SFDC 1015

The JANUS Application Development Environment—Generating More than the User Interface

Helmut Balzert, Frank Hofmann, Volker Kruschinski, and Christoph Niemann

Lehrstuhl für Software-Technik, Ruhr-Universität Bochum, Universitätstraße, 150, D-44780 Bochum, Germany
Phone: +49-(0)234-700-{6880, 6791, 5918, 7982}
Fax: +49-(0)234-700-6914
E-mail: {hb, hofmann, krusch, niemann }@swt.ruhr-uni-bochum.de, janus@swt.ruhr-uni-bochum.de
WWW:http://www.swt.ruhr-uni-bochum.de/forschung/veroeffentlichungen.html

Abstract

The increasing pressures of competition demand greater productivity and quality in the development of software. These goals are attainable by generating as much as possible and programming as little as necessary. Beginning with an OOA modeling of the problem domain component, this article will show how the user interface as well as the linkage to data keeping can be generated through an integrated approach. In addition, a client/server configuration is also possible. A OOA model upon which two generator systems are installed is the basis for generating.

Keywords

User interface generation, OOA model, object oriented database, rapid prototyping, application framework.

Introduction

The ever increasing demands on the productivity and quality of software development necessitates extensive automated support for application development. If one examines object oriented application development (figure 1), the way from the problem domain to an object oriented analysis model (OOA model) cannot be automated. This step shall continue to belong to one of the most ambitious tasks of software development.

If an OOA model is created, it forms the basis for any additional steps of development. The concepts available today describing an OOA model (class, inheritance, association, aggregation, object life cycle, interaction diagrams, subsystems, see also [Coad91a, Booch94, Rumbaugh91]) allow close to real-world situation modeling of the problem domain.

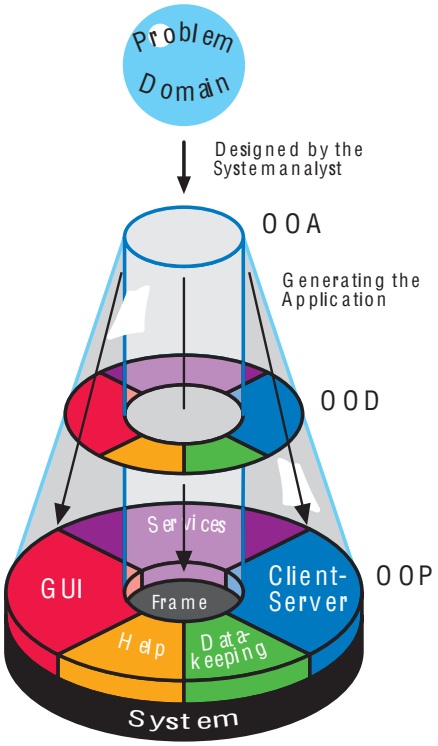

Figure 1. The way to an application starting at the problem domain

The following must be done to obtain a usable application from a OOA model:

- Integration into the system software of the target system.
- Design and connection of the user interface to the problem domain components.
- Connection to the desired data base management system (DBMS).
- Design and connection of the help system.
- Creation and connection of various services (e.g., multiple user administration, client administration, etc.).

Analyzing the jobs to be completed, one ascertains that a large part of these tasks can be automated by generators. The term "automated" is intentionally used instead of "automatic". Automated is intended to express that generating does not run fully automatically, but rather that the developer retains the possibilities to intervene and make decisions during the generating process.

Therefore, the optimal goal consists of generating nearly all additional necessary tasks from an OOA model. The semantics of a problem domain component are principally incapable of being generated, i.e., the technical semantics have to be implemented by the software developer. He uses the desired programming language (inner column of figure 1).

Even in this area, however, much can be generated. Today's OOA/OOD tools allow the corresponding program frameworks to be generated from the OOA model, e.g., the tools Together/C++, Paradigm Plus and ObjectiF. To have a practical benefit of generating system components the developer needs an integrated system which will combine all fragments.

Furthermore, it is not enough to generate all components from the same starting point (e.g., an OOA-model) an integration of all generated parts can be done automated. Therefore we have developed the JANUS Application Development Framework (JADE[1]). It is a further development of the JANUS-system [Balzert93, Balzert94, Balzert95a, Balzert95b]. The JANUS-system was capable of generating and animating a graphical user interface from an OOA model using the capabilities of an UIMS.

The advanced system now produces the user interface, the code frame for the application domain, the database schema, further services (e.g., a help system, printing facility) and 'last but not least' the connection between all these parts. The starting point is still an OOA-Model. JANUS requires the model in a well defined input language, the JDL (JANUS Definition Languages) which is an extension of ODL and IDL. To avoid that the user has to code his OOA model using this language directly, we have built interfaces to some popular OO CASE tools. Currently JDL can be exported by the case tools Paradigm Plus and Together C++.

The result of the generation process is a ready-to-work-with application offering basic functionality. The user is able to create and modify objects of classes defined in OOA by using entry forms. If a corresponding relationship (association or aggregation) exists in the OOA model the user can establish links between objects, too. Additionally a list view of all objects that have been created for each class is provided.

Functionality for sorting and deleting objects is also generated. All data entries are kept persistent in an underlying database. Until now the software developer has not written a single line of code. The only work that has been done was defining an exact OOA model of the application's problem domain.

The generated program will however be the fundamental frame of a final system. A programmer will have to complete the application. He has to implement the operations defined in the OOA model to provide the application's core functionality. Additional features—especially regarding the GUI—can be added to the generated code. To ease this JANUS generates C++ source code for all parts of the program. These can be edited and compiled the normal way. This paper describes the concepts of integrating all parts. It gives examples of the transformation process and its results.

---

[1] This JADE system has nothing in common with JADE [VanderZanden90] but the name. It seems that we have no luck in choosing the right name for our system.

1 As to the Situation

The situation today is characterized by increasing attempts to automate separate areas of the software development process. Class libraries in combination with a graphical editor are used today in the development of GUIs. GUI class libraries are hierarchically organized and provide predefined interface objects at higher abstraction levels. The activation of the underlying window system is undertaken by internal operations and remains hidden from the developer. The design of the GUI using this technique leads to two results:

- A code frame will be generated in the desired programming language (usually C++). The combined interface objects can be created dynamically using this code. The I/O operations of these objects have to be manually linked to the OOA model.
- Characteristics of interface objects such as position, size, labeling, and shape will be placed in resource files. Each resource object contains an identification through which the connection to the objects implemented in the programming language is made. A special resource translator transforms the resources into object code, which will later be linked to the application.

It was shown under the JANUS system [Balzert93, Balzert94, Balzert95a, Balzert95b] that a GUI can be generated and subsequently animated from an OOA model based upon expert knowledge of software ergonomics.

However, the linkage to data keeping in particular is missing in order to attain a usable application. When using an object oriented database (OODB), the object model is defined in an Object Definition Language (ODL). The developer separates the declaration (data and interfaces) of an application from the implementation. A declaration preprocessor for the ODL takes over the following tasks:

- The ODL is transformed into a declaration conforming to a programming language which then can be translated by a compiler together with the implementation of the application.
- A database with the database schema obtained from the ODL declaration is created in which the object model of the application is also established as a meta schema.

The implementation of the technical semantics of the OOA model occurs in the selected programming language. To handle persistent objects, an Object Manipulation Language (OML) is provided by an external library. This library comes with the chosen database management system. With this, the programmer can manipulate persistent objects with the same concepts (pointer, list,...) known from the programming language as usual.

The declarations transformed in the programming language and the implementation are translated by the compiler into object code. The runtime system ODBMS is added to the object code during linkage so that the finished application can ac-

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.