PFAST: An Eclipse-based Integrated Tool Workbench for Facilities Design

Thomas Mampilly
Department of Computer Science and Engineering
The Ohio State University
mampilly@cse.ohio-state.edu

Rajiv Ramnath
Department of Computer Science and Engineering
The Ohio State University
ramnath@cse.ohio-state.edu

Shahrukh Irani
Department of Industrial, Welding and Systems Engineering
The Ohio State University
irani.4@osu.edu

ABSTRACT
In this paper, we examine PFAST, an Eclipse-based integrated tool workbench targeted at rapidly integrating software tools for planning and optimizing manufacturing facilities. We describe the integrated-tool architecture, built on top of the Eclipse Rich Client Platform, which alleviates many of the problems faced by an earlier version of the system. We also describe our experiences in analyzing the requirements of the disparate tools that compose the system, the problems we faced in implementing the system, and the lessons learned. This paper highlights the successful introduction of Eclipse-based tool integration into the manufacturing facilities planning domain.

Categories and Subject Descriptors
D.2.6 [**Programming Environments**]: Eclipse

General Terms
Performance, Design, Experimentation, Human Factors, Standardization.

Keywords
Eclipse, Integration, Tool Integration, Rich Client Platform, Plug-in Architecture, Facility Design, Production Flow Analysis, Interdisciplinary Engineering.

1. INTRODUCTION
In order to adapt to a changing environment, a system must have the ability to be quickly modified or reconfigured to meet a new set of requirements. This ability to adapt is even more prominent when new technologies are being researched and developed – prototypes may need to be rapidly developed in order to expedite the testing and validation of new research. In the manufacturing domain, research is being conducted into finding new methods to design efficient factories that can adapt to changing product requirements and volumes. The move from monolithic architectures to increasingly modular systems has promoted adaptability to a great extent, but analysis tools are needed to plan and optimize how the modular factory should be designed or reconfigured.

The Production Flow Analysis and Simplification Toolkit (PFAST) [1] allows manufacturing facilities to be designed based on grouping similar products into product families and complementary groups of machines into machine cells. These groups or units can then be configured and recomposed to form manufacturing facilities that conform to "Lean Thinking." PFAST is continuously evolving, as an application, with new research in algorithms and strategies for facility layout being employed to create and update various analysis tools. The design process for facilities to be designed or reconfigured using PFAST is also not fixed, so tools have to be used in various combinations and sequences, with a high degree of user interactivity. Next, each new design process needs to be analyzed, tested, validated and compared to other processes in order to arrive at an optimal design process. Third party tools also need to be employed to increase the functionality of PFAST while minimizing development time. Integration requirements of these tools also need to be met by the architecture of PFAST.

The ideal solution that meets the requirements of PFAST is an integrated tool *framework*. This would allow developers to modify each tool, relatively, independently of one another, and compose each system or subsystem as a configurable composition of tools. Tool integration frameworks are often built, from the bottom up, based on requirements specific to the project. In order to enable the focus of effort to be on developing research, rather than application development, it is important to leverage existing tool integration infrastructure. The Eclipse platform and the Eclipse RCP provide an ideal infrastructure that enables the rapid development and integration of tools. In this paper we describe our experiences in introducing Eclipse to a new domain by building an integrated tool workbench for factory design using the Eclipse RCP.

2. BACKGROUND
2.1 Eclipse
Eclipse [2] is an infrastructure for building integrated development tools. Integration in Eclipse is enabled by an XML-based mechanism to define plug-ins. Each plug-in is a component that provides a specific service within the context of the system. The system is composed of several plug-ins each of which is integrated either by connecting to an extension point of another plug-in or by providing extension points into which other plug-ins can connect. The Eclipse platform provides dynamic discovery, linking and execution of plug-ins. Tool integration can be achieved in Eclipse by employing four levels of integration [3] within the context of tool relationships [4][5]:

Invocation Integration - Eclipse provides an operating system independent means of registering specific resource types with specific tools. The resources can then be launched in separate window instances with the corresponding tool being responsible for handling all aspects of the resource's contents.

Data Integration - Data integration allows tools to share data between each other using the underlying file system. Eclipse enables data integration using a resource manager that accesses shared data using standard file access.

API Integration - Eclipse's plug-in architecture enables API level integration by allowing tools to be integrated into the system by describing tool APIs in the plug-in manifest.

UI Integration - Eclipse provides several UI frameworks to enable disparate tools to be integrated into a single seamless application. Tools can also be integrated by registering themselves as interested in events generated by other tools.

The Eclipse RCP (Rich Client Platform) is a subset of Eclipse that enables sets of plug-ins to be developed and deployed as standalone applications, independent of the Eclipse development environment. The integrated tool workbench discussed in this paper is built on the Eclipse RCP.

2.2 PFAST

PFAST (Production Flow Analysis and Simplification Toolkit) is software that has automated the manual methods of Production Flow Analysis (PFA) [6]. PFA is a comprehensive method for material flow analysis, part family formation, design of manufacturing cells, and facility layout design. Each stage in PFA attempts to eliminate delays in production flows and operational wastes in a progressively smaller area of the factory. PFAST offers a facilities planner the ability to design four types of layouts to achieve the goals of PFA: Functional layout, Cellular layout, Modular layout, and Hybrid layout.

Research has shown [1] [7] that factory design cannot be thought of as a cookie-cutter process with one strategy being applicable to all types of manufacturing facilities. Specific design strategies must be chosen to meet the particular requirements of a specific facility. Even within a particular design strategy, tools may need to be used within the context of a specific process with specific algorithms and visualization techniques being applicable in different circumstances. In this paper we will examine one design strategy, the Cellular layout strategy, in depth to gain an understanding of the specific requirements of tool integration within the context of the strategy and to demonstrate how the Eclipse RCP can meet these requirements.

2.3 Cellular Layout Strategy for Facility Design

Each tool in this section is analyzed with respect to the integration requirements of both the input and output data of the tool. These requirements are described in terms of the 4 levels of integration as described in section 2.1 along with user interaction requirements. Also, certain tools are optional to the design process. The integration framework must provide the capability to leave such tools out.

The cellular layout strategy can be decomposed into the following constituent tool types:

2.3.1 Product Mix Segmentation

This is an optional tool within the Cellular layout strategy. This tool allows users to filter the input data based on certain criteria as described below:

Tool Criteria 1: Part-Quantity – The product mix is sorted in descending order of volume of production of each part.

Tool Criteria 2: Part-Quantity-Revenue – The product mix is sorted in descending order of volume of production and revenue generated by each part.

Input integration requirements – Data level integration. The input to the product mix segmentation tool is independent of other tools within the framework and can be read directly from the input file.

Output integration requirements - Data level integration - The output of the tool produces a filtered version of the original input file, in the same format as the original input file.

User interaction - The user needs to be able to select the criteria with which to filter the input data and needs to be able to select sets of input data points.

2.3.2 Cluster Analysis

This is an essential tool that clusters the product mix based on the similarities of flow routes (manufacturing process steps) between products.

Input integration requirements – Data level integration. The input to the Cluster Analysis tool is dependent on the product mix segmentation tool and can be read directly from the output file of that tool or directly from the input file if the product mix segmentation tool is unavailable.

Output integration requirements - Data level integration. Since the tool needs to be run only once per analysis session, the output of the tool (similarity and cluster information structures) can be defined using standard data integration formats such as XML.

User interaction - Depending on user requirements, the ability to choose a specific clustering algorithm may need to be made available to the user.

2.3.3 Multivariate Analysis

This is an optional tool that heuristically determines the optimum number of clusters or cells in the product mix.

Input integration requirements – Data level integration. The input to the Multivariate Analysis tool is dependent on the Cluster Analysis tool and can be read directly from that output file.

Output integration requirements - UI level integration. The output of the tool is the optimum range of clusters or cells in the facility layout. This tool can be run multiple times within a single design session and must therefore automatically update any visualization of these clusters.

User interaction - The user must have control over the criteria with which to establish the optimum cluster range.

2.3.4 Visualization Tools

Input integration requirements – UI level integration. The input to the visualization tools are dependent upon other tools in the process and must be dynamically updated.

Output integration requirements - UI level integration. The output of the tool is a visualization of certain attributes of the data and must be constantly updated.

User interaction - Selections made in one visualization tool must be made visible to other visualization tools

2.3.4.1 Visualizations

Each visualization tool is other than the Tree view tool is an optional tool. However, each additional visualization tool provides enables the user to make better design decisions.

Tree view - Displays the clusters in a tree format and allows the user to select the number of clusters (value of K) to be formed.

Load Profile view - 3D bar chart showing the workload on each machine within a cell or cluster.

PQ Analysis view - Line plot showing the relationship between products and their production volumes.

PQ$ Analysis view - Scatter plot showing the relationship between products, their production volumes and the revenue generated by them.

Flow Diagram view - Shows details of flow between machines within a cell or cluster.

Details view - Shows details of each product in the input data in a tabular format.

2.4 Summary of Integration Requirements

Figure 1 below shows how the tools are interconnected.

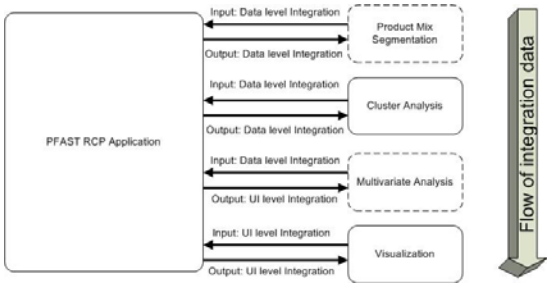

Figure 1. PFAST RCP Application Tool Interconnection. Tools shown as dotted boxes represent optional tools. The flow of data needed for integrating the tools within the process is shown by the thick arrow on the right side of the diagram.

It can be seen from the above section (section 2.3) that integration requirements are not fixed across all tools. The level of integration varies from simple data integration to seamless UI integration. Since the design process is flexible, user interaction is of high importance. Users need to be able to adjust the design process based on specific design requirements and select tools and their functionalities in as simple a manner as possible. Not all tools are essential to the factory design process; therefore, the ability to include or exclude certain tools from the workbench must be available. All features of a particular tool may not be required within specific design processes. The ability to activate and de-activate certain tool features must be available. Ongoing research and changing user requirements may require existing tools to be modified, extended or replaced, and new tools to be incorporated into the design process. As an aside, the analysis of requirements (see Section 2.3) assumes that the tools are written in a common language. However, the use of external tools written in other languages may be required. The level of integration may, therefore, encompass API integration, in addition to the other levels of integration identified.

3. INTEGRATED TOOL WORKBENCH FOR FACTORY DESIGN

3.1 Existing Architecture

PFAST is currently implemented as a standalone MFC application. The modularity of the current design is minimal with limited use of object-oriented concepts. There are; however, some benefits to the model-view architecture used by MFC applications [8]. Separation of the document or data of the application from its presentation or views allows for different levels of integration. Data level integration can be achieved by loading data into different document structures. UI level integration can be achieved by associating multiple views with different document structures.

Integration with external tools, written in different languages, is difficult but achievable through technologies such as CORBA, DCOM and RMI [9]. The current architecture does not permit tools to be deactivated or limited in functionality easily, although this can be achieved through (mostly minor) code changes. Since the modularity of the existing architecture is minimal, updating existing tools or adding new tools requires existing code to be modified with the possibility of changes affecting the system in an unpredictable manner [10].

3.2 The PFAST Integrated Tool Workbench

3.2.1 Implementation

PFAST was implemented within the Eclipse RCP framework, taking advantage of the existing tool integration infrastructure and other features as described below.

Each tool described in section (2.3) was implemented as a separate Eclipse plug-in.

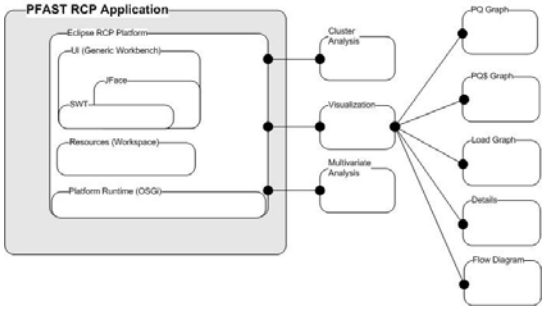

Figure 2. PFAST RCP Application architecture.

The main plug-in, essential to the Cellular strategy, is the Cluster Analysis plug-in. The algorithms related to this plug-in were implemented in the MFC version of PFAST. These algorithms were bundled into a shared library, which can be called using the Java Native Interface (JNI). The output produced by the calls to native code is stored in the shared workspace and can therefore be used through data level integration by the main RCP application.

The Multivariate Analysis plug-in was implemented using a third party open source library. Data level integration was used to obtain input for the plug-in. UI level integration was enabled

using a standard SWT widget (Text box) along with the Eclipse selection listener.

The Visualization plug-in extends the main RCP application. All individual visualization tool plug-ins extend the main visualization plug-in and can therefore be notified of changes simultaneously. Each plug-in is notified of events generated by other tools in the system according to the Observer pattern [11]. These plug-ins implement event handlers to perform the specific functions in response to these events, and according to the visualization requirements.

Dynamic plug-in discovery allows non-essential tools to be omitted from the plug-in folder as and when necessary. When the RCP application is loaded by the Eclipse platform, only the plug-ins in the plug-in folder will be loaded and integrated. Since there is a minimal amount of coupling between plug-ins, omitting any non-essential plug-ins from the plug-in folder will not have a detrimental effect on the system and it will perform according to expectations.

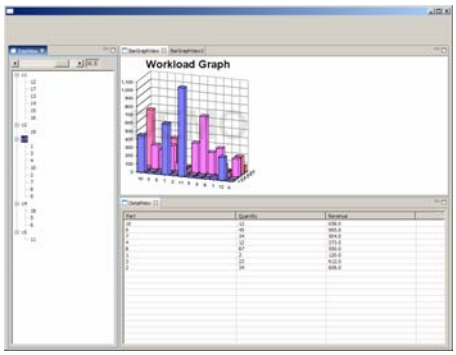

Figure 3. PFAST RCP Application.

The integration requirements for each tool as described earlier (see Section 2.3) were adhered to and several features of Eclipse were leveraged to enable a quick realization of the requirements. The relationship between the integration requirements of each tool and the Eclipse features used to realize them are shown below (see Table 1).

3.2.2 Problems Encountered

As mentioned in section (3.2.1), individual visualization tools extend the Visualization plug-in. The Eclipse plug-in registry API enables the processing of extensions iteratively, allowing a host plug-in to query the members of its extensions and process them as required. Although this feature allows the decoupling of extensions from their host plug-ins, the plug-in registry API involves a definite learning curve. Implementing extension processing had to be done with care making sure that the design was unambiguous while providing enough information for specific callbacks to be implemented.

Integration with third party or open source libraries was problematic at times due to the move from Java 1.4 to Java 1.5. It was necessary to roll back to Java 1.4 frequently to find the best solution to the integration problems. On one occasion it was necessary to switch to a different library due to Java version incompatibilities. This was however relatively painless and was accomplished simply by creating a new plug-in that used the new library. This allowed plug-ins to be swapped in and out of use when Java version compatibility became an issue.

The process of creating shared libraries from existing native code was a laborious, time-taking process involving large amounts of movement of code between classes and the creation of new classes to replace procedural code.

3.2.3 Lessons Learned

Eclipse's dynamic plug-in discovery allows tools to be added and removed as and when necessary while allowing the system to behave as expected. This was particularly useful when testing different visualization libraries as it permitted similar plug-ins to be created for each one and compared with one another. Overcoming problems due to Java version differences were also made simpler through the ability to swap plug-ins in and out of the application.

Initially, various visualization plug-ins were implemented as extensions to the main RCP Application. This proved to be quite inefficient in terms of how each event was handled: each plug-in needed to listen for changes in the system. The move to an inheritance-type design for the visualization plug-ins allowed this problem to be overcome. A single host plug-in (the Visualization plug-in) is notified of relevant changes in the system, and this plug-in then processes the notification as required, calling specific members of its extensions. This type of design allows plug-ins to be categorized and handled accordingly.

One of the most important advantages that the new architecture provides is the ability to test and validate new ideas. In a matter of weeks, two new ideas were implemented and tested. A method to determine the optimal range of clusters or cells in a facility layout was implemented, tested and validated. A multi-attribute visualization method to visualize clusters was tested on several datasets and was determined to not be as useful as initially assumed.

The rapid development of multiple plug-ins was accomplished by leveraging the user interface features available in the Eclipse platform. SWT widgets and JFace UI components allowed the focus of the development to be on choosing the right tools for the facility design strategy rather than getting mired in developing user interface features such menu bars, data structure viewers, and dialogs.

4. FUTURE WORK

The process by which a facility designer arrives at a solution to a design problem may involve using specific tools in a specific order, and as new tools are added to the tool base, this process may have to evolve as well. Tool integration within the context of a particular process [12] can be implemented by defining tools using a tool modeling language and the process in which they act using a process modeling language. Future work on PFAST will attempt to make use of existing modeling languages and, if necessary, adapt them to the requirements of PFAST.

PFAST consists of four layout strategies: Functional layout, Cellular layout, Modular layout, and Hybrid layout; however, only the Cellular layout strategy been implemented in the Eclipse RCP framework to date. We intend to implement all four strategies, thereby allowing the user to choose a design strategy based on the specific requirements of the facility to be designed or reconfigured.

Table 1. Integration requirements and their corresponding Eclipse features

Tool Type	Tool	Integration Requirements	Eclipse Features Used
Analysis	Clustering Tool	Data level integration & API level integration	Private workspace file access, JNI calls
	Multivariate Analysis	Data level integration & UI level integration	Private workspace file access SWT Text box & selection listener
Visualization	Tree view	Data level integration & UI level integration	Private workspace file access JFace Tree viewer & selection listener
	Graph visualizations	Data level integration, UI level integration	Private workspace file access & selection listener
	Details table	Data level integration, UI level integration	Private workspace file access JFace Table viewer & selection listener
	Flow Diagram	Data level integration, API level integration, UI level integration	Private workspace file access, JNI calls & selection listener

5. CONCLUSIONS

The Eclipse RCP in conjunction with our integrated tool architecture enables PFAST to keep up with technological advancements with minimal integration overhead. Using the Eclipse RCP, we were able to interchange tool plug-ins within a given facility design strategy without the need to modify any source code.

The PFAST RCP application was tested on various data sets and the ability to use different tools depending on specific requirements of the data translated into a huge advantage in terms of testing and validating research ideas. In the past, very large amounts of time were spent in analyzing and formalizing new methodologies before their implementation due to the high cost and turnaround time for integrating the new method into the largely monolithic system. With the new architecture, however, it is easily apparent that new ideas can be implemented with negligible development effort and they can be then tested and if necessary disregarded before spending unnecessarily large amounts of analysis time. This result highlights the success of the introduction of Eclipse into the facilities planning domain.

A few commercial and open source tools were integrated into PFAST. The process was simple and efficient, even in light of some compatibility issues. This kind of modularity will help PFAST grow quickly by allowing the quick integration of research tools being developed at other locations and by allowing PFAST to be integrated with commercial software systems.

6. REFERENCES

[1] Irani, S. A., Zhang, H., Zhou, J., Huang, H., Udai, T. K. & Subramanian S. *Production Flow Analysis and Simplification Toolkit*. International Journal of Production Research, 38(8), 1855-1874. 2000.

[2] www.eclipse.org.

[3] Amsden, J. *Levels of Integration: Five Ways You Can Integrate with the Eclipse Platform*. Eclipse Corner Article. March, 2001.

[4] Wasserman, A. *Tool Integration in Software Engineering Environments*. Lecture Notes in Computer Science, #467, Springer-Verlag, Fred Long, ed., ISBN 3-540-53452-0. 1990.

[5] Thomas, I., Nejmeh, B. A. *Definitions of Tool Integration for Environments*. IEEE Software, v.9 n.2, p.29-35, March 1992.

[6] Irani, S.A. & Zhou, J. (1999). *Production Flow Analysis.* Industrial Engineering Applications and Practice: A Users' Encyclopedia (CD-ROM, ISBN 0-9654599-0-X), A.K. Mital & J.G. Chen (Eds), International Journal of Industrial Engineering, Cincinnati, OH. 1999.

[7] Irani, S.A., Zhou, J., Huang, H. & Udai, T.K. *Enhancements in Facility Layout Tools using Cell Formation Techniques.* Proceedings of the 2000 NSF Design and Manufacturing Research Conference, Vancouver, BC (Canada), January 3-6, 2000.

[8] G. Shepherd, S. Wingo, *MFC Internals: Inside the MFC Architecture*. Addison-Wesley, 1994.

[9] Bechini, A., Foglia, P., and Prete, C. A. *Use of a CORBA/RMI gateway: characterization of communication overhead*. In Proceedings of the 3rd international Workshop on Software and Performance. ACM Press, New York, NY, 150-157. 2002

[10] Parnas, D. L. *On the criteria to be used in decomposing systems into modules*. Communications of ACM 15, 12 (Dec. 1972), 1053-1058. 1972.

[11] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc. 1995.

[12] Pohl, K. and Weidenhaupt, K. *A contextual approach for process-integrated tools*. In Proceedings of 5th ACM SIGSOFT International Symposium on Foundations of Software Engineering. Springer-Verlag New York, New York, NY, 176-192. 1997.