CASE INTEGRATION APPROACHES AND A PROPOSAL FOR AN OBJECT ORIENTED CASE ENVIRONMENT

F. Losavio, A. Matteo
Centro de Ingeniería de Software y Sistemas, ISYS
Facultad de Ciencias, Universidad Central de Venezuela
Aptdo. 47567, Los Chaguaramos 1041-A
Caracas, Venezuela
e-mail: flosavio@conicit.ve, amatteo@conicit.ve

M. Perez
Departamento de Procesos y Sistemas
Universidad Simón Bolívar
Aptdo 89000
Caracas, Venezuela
e-mail: movalles@usb.ve

ABSTRACT
Software systems are becoming increasingly complex, thereby requiring a better support for their development process. This process is long, highly dynamic in its requirements, and expects quality features such as extensibility, reusability, efficiency, etc. The need of counting on integrated environments to support this process. This paper presents a review of the historic development of the CASE tools architectures and the characterization of the integration of this type of environments. Finally, the object-oriented CASE environments are qualified on the bases of a conceptual model.

Keywords
CASE environment, integration, CASE tools, object-oriented CASE.

1. INTRODUCTION
Throughout history, productivity in manufacturing processes significantly increases when human capabilities are backed by powerful tools. A poorly designed, implemented and maintained software is a major problem for many companies using computer systems. These aspects have focused attention on the processes by means of which software is developed and maintained and on the computer-based technology that supports it. The Computer Assisted Software Engineering (CASE) approach suggests that automation should fully or partially support the software development process. This definition means giving support to the management, administrative or technical aspects of any part of a software project.

Initially, the CASE approach was dedicated to the construction of support tools for the development of programs such as translators, compilers, assemblers, linkers, etc. and we called programming environments. A CASE tool is a computer-based product aimed at supporting one or more software engineering activities within a software development process. In the measure that computers have become increasingly more powerful and their use more widespread, the notion of a broader development process has appeared. The software development process can be seen as:

- A large-scale activity involving significant efforts to establish the requirements, design an appropriate solution, implement that solution, test the solution correctly and document the functionalities of the definitive system.
- A long term process. The implications of this is that the software structure produced must be capable of allowing easy addition of new functionalities. In other words, it has to be extensible and a detailed record of its design, testing and implementation must be made in order to help the analysts carry out its maintenance. Likewise, many versions of the artifact produced during project development must be maintained in order to facilitate software construction by a team (groupware).
- A group activity entailing the interaction among a large number of persons during each stage. These teams must cooperate in a controlled manner and must have consistent visions of the status of the project.

A typical CASE environment is a set of CASE tools and other components assembled with an integration approach backing all or most of the interactions between the environment components and the users, and the environment itself. Other names for a CASE environment are Software Engineering Environment (SEE), Integrated Project Support Environment (IPSE) and Software Development Environment (SDE). The definition of an SEE implies that its structure shall support the integration or the cooperative use of CASE tools for the tasks execution. This cooperation can be accomplished through several dimensions: data, process, control and presentation, are among the more accepted ones [Cuthill, 1994].

The crucial part of the definition of a CASE environment is that the interactions among the components must be backed by the environment. What differentiates a CASE environment from a conglomerate of CASE tools is that there is an element provided by the environment, which
facilitates the interaction between these tools. This element could be a physical mechanism such as a shared database, a message distribution system, a conceptual notion such as a shared philosophy on tool architectures, a common semantics on the objects that are manipulated or a combination of all these. The range of all the possible forms of providing the glue that links the CASE tools invariably leads to a spectrum of implementation approaches for a CASE environment [Brown et al, 1994], which shall be discussed in the following section.

Current experiences, however, suggest that the technology the CASE tools and environments is still insufficient to provide all the benefits offered, for the following reasons [Brown et al, 1994]:

- Inability to an easy combination of tools to cover the complete software development life cycle;
- Overlapping of tools when they cover similar services;
- A lack of well-defined procedures for data exchange from one tool to another in the synchronization of communication, so that a tool can be invoked from another one;
- Poor managerial visibility of the progress of the tasks in a project and the intermediate states of the products produced by the tools;
- Poor facilities to link and adapt the tools to the company's needs;
- System management and maintenance problems as well as problems in the evolution of the tools due to their size and complexity.

2. THE EVOLUTION OF THE CASE ENVIRONMENT ARCHITECTURES

Two independent state-of-the-art approaches have been taken in practice for describing CASE environment architectures. The first is based on an architecture able to support the complete software development process based on a holistic set of shared mechanisms (top-down approach) facilitating the integration and coordination of the tools. This approach is called integrated project support environment (IPSE). The second approach takes into account the coordination of independent CASE tools, called frequently Case Tool Market Technology (CTMT), a bottom-up approach. The IPSE approach has focused more on the development of a generic framework technological infrastructure to give support to large-scale development software. Figure 1.

Three environment categories are detected within the IPSE approach [Sommerville, 1992]: Programming environments: these are environments which are principally intended to support the processes of programming, testing and debugging; CASE workbenches, which are oriented towards software specification and design, and Software Engineering Environments (SDE). This last class is intended to support the production of large, long-lifetime software systems whose maintenance cost typically exceed development costs and which are produced by a team and not by individual programmers.

Figure 1. The Evolution of CASE Environments

They provide support for development and maintenance activities. Two types of software engineering environments are identified within the SDE: language-oriented are environments for systems development in a single programming language, for instance Cedar [Sommerville, 1992] environment and Ada environment [Sommerville, 1992] and the actual IPSE themselves, for instance ISTAR, ECLIPSE and Arcadia environments [Sommerville, 1992].

The second approach CTMT has been followed by the suppliers [Brown et al, 1994] who have gradually understood that a simple tool does not support all the engineering and managerial tasks involved in the development of software and so multiple tools were joined and produced until a coalition of CASE tools, covering a wide range of functionalities inherent to a CASE environment, was formed. On the basis of the rapidly growing CASE market development, there has currently been a noticeable turn from the IPSE approach towards the coalition CASE approach.

The features of these approaches, also according to Brown [Brown et al, 1994] are:

- IPSE (TOP-DOWN): Characterized by a centralized Object Management System (OMS) with its services and possible managerial support centered in the software development process. These include strong support elements for the software development process. The IPSE approach and its support in a framework of common capabilities is quite a generalized approach. The main disadvantages of this architecture are the following:
1. Large-scale holistic solutions: Large investments were made for the global large-scale development of this kind of environments and it is precisely this globality that affects the construction of the environment, its performance and its use.

2. Conventional filing systems provide a good support for data distribution and the management of versions but not for the handling of semantic and information models.

3. IPSE frameworks need to have sufficient genericity to meet the needs of different users. Inasmuch as the process answers not only to several organizations but also to different project types, the backing that IPSE can provide is rather superficial.

For these reasons, IPSE is not very widely accepted within the possible customer organizations. However it is known, based on experiences such as Gohner's [Gohner, 1991], who introduced two architecture alternatives for IPSE-type environments. The first coincides with Brown's proposal - the traditional approach- and consists in the development of integrated tools that carry a homogeneous IPSE and a second a posteriori integration proposal of heterogeneous tools to produce an heterogeneous IPSE. The degree of integration is higher in the homogeneous IPSE inasmuch as the tools share the data and they are more linked. The heterogeneous IPSE shall have a dominant tool responsible for the project database. Integration with other front-end tools, i.e. those which support the initial activities of the software development cycle and back-end tools which support the final activities thereof, are achieved through temporary files. Gohner coincides with Brown in stating that the main difficulty of the homogeneous IPSE is the effort needed to develop a priori integrated tools. In this case the 5-5-5 rule is fulfilled. This means that it can takes 5 years for the research and development of the prototype, 5 years for the development of the product and 5 years for its marketing. Therefore one of the ways to enjoy the benefits of an IPSE nowadays is through an architecture of heterogeneous tools. Gohner's global integration idea [Gohner, 1991] is to store the complete information produced by the front-end tools on the Engineering and Project Management-Oriented Specification System (EPOS) project database and to give part of this information to the back-end tools. EPOS represents the most complete IPSE, with integrated front-end and back-end tools, available for use in real industrial projects.

• Coalition CASE (BOTTOM-UP): These environments are in a large extent the reverse of IPSE. Each CASE tools participating in a CASE coalition provide their own database. The services are also separately accessible. The CASE coalition defines its own integration and owner services model and counts on remote execution and other control-oriented integration forms. The CASE tools participating in the coalition support finer-grained processes in a localized fashion. The development of the CASE coalition environment is a bottom-up process. These include tools that were developed separately and thereby reflect their own philosophy with respect to the software development process. The key problems in this type of approach are the absence of generality and poor adaptability of the tools, reflecting an irregular ad hoc support of the software development process. Examples of these mixes: Interactive Development Environment's Software Through Pictures, CodeCenter and Frame Technology's FrameMaker [Brown et al, 1994]. A modality of this approach is the proposal by Clemm and Osterweil who achieve the integration of a set of tools through the Odin software system [Clemm and Osterweil, 1990]. They used it in principle to integrate a family of tools for development, testing and maintenance in FORTRAN. It was later used to integrate tools for the development in C.

3. DIFFERENT APPROACHES FOR THE INTEGRATION OF CASE TOOLS

The two architectures described lead us to examine the definition of integration. Integration is a key element to provide an effective support to software development process, which is not clearly understood. The success of the integration of a set of CASE tools is a complex problem that also depends on many factors and the more important aspect is to delve deeper into the understanding of the meaning and role of integration in a CASE environment. So long as there are different ways to solve this problem, a necessary step is to consider the properties of integration in order to clarify the meaning of integration. Another necessary step is to differentiate the semantic elements of integration mechanisms, in other words, to separate the how the tools may be integrated (in terms of underlying mechanisms) from the fact of defining what is integrated (in terms of the required semantics).

Brown [Brown et al, 1994] identifies three integration strategies:

i.- As a set of attributes characterizing a CASE environment. The proposals of Wasserman [Wasserman, 1990], Mathias [Mathias, 1992] and Mi [Mi and Scacchi, 1992] are included in this strategy.

ii.- As the goals for the relations among the components in the environment, a strategy proposed by Thomas and Nejmeh [Thomas and Nejmeh, 1992]
As a central repository through which the components of the CASE environment share data. This is the standard Portable Common Tool Environment (PCTE) strategy [ECMA TR/55, 1993] and the proposal of Kusuki [Kusuki et al, 1992].

In what follows, these strategies will be considered in details.

**Strategy i.** Five dimensions are considered:

1. **Platform:** related to the services of the structure.
2. **Presentation:** related to interactions with the user.
3. **Data:** related to the use of the data by the tools.
4. **Control:** related to communication and interoperations.
5. **Processes:** related to the role of the tools in the process.

According to Wasserman, there is consensus in the software engineering community as to the four integration relations: presentation, data, control and process. The platform is not developed by Wasserman. Figure 2 shows three of these dimensions.

- **Processes:** To ensure that the tools interact effectively in the backup of a predetermined process.

This strategy has proven its usefulness inasmuch as it provides a basic understanding of the integration concepts. The orthogonal integration dimensions allow attributes to be assigned to the environment. If an environment is through the light of these dimensions, then the set of environment attributes can be defined and their integration approach can thus be characterized. This somehow allows the comparison of the environments in terms of their approach. The criticism to this proposal, that of orthogonal dimensions, is whether they are really orthogonal and if they can be studied separately. Another problem is whether to consider that, the farther it is from the axis, better integrated is the environment.

Mathias presents an experimental information system environment called Development Assistance for Integrated Database Applications (DAIDA). Based on his experience, the author detected that the formulation of process-oriented conceptual models by means of a representation of the database knowledge and techniques could integrate development stages and tasks. DAIDA is more than a technique based on knowledge to integrate CASE tools. It is directed towards three important integration dimensions within a process-oriented conceptual model, such as: how to handle the dependencies within the development stages, how to handle the relations between the systems and their technical and social environments and how to integrate the development tasks from two points of view: small-scale development (the content of the actions and their results) and large-scale development (management of objects and processes). One of DAIDA's goals is to make the process explicit and to back it over long periods.
Mi and Scacchi propose a high integration level, the integration of the process which explicitly represents the development activities in a process model to guide and coordinate the development and to integrate tools and objects. A CASE environment based on the integration of processes is known as a process-driven CASE environment. Its implementation strategy is to carry out process integration using existing CASE tools. These represent the task-execution chain as a software development process model based on a Software Process Model (SPM) representation, thus achieving an integrated environment. Process integration aims at making the conceptual task-execution chain explicit, reusable and flexible. In the opinion of these authors, the integration of processes provides mechanisms to guide the software process and to manage work spaces and the invocation of tools and objects. The integration process also enables the managers to monitor and control the progress of the development. As can be observed in Figure 3, the key mechanisms for the integration per process are the SPM, the process driver and the interfaces both for the developer and the manager. These were identified as key components to integrate their process-oriented CASE environment.

![Process Integration Diagram](image)

**Figure 3. Architecture of a process driven CASE environment**

**Strategy ii** This strategy does not treat integration as a property of a component but as a property of the relations between two or more components in a CASE environment: once again it considers the presentation, data, control and process dimensions. A set of attributes is defined for each dimension responding to a set of questions that qualify integration.

- **Presentation:** Two elements are herein identified:
  - Appearance and behavior: How easy is it to interact with a tool after having learned to interact with another?
  - Interaction Paradigm: How easy is it to interact after having learned the interaction paradigm of another? How similar are its metaphors and mental models?

- **Data:** Five properties are identified here:
  - Interoperability: How much work must be carried out for the data manipulated by a tool to be manipulated by another? What must we do for the data to be seen as a consistent whole?
  - Non-redundancy: How much of the data handled by a tool is duplicated or can be derived from the data handled by the other?
  - Consistency: How well do the two tools cooperate to maintain the semantic restrictions of the data that they handle?
  - Exchange: How much work must be carried out for the data generated by a tool to be used by the other?
  - Synchronization: Two tools are integrated with respect to the integration by synchronization if all the changes on the non-persistent data made by one tool are communicated to the other.

**Figure 4. Entity-relationship diagram depicting a single tool, four relationships and elaborated properties for each relationship**

- **Control:** Two elements are identified for this case:
  - Provision: How vast is the use of a service offered by one tool to another?
  - Use: To what degree are the services offered by a tool used by another in the environment?

- **Process:** Three elements are present:
  - Step: How well do two relevant tools combine to improve the performance of a step?
  - Event: How well do relevant tools in the environment agree in the events they need to support a particular process?
  - Restriction: How well do relevant tools cooperate to enforce a restriction?
Strategy iii. The last strategy identified by Brown, is widely known in the CASE community. There is almost the conviction that the repository is the center of any CASE environment. The concept of repository is used in a broad variety of concepts. For [Brown et al, 1994] a repository has the following four characteristics: it is a data storage mechanism, it is an interface for persistent data, it is a set of information schemes or models and it contains a set of concepts for manipulating data. In terms thereof, the current state-of-the-art of the repositories is as follows:

- In spite of the advancements in OMS developments, the use of UNIX file systems continue to predominate. The implementation of PCTE environments continue to be scarce or rare.
- The standard SQL and UNIX are established for the interfaces.
- At scheme level, many tests are being carried out to define a generic information model to be used as a semantic base of agreement between the application domains. Up to now, none of these generic schemes has had a complete success.

It can be concluded that the existence of a repository does not guarantee integration; however it does facilitate it. The proposal of Kusuki establishes that the integration mechanism can be achieved with the execution of transformations from model to model in each integration stage. Knowledge bases were applied for this, aimed at managing and controlling this type of integration. This means that a CASE tools integration platform was built in a PCTE environment with a knowledge-based integration mechanism. This mechanism executes a model-to-model mapping using rules and also using an object model for data integration, one for communication and messages for control integration and three models for the integration of the processes.

4. A CONCEPTUAL INTEGRATION MODEL

It seems to be a natural conclusion, from the considerations of section 3, to define more appropriately the aspects involving the design, construction and use of an integrated software development environment. The conceptual model proposed in [Brown et al, 1994] is used, for the purpose of understanding integration in a CASE environment. Three levels are identified in this model: a first central level, consisting of the services available to the end users; a second level, consisting of the mechanisms implementing these services and a third level, the processes level, which deciphers the set of goals and restrictions of the project or organization, providing the context in which the end user services will be executed. Figure 5.

- The mechanism level includes the architectural aspects and the technological components that the CASE environment shall implement. Integration at this level directs the implementation aspects, including the necessary interfaces for the environment infrastructure, the interfaces proposed by the tools and the specific integration mechanisms that shall be used to connect the different environment components. The question: how do the different components of the environment connect? is answered at this level. The mechanisms level is related to the technology available and the techniques that must be applied to connect the different components in the environment.
- The level of services for the end user corresponds to an abstract description of the functionality offered by the CASE environment to the end user. Integration at this level can be considered as the specification of the services that shall be provided to the end user and how these services relate to one another (at an abstract level). It includes the definition of the relations among the services of control of versions and of data management, for instance. The question: what does the CASE environment do? is answered at this level, describing its functionality in abstract terms.

![Figure 5. A Three Levels Conceptual Model of CASE Environment Integration](image)

1 Note that this figure illustrates examples of the elements at each level, and is not intended to be exhaustive.
organizational perspectives by means of different roles. Which activities support the CASE environment? Is the question to be answered at this level. There may be a broad range of supported activities with many degrees of liberty in the use of the environment or the environment may be specific for a small number of well-defined processes.

This means that integration occurs at three levels. However, relations among them are also observed. The relationship between the level of mechanisms and end user services is an implementation relationship, which can be enforced by means of different mechanisms and one simple mechanism can implement more than one service (N:M) taken from the E-R model. The relationship between end user services and processes is a relationship of adaptation.

The processes to be backed act as a set of guides and restrictions to make the combination of the set of end user services. This three-level model provides a context for the discussion of the integration of a CASE environment that had been previously proposed in an ad hoc fashion. The functionality required in a CASE environment can be related to the technology and tools that implement it. Besides, it can be observed how the organizational context and the software development process can restrict or guide the services provided in the environment. We think that this model provides a conceptual framework to understand and discuss several aspects concerning the integration of an object-oriented CASE environment, which will be discussed in the next section.

5. OBJECT-ORIENTED CASE ENVIRONMENT (OOCE)

In the three-levels model proposed by Brown [Brown et al, 1994], the integration process is orthogonal to the integration level of mechanisms and user services. This approach shows that integration is not simply a conglomerate of related tools but rather a combination of various integration mechanisms used over a set of services to achieve specific objectives towards the system development process. The vision focused on improving the process is taken by Brown from the proposal of Humphrey [Humphrey, 1990], who states the need of having a clear definition, a consistent execution and a continuous monitoring of the software development process in the organization. This is reflected through the Capability Maturity Model (CMM) which illustrates the maturity of the organization. The CMM does not focus on the integration of the process as the other authors do, but rather sees the process as an intervention in all the aspects of the software development activity. This is the reason for defining at a first step the features of the software development process that will support OOCE must be defined as a first step. These are:

- The CASE environment must be support the object oriented paradigm
- It must contemplate the reuse of software artifacts.
- It must allow the use of multiple object-oriented analysis and design methods in the different phases or the use of one method in all the phases.

Regarding the level of user service the following is proposed:

- Two types of users: manager and developer.
- Developer shall work concurrently as a team (groupware).
- Developer shall count on version management services.
- Code reading necessitates frequent context switches, which is a challenge to the navigational capabilities of the environment.
- Support the inspection of objects and objects graphs at runtime.
- Design and programming are closely coupled and carried out iteratively.
- Support the visualization of information.
- Support the developer with the information retrieval (browsing). Not only means to answer the corresponding questions, but also to provide appropriate navigation aids.
- Homogeneity of the user interface ans short response times.
- Support in the definition and control of all artefacts belonging to a software system.
- Support in defining and managing the project structure.

For the level of mechanisms, the following technology shall be available:

- Client-server architecture.
- PCTE standard.
- Object-oriented databases.

6. CONCLUSIONS

This paper presents the historical evolution of the architectures proposed for CASE environments to find a way to establish a survey of integration approaches. We described different integration strategies in order to identify architectural requirements that an object oriented CASE environment should fulfill. The purpose of a future research is to study the special features that an object oriented CASE environment should have. In short, our proposal is an instantiation of Brown’s three levels conceptual model, which is a conceptual framework of integration of a CASE environment.

7. REFERENCES