*To appear in Proc. IFIP/IEEE IM'97*

# ACE: An environment for specifying, developing and generating TINA services

*P.G. Bosco, D. Lo Giudice, G. Martini, C. Moiso*
*CSELT*
*Via G. Reiss Romoli 274, 10147 - Torino - Italy ,*
*tel: +39 11 2286806, fax: +39 11 2286862,*
*{Piergiorgio.Bosco,Giovanni.Martini,Corrado.Moiso}@cselt.it*

**Abstract**

The objective of the paper is to present an innovative environment (Application Construction Environment - ACE) for the specification, development and generation of Telecommunication services according to emerging standards such as TINA and CORBA. We will briefly describe the service specifications and development lifecycle requirements for distributed object applications in the telecom domain and how ACE functionalities addressthem.

**Keywords**

TINA, CORBA, CASE, Specifications validation, Behaviour modelling

## 1 INTRODUCTION

The adoption trend of client-server and object oriented paradigms (e.g., ISO/ODP (ISO/ODP, 1993), CORBA(OMG, 1991) and TINA (Handegard, 1995)) is now occurring in various IT domains as well as in telecom ones. Actually, no matter the

domain in consideration, today no formal description techniques are available to express in a complete, compact and natural way the concepts present in the models mentioned above. This is especially true about the design phase of the development lifecycle of distributed applications.
In the IT world the available object-oriented CASE tools for distributed applications are currently more oriented to code generation than to support the whole specification process (included behavioural description).
However there is some attempt at improvement due to the rise of *defacto* standard architectures such as CORBA (OMG, 1991) and MS OLE (distributed OLE) (Brockschmidt,1995) which allow to consider architectural principles in the early stages of specification. In fact some CASE tools are able to generate applications based on these architectures. These typically generate description languages such as IDL for CORBA and OLE types for MS OLE, from the analysis models (GUI, architecture early phases, application logic and Data logic are forced to be kept separate) and finally they also generate the stubs that deal with the distribution mechanisms. This approach offers a good starting point for later design/development phases, though it represents only a small step further, since it still lacks the expression of constructs or "services" such as transactions, concurrency, typical of distributed object environments. Not to say, other requirements traditionally well considered in the telecom industry, such as more formal behavioural description, validation of specifications, performance evaluation etc. are not usually addressed in a integrated way. Notably, only SDL-based CASE environments, approximate this level of requirement coverage.
On the contrary, standard languages familiar to the telecommunication community such as SDL or LOTOS and their latest evolutions do not have sufficiently matched the object-oriented models and behavioural concepts of the recently emerged architectures mentioned above.

In general, the lack of a complete formalism for object-oriented distributed processing models is one of the causes of the absence of CASE tools suited to support the design phases of a system according to such models. This problem can be addressed to some degree by following a pragmatic approach to apply and extend existing notations such as OMT (Rumbaugh,1991), SDL, etc. to distributed computational architectures.

*ACE Objectives*
The motivations to develop ACE were to allow the specification and development of applications according to the distributed processing model of TINA and CORBA, with a particular emphasis on the aspects which nowadays are more difficult to be found in commercial tools, i.e. behavioural modelling. Also "openness" wrt the actual OO

distributed platform architecture was an additional requirement, which CSELT had already started to address in previous versions (Bersia,1995).

There two main strategic reasons to the development of ACE are:

- To use ACE internally to support the specification and prototyping of services according to overall TINA architecture (Service Architecture, Computing and Information Architecture), to CORBA and OMG/COSS1 and OMG/COSS2 for the basic Object Services (OMG,1995).
- To stimulate in the industry the development of "industrial quality" tools "á la ACE": ACE today is a running prototype which acts as a "live" requirement.

Last but not least, ACE aims at providing support for the main lifecycle requirements as expressed in the TINA Service Architecture (Kobayashi,1995). It is a nut-shell for a number of tools (specification editors, performance evaluators, code generators, etc.) from which the analysts/designers/developers all together orchestrate.

## 2 REQUIREMENTS FOR TINA SERVICES DEVELOPMENT TOOLS

It is useful to identify a set of functional requirements that should be supported by a development environment during the main stages of a TINA service lifecycle.

*Browsers/Editors*

The specification and the development of the components of a service should be done by means of proper editors, customised ad hoc to the formalism used for the specification. Advanced graphics, icons, links should be supported. The resulting GUI of the tool should be highly customisable and adaptive to the developer's needs.
The editors should be integrated with the other functionalities of the environment (e.g. syntactic verification could interactively be invoked when new icons are introduced in a diagram).

*Validation & Simulation*

Integrated provisioning of functionalities for the validation and verification of the correctness of a service should be considered as a mandatory aspect to be supported. It could be done according to different criteria e.g. wrt the specification, wrt implementation (no programming errors), or wrt the performance and timing constraints. The confidence about the suitability of an application is gained by challenging its specification against a set of *dynamic properties*:

- Global properties: a set of constraints holds for all the system states and/or for subsets of the states (e.g. a single critical resource can be always allocated to either zero or one resource client).

- Local properties: a set of constraints holds for the trajectories across the state space of the system and/or for the trajectories across a subset of the state space.

In both cases system state exploration (with interactive emulation or with exhaustive emulation) seems to be the only practical approach to partially achieve the goal.

*Performance evaluators*

Performance of a service is one of the most important non-functional requirements. The evaluation of the performances (e.g. response time or throughput of a service) can take place at two main points in the lifecycle of application creation: the specification phase and the acceptance phase.

In both cases the system specification can be taken as the model to study, provided that the service has been implemented as specified. In the former case the objective is to estimate the upper limits in the performances of the service, as functions of the size of the resources; in the latter case the goal is to forecast the performances of the real implementation with particular resource configurations and load situations.

The distributed nature of the computing environment, adds a degree of complexity, since the communication infrastructure plays its own role in the whole performances.

*Code Generation*

Given the adoption of CORBA for what regards TINA DPE (Distributed Processing Environment, computational model and services), the first main requirement of code generation is to generate CORBA IDL.

Further sophisticated code generation is also required in order to meet TINA extensions such as the support for "multiple interfaces".

Also given the requirement of better satisfying "behavioural" aspects of the Telecommunication services based on a distributed platform, generation that integrates COSS basic services such as Transactions, Events and Trading could also be foreseen. Main reference languages are the ones for which OMG has already defined a mapping (C++ ) or is going to (JAVA).

Finally we also believe that targeting to environments such as MS OLE/COM is also a requirement. In developing peripheral telecom services where MS Lans or MS Desktops are part of the scenario, one would want to take advantage of the advanced development environment present on MS platforms (e.g. for Service Creation) and of the high number of "prefabricated" SW components based on MS OCX and OLE Automation objects (Brockshmidt, 1995) already available on such platforms.

Furthermore, whatever the target platform is, it is required to be able to trace the behaviour of the service components (CORBA objects, OLE objects, ...) when they are deployed. Specific selectable mechanisms for their tracing should be embedded in the

generated code. The tracing tool should provide enough information in order to highlight the various interactions among service components themselves and the DPE services.

## 3 ACE FUNCTIONALITIES

This section outlines the main functionalities supported by the ACE environment satisfying the above requirements.

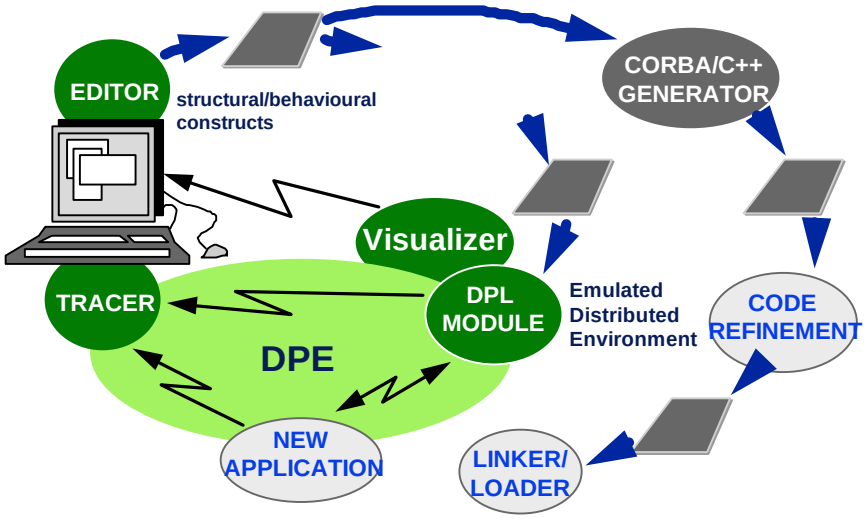


**Figure 1** Overview of the development cycle in ACE.

For the sake of clarity, in Figure 1 an overview of the main phases of the development lifecycle supported in ACE is illustrated. From the set of editors it is possible to produce code both for the simulation environment and for the real platform. The specifications can be executed in a simulated environment, for a first functional validation of the service logics, that can be traced directly on the editors in order to achieve a better comprehension of the execution of the user-level specifications. The code generated for the real platform, after a refinement process of the implementation skeletons, is ready to be installed and deployed on the DPE.

### 3.1 ACE COMPUTATIONAL MODEL CONFORMANCE

As far as conformance objectives towards TINA DPE (Handegard,1995) the following computational model (Bosco,1993) is realised in ACE:

- Creation/Deletion of object and group instances (building-blocks).
- Asynchronous/Synchronous invocations of operations: invocations can be either blocking or non-blocking.
- Accesses to attributes, to read/write their values.
- Subscription to a notification emitted by an object, indicating the notification handler, i.e. the interface to be invoked when these notifications are received.
- Emission of a notification (to all the subscribers).
- Concurrence control in multi-threaded objects: guards, semaphores and close/open nested transactions.

### 3.2 EDITING COMPONENT

The editing component consists of a suite of graphical editors. They support the specification of templates for building-blocks (BBs) and objects, including their behavioural parts, and interface templates.

Each graphical editor is associated to a (graphical) language, consisting of a set of icons and rules on their composition and interconnection through (labelled) links, as depicted in Figure 2. The icons can have attributes that are specified through dialogue-boxes: the attributes are introduced by a sequence of dialogue-boxes, where the structure of a dialogue-box depends on the attributes introduced in the previous ones. Checks for syntax and type constraints are performed during the data input.
The selection of an icon associated to a specification sheet automatically opens the corresponding editor on the associated sheet.

The BB editor (BBE) is the graphical editor to specify BB templates. Special icons represent the object templates included in the BB template, and the contract declarations (i.e. the interfaces accessible from the outside of the BB).

The object editor (OE) is the graphical editor suited to specify object templates, and relies on the behaviour editor (BE) to specify their behaviour.
The specification of an object template is structured into a hierarchy of sheets.

- The declaration part: declaration of parameters, interfaces, internal state, methods, local functions.
- The methods and functions, specified in graphical way.

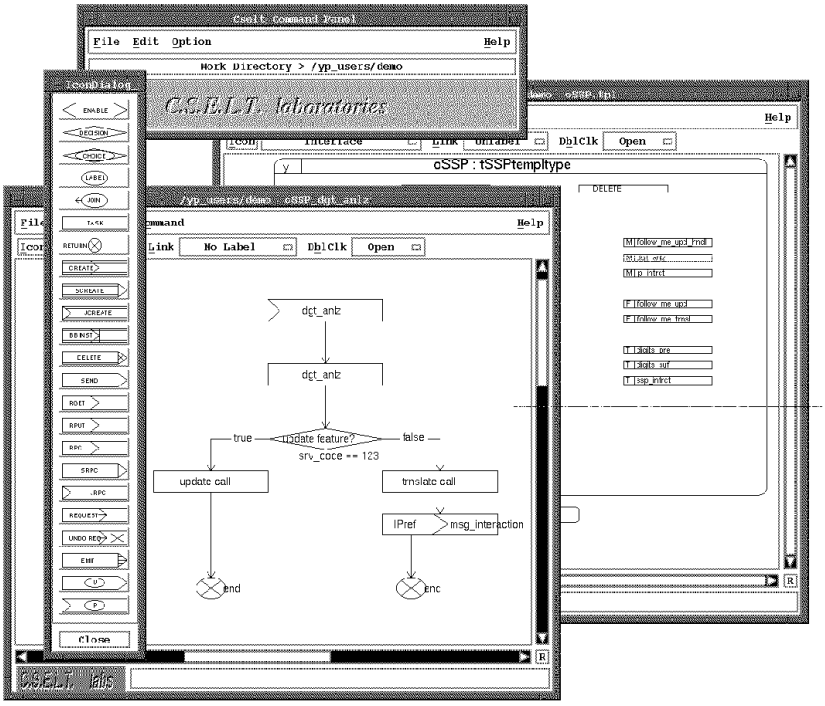


**Figure 2** Behavioural and object template editors.

The OE works on the declaration part, while the BE works on the methods/functions graphically specified (Figure 2). The language associated to OE is a set of icon types that represent the different declaration components of an object template.

Methods are specified by the BE. The language associated to this editor is a flow chart language, where the icons, derived from SDL symbols, are specialised to represent behavioural constructs of the computational model (e.g. invocation of an operation), control constructs (e.g. method start, if-then-else, return, etc) and synchronisation constructs (guards and locks).
The local functions of an object can be specified either in a graphical way through the BE or in a textual way in DPL (see next section 3.4).

The interface editor (IE) is the graphical editor to specify interfaces, and declaring in

CORBA IDL operations, attributes, data types

*Interactive editing features*
All the graphical editors provide interactive/adaptive editing features to drive the production of specifications. For example when a new specification sheet is created the editor initialises it with the mandatory icons, or when an icon is instantiated the editor automatically opens the associated dialogue-box and highlights the mandatory editing-fields.
Syntactic controls and type checking are performed as soon as the values of a dialogue-box are confirmed.

### 3.3 SYNTACTIC AND SEMANTIC CHECKING

ACE performs a global syntax and type check of the specified templates. All the constructs and the declarations are type-checked, the syntax of the attributes of the icons is controlled and the syntax of graphical specifications is checked.
This global check can detect errors even if no errors were signalled during the editing of a template: in fact, for example, after the change of a variable declaration all the expressions where it occurs may be badly typed.
If the global analysis successfully terminates, it generates an intermediate code representation of the template, which is used as input to all the final code generators.

### 3.4 SIMULATION AND VISUALIZATION OF SPECIFICATIONS

At present, functional validation of specification is assumed to be carried on in a user-based interactive way, through the simulated execution. DPL (Distributed Processing Language) is a multiparadigm language that can be used to fast-prototype applications according to the computational model and to execute them in a (sequential) simulated environment. By exploiting Prolog features, DPL is able to model behavioural evolutions of objects dynamics, and, with the provision of an internal scheduler, to support the execution of concurrent entities within the simulation environment, by providing in this way interleaving semantics.

*Mixed mode simulation*
Though the provision of a simulated environments has proved to be sufficient for the validation of user specified service logics, it shows a certain degree of inadequacy when developing service components making intense use of already deployed services. In this case, providing support from within ACE for their modelling does not seem very practical and feasible. This issue has been tackled devising a mixed mode

simulation mechanism, that allows simulated components to interact with applications and services executing on the real DPE (CORBA platform).

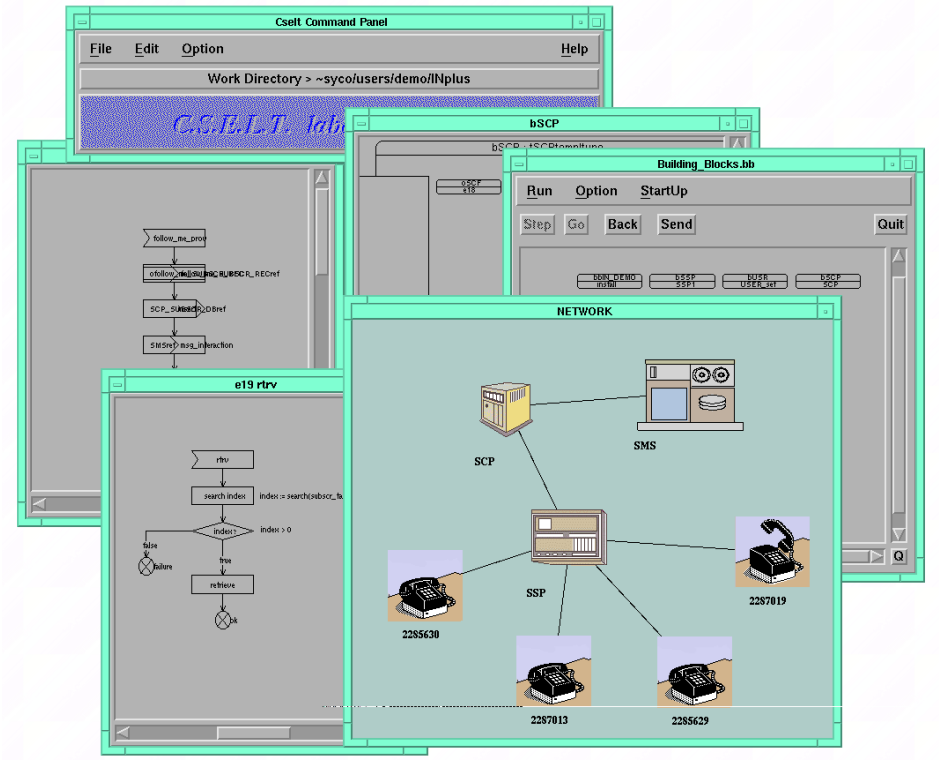


**Figure 3** A simulation session.

*Visualisation and tracing of executions*

The execution of specifications can be visualised in two different ways:

- Model oriented: the executions are traced in terms of concepts of the computational model(e.g. interactions through interfaces).
- Application oriented: the specifications are coupled with a user defined scenario that represents the agents and the artefacts involved in the application (e.g., the telephones, the switches, calls, and connections, as illustrated in Figure 3).

The model oriented visualisation is based on user selectable tracing of the behaviour diagrams and of the object states, directly on the same graphical representation of specifications. The users can interact with the execution by asynchronously invoking

announcements to the objects, defining break-points or inspecting state variables.
The application-oriented visualisation is based on an interface through which the users can introduce, in an asynchronous way, inputs to the application (e.g. by means of the mouse, dialogue-boxes,...) and the application can provide outputs in a graphical way (e.g. through some animations).
The behaviour of this interface depends on the characteristics of the application and must be explicitly programmed, together with some enrichment of the specification to interact with the visualisation interface.

### 3.6 PERFORMANCE EVALUATION OF SPECIFICATIONS

The technical approach chosen for the performance evaluation of an application is the simulation of its specification, augmented with a non-functional characterisation of the significant activities, i.e.:

- Durations of activities: single actions or groups of actions can be given a duration (possibly stochastic).
- Mapping of objects and messages on computing and communication resources: each BB is 'deployed' on a physical processor.

An already available process-based timed executor is being integrated into ACE ( $SPIN^+$ (Chiocchetti,1994)). A number of executions of the model are run and, out of them, estimates are calculated for the desired system parameters. The definition of the performance indexes is embedded within the system specification, as well as validation properties.

### 3.7 CODE GENERATION FOR REAL DISTRIBUTED EXECUTION

ACE provides a code generator to produce CORBA/IDL and C++ for deploying and running the applications on a real CORBA platform. At the moment, the generation is targeted for the Iona Orbix (Iona,1995) realisation, though the targeting to other platforms (Chorus,1994) is an ongoing activity. Among them, of particular relevance is the ReTINA distributed platform (Chorus,1995), a real-time CORBA platform.
The actual mapping tackles and solves the differences between the object model of the TINA computational model (objects having multiple interfaces) and the one supported by the CORBA binding with C++ language. In the process of the code generation this semantic gap is fulfilled by proper object composition and dedicated "glue code" that reduce the mismatch between the two different models.
The code generation covers the production of both the declaration part of templates and the behaviour parts which are specified through the graphical diagrams.
As far as interfaces are concerned, CORBA IDL code is generated automatically

starting from the graphical specification of an interface, as well as the contrary (i.e. it is possible to import externally defined interfaces specified in CORBA IDL). Likewise, starting from the object specifications (structural and behavioural sections), skeletons C++ implementations are generated automatically from the SDL-like behavioural notation.

## 4 REFERENCE APPLICATIONS

ACE is the reference environment within CSELT for developing TINA compliant services. Experience and useful suggestions have been gained by employing the environment (though for minimal parts, and not in all its components) in the definition of the specifications and functionalities for small scale projects within our research group, regarded as proofs of concepts for the TINA Service Architecture.
Among them, it has provided a useful support for the definition and observation of the specifications of the Connection Management module during the TINA World Wide Demo (Spinelli,1996) allowing to trace and analyse the service specifications.
Within the ACTS ReTINA project, it is the core tool providing support functionalities for the definition, validation, and code generation of service components.

## 5 FUTURE EVOLUTIONS

The activity on ACE is still evolving, and during the next months we'll focus on 1) the extension and the upgrade of existing functionalities towards TINA and OMG standards, and 2) the inclusion of functionalities to keep pace with recent technology evolutions in the market place (e.g. support for JAVA and for the Unified Modelling Language (Booch,1996)).

Evolutions regarding 1) comprise the targeting code generation to different CORBA (2.0) platforms (ReTINA DPE) and the support for specification written in TINA ODL(Leydekkers,1995); the inclusion of other languages for expressing application functionalities (e.g. Message Sequence Charts, Use Cases) and DPE basic services abstraction. In this case, the idea is to simplify the use of low-level object services (e.g.: naming, transactions, concurrency, etc.) by providing graphical entities (Icons, symbols, etc.).

During '96 we plan also to expose the tool at major events such as: OMG Object World Trade shows, etc. In order to stimulate and divulge ACE basic philosophy and

approach contacts with major vendors and tools developers various discussions are underway. A public domain version of ACE can be obtained from the authors.

## 6 CONCLUSIONS

At the present stage ACE is an advanced integrated CASE environment for distributed object oriented applications compliant to the emerging general and telecom-oriented distributed processing standards. Even if not yet exhaustively satisfied many of the requirements and gaps described in initial sections are addressed in ACE.

*Part of this work has been supported by the EEC ACTS AC048 project ReTINA.*

## 7 REFERENCES

N.Bersia, P.G.Bosco, G.Canal, R.Manione, C.Moiso, M.Spinolo (1995) A Case Environment for TINA-oriented applications, in *Proc. ISS'95*.

G.Booch, J.Rumbaugh, I.Jacobson (1996) Unified Modeling Language, Rational Rose.

P.G.Bosco, G.Giandonato, C.Moiso (1993) A distributed processing model for telecommunication services and operations software, in *Proc. TINA'93*.

K.Brockschmidt (1995) Inside OLE 2. MS Press, Redmond.

E.Chiocchetti, R.Manione, P.Renditore (1994) Specification based performance evaluation of distributed systems for telecommunications, in *Proc. Tools'94*.

Chorus Systeme (1994) COOL-ORB Programming Model. Technical Report, Paris.

Chorus Systeme (1995) , ReTINA, http://www.chorus.com/Research/retina.html

T. Handegard, N. Mercouroff (1995) Computational modelling concepts. TINA-C.

Iona (1995) Advanced Programmer Guide. Iona, Dublin.

ISO/ODP (1993) Basic Reference Model of ODP-part 3: Prescriptive Model. ISO/SC21 N8125.

H.Kobayashi, K.Moor, C.Abarca (1995) TINA Service Architecture. TINA-C.

P.Leydekkers, N.Mercouroff (1995) TINA Object Definition Language. TINA-C.

OMG (1991) The common object request broker. OMG Document Number 91.8.1.

OMG (1995) CORBA Services: Common Object Service Specifications. OMG Document Number 95.3.31.

J.Rumbaugh et al. (1991) Object-Oriented Modeling and Design, Prentice Hall.

G.Spinelli, W.Takita, G.Martini, S.Chikara (1996) TINA Connection Management Implementation over Distributed Processing Platforms, in *Proc. NOMS '96*.