start using the standard pretty much on their own to do that?

MR. MARONE: Well, they're on their own. Since not all of the implementations are from members, there may be implementations that we're not aware of. But in order to encourage implementation, we've had implementation awards that we give out every year at our technology conference in May. So, this particular year, we've had 13 people on the life side and some 17 on the property and casualty side that implemented the XML standards in the latest 12-month period. So we got a kind of feeling for at least the ones that had submitted certification.

We're also doing a survey right now to try to get a better feeling for all the implementations, including the non-certified ones. So we have better statistics to track the implementation. That's all part of getting the critical mass that makes standards successful—knowing how many implementations we have.

MR. EDWARD C. JARRETT: One of the challenges that we have in our systems is that we have evaluation systems that deal with very large extracts and data feeds from the advent systems and various other sources. In dealing with gigabyte files and changing those from our fixed layout form into an XML form, it looks just like the size of a file that's going to go from, let's say, 2 gig to 10 gig. Just processing that file both from the input and the output standpoint, how do you deal with those types of things in XML?

MR. MARONE: What we've done in the ACORD standard is we've adopted the use of multipurpose internet mail extensions (MIME) in order to break the XML messages up into discreet components, typically one transaction or one policy at a time, depending on what your application is.

Let's say you were doing a 2-gigabyte extract. That would be, you know, 500,000 policies that you processed in a nightly run. It's not practical to put those into a single XML stream. In real computers, it's difficult to parse some deal with that much information. So instead, we specify the use of MIME in order to break them up into individual XML documents. Then you can use standard tools for taking the MIME file and breaking, disassembling it on the other side, and then processing each one individually.

Does that answer your question? That's what the ACORD standard calls for you to do if you need to deal with a very large data set. I know it's difficult to deal with a huge 10-gigabyte file in general. That's another problem someone would have to talk about implementation strategies.