Automating Component Integration for Web-Based Data Analysis[1]

John Penix, Bernd Fischer*, Jon Whittle**
Computational Sciences Division, M/S 269-2
NASA Ames Research Center / *RIACS/ **Recom Technologies
Moffett Field, CA 94035
(650)604-6576
jpenix,fisch,jonathw@ptolemy.arc.nasa.gov

Gilda Pour
Department of Computer, Information and Systems Engineering
San Jose State University
One Washington Sqare
San Jose, CA 95192-0180
gpour@email.sjsu.edu

Jeffery Van Baalen
Computer Science Department
University of Wyoming
Laramie, WY 82071
jvb@uwyo.edu

Abstract—In this paper, we describe a component-based infrastructure for Web-based data analysis applications that reduces the need for custom integration programming. Our vision is that, to construct an application, scientists will give a specification of three kinds of components: the desired data source, a data analysis filter, and a visualization device. From this specification, our infrastructure will automatically generate the desired application. To support this vision, the infrastructure contains an intelligent component integration system (iCIS) that automates component retrieval, adaptation, and integration. Within iCIS, component knowledge is represented using formal mathematical specifications, to provide a common framework for reasoning about problem decomposition, component retrieval and adapter generation. Our current focus is on integration of data analysis filters with online data archives described using XML-based languages. We also discuss how iCIS is integrated with a 3-tier client/server application architecture to support in a heterogeneous and distributed computing environment

TABLE OF CONTENTS

1. INTRODUCTION

Data analysis and visualization are major activities required for the support of NASA science missions. Numerous software toolkits and libraries are available to support the construction of custom data analysis and visualization applications. However, due to differences in data formats and system implementation styles, using these systems often requires space scientists to perform extensive custom programming that is time-consuming, expensive and not reusable between missions. In addition, current approaches to developing scientific software do not scale well. Scientific software is commonly developed from prototype systems and is evolved through experimentation. As the software is expanded and generalized, it becomes difficult to modify and maintain due to this ad-hoc development style. Because scientific problems are becoming larger and more complex, these software engineering issues are unavoidable.

To circumvent these problems, our approach is based on component-based software development (CBSD) [Jell 1998], a rapidly emerging trend in industrial software engineering. CBSD is based on the concept of building software systems by selecting, adapting and integrating a set of pre-engineered and pre-tested reusable software components. CBSD has the potential to: (1) reduce cost and development by allowing systems to be built from reusable components, (2) enhance software reliability as components undergo evaluation during each use, (3) improve maintainability and extensibility by allowing plug-and-play component replacement , and (4) enhance the quality of

[1] 0-7803-5846-5/00/$10.00 © 2000 IEEE

enterprise applications by allowing application-domain experts to develop components [Pour 1999a]. Of these factors, increased reliability is the primary benefit from the perspective of scientific applications [Dubois 1999]. In addition, CBSD simplifies component integration and addresses the scalability and maintainability problems of the current development process used to construct scientific applications.

To support the use of CBSD for web-based data analysis applications, we are integrating application generation technology into a middleware infrastructure. Domain specific application generation tools can then be developed using this infrastructure to support application specification and integration. Using one of these tools, scientists will provide a specification of three kinds of components: (1) the desired data source, (2) a data analysis filter, and (3) a visualization device. From this information, the application generator will automatically generate the desired application.

To provide this automated capability, the infrastructure contains an intelligent component integration system (iCIS) that automates component retrieval, adaptation, and integration using automated reasoning. It takes the user's request, locates the corresponding components in a heterogeneous and distributed environment, and uses knowledge about the components to automatically generate any adapters required for component integration. Thus, the need for time-consuming and expensive custom programming will be reduced.

For example, a space scientist may request the image of a certain astronomical object over a specified time period, without having to specify the specific instrument or data archive where the data is to be retrieved from. The scientist could then request that these images be sent through an analysis filter that highlights certain frequencies and displays the results as a Mpeg movie. In this case, iCIS will locate the appropriate filter, generate an adapter to allow the filter to use the image archive data format and generate an adapter to feed the highlighted images into the Mpeg generation program.

In the next section, we describe the formal framework for component-based software development that is the foundation for the intelligent component integration system. We then describe how iCIS is integrated with a 3-tier client/server architecture to support scalability and concurrent generation of multiple applications in a heterogeneous and distributed computing environment. The next section discusses our current work integration of data analysis filters with data archives described by XML metadata.

2. Component Integration Framework

The fact that components have well defined interfaces and behaviors can be leveraged to provide tools to automate some software development tasks. By specifying component interfaces and behaviors in machine-readable form, we can create algorithms that compare two components by calculating the difference between their interfaces and behavior. In our specific approach, we model component specifications using mathematical logic and base our algorithms on automated logical inference. Within iCIS, component knowledge is represented using formal mathematical specifications, specifically, higher-order parameterized theories. This provides a common framework to reason about problem decomposition, component retrieval and adapter generation.

Component Retrieval and Adaptation

For each component there is a specification that describes its interface and behavior. The interface specification describes the types and methods provided by the component. The behavioral specification describes the legal inputs to the component and the desired relationship between the inputs, outputs and abstract state of the component. These specifications are used to retrieve components and to determine the type of adapters that must be generated for integration. This is done using specification matching, where automated reasoning is used to compare component specifications. We currently have prototype implementations of component retrieval and matching that have performed very well in several empirical studies [Fischer 1998a,1998a, Penix 1999a].

The results of specification retrieval and matching are used to guide the selection of adapters that are used to integrate components. The system contains specifications of standard adaptation strategies that are used in the system. A strategy determines which adapter to generate and provides heuristics describing how to specialize it to the application being generated. Adapters are associated with each of the matching relationships that could exist between user request and library components. These adapters can be mapped into JavaBeans components using event-based component integration [Penix 1999b].

Synthesis Framework

The component retrieval and adaptation approach outlined above supports a bottom-up development style. This technique is successful when the problem specification and the component descriptions are conceptually close. In large-scale applications, there tends to be a larger gap between specification and components which must be bridged by a complementary top-down method. Deductive program synthesis [Manna 1992, Smith 1990] is such a method. At its core, it is based on the formal equivalence between proofs and programs, the so-called Curry-Howard isomorphism

[Howard 1980]. Thus, the technique fits well into our specification-based framework.

Deductive synthesis starts with a specification of a program to be synthesized. This specification is gradually refined by applying inference rules. The inference rules represent the relationship between domain-specific specification constructs and the behavior of library components. Therefore, the rules guide the system toward a reformulation of the problem in terms of integrated components. In the higher-order setting, program fragments to be synthesized are represented by higher-order meta-variables which are instantiated as the inference rules are applied. The result a flexible method for synthesizing programs that are guaranteed correct with respect to their specifications.

The full potential of the top-down and bottom-up approaches can be achieved by integrating deductive retrieval and deductive synthesis. The main problem is that each technique is suited to its own particular logic. Synthesis works best in a very expressive higher-order framework whereas retrieval works well in a first-order framework more amenable to automation. We have developed an integration algorithm [Fischer 1999b] which provides a bridge between these two logics. As input, the algorithm takes the internal state of the synthesis proof and identifies opportunities to reuse library code. In the higher-order setting, these opportunities correspond to the introduction of new, higher-order meta-variables (i.e., new program fragments) into the proof state. Each meta-variable is then linked to a retrieval query which is automatically extracted from the proof state. These queries are submitted to the retrieval system and the retrieved components are plugged back into the proof state, by instantiating the meta-variables

Intelligent Component Integration System

An overview of the information flow in the iCIS architecture is shown in Figure 1. The input to the system is a user request that describes a data source, a data analysis task and a data visualization component. There may also be additional information in the request that describes the intended data flow in the application indicating, for example, which data fields should map to data analysis parameters. The data source components are described by metadata that are used to locate the appropriate data set from a collection of registered data archives. Top-down synthesis is applied to decompose complex data analysis operations into problems that closer match library functionality. The component specifications generated during top-down synthesis are used to retrieve and adapt analysis components from libraries.

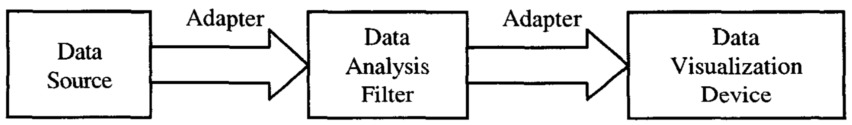

Figure 2: High-level Architecture of an Integrated Data Analysis System

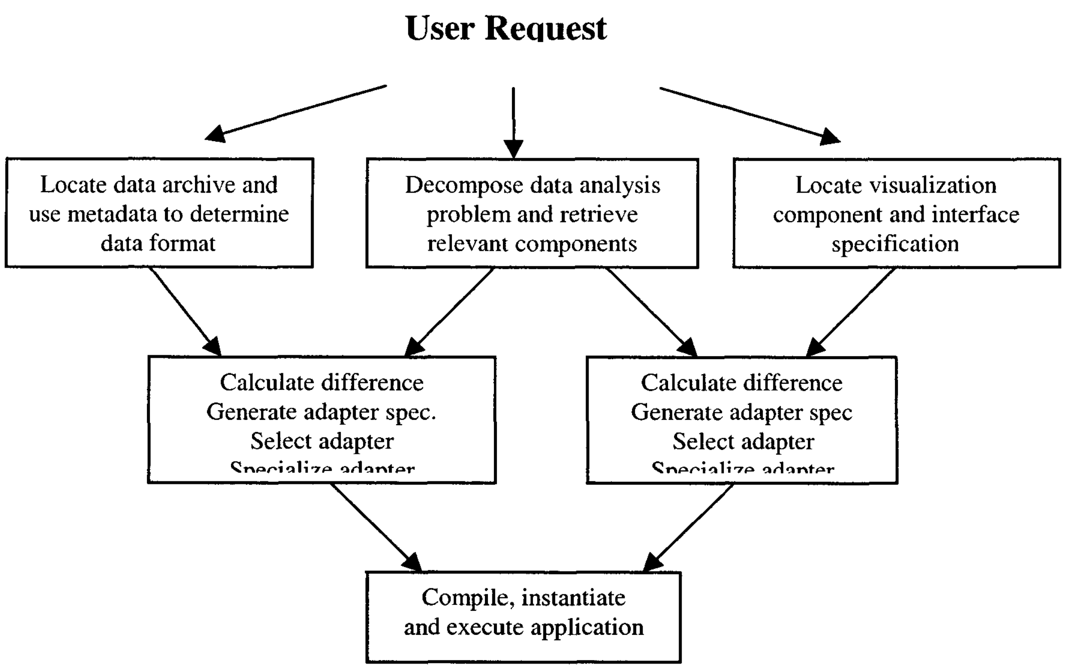
User Request
Locate data archive and use metadata to determine data format
Decompose data analysis problem and retrieve relevant components
Locate visualization component and interface specification
Calculate difference Generate adapter spec. Select adapter Specialize adapter
Calculate difference Generate adapter spec Select adapter Specialize adapter
Compile, instantiate and execute application

Figure 1: Information Flow in the Intelligent Component Integration System (iCIS)

Once the data and components are located, the metadata and component specifications are used to compare formats and calculate the difference that must be overcome for integration. This difference is converted into a specification that is used to select the type of adapter that will be created for integration and specialize it to the specific components being integrated [Penix 1997]. The adapters are stored as incomplete source code templates (Java Classes) and specialized by adding information (such as data type and function names) from the interface. The specialized adapters are compiled into executable form and instantiated into JavaBeans components [Penix 1999b]. Signaling the adapters to execute starts the data analysis application. Figure 2 shows the high-level architecture of an application generated by iCIS. The large arrows between the components represent the adapters that are automatically generated by the system.

Example

To illustrate how iCIS works, we will use and example from NASA's Vulcan Project, which is attempting to detect large planets in other solar systems. Their method for locating planets is to attempt to detect a periodic fluctuation in the intensity of a star that might indicate a large planet passing across it's disk. To perform this analysis, they track a collection of stars as they move across the sky each night. For each star, a sub-image is cut from the star field and then the sequence of these sub-images is realigned to correct for nightly drift. Image realignment is performed using statistical data analysis techniques.

To support Vulcan image extraction and realignment using iCIS, the format of the image data and the interface and behavior of the data analysis components must be specified. Inference rules must also be specified to capture the relationship between the abstract domain and the data analysis components. We can then retrieve, evaluate and integrate components to assemble the application.

The input to the analysis is a set of files that describe an area of the sky of a specified time window. Vulcan used the Flexible Image Transport System (FITS) as the data format to store telescope observations. FITS files contain a header which includes metadata such as bits/pixel, number of dimensions, time of imaging, object observed and telescope used. Indexing based on the FITS header metadata is used to catalogue and retrieve the data files and to determine data file format.

Given a sequence of sky images, the goal of data analysis is to construct a set of realigned image stacks for a specified set of stars. This type of analysis is supported by a statistical analysis software library. The functionality of the library components is specified in terms of a formal model of the statistical analysis domain. However, to support component retrieval and integration, there must be a mapping from the abstract application domain (star images and sky images) to the statistical domain of the component library. For example, it is necessary to realize that a star image is a 2 dimensional image with light intensity corresponding to a Gausian distribution in both dimentions. This information is represented in the form of rewrite rules over an algebraic specification language. For example, the following is a segment of the abstract domain specification for the star image domain:

```
type coord = (int,int);
type image[X] = seq[seq[X]]
op reow : image[X] -> seq[X]
op subimage : image, coord, coord -> image;
type star_image = {I:image[int] | gausian(I)};
type sky_image = image[int];
type star_name = string;
op star_location : sky_image, star_name -> star_image;
op inscope : star_name, sky_image -> bool;
op align : star_image -> (float,float);
```

Based on these abstract types and operators, we can construct a formal specification of the Vulcan data analysis problem in terms of a domain, range, precondition and a post condition:

```
Domain: seq[star_name] x seq[sky_image];
Range: seq[float,float];
Pre: s in starlist, I in skyimages => inscope(x,I);
Post: map(align, map(extract(starlist),skyimages));
```

This type of specification can be automatically extracted from more intuitive graphical representations, such as block diagrams or constraint graphs. In this case, the specification could be visualized similar to:

```
        (starlist,skyimages) -> extract -> align
```

We are currently working on developing a graphical representation of the star image domain.

The formalized problem specification can be decomposed using top-down synthesis by applying rules that describe how functions (such as extract and align) can be applied to sequences of data. The result are specifications describing components required to perform the extract and align functions for star images. For example, it is inferred that align must consider a Gausian distribution based on the type restriction specified for star images.

After the appropriate components are retrieved, adapters are generated to handle conversion from FITS to image sequences as well as data buffering between components. The buffers are inferred from the mismatch between the operator types in the graphical specification which are compensated for by the map operators in the formal postcondition. The adapter to perform the conversion from FITS is selected from a small collection of adapters that map from FITS tables into data sequences and streams. We anticipate that several such adapter templates will be required to support each data format standard.

3. 3-Tier Application Architecture

Three groups of components are identified in Web-based enterprise applications: (1) presentation, (2) application logic, and (3) data. The most commonly used architectures for Web-based applications are 2-tier and 3-tier architectures.

In 2-tier architectures, data components run on the server side (tier 2), and both presentation and application logic components run on the client side (tier 1). 2-tier architectures have been shown to have several drawbacks affecting their scalability, maintainability and performance. For example, integrating application and presentation logic lowers maintainability because evolution of one aspect cannot be performed independent of the other. In addition, a user of a 2-tier system can access only one data source at a time; access to other data sources must be done via gateways, raising serious performance issues for enterprise systems. Finally, 2-tier applications have limited scalability due to lock contention. Lock resolution is independent of server speed, so installing more powerful DBMS servers does not provide significantly greater performance.

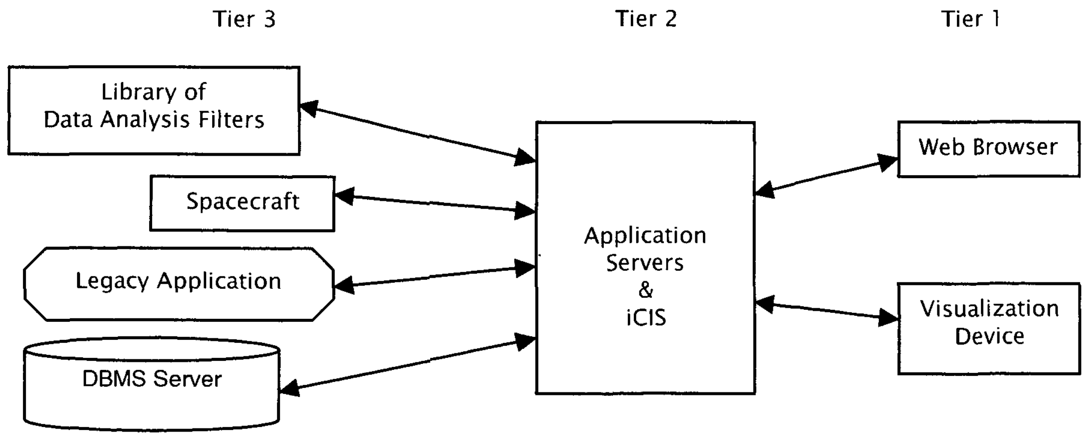

Figure 3: 3-Tier Web-Based Data Analysis and Visualization

In 3-tier client/server architectures, each component group (i.e. presentation, application logic, and data components) forms a distinct unit. Presentation components operate in tier 1, application logic components in tier 2, and data components in tier 3. Presentation components manage the interaction between users and the software system, and make requests for application services by calling application logic components in the middle tier. Application logic components manage the clients' requests for connections to the data components in tier 3. The client may request connections to multiple heterogeneous servers including back-end servers, DBMS servers, legacy applications through the use of their native interface, such as SQL for relational databases. The middle tier also resolves many difficult infrastructure issues such as naming, location, security and authentication, allowing application developers to focus on putting their domain expertise into the application logic.

3-tier architectures allow application components to run on middle-tier servers, independent of presentation interface and database implementation. As a result, they provide developers with more choices for enhancement of application scalability, performance, reliability, and security [Pour 1999b-d, Orfali 1998]. Additionally, 3-tier architectures have the following major benefits:

- System administrators can replicate application components and run them on different machines simultaneously. This will enhance the software availability, scalability, and performance.
- Application logic components can share database connections. This will improve software performance by lowering the number of total sessions that a database server must support.
- They provide access to other sources through native protocols & application interfaces rather than data gateways. This improves performance and allows users to control the data access.
- They allow developers to make the most of reusable application logic components, improving the software development and maintenance process [Pour 1999d].

To support scalability and concurrent generation of multiple data analysis applications in a heterogeneous and distributed computing environment, iCIS is being integrated with a 3-tier client/server architecture as shown in Figure 3. Web browsers and data visualization components operate in tier 1, the proposed iCIS and application servers in tier 2, and DBMS servers, spacecraft, legacy applications, and libraries of data analysis filters in tier 3.

After evaluating the major industry-supported component models, chose to use JavaBeans for client-side component development and Enterprise JavaBeans (EJB) [Thomas 1998] for server-side component development in this project. The Java-based component model is the best candidate for Web-based enterprise application development for several reasons including the portability and security features of Java [Pour 2000a,b,1999b,1998]. The EJB specification defines an application programming interface (API) that allows application developers to create, deploy, and manage cross-platform component-based enterprise applications easily. EJB also provides efficient data access across heterogeneous servers and allows scalability, reliability, load balancing, and atomic transactions. In addition, we have determined that the JavaBeans event model to be well suited for supporting adapter generation within iCIS [Penix 1999a].

4. Data Archive Integration

To integrate data analysis filters with various data archives, we propose to use technologies based on the eXtensible Markup Language (XML) [Bosak 1999]. XML is a new standard developed by the World Wide Web Consortium (W3C) for describing and exchanging structured data and metadata on the Web [Chang 1998, Maruyama 1999]. A large number of commercial and non-commercial tools are being developed to support the development of XML-based applications. These include many Java/XML software packages that can be used to integrate XML support into the Java-based component framework [Pour 1999b].

There are several existing efforts to use XML for storage and retrieval of space science data. We plan to use the results of two of these projects. The SSDOO's Astronomical Data Center (ADC) (http://adc.gsfc.nasa.gov) is currently developing XML-based data representations to support the storage and retrieval of data and metadata in astronomical catalogues [Shaya 1998,1999]. They are also developing an XML toolbox for importation, enhancement, and distribution of data and metadata documents written in XML. There is also an ongoing international effort to develop the Astronomical Markup Language (AML) for similar purposes [McGrath 1999].

By using XML, these efforts can leverage off the existing commercial tool support for creating and manipulating XML-based data representations. However, these projects do not address specific issues of integrating data with existing data analysis components or tools. Our proposed effort will build upon these projects by using XML-based descriptions of table formats (part of the metadata) to automatically generate adapter components that extract data from tables. We will also investigate the use of XML metadata to specify "data source components."

Our initial target data analysis application is from the Vulcan project at Ames, which is actively searching for planets orbiting distant stars. To support this effort, they must currently develop and maintain software to perform data file input/output, subsetting and reformatting, in addition to data analysis algorithms. Vulcan currently uses the Flexible Image Transport System (FITS) [NOST 1999] for data archival. In the proposed system, the FITS data will be retrieved from a remote site, and an adapter will be generated to read the FITS data, perform subsetting and reformatting, and direct the data to the analysis components. AML is currently undergoing revisions to unify the metadata format with the header file format from FITS, making this work relevant to supporting Vulcan.

In the Astronomical Markup Language (AML), metadata is used to describe the format of table data in the same file or in an external file. There is currently an international effort to unify the table metadata format in AML with the header format in FITS which will allow us to use AML to describe the data formats used in the Vulcan project. To support adapter generation, the metadata describing the data format will be read in using XML/Java tools. Then, this information will be used in combination with user specified data field selections to specialize an adapter template that will read the table and extract the desired fields. The interface specification of the data analysis component will be used to determine how to package the selected data for analysis.

Our initial source of data analysis components data analysis components will be from an ongoing collaboration between NASA Ames and U.C. Berkeley [Buntine 1999]. This is an effort to develop techniques to automatically generate probabilistic data analysis components from Bayesian network specifications, with the initial application also being the Vulcan project. The generated algorithms will form the initial component set to be integrated into the proposed infrastructure.

5. Conclusion

We have outlined the architecture of an intelligent component integration system that supports the component-based development of scientific data analysis applications. Our system operates on the middle tier of a three-tier web-based architecture and thus isolates data source, analysis, and visualization from each other. It takes as input high-level specification of a data analysis problem (i.e., meta-level descriptions of data contents, analysis filters, and visualization methods) and identifies appropriate data sources and analysis and visualization components. From these, it then synthesizes the application. The system relies on formal specifications to describe as the components, wrappers, and architectures and uses automated reasoning techniques to retrieve, adapt, and integrate components intelligently.

We have already implemented some of the computationally more intensive kernel reasoning components. Experiments with them have demonstrated that careful implementations can mitigate the inherent computational complexity of the specification-based component operations as retrieval and adaptation, which are required for component-based software development. We are currently working on integrating these reasoning components into an iCIS prototype which will cover the full functionality described in the paper.

References

[Bosak 1999] Bosak, J. and Bray, T., "XML and the Second-Generation Web", *Scientific American*, May 1999.

[Buntine 1999] W. Buntine, B. Fischer, T. Pressburger: Towards Automated Synthesis of Data Mining Programs. *Proceedings of the. ACM International Conference on Knowledge Discovery & Data Mining*, August 1999.

[Chang 1998] Chang, D. and Harkey, D., *Client/Server Data Access with Java and XML*, Wiley, 1998.

[Dubois 1999] Dubois, P.F., "Scientific Components Are Coming", IEEE Computer, March 1999.

[Fischer 1999a] Fischer, B., Lowry, M. and Penix, J., "Intelligent Component Retrieval via Automated Reasoning", *AAAI Workshop on Intelligent Software Engineering*, July 1999.

[Fischer 1999b] B. Fischer, J. Whittle: An Integration of Deductive Retrieval into Deductive Synthesis. Proceedings of the 14th International Conference on Automated Software Engineering (ASE-99, October 1999.

[Fischer 1998a] B. Fischer, J. Schumann, G. Snelting: Deduction-Based Software Component Retrieval. *In Automated Deduction – A basis for applications, Volume III: Applications*, Kluwer 1998.

[Fischer 1998b] B. Fischer: Specification-Based Browsing of Software Component Libraries. *Proceedings of the 13th International Conference on Automated Software Engineering*, October 1998.

[Howard 1980] Howard, W., "The formulas-as-types notion of construction", in J.P.Seldin and J.R. Hindley (eds.) *To H.B.Curry:Essays on Combinatory Logic, Lambda-Calculus and Formalism*, pp. 479-490, Academic Press, 1980.

[Jell 1998] Jell T., *Component-Based Software Engineering*, Cambridge University Press, 1998.

[McGrath 1999] McGrath, R., Futrelle, J., Plante, R. and Guillaume, D., "Digital Library Technology for Locating and Accessing Scientific Data", *The Fourth ACM Conference on Digital Libraries*, August 1999.

[Manna 1992] Manna, Z. and Waldinger, R., "Fundamentals of Deductive Program Synthesis", IEEE Transactions on Software Engineering, 18(8):674.

[Maruyama 1999] Maruyama H., Tamura K. and Uramoto N., *XML and Java: Developing Web Applications*, Addison-Wesley, 1999.

[NOST 1999] NASA/Science Office of Standards and Technology, Definition of the Flexible Image Transport System (FITS), NOST 100-2.0, NASA Goddard Space Flight Center, Greenbelt, Maryland, March 1999.

[Orfali 1998] Orfali, B. and Harkey, D., *Client/Server Programming with Java and CORBA*, Wiley, 1998.

[Penix 1999a] Penix, J. and Alexander, P., "Efficient Specification-Based Component Retrieval", *Automated Software Engineering Journal*, (6) pp. 139-170, April 1999.

[Penix 1999b] Penix, J., "Deductive Synthesis of Event-Based Software Architectures", *Proceedings of the 14th International Conference on Automated Software Engineering*, October 1999.

[Penix 1997] Penix, J. and Alexander, P., "Toward Automated Component Adaptation," *Proceedings of the Ninth International Conference on Software Engineering and Knowledge Engineering (SEKE)*, June 1997.

[Pour 2000a] Pour, G., "Component Technologies: Expanding the Possibilities for Web-Based Enterprise Application Development, "Chapter in, *Internet Technologies and Applications: Advanced Research, Theory, and Practice*, To appear, 2000.

[Pour 2000b] Pour, G., "EJB Server Component Model Revolutionizing Distributed Enterprise Software Development" *Software Development Magazine*, To appear, Jan. 2000.

[Pour 1999a] Pour, G., "Java-Based Component Model for Enterprise Application Development," *30th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS USA)*, IEEE CS Press, Aug. 1999.

[Pour 1999b] Pour, G., "Enterprise JavaBeans, JavaBeans and XML Expanding Possibilities for Web-Based Enterprise Application Development," *31st International Conference on Technology of Object-Oriented Languages and Systems*, IEEE CS Press, China, Sept. 1999.

[Pour 1999c] Pour, G. and Vadlakonda, V., "Web-Based Manufacturing Applications: Implementing Statistical Process Control with JavaBeans," *International Conference on Internet and Multimedia Systems and Applications (IMSA'99)*, Grand Bahamas, Oct. 1999.

[Pour 1999d] Pour, G. and Xu, J., "JavaBeans, Java, Java Servlets, and CORBA Revolutionizing Web-Based Enterprise Application Development," *World Conference of the WWW, Internet, and Intranet (WebNet)*, Oct. 1999.

[Pour 1998] Pour, G., "Developing Web-Based Enterprise Applications with Java, JavaBeans, and CORBA," *World Conference of the WWW, Internet, and Intranet (WebNet)*, AACE, Orlando, FL, Nov. 1998.

[Shaya 1999] Shaya, E., "XML at ADC: Steps to a Next-Generation Data Archive", *NSSDC News Letter*, http://nssdc.gsfc.nasa.gov/nssdc_news/, Vol 15, No. 1, March-June 1999.

[Shaya 1998] Shaya E., Blackwell, J., Gass, J., Weiland, J., Cheung, C. and White, R., "Formatting Journal Tables in XML at the ADC", *Astronomical Data Analysis Software and Systems VIII*, November 1998.

[Smith 1990] Smith, D.R., KIDS: A semi-automatic program development system", IEEE Transactions on Software Engineering, 16(9):1024-1043, 1990.

[Thomas 1998] Thomas, A., "Enterprise JavaBeans Technology: Server Component Model for Java Platform,"

John Penix received a BS in Electrical Engineering and a PhD in Computer Engineering from the University of Cincinnati. He is currently a member of the Automated

Software Engineering Group at NASA Ames Research Center and a part-time faculty member at San Jose State University. His research interests are component-based program synthesis and automated program abstraction and verification.

Bernd Fischer received an MS in Computer Science with distinction from TU Braunschweig, Germany, where he is a PhD Candidate in Computer Science. He is currently a member of the Automated Software Engineering Group at NASA Ames Research Center. His research interests lie in the area of deductive component retrieval and automated program transformation and synthesis.

Jon Whittle received a BA in Mathematics from St. Peter's College, Oxford and PhD in Computer Science from the University of Edinburgh. He is currently a member of the Automated Software Engineering Group at NASA Ames Research Center. He research interests are in the area of formal approaches to program and design synthesis, and empirical evaluation of software engineering tools.

Gilda Pour holds a BS in Electrical Engineering and an MS in Electrical and Computer Engineering from Florida State University, a Doctor of Engineering from the University of Florida and a PhD in Computer Science from the University of Massachusetts. She is currently on the faculty of Computer, Information and Systems Engineering at San Jose State University. Prior to joining SJSU, she was a Senior Software Engineer at Hewlett-Packard R&D Laboratories. Her research interests lie in the areas of component-based software engineering and web-based enterprise application development.

Jeff Van Baalen received a BS and MS from the University of Wyoming and a PhD in Computer Science from the Massachusetts Institute of Technology. He is currently an Associate Professor of Computer Science at the University of Wyoming and a part-time member of the Automated Software Engineering Group at NASA Ames Research Center. His research is in the area of deductive program synthesis, automated theorem proving and problem reformulation.