A Plan for Producing a Conventional Ada Library

Marin D. Condic
mcondic@acm.org

Abstract

There is a general consensus that Ada would benefit from having a library of tools available that would be standard across most or all implementations. This paper discusses how to get a "standard" library for Ada without impacting the ARM, increasing the leverage available to Ada developers, providing an enhanced product for the Ada compiler vendors, sparking new interest in Ada and perhaps building a financially successful business in the process.

Introduction – Why a Conventional Ada Library?

Why do we want a "Conventional" Ada library? The short answer is "Because everyone else has one and we don't." That is of course an oversimplification, but it makes the point. It is also not simply a matter of catching up to the features available in other languages. Developing a large, diverse library is a way of exceeding the capabilities of other languages. Because Ada doesn't need to have a library of objects that make up for shortfalls in the language (like lack of multitasking or lack of range checking) it can concentrate on providing developmental leverage above and beyond what is available from other languages. It can add library wings that provide capabilities not traditionally associated with a language library.

For example, an object oriented class that provides the inner workings of a Rolodex application – a linked list of names/addresses/phone numbers, etc. – adds a standard utility that speeds development of any application that needs to track people. Such a class would provide a commonly needed data collection in a way that would be portable from one application to another, along with all the tools needed to manipulate it. Inheritance allows easy extension for specialized needs and as a member of a common library, it would be well integrated with other utilities. Thinking outside the box means we can offer more to the developer. Giving the developer more leverage in new and interesting ways makes Ada a more attractive implementation choice and will spark renewed interest in the language.

The customary objection to Ada from those who aren't just totally bigoted against it usually takes on this form: Ada may have all sorts of theoretical advantages over Java or C++, but when it comes to getting the work done, these other languages do it better for a variety of reasons. The reasons range from lack of skilled Ada programmers, required training, cost and/or availability of compilers and development tools, interfaces to existing utilities, and availability of libraries. Building a Conventional Ada Library doesn't solve all of the problems, but it can help address many of them. The objections can usually be summed up as "I can't get my product to market as quickly with Ada as I can with something else." If a substantial library exists that gets a developer to market

50% quicker than some other language, perhaps the notion of finding skilled programmers or providing training doesn't look so onerous? The expense of training programmers is more than offset by the advantage of getting to market ahead of one's competitors.

Having a library of things people take for granted in other languages is a must. This means at minimum developing some kind of standard container library. It also implies some OS interfacing and networking capabilities. We should look at what is provided in other languages to insure that a Conventional Ada Library will contain the kinds of capabilities they provide. But we should definitely not stop there. Much of the reason for the popularity of C has to do with the availability of libraries totally outside of the language, such as databases, realtime operating systems, graphics libraries, network communications, and so forth. The fact that Ada can bind to these libraries is not of much help. Ada has to provide compelling reasons why it should be adopted over some other language. At best, Ada can say "I'm just as good as…." when discussing binding to libraries provided by other languages or written in other languages. Why would someone want to change from what they already know and use? Why would someone want to buy and support two compilers when they could just use the one?

Other languages may not so much have a huge standard library as it is a case of an individual, dominant implementation having a large library. To the end user, the distinction is irrelevant. They get the leverage by using Sun/Java or Microsoft Visual C++ and they don't really care that it is not "standard". By having a Conventional Ada Library that is adopted across a number of compilers, the end user gets even more from Ada because it **is** common. Programmer skills transfer from one project to another, books and manuals get written presuming the common capability, code is more easily ported from place to place and so on. Even if Ada provided nothing more than what other specific language implementations offered, but provided it in a standard way, it has a competitive advantage because Ada is so good at being "Standard".

If Ada can find a way of adopting a large, diverse library that adds leverage and capabilities not found elsewhere, it has developed a competitive advantage. A library that provides things that are "new" and "innovative" will spark the interest of developers who have not yet given it a serious look. A library that exceeds the customary expectations developers have from a language library makes it possible for the language to capture market share and spur more growth. Developing a Conventional Ada Library and getting its acceptance within the Ada community is a winning proposition for everyone. This paper attempts to outline how we might get there.

Compiler Vendors, Standards Committees, Et Alia. Why do we need the vendors and others involved in developing a Conventional Ada Library?

Any individual could take it upon himself to produce a Conventional Ada Library. He could build it, put it on a website, encourage others to use it and hope that one day, it becomes dominant enough to be "The Answer" whenever someone goes looking for a library of tools in Ada. It has been tried by any number of very intelligent and dedicated

individuals. I admire their effort. However, if we look around, we still don't see something that looks like a "Conventional Ada Library." What we have is a variety of unrelated, disbursed libraries serving diverse needs and with no consensus as to what to use when solving common problems.

At the minimal level of having a library of containers, Ada has no answer except the traditional answer of "Well, you can always build your own, or download one of a few dozen libraries out there somewhere on the Internet, or you could always go build a binding to the one provided in Language X's library…." This is the sort of paralysis that leads users to look towards Java, C++ or other languages – because they get an answer right out of the box rather than have to cobble together their own answer.

What Ada needs to provide leverage to its users is a library that most everyone accepts as "The Place To Go" when looking for answers to common problems. Traditionally, users look at the libraries supplied by their compiler vendors. If there is already a library in there to satisfy some need, they look no further. Often, a programmer looks at a library supplied by the compiler and just assumes this is part of the language itself. That may be incorrect, but from a practical perspective, how is it different? If you only develop code on platform X with compiler Y and it has some library that is useful, it might as well be part of the language because there is no practical distinction.

Because of the way people view the libraries that come with their compiler, the vendors are in a unique position to impose an answer and make it a de facto standard. Everyone seems to agree that a library of some form is a good thing, but nobody agrees on which library that should be. If the vendors said "Here's what you get with the compiler… You're free to use another answer, but this is the convenient one…" we would quickly see nearly everyone gravitating towards that answer. With a Conventional Ada Library imposed (or perhaps "encouraged" is a better term?) by the vendors, everyone wins.

What is in it for the vendors?

The vendors get an enhanced product that is more competitive against other languages. It should be clear that the value of someone's compiler goes up if it comes with more features, even if those features are vendor specific. Increased features that are standard across most compilers has the benefit of increasing the overall demand for Ada compilers and a Conventional Ada Library is a way to obtain more features. Increased demand implies that all vendors would benefit.

At present, any given vendor, or all vendors, could choose to do a number of different things:

Develop their own libraries. This is done to some extent. The GNAT packages, for example, make the GNAT compiler more attractive and are not likely to be adopted by their competitors. A library that is specific to one compiler may help that vendor to get an increasingly larger share of the pie, but it doesn't help grow the size of the pie. It is not in the interest of any of the vendors to continue down the existing path of allowing Ada to

slowly fade away into obscurity while increasing their own market share. Vendor specific libraries are also costly to build, maintain and support. Sharing the cost of library building across vendors makes it easier to accomplish and provides more utility.

Identify and incorporate existing libraries. A vendor could pull together dozens of smaller libraries available under the GPL or similar licenses, cobble them into something and call that their library. It could even be agreed upon and adopted by multiple vendors. One problem with assembling a collection of utilities from a variety of sources is that you wouldn't get a comprehensive, well integrated product. Another problem is that there may be a variety of licenses involved and a large number of authors from whom to obtain copyright. Additionally, there is generally a lack of reliable support, questionable documentation and uncertain quality. Whatever the reasons may be, we can make this observation: A variety of library components already exist and could have been pulled together into a Conventional Ada Library, yet this has not been done. There must be something unattractive about doing so, or it should have happened by now.

Try to identify some library interfaces for the next Ada standard. This suffers from a host of problems. The biggest problem is that to get anything into the standard, you are looking at a 10 year turnaround cycle. A related problem is that as a "Standard" there is a demand for a much more rigorous level of validation than can easily be achieved for a library of any magnitude. Realistically, it would only be possible to get relatively small, extremely portable components into the standard (such as a minimal container library) and once done, they would not be able to react and adapt easily to changing circumstances. Even something as simple as a minimal container library becomes problematic because there are multiple, competing requirements that cannot all be satisfied easily within the context of a standard. While we would hope that at least some components of a library become available in the Ada standard, I don't think this approach is an attractive way of providing substantial new leverage to Ada developers.

Do nothing and hope the problem goes away. Because there is no simple answer to the question of libraries, there tends to be an endless debate that results in inaction. Vendors tend to want to wait until they see some paying customers demanding something specific. This is understandable, but it doesn't accomplish anything in terms of actually improving the capabilities of Ada. End users all seem to want something, but can't seem to arrive at a consensus as to what that is. Perhaps they don't really know what they want until they see it and are waiting for something to emerge that they can recognize as filling their needs. In any event, waiting around for a library to emerge on its own is foolish optimism. Its like waiting around for an oil well to spring up in your back yard and make you wealthy. It might happen one day if you just wait long enough. But its far more likely to happen if you go out and actually start drilling wells.

Get together and build a library. As an alternative, the vendors, in conjunction with other parties, could form up a corporation, the purpose of which would be to produce an Ada library that all would adopt. It is possible to structure such an R&D corporation in a way that does not require a fortune in investment, yet produces the desired end result. The corporation would earn revenue through royalties and support contracts, in which all the

participants could share. All vendors would get an enhancement to their product at a lower cost than would be possible if they each pursued it individually. The increased capabilities that would result for Ada will make it more attractive against other languages. In other words, the pie gets bigger and everybody's slice grows with it.

What is in it for the Ada Standard?

From an end user's perspective, it would be attractive to have an extremely large library incorporated in the standard. However, as observed above, this has many impracticalities. How would the Ada Standard benefit by having a Conventional Ada Library developed outside of the standard?

A Conventional Ada Library that was widely accepted as being part of Ada without actually being in the standard would be a benefit to the standard in a number of ways. Because the library could react more quickly than a standard revision, when weaknesses or gaps were found in the standard, they could much more readily be added to the library. This would allow the standard to experiment with solutions without having to commit to them.

A library would create a laboratory in which new features & capabilities could be tested to see how well they worked and how well they were accepted by the user community. Specifications could be tinkered with until there seemed to be a level of stability and acceptance. This would provide confidence that the solution was the correct one and suitable for incorporation in the standard.

Having a reference implementation helps to clarify the behavior one would expect from a given package within a standard. The Ada standard would be able to better describe what it would expect from a competing implementation if it had a reference implementation to which it could look.

Eventually, the standard could choose to adopt individual parts of the library and declare them to be officially part of Ada. It would likely want only the specification of a given part of the library, allowing for competing implementations to exist. The vendors, having a reference implementation available, would not find an addition to the standard to be nearly as much work as if they had to go build one from bottom-dead-center. This allows for growth in the language while mitigating risk and easing the entry of new features into the market.

An Ada Library R & D Corporation - Structure & Funding

Realistically, nothing is going to happen to get a Conventional Ada Library built without some sort of funding. Even if one hypothesizes that the whole job will be done by volunteers out of the goodness of their hearts, in effect, that amounts to funding the development by the volunteers. It is not impossible to find volunteers, but such efforts are difficult to direct and control and historically, they have tended to produce only small and/or incomplete products. Yes, exceptions can be found, but for every successful major

product built by volunteers that one would care to name, thousands, if not millions, of failed attempts can be identified. If you want something done according to your requirements and on your schedule, you had better get out your checkbook.

Since there is no longer any big, institutional sponsor for Ada, it is problematic as to where funding could be found. It would be easy to put forward a grand proposal that would result in a Conventional Ada Library being built in a short span of time and with all the professional look one would expect from some major software vendor. A reasonable SWAG at this would probably be somewhere on the order of $10-$20 million to get a first generation product out the door and support it long enough to see it start generating revenue. Given that it would be difficult to raise such an ambitious amount of money on speculation, we might want to consider a leaner operation.

Assuming for the moment that nobody has a huge pile of money to throw at this project, how could it possibly be done on a tight budget? We might be able to get the development work done for less investment by sharing the potential rewards. Those who might be inclined to contribute some level of work on a voluntary basis could be persuaded to produce larger amounts of the work and meet with more requirements if they were paid something for an acceptable effort and offered a share in the revenues that might follow. A model for such an effort exists in the book publishing industry. Let's look at that for a moment.

A book publisher does not directly produce the raw material that it packages and sells. The publisher finds authors and looks over submissions, determining what meets the requirements of the publishing house and accepting these submissions for inclusion in the catalog of works to be sold. The publisher works with the authors to develop the products to fit within the standards of the publishing house and assembles the materials for distribution. The publisher pays the authors some negotiated amount of money as an advance against future royalties and then pays the authors something for every unit sold. This model spreads some of the risks to the authors. Unlike direct employees, they have to do work on speculation that there will be a payoff down the road. Yet they still receive some compensation for producing material that meets with the expectations of the publisher. The model reduces the amount of up-front investment needed by the publisher to get the products built and shares the success of the products equitably with the authors. It also provides good control on the part of the publisher to get the products that are believed to be marketable. A similar model could be used for publishing a Conventional Ada Library.

Suppose that an R & D corporation were formed and shares were sold to those who have a stake in seeing a successful Conventional Ada Library created and maintained. The target shareholders would be Ada compiler vendors and some of their larger customers. As investors, they would direct the requirements and prioritize the list of components to be built. They would also gain rights to use and distribute the end products under some reasonable licensing restrictions that meet with their individual objectives and work to sustain the R & D corporation. The funded organization might only require a very few people to begin operations – perhaps a single individual at the start to get the ball rolling

and a few more as the work load increased. The corporation would have the goal of becoming self sustaining within two years from revenues generated by support contracts and royalties. The shareholders get the immediate benefits of enhanced products by virtue of having a library to package with their compilers or products and the future benefits of sharing the revenues as the R & D corporation achieves profitability.

The R & D corporation would set up the structure of the Conventional Ada Library and find willing authors to produce the code, test suites and documentation. When an author produces some part of the library that is found acceptable by the editors, he is paid an advance and he assigns copyright to the corporation. Some percentage of the corporation's revenues would be set aside as a royalty pool. At reasonable intervals, the contributions of the various authors are counted as a percentage of the total library and they are paid a percentage of the royalty pool as a result. The scheme is not perfectly equitable as we all know from discussions about code measurement – not all code is of equal value, code can be inflated, etc. However, it does provide a reasonable mechanism for getting some level of compensation to the authors, encourages more development and does not require nearly so much up-front capital to get a library developed. In effect, the authors become a kind of partner in the venture and have a stake in seeing it succeed. Special arrangements could be made with authors who had what might be considered more valuable contributions and the editors of the library would be able to insure that acceptable quality standards were met.

We need to give serious consideration to how the R & D corporation would generate revenues. It cannot succeed long term unless it has customers willing to pay to keep the library alive and well. One part of the model is to sell support. The library could be made publicly available at various intervals, much like the GNAT compiler is made available. This encourages use, but doesn't directly generate revenue. If the library is large enough and useful enough, there would be customers willing to get regular releases, reactions to bug reports, priority on addition of new features and so on. Documentation, books and various other "extras" would also attract users to a support contract. But support alone may not be enough, nor is it an equitable arrangement when one is agreeing to pay the authors some form of royalty. Why should an author receive a royalty if the only revenue is from the labor of the R & D corporation to provide support? Does an author receive a royalty from a professor who earns money teaching from the author's book? Equity is achieved for the author when the student purchases the book, so we would want to consider how to gain some revenue from the author's contribution. This brings us to a discussion of licensing.

There are a large number of developers who are fond of "Open Source" licenses. In particular, the GPL and some of its variants. For a variety of reasons, I don't think that the GPL itself would be acceptable, nor is it the best possible model for building and sustaining a Conventional Ada Library. We would definitely want to distribute the library in source code format and would not want to discourage others from sharing the library with those who want it. We would want to see others involved in adapting and extending the library, potentially adding their enhancements back in as contributions for publication. Openness in one thing, but something has to happen to generate revenue or

the project withers quickly. An unlimited right to use and distribute the library in any manner and for whatever purposes is giving away a bit too much. We need to be concerned with keeping the R & D corporation alive as well as paying something to the authors in the form of royalties.

Suppose that a license was constructed that allowed most uses of the library for what could be considered "internal" purposes. You can use the library for any software development you like so long as it is not incorporated in a product or service that is sold. You can develop programs for yourself at home or in school or within your business and this is allowed. You can give the library to someone else in its source code form provided they accept the same license. The license for the library doesn't need to "infect" your software by forcing you to adopt the same license for anything you develop that may incorporate it. The only restriction is that you cannot use the library in some form of commercial product.

But how does one then use the library if the objective is to build some software that is sold to someone else? The simple answer is that you can't – not under the general license that you may get by virtue of getting the source code for the library. What you would need to do is execute a different license with the R & D corporation that owns the copyright. Would this be a problem? Would this discourage use? I think not – provided that certain guidelines were followed.

The key guideline is that a "for-commercial-resale" license must be easy and inexpensive to obtain. You make it easy and free for someone to use the library in a large segment of the possible development scenarios. The instant they determine "Hey! We could sell this thing we developed…" they can come to the R & D corporation and select from one of several licenses that meet their needs. The licenses are available at a price that doesn't discourage the developer from using the product. If a license gets too expensive or difficult, the developer simply builds his own library or goes to some other source to get what he needs.

The exact terms of the licenses would be something that would need to be worked out with competent legal advice and taking into account all of the possible ways of utilizing the library. (Translations to object code, usage and distribution on a network, incorporation in a textbook, etc.) One could imagine a variety of usage scenarios that might be dealt with.

Suppose that a user has a support contract with the R & D organization. This might give the user a license to incorporate the library in products for sale without additional fees for as long as they maintain their support contract – perhaps to some maximum number of copies. This encourages developers to purchase a support contract if they anticipate heavy usage.

Imagine that a user develops something for a single customer – or some small number of customers – and doesn't want the expense of a full-up support contract. For this user you have a small-scale license that allows some limited number of copies to be sold for a

relatively small fee. What would a small-scale developer pay for a copy of MSVC++ and its associated MFC library? The license should scale the cost to a similar level.

Perhaps a user wants to incorporate the library in some product that will possibly have wider sales, but the end result is unknown and he doesn't want the up-front expense of a full support contract. For this user, you have a license that involves a reasonable per-copy fee that might be convertible into a support contract if his product is enormously successful.

The important concept is that the R & D corporation can make some money on licensing the library so long as it doesn't get too greedy about it and price itself out of existence. It must not make it so difficult for the user to get an appropriately scaled license that they give up and go somewhere else. If compliance is easy and inexpensive, enough honest people will opt for the license and revenue will result.

A variety of schemes could be investigated for structuring a corporation to produce and maintain a Conventional Ada Library and for rewarding those who contribute to its success. The important point is that with some creative structuring and innovative organization, we could get a library built at an affordable cost. Such a library could be accepted as a de facto standard and provide a means of making it pay for itself.

What would the R & D Corporation produce?

A Public Domain Collection Of Interfaces. If the various package specifications of a Conventional Ada Library were placed in the public domain, this would allow for easy adoption of the interfaces into the Ada standard if that became desirable. It would mean that alternate implementations could be built for specialized purposes where the reference implementation may not be acceptable. It would open the door to competitors, but realistically, this would not be much of a threat to the organization supporting the library. Competition would imply a measure of success and an attractive market. That would actually be a good thing. If a competitor were to arise, they would be in a position of playing "follow the leader" which is never a really attractive business proposition, so I would not consider this to be an issue. Placing the interfaces in the public domain makes them totally open to anybody who wishes to publish anything about Ada while not hurting the implementation at all.

A Proprietary Reference Implementation. The Reference Implementation should be kept under some semi-restrictive license in order to make sure it can generate some revenue to keep the R & D Corporation alive and reward those who contribute to it. The level of restriction should be kept minimal, but sufficient to insure that if a profitable use is found for the library that the developers of the library gain something for their efforts. The philosophy should be one of "Use the library any way you want, but the instant money changes hands, we want a piece of the action."

The Reference Implementation is periodically made publicly available under a license permitting personal and business use, but no "For Profit" use. For those who wish to

embed the Reference Implementation in some end product, the Organization will have one or more commercial licenses available that provide easy terms and inexpensive rates for use. The details are negotiable, but the goal is to make it easy to use & not excessively expensive to comply with a license.

Support, bug fixes and regular releases to paying users. The R & D corporation would see the bulk of its initial revenues coming from support contracts wherein it provides frequent updates to the library to its supported customers. There are any number of developers who would pay to gain rapid response to problems or changing needs, much as one sees from other successful "Open Source" endeavors. The organization would have frequently scheduled releases to add new features and fixes to satisfy this need. It would also be able to provide telephone support, consulting and other services to give value to its supported customers.

Documentation. Beyond the interfaces and whatever comments exist in them, users would have a need for a manual of some form. A user's manual that provided detailed explanations of the purposes and uses of the library components would be a valuable asset that could be made available to supported users and those who purchase licenses. This is another value added product that makes it attractive to spend some money instead of waiting for the next publicly available, free release of the library.

A comprehensive test suite. Built into the library would be packages that contain test procedures for each library component. This is important in part because the R & D organization needs to insure the quality of its work. It is also important because it serves as a proper example of how to utilize the components and acts as a guide for the end user. If the library documentation is hyperlinked to the test procedures, it provides quick illustrations to the user of how to properly get results from the library. The test components are naturally released as part of the library and are thus publicly visible, but this still provides value added to supported users because they get it first and with the documentation.

Authors & Publishing of Components.

Anyone can submit an extension to the library. The R & D organization is acting as a publisher. As such, someone with a good idea is free to submit some extension to the library with the hope that it will be accepted. Minimally acceptable submissions are reviewed to determine if they are appropriate additions to the library and if so, the organization pursues whatever is needed to secure the copyright and incorporate the work in the library. The author is paid some form of cash advance and gains the right to share in the royalty pool according to the agreement executed. The length of time in which the author would be eligible to participate in the pool would be fixed to encourage continued development.

Anyone can submit an enhancement to an existing branch of the library. (Bug fixes, additional capabilities, etc.) It is problematic as to how one would count a contribution to an existing library for purposes of royalties. Generally, one would expect that most of the

time, the original author would be the one proposing enhancements and fixes. This might be the mechanism by which an author extends his right to share in the royalty pool. If the enhancements or fixes come from some other source, the organization might pay some fixed fee to the author to obtain the copyright with the exact amount negotiated depending on the perceived value of the enhancements.

Submissions must meet certain requirements for format, style, content, etc. & must include documentation. The R & D corporation needs to insure the quality of its end product and not waste its time evaluating submissions that are too far away from its expectations. The organization can produce an author's guide that outlines exactly what it expects for a submission to be reviewed. The author would have to submit the source code developed in accordance with some guidelines for style and format. Documentation in the style used by the library as well as test components would be expected. The source must compile and execute its test suite using some selected reference compiler and platform. If a submission makes it that far, the organization would be in a position to decide on accepting it or not based on the perceived market needs. It would be expected that the R & D corporation would make known to potential authors what sort of components would be most desirable and encourage early contact with authors interested in developing them. It might even be desirable for the organization to develop a specification and look for an author who would want to provide an implementation.

What Would a Conventional Ada Library Contain?

First cut – A Basic Container Library. It would be expected that the Board Of Directors would set the requirements for a basic container library. These are, after all, the very first users of the library and the ones paying the bills, so they should establish what is needed based on their use or that of their customers. We have seen a variety of container libraries available for Ada and debated many others, so it is clear that there is no one, single, obviously correct answer. The goal ought to be to maximize the usefulness of the basic containers for general programming and build the remainder of the library components on top of these. With that goal in mind, some suggested requirements are given here:

A simple, generic implementation of dynamically allocated lists and maps. Something like this might easily make it into the next Ada standard and the objective here would be to have some packages that would be useful for relatively simple programming problems. They should allow the user to build lists and maps of homogeneous types and be able to serialize these containers to/from streams so that they might be stored and retrieved readily. This would be the starting point for educational usage, light duty programming, and other sorts of basic use. I would start here, but not base the remainder of the library on it.

An object oriented container library that covered a variety of data structures. This should be suitable for general programming, utilizing dynamic allocation and providing serialization to/from streams. One would want to look at what is provided in existing libraries such as the Booch Components, the MFC, the STL and so forth to see what has gone before and adapt the best features for the Conventional Ada Library. Simplicity and

ease of use should not be the primary objectives – although it doesn't hurt to have those things where possible. Power, reliability and comprehensiveness should be the goal. We would want the widest variety of useful data structures with the most possible leverage. If it gets complex to use, this is all right. Those with simple needs can resort to the primitive generic packages described above. The complexity is handled with proper documentation and examples provided by the test suites. This is what you want to build the remainder of the library on so it needs maximum power and reliability.

The container portion of the library should leave branches open so that variations on the basic containers could be added if special needs are perceived. If eventually it is determined that it would be attractive to have similar containers built upon static allocation, realtime constraints, task safety or other design goals, you want to be able to add these easily. Since such special needs are not as common, these goals should not be important for a first draft. The structure of the library simply needs to leave itself open to such extensions so that they can be added in an orthogonal way with similar "look and feel" to the basic components.

Additional Builds – Include libraries for things not usually found in other languages.

This is where things get exciting. Why shouldn't a library contain components that provide the basic "guts" for many common applications? Ada is a very portable language that could easily provide most of the "in memory" parts of an application and thus provide leverage to a developer. Much of the I/O in Ada is equally as portable and that, coupled with other standards, ought to allow the underlying parts of an application to be built in a way that is portable across a wide variety of platforms. Let's call these "engines" for lack of a better term and examine what I mean here.

As an example, let's look at some of the things commonly found on a desktop computer. You typically have a calendar, an e-mail program, a spreadsheet and so forth. Start with a calendar. What does it do? It holds data about appointments, loads and stores them to disk, looks at the clock and generates alarms when an appointment comes up. It doesn't generally deal with gigabytes of complex data, so it doesn't need a relational database under it. Mostly, it can load its data from some file and operate on it in memory just fine. Everything about it, except the GUI interface, could be built in a portable way using standard Ada features. Hence a "Calendar Object" could be a component of a library and if someone needs a calendar within the application they have at hand, they've got one right off the shelf. Why couldn't a Conventional Ada Library contain such an object and offer this leverage to a developer?

One might argue that there are an infinite number of possible features that are desired in a calendar and so providing a "standard" calendar is undesirable. A dozen different developers would come up with two dozen different sets of requirements – how could a Conventional Ada Library expect to satisfy them all? The answer is that it doesn't have to. It only needs to provide *an* answer – not *the* answer. If a developer is trying to build the next Great American Calendar program, he is always free to build one from the

ground up on his own. He might use the basic containers that are provided by the Conventional Ada Library to do so. He may even submit his engine to the R & D Corporation as a library extension. But if he is putting together an application that doesn't have all sorts of specialized needs and the calendar engine in the Conventional Ada Library comes close, he got most of what he needs right out of the box. If the calendar engine is built on an object oriented basis and source code is provided, chances are he can add whatever capabilities he wants and inherit the rest.

What if there were a whole suite of such engines? Wouldn't this make Ada an exciting and new thing for developers? What if the engines extended into areas involving the Internet? The mechanics of sending and receiving an e-mail on the Internet might be rather complicated, but what if they were hidden beneath an e-mail engine that just looked like an object to the programmer? He just declares an object of the appropriate type, feeds it the necessary data via the interface and automagically, e-mail gets sent and received.

Other languages may provide some of these capabilities either as standard parts of their library or as components that can be added from some other source. What would differentiate Ada is that it has not just a few such components, but a whole suite of them covering a wide array of possible applications. They would be well integrated and standard across many implementations. If Ada sets her mind on providing the maximum possible leverage to the developer with a wide variety of portable components, she can do it better than any other language. It would be new and exciting, providing something that would attract developers to Ada.

The idea here is to dream up as wide a variety of application components as possible and provide the portable parts to the developer to increase developmental leverage. "Everything but the GUI!" ought to be the motto. "…And the GUI comes later!" might be a possibility as well. In no particular order of importance, I have a list of potential candidates for a Conventional Ada Library below. It is not by any means complete, but it constitutes a good start and provides some topics for discussion.

Networking
- Transport layer - Sockets
- Session layer – Client/Server
- Presentation layer – Messaging protocol.

Internet Application Engines
- Web browsing / search engine.
- FTP engine
- E-Mail engine
- Telnet engine.
- Newsgroup engine
- Audio/Video engine
- Instant Messaging Engine
- NIST Time Reference engine

OS Interfacing

- File System management
- Process management (initiation and communication with other processes)
- Backup/Restore (ZIP file creation & management)
- Hardware Interfaces (sound card, ports, printers, specialized devices, etc)
- Image management (graphics rendering, formatting, etc.)
- Type Font management

Mathematics
- Statistics & Probability
- Matrix & Vector
- Big Number Arithmetic
- Symbolic Algebra
- Encryption/Decryption of streams.
- Compression/Decompression of streams.

Business
- Financial math packages
- Accounting engine
- Rolodex engine
- Calendar engine
- Word Processor engine
- Spreadsheet engine
- Presentation engine

Database
- Interface to SQL databases
- Back-end connection to one or more generally available databases – MySQL?

XML Document Object Model
- Read in & parse XML and assist the user in dynamically building XML.

Let's talk about a GUI interface for a bit as well. This is problematic because of the variety of OS provided GUIs available on everything from PC's to cell phones. It is difficult to find a way to make something that is portable, yet takes full advantage of the native "look and feel" provided on each platform. Should a standard GUI interface be an objective for the Conventional Ada Library? It ought to be considered.

We can note that GtkAda succeeds in providing a GUI that is usable on both Linux and Windows. We can also note that, in effect, browsers such as Netscape succeed in providing a standard interface on a variety of platforms. In both cases, there is a platform specific part and an interfacing protocol that allows platform independent application development. In the case of GtkAda, you have Ada package specs that connect to platform specific implementations. In the case of Netscape, you have a display/interaction engine that interfaces to the application via HTML and other mechanisms. A semi-portable GUI is at least possible to achieve.

Thought needs to be given to how a Conventional Ada Library might utilize existing protocols or develop its own to create a portable application interface to a platform specific GUI component. For practical purposes, if the implementation dealt with

Windows and X-Windows, it would cover the bulk of the territory in current use. Beyond that, we can't spend our lives worrying how to make everyone happy.

One solution I find attractive is to construct a platform specific display engine that communicates with an application using an XML based protocol. If a browser-like server was built and a communication protocol based on XML was developed, it is possible that an application program could send GUI descriptions and data to the server and expect back the results of user interaction. This separates the application from needing to know anything about the platform specifics on which the server is running, so it remains portable and rather naturally networkable. Could a Conventional Ada Library provide something along these lines and thus create a portable GUI for Ada? I think this is possible and utilizing other components of the library, the developer would see a well integrated product.

A GUI engine that used an open XML based protocol via a standard socket link might be attractive enough that users of other languages would want to utilize the engine on its own. After all, once you go to an open XML protocol, there is no reason the XML can't be produced by or read by a C++ program. If the GUI engines are available and the protocol is known, anyone can decide they would like to play with the new toys. That starts recruiting potential converts to Ada. Would it detract from Ada and just help promote the use of C++ or some other language? Not with any more success than Ada has had arguing that we can always bind to all those wonderful C/C++ libraries. As long as Ada is out front leading the way, it is always one jump ahead of the competition. Extend the invitation to users of other languages to utilize the GUI engine. It will always and forever be easier to do with Ada because the Conventional Ada Library is always providing more and better tools to go with it.

A standard GUI for Ada would be an attractive feature. It does, however, raise a large variety of questions. What sort of GUI model and interface should it use? What other tools would need to be made available in order for it to be practical? Should it attempt to incorporate any existing standards or develop something on its own? This could easily be the topic of another paper, so let's not try to go too far with it right now. There are two key points to keep in mind when thinking about a GUI for Ada. One is that a solution does exist. We should not dismiss the idea on the basis of impossibility. The other is that the solution need not be perfect. We can make compromises and live without absolute portability or other concerns so long as we get something that is useful in a majority of the potential uses.

Conclusion.

A Conventional Ada Library is certainly possible to build and incorporate as part of the Ada language if it is done outside of the Ada standard itself. All it really requires is a willingness on the part of the Ada community to agree to a plan and find the resources to build one. With such a library in place, Ada can offer important developmental leverage to its existing users and create an exciting environment that will attract new users. It doesn't have to be "perfect" and it doesn't have to be 100% portable to every imaginable

platform. It merely needs to cover a large enough variety of general uses and work with enough reliability that it creates leverage for the average developer.

Getting a Conventional Ada Library would require the cooperation of some key members of the Ada community and would require some financial commitments to see it through. It could be done on a lean budget if necessary by sharing the risks and rewards with those willing to participate. It can be made to happen so long as we see it as an important key to the future success of Ada and are willing to commit to a plan to achieve it.