The ITHACA Application Development Environment *

Vicki de Mey and Oscar Nierstrasz†

University of Geneva

Abstract

The goal of ITHACA is to produce a complete object-oriented application development environment. This paper reports on the status of ITHACA in relation to this ambitious goal concentrating on the tools comprising the application development environment. Some general observations and recommendations are made concerning the integration of the tools. Future directions of the project are also outlined.

*.In *Visual Objects*, ed. D. Tsichritzis, Centre Universitaire d'Informatique, University of Geneva, July 1993, pp. 267-280.

†.*Author's current address:* Institut für Informatik und angewandte Mathematik (IAM), University of Berne , Länggassstrasse 51, CH-3012 Berne, Switzerland. *Tel:* +41 (31) 631.4618. *E-mail:* oscar@iam.unibe.ch. *WWW:* http://www.iam.unibe.ch/~oscar.

1 Introduction

The goal of ITHACA is to produce a complete object-oriented application development environment that can be easily adapted to various application domains [1]. The ITHACA platform is an application support system that offers advanced object-oriented technology consisting of an object-oriented programming language and environment, application development tools and application workbenches. ITHACA targets the flexible construction of large-scale open applications characterized by continuously expanding hardware and software platforms and evolving requirements. More precisely, the ITHACA platform consists of:

- An *Object-Oriented Programming Environment* including an object-oriented language, called CooL, which has support for persistence and transactions, an object-oriented database, and programming and debugging tools;
- An *Application Development Environment* including:
  - A Software Information Base (SIB): for storing all information concerning software reuse and the development process, and including a graphical browsing and querying interface;
  - A Requirements Collection and Specification Tool (RECAST): for guiding the application developer in the identification of reusable software information;
  - A Visual Composition Tool (Vista): for interactively constructing applications from pre-packaged, plug-compatible software components;
- *Application Workbenches:* for selected application domains, including Public Administration, Office Systems and Financial Systems.

ITHACA is a project of two phases: ITHACA-2 and ITHACA-IT. ITHACA-2‡ was a Technology Integration Project in the Office & Business section of the European Community's Esprit II Programme and ended in 1992. ITHACA-IT** (Innovation Transfer) is a B-Type project in the field of "Integrated Business Systems" and runs to mid-1994. ITHACA-IT has the goal of finalizing and industrializing the work started in ITHACA-2. Significant progress was made during ITHACA-2 on the object-oriented programming environment and application workbenches. The application development environment addresses many current research issues and is still evolving.

This document will briefly describe the component-oriented software life cycle which is the base for the work in the ITHACA project. In section 3, a summary of the tools developed in ITHACA-2 will be made. Section 4 will reflect on the work done in ITHACA-2 and report some observations and recommendations for continued work on the tools [20]. Section 5 will introduce ITHACA-IT and the work Geneva will be doing in this phase of the project. Section 6 closes the paper with some final conclusions.

2 Component-Oriented Software Life Cycle

The ITHACA software development environment [12] is based on a component-oriented software life cycle [22]. The scenario for application development includes the following steps:

1. Starting with some very general application requirements, select a *generic application frame* (GAF) from a software information base (SIB). The GAF encapsulates for a given application domain the domain knowledge, the requirements model, generic designs, and abstract and concrete classes (software components).
2. Guided by the GAF, specify the requirements for the specific application and identify and refine relevant designs and software components.
3. Bind reusable components together to form a running application.
4. Monitor the application for correct behaviour and adapt it to evolving requirements.

‡.The partners are Siemens/Nixdorf (Berlin), Bull (Paris), Datamont (Milan), TAO—Tècnics en Automatitzaciò d'Oficines (Barcelona), FORTH—the Foundation of Research and Technology, Hellas (Iraklion) and CUI—the Centre Universitaire d'Informatique of the University of Geneva.

**.ITHACA-2 partners minus Datamont.

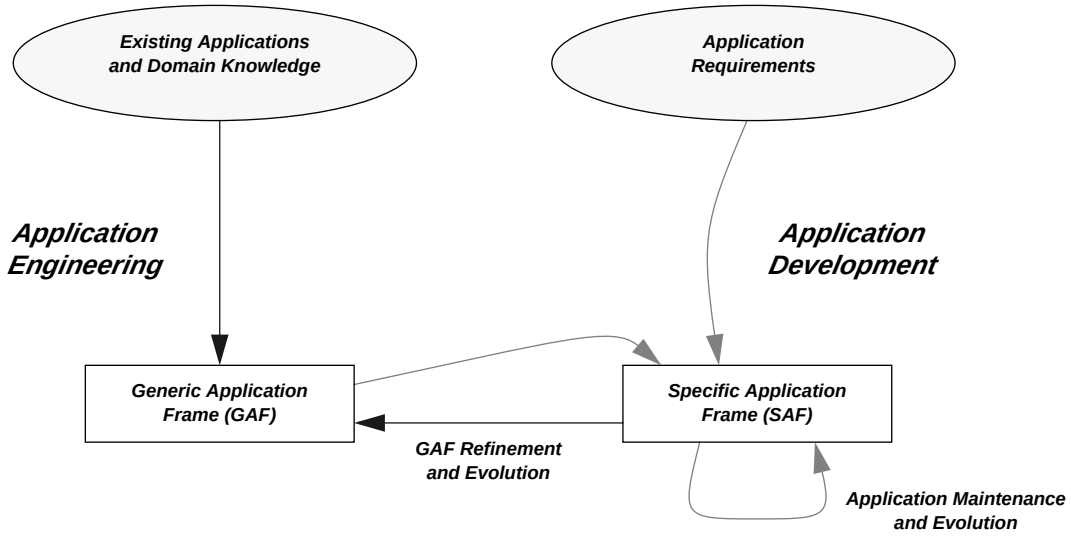

Figure 1 Component-oriented software life cycle.

5. Use experience gained from the new application to refine the GAF.

A key assumption behind this scenario is that sufficient domain knowledge and experience have been gathered from specific applications to make it possible to abstract general knowledge. This approach relies on two quite different activities:

1. ***Application Engineering*** is the activity of abstracting the domain knowledge for selected application domains, developing reusable software components to address these domains, and encapsulating this knowledge into GAFs;
2. ***Application Development*** is the activity of instantiating a specific application from a GAF to meet some particular requirements.

Traditionally, the roles of application engineering and application development have not been so sharply divided. In this scenario however, these two activities must go on concurrently. Feedback from the application developer goes to the application engineer and the application engineer supplies new "parts" for the application developer to build with as seen in figure 1. The application developer and the application engineering are not necessarily two different people but the activities they perform are separate. Capital investment occurs during application engineering. This is an expert activity that requires detailed understanding of both the application domain and of the mechanisms available for packaging and composing software components. The return on investment should occur during application development. Applications will be easier to develop, will be more robust and reliable, and will be more flexible and easy to adapt to evolving requirements. The degree to which application development is streamlined depends on (1) the quality of application engineering preceding it, and (2) application development tools to support reuse. The essential difference between a component-oriented software life-cycle and a traditional one is that application engineering introduces design for reuse. Two important points need to be stressed:

- GAF development is iterative and evolutionary.
- Software components are not reusable in isolation, but only as part of a generic design i.e. reuse occurs by design.

3 ITHACA-2 Tool Summary

The architecture of the application development environment is depicted in figure 2. It is composed of tools for application specification (RECAST), for composition support (Vista) and for storage and extraction of development information (SIB system). The environment supports reuse of development information at every stage of application creation.

All the tools have found the need for some type of graphical editing facility. This support for implementing the tools is supplied by Labyrinth [14] [15]. Labyrinth is a graphical editor for hierarchical networks of interconnected, parameterized objects with an event-driven evaluator that maintains acyclic constraints. Labyrinth defines a set of classes for the data structure that represents a network. There is a set of routines defined to search and edit the network and to display the network on the screen. These routines are methods on the classes or functions. This section continues by summarizing the status of the tools at the close of the ITHACA-2 phase of the ITHACA project.

3.1 The SIB System

The ITHACA tools assume and support the existence of a common repository for reusable software development information. The SIB system [6] [7][8][26] is the repository used in ITHACA. The SIB provides the underlying mechanisms for storing, representing, and maintaining reusable component descriptions in the application development environment. The SIB elements are structured *descriptions* of software components, and of their associated requirements and designs in terms of knowledge concerning *application domains*. Since descriptions may refer to one another, the contents of the SIB is organized as a semantic network, sharing properties of both object-oriented databases and hypertext systems.

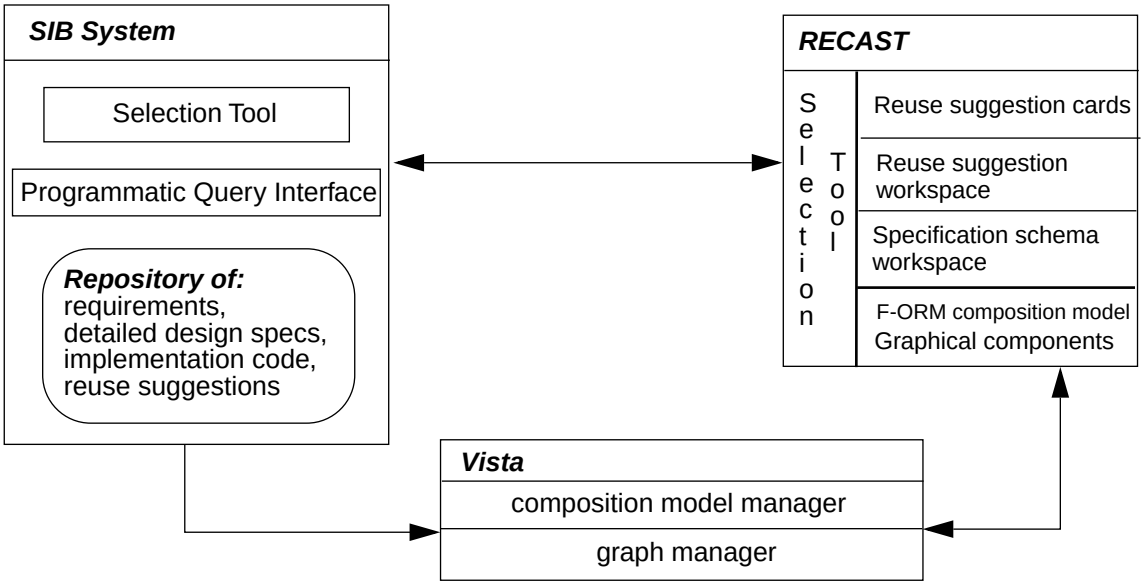

Figure 2 ITHACA-2 application development environment.

3.1.1 SIB basics

The SIB provides for persistent storing, sharing, accessing, managing and controlling data. The SIB offers support for conceptual modelling, flexible retrieval and browsing of multimedia data, and is optimized for efficiency with network structures consisting of a large variety of classes rather than relatively few classes with large populations per class.

The SIB is structured as an attributed directed graph. The nodes and links of the graph represent descriptions of software objects and semantic relations, respectively. The SIB is defined in terms of the Telos [21] knowledge representation language which supports creating an infinite instantiation hierarchy and treats attributes as objects in their own right. There are three kinds of descriptions:

- requirements descriptions (RD)
- design descriptions (DD)
- implementation descriptions (ID)

These descriptions provide three corresponding views of a software object:

- an application view, according to a requirements model;
- a system view, according to a design model;
- an implementation view, according to an implementation model.

Descriptions can be simple or composite. The term "description" reflects the fact that these entities only describe software objects; the objects themselves reside outside the SIB. Descriptions are related to each other through a number of semantic relations such as isA, instanceOf, attribute, hasPart, correspondsTo, similarTo, specialCaseOf and association.

3.1.2 SIB prototype

The SIB system [5] consists of the following major parts

- The *SIB Interactive User Interface* implemented using the OSF/Motif toolkit.
- A *Graphical Browser* presenting parts of the SIB network graphically and allowing the user to browse through the SIB by sending messages to the SIB Interactive User Interface in response to user actions.
- *Display Forms* are used to provide information about nodes. The forms are designed to support multimedia information (text, graphics, images, animation, etc.).
- A special *Data Entry Form* allows data to be entered into the SIB.
- A Q*uery Interface* handles queries about SIB objects.

The selection of software descriptions from the SIB is accomplished through the SIB system Selection Tool by an iterative process comprising alternate stages of explicit queries and browsing. The selection tool provides facilities for filtering the display of links in the graphical browser according to their types, a history mechanism that keeps a record of users' moves through the SIB network, a list of the contents of the SIB organized by application frame and forms that display information about particular objects.

3.1.3 Relationship to other tools

To access the SIB from another application, a programmatic query interface is available [8]. The interface can be used in a client-server fashion, where communication between the application and the SIB is by named pipes, or by direct access. For direct access, the code for accessing the SIB must be linked into the application. Most of the functions used to retrieve information result in a set of answers. To minimize the traffic on the pipes, temporary sets are stored on the server side and later accessed for relevant information. Queries are either simple or recursive. Simple queries refer only to objects that are connected by immediate links to the current node or to objects in an answer set of a previous query. Recursive queries traverse existing links according to some restrictions and calculate the answer set from the traversal. Both Vista and RECAST use the programmatic query interface in a client-server fashion.

3.2 Vista

Vista allows application developers to build applications as sets of cooperating components.

3.2.1 Vista basics

Visual composition[*] [23] is the interactive construction of applications from pre-packaged, plug-compatible software components by direct manipulation and graphical editing. This approach allows both the internal behaviour of applications, as well as their user interfaces, to be directly edited and manipulated. For visual composition to work, *component sets* and *composition models* must be available. A component set is the collection of software components from which applications may be constructed. A composition model defines the allowable composition interfaces for a given component set and the possible connections between components.

In order to evaluate visual composition as an approach to application construction, we developed *Vista* [16][17], a prototype visual composition tool, and we applied the tool to a number of selected application domains. Vista supports visual composition in two ways: (1) by imposing a reasonably flexible structure for component definition, and (2) by defining a framework, referred to as the Component-Port-Link (CPL) framework, for component composition.

A component is made up of three parts: a *presentation*, a *behaviour* and a *composition interface*. The presentation of a component is how it looks on the screen. The presentation can be the actual user interface the component will have in the final application or just a visualization for tool purposes. The behaviour of a component consists of whatever it has been designed to do. The most critical part, however, is the composition interface, which allows the behaviour of the component to be bound to a given context. The composition interface of a component consists of a set of *ports*, each of which has a name, a type and a polarity. It is also possible to define a group of components as a *composite component*, thereby promoting hierarchical composition. To define a composite component, one must provide (1) a composition interface (by specifying which ports of the constituent components are to become ports of the composite component), and (2) a visual presentation (which can be composed of existing presentation components). The behaviour of a composite component is defined by the behaviour of the components it contains.

A *composition model* declares what port and link types are valid in a particular application domain and which ports are *compatible*, i.e., which may be linked together. Composition models can be related to each other by inheritance allowing the reuse of models in the declaration of other models. By decoupling composition models from component sets, a variety of different software composition paradigms can be supported. Components share information with each other once they are linked together.

3.2.2 Vista prototype

- User Interface

  The user interface is composed of three windows: message window, control window and tool window. The message window displays system messages which supply information or error messages to the user. The control window contains a scroll list of the components and composite components that can be used or edited. There is also an area to select link presentation (color, style, width) on a per link type basis. The tool window contains the *command menu* and *drawing area*. The drawing area is where the user instantiates components. The command menu is used to execute an operation on a component or a link. The user interface of the tool, components and component ports are sensitive to user input.

- Components and Composite Components

  When the tool is launched with a composition model, only those components which are designed for that particular composition model are shown in the control window. When the user does not specify a composition model, Vista uses a default composition model (data flow) and component set (user interface components).

  Components can be added to the tool either by programming or by composition. Components added to the tool by programming are called *primitive* components. Components added to the tool by composition are composite components as described above. Adding new primitive components to Vista requires the definition of a C++ class for the behavior of the component and, if an appropriate C++/Motif widget is not available, the definition of a C++/Motif widget for the presentation of the component.

- Composition Models

  A composition model allows the user to define compatibility rules between ports and specify what C++ classes implement the behaviours (i.e. checking, modifying or intercepting data) of ports and links. A composition model is dynamically loaded into the tool at run-time. The tool parses a composition model file and creates a composition model manager which manages the compatibility rules. To link two components, the user can select a link type otherwise a default link type will be used. The connection is made only if the particular rules for the link are satisfied. This is controlled by the composition model manager and the object implementing the behaviour of the link.

3.2.3 Relationship to other tools

For the implementation of Vista, useful parts of Labyrinth have been extracted and new functionality has been added. Vista compositions can be seen as augmented Labyrinth networks.

Presently, the persistent data created and used by Vista consists of (1) composite components, and (2) composition models. Vista generates Telos descriptions of both primitive components and composite components according to the Vista model in the SIB. As defined now, the Vista model in the SIB reflects the underlying implementation of Vista i.e. compositions (augmented Labyrinth networks). The compositions are mapped directly onto SIB semantic networks. This model is not at a high

*.Previously referred to as “visual scripting.”

enough level and should be redefined to better reflect the needs of the application engineer and application developer.

RECAST has used Vista for its implementation.Vista has been used to compose applications in both the workflow and multimedia domains [19].

3.3 RECAST

RECAST [3] provides "guided tours" of the SIB, attempting to construct a specific application on the basis of: 1) the user requirements; 2) generic information pertaining to the selected application domain. Software components selected with RECAST have an associated set of design guidelines describing how components can be reused and what design classes should be selected and composed in the subsequent phases of the development.

3.3.1 RECAST basics

RECAST is based on the following assumptions:

1. Requirements are specified according to an object-oriented specification model, called F-ORM.
   The Functionality in the Objects with Roles Model (F-ORM) is used for requirements representation. It is based on the object-oriented paradigm enlarged with the concept of role [25] to represent the different behaviors that an object can have during its life-cycle. The role dynamics is represented via state/transition diagrams.
2. The specification process is accomplished according to the ITHACA Object-Oriented Methodology (IOOM).
   According to IOOM [9] [10], requirements are organized at various detail levels and the specification task involves both the definition of requirements at a given detail level (*schema definition*) and the transformation of the schema (*schema transformation* via refinement or abstraction). A set of primitives for object-oriented analysis and design under reuse are defined.
3. A composition approach is at the basis of the specification process.
   Requirements specification is executed by composing reusable requirements. The identification of reusable components representing requirements is at the basis of the approach; reusable components are selected, personalized and interconnected.
4. The reuse process is guided by suggestions supplied by the tool.
   The component reuse process is supported by suggestions concerning the possible /necessary design actions to adapt the component to a specific application's needs. Various reuse strategies are associated with reusable components and the application developer design actions guide the reuse process. Suggestions are represented as *reuse guidelines* stored in the SIB.
5. Requirements and design specification are interleaved phases.
   The mapping of requirements specifications into detailed design specifications is also guided by suggestions. Various mapping strategies are provided for reusable components; the selection of a strategy supports the identification of a set of the basic detailed design components that correspond to the requirement component.

Generic F-ORM classes and reuse guidelines represent the reuse information used by RECAST and are stored in the SIB and organized in GAFs.

3.3.2 RECAST prototype

The RECAST prototype [2][4] supports requirements tailoring and composition. A *component set* has been defined to represent F-ORM requirements. Class and role components are used to achieve the requirements representation. Role components are in effect composite components that include a state/transition diagram definition representing the role behaviour.

The *composition model* for F-ORM defines the allowed interconnections among class, role and state/transition diagram components. In particular, IsA and PartOf relations can be established among classes. Role composition is based on the *role message interface*; each role can receive and send a fixed set of messages and can be connected only to roles having a compatible message interface, that is, a message must be provided as outgoing in one role and incoming in the other or vice versa. State/transition diagram composition involves the connection of diagram components (e.g., states and messages) according to the possible flows of execution among role states.

RECAST retrieves reusable components from the SIB either by query or browsing. Browsing uses the SIB system graphical browser with a view of the F-ORM reusable classes stored in the SIB and of the semantic relations between them.

When the application developer selects a generic class for reuse, new objects are created as instances of the meta classes that describe the reuse of the selected class. Information about the design actions executed by the application developer on a class is sent to the meta classes associated with the class and new reuse suggestions are produced. Suggestions guide the application developer in the execution of cluster definition, transformation and mapping (into detailed design specifications), and are represented as suggestion cards.

3.3.3 Relationship to other tools

RECAST enhances the composition support Vista provides by introducing *reuse facilities*. Requirements specification is mainly achieved from reusable classes stored in the SIB which have associated *reuse suggestions* that drive the specification task. Requirements composition is guided by suggestions: classes that are necessary and useful to a class being reused are automatically proposed, together with their interconnections [11].

The specification of requirements at a given level involves the identification of reusable classes from the SIB repository and their composition in the RECAST workspace. New classes can also be created if no reusable classes satisfy the needed functionalities. Schema transformation involves the refinement of roles into classes, of messages into roles, etc. according to the method primitives. Transformation rules are out of the scope of the composition model and are not (yet) supported via Vista facilities but directly by RECAST.

Vista is used, at the implementation level, via a programmatic interface, in order to graphically represent and compose

F-ORM and reuse guideline components. Vista, with the ADL component set and composition model [24], is accessed in order to establish a connection with the detailed design specification phases. RECAST accesses the SIB contents via the SIB programmatic query interface and the SIB system graphical browser.

4 Experience, Observations and Recommendations

In general, the experience of the partners in the ITHACA-2 phase has been positive. All industrial partners have committed to object-oriented programming and the ITHACA scenario. Siemens Nixdorf and Bull are marketing products directly resulting from ITHACA. All academic partners have found the ITHACA project fruitful and rewarding, and a catalyst for further research. In particular, we at the CUI have put much of the experience in developing Vista toward developing a multimedia programming environment [18] that includes a composition tool addressing the domain of multimedia applications. FORTH, the developers of the SIB, have made major steps towards a more robust software repository technology.

Although there is very positive feedback concerning ITHACA-2, the goal of a complete and integrated application development environment was not met. Our experience and research shows that efforts towards integrating the tools into such an application development environment should be continued. Two general observations are that (1) a generic graph management and presentation layer is useful since all tools use a graph-based data structure and (2) all tools should communicate through a common network communications facility. More specifically, the integration of the tools can be pursued on two fronts: (1) augmenting the SIB system and (2) developing a general composition tool. This integration would supply a more consolidated environment on which to base further research.

4.1 Augmenting the SIB System

4.1.1 SIB contents

The SIB is populated with descriptions of software artifacts. Descriptions are of three types: requirements, design or implementation. Each type can define a number of different models. A model is a collection of classes (a schema) defining the structure of software artifact representation. There is a meta model that defines the basic components to be used in definitions of all other models for the SIB. Currently there are the following models:

- requirements: RECAST models for F-ORM generic classes and reuse guidelines
- design: Vista (composition networks)
- implementation: PL (template for other implementation models), CooL.

Work could be continued in the following areas to enable better selection of useful software artifacts:

- ways of relating all the models;
- a model for composition models is needed;
- composition information can be added to existing models;
- new models for composition are needed that include information about types of ports, allowable interconnections, the composition interfaces of components, and the part-of relationship between composite components and their components;
- Models describing the structure of components and how components can be composed could be merged.

4.1.2 SIB selection tool

The principal tool for accessing the SIB is the SIB supplied selection tool. RECAST also supplies a selection tool. The selection tool of the SIB system can be augmented with some of the functionality of the RECAST selection tool so as not to duplicate functionality in the system. For example, the RECAST selection tool supports retrieval of reusable components via fuzzy queries which are currently tailored to F-ORM class queries. This functionality could easily be made notation-independent and then used for any query on the SIB.

4.2 General Composition Tool

The activities of requirements collection and specification and composition can be supported by a common composition tool. It was observed that requirements collection and specification is a composition activity with its own particular component sets and composition models. Therefore, the composition tool supports general composition behaviour and multiple composition models and workspaces for specialized activities. As mentioned earlier, Vista does not support all the requirements of RECAST, such as transformation rules in the composition model. To move Vista closer to a general composition tool, two issues could be addressed: (1) enhancements to composition models and (2) facilities for adding components. Point 1 involves expanding the composition model to handle more information concerning components, ports and links including application constraints. Point 2 involves supporting the addition of new components with tools that make it easier to declare a component's composition interface.

5 What's Next: ITHACA-IT

The goal of ITHACA-IT [13] is to industrialize the object-oriented application development and programming technology, as well as the object-oriented workflow application, provided by ITHACA-2 and to introduce this technology to a wide marketplace. ITHACA-IT technology will target large commercial application projects with the aim of reducing the costs of those applications by enhancing productivity. The ITHACA-IT technology will include the following components:

- object-oriented system kernel
- application development environment
- object-oriented software information base.

ITHACA-IT is smaller than ITHACA-2, from the point of view of both manpower and budget. A close cooperation is planned between ITHACA-IT industrial partners and standardization organisations, such as the OMG, on all matters relating to the propagation of object-orientation at an industrial level.

The ITHACA-IT system will support large scale information system development and will, like ITHACA-2, employ object technology at every level. The system will be designed to:

- ensure maximum reusability at various levels of application development,
- reduce the cost of industrial software development,
- guarantee a high application quality.

Unlike other developments on the market, ITHACA-IT is not aimed at creating niche applications to be used in dedicated environments. Rather, the focus is on achieving a high level of openness with the goal of providing comprehensive support for object-oriented software development methods. Geneva's participation in ITHACA-IT will focus on visualization of (1) the software development life cycle, (2) software information and (3) office procedures as needed by the object-oriented programming environment and the workflow application. We will also be involved in looking at the interoperation of applications focusing on the interoperability of CooL applications with applications developed in other programming languages or those based on different CooL class-hierarchies.

6 Conclusions

Significant progress was made during ITHACA-2 on the object-oriented programming environment and the workbenches allowing the major industrial partners to market some of the results. Siemens Nixdorf is marketing a non-persistent version of the programming environment (support for persistence is being developed in ITHACA-IT). As part of their experience with the Office Systems workbench, Bull is committed to a fully object-oriented workflow application based on ITHACA technology and has completed a short-term workflow product which is currently being marketed. TAO, as a result of its work in the Public Administration workbench, is investing in a Coordination Procedure system which it intends to use in future products. Datamont will use the results from the Financial workbench for internal clients.

The application development environment is still at the level of a research prototype. The SIB is quite stable and is being incorporated into the programming environment. As a result of ITHACA-IT, Siemens Nixdorf plans to industrialize and market the SIB. Both RECAST and Vista are prototypes. Within the scope of ITHACA-IT, no further development is planned for RECAST. Vista will be used for investigations into visual composition of CooL components. Both Geneva and the Politecnico di Milano, the developers of RECAST, have further research planned involving ITHACA-2 results. The experience from ITHACA-2 has indicated that the original goals of the project are still very relevant and should be further exploited.

References

[1] M. Ader, O. Nierstrasz, S. McMahon, G. Müller and A-K. Prö-frock, "The ITHACA Technology: A Landscape for Object-Oriented Application Development," Proceedings, Esprit 1990 Conference, Kluwer Academic Publishers, Dordrecht, NL, 1990, pp. 31-51.

[2] R. Bellinzona, M. G. Fugini, G. Bracchi, "Scripting Reusable Requirements Through RECAST," ITHACA.POLI-MI.92.E.2.9.#1, Politecnico di Milano, June, 1992.

[3] R. Bellinzona, M. G. Fugini, B. Pernici, "An environment for requirement reuse", ITHACA.POLIMI-UDUNIV.92.E.2.9-E.8.4.#1, November 1992.

[4] R. Bellinzona, M. G. Fugini, "RECAST Manual", ITH-ACA.POLIMI.92.E.2.9.#3, December 1992.

[5] P. Constantopoulos, E. Pataki, "A Browser for Software Reuse," 4th Int'l conf. CAiSE '92, Manchester, U.K., May 1992.

[6] P. Constantopoulos, M. Jarke, J. Mylopoulos and Y. Vassiliou, "The Software Information Base: A Server for Reuse," ITH-ACA.FORTH.92.E2.#1, Foundation of Research and Technology — Hellas, Iraklion, Crete, January 12, 1992.

[7] P. Constantopoulos, M. Dörr, Y. Vassiliou, "Repositories for Software Reuse: The Software Information Base", Foundation of Research and Technology — Hellas, Iraklion, Crete, 1992.

[8] C. Dadouris, M. Dörr, I. Gardiki, M. Marakakis, E. Pataki, E. Petra, G. Spanoudakis and G. Yeorgiannakis, "Implementation of the SIB System," ITHACA.FORTH.92.E2.#3, Foundation of Research and Technology — Hellas, Iraklion, Crete.

[9] V. De Antonellis, B. Pernici, P. Samarati, "F-ORM METHOD: A F-ORM Methodology for Reusing Specifications," IFIP WG 8.4 Working Conf. on Object-Oriented Aspects in Information Systems, Quebec, Oct. 1991.

[10] V. De Antonellis, B. Pernici, "ITHACA Object-Oriented Methodology Manual - Introduction and Application Developer Manual (IOOM/AD)," ITHACA.POLIMI.UDUNIV.91.E.8.1, October, 1991.

[11] M. G. Fugini, M. Guggino, B. Pernici, "Reusing Requirements through a Modelling and Composition Support Tool", Third International Conference CAiSE'91, Trondheim, Norway, May 1991.

[12] M. G. Fugini, O. M. Nierstrasz and B. Pernici, "Application Development Through Reuse: The ITHACA Tools Environment", ACM SIG-OIS Bulletin Vol. 13, No. 2, August 1992, pp. 38-47.

[13] ITHACA-IT Technical Annex.

[14] M. Katevenis, T. Sorilos and P. Kalogerakis, "Labyrinth Programmer's Manual (version3.0)", ITH-ACA.FORTH.92.E3.3.#1, Foundation of Research and Technology – Hellas, Heraklio, Crete, January, 1992.

[15] M. Katevenis, T. Sorilos, C. Georgis and P. Kalogerakis, "Labyrinth User's Manual (version 2.10)," ITHACA report FORTH.90.E3.3.#7, Foundation of Research and Technology – Hellas, Heraklio, Crete, Dec 31, 1990.

[16] V. de Mey, B. Junod, S. Renfer, "Vista Implementation," ITH-ACA.CUI.92.Vista.#1, Centre Universitaire d'Informatique, University of Geneva, Dec. 1992.

[17] V. de Mey, B. Junod, S. Renfer, "Vista User's Guide," ITH-ACA.CUI.92.Vista.#2, Centre Universitaire d'Informatique, University of Geneva, Dec. 1992.

[18] V. de Mey and S. Gibbs, "A Multimedia Component Kit: Experiences with Visual Composition of Applications," To appear, ACM Multimedia Conf., 1993.

[19] V. de Mey, "Experimenting with Component-Oriented Software Development," in Object Frameworks, ed. Dennis Tsichritzis, Centre Universitaire d'Informatique, University of Geneva, July 1992, pp. 221-241.

[20] V. de Mey, O. Nierstrasz, S. Renfer, R. Bellinzona, M. G. Fugini, P. Constantopoulos, M. Dörr, M. Theodoridou, "RECAST/Vista/SIB Integration," ITHACA.CUI-POLIMI-FORTH.92.Vista.Recast.SIB.#1, Centre Universitaire d'Informatique, University of Geneva, Dec. 1992.

[21] J. Mylopoulos et al., “Telos: Representing Knowledge about Information Systems” ACM Trans. on Information Systems, October 1990.

[22] O. Nierstrasz, S. Gibbs, D. Tsichritzis, “Component-Oriented Software Development”, Communication of the ACM, Vol. 35, No. 9, September 1992.

[23] O. Nierstrasz, D. Tsichritzis, V. de Mey and M. Stadelmann, “Objects + Scripts = Applications,” Proceedings, Esprit 1991 Conference, Kluwer Academic Publishers, Dordrecht, NL, 1991, 534-552.

[24] O. Nierstrasz, “The ADL Scripting Model and Component Set”, ITHACA.CUI.91.Vista.#6.1, Dec. 1991.

[25] B. Pernici, “Objects with Roles”, Conf. on Office Information Systems, April 25-27 1990, Cambridge, Massachusetts.

[26] E. Petra and C. Vezerides, “SIB Contents Manual,” ITHACA.FORTH.91.E2.#4, ICS-FORTH, Jan., 1991.