Specifying and Developing TINA Applications with ACE

PiergiorgioBosco, Gianni Canal, Giovanni Martini, Corrado Moiso
CSELT
Via Reiss Romoli, 274
10148 Torino, Italy
{piergiorgio.bosco,gianni.canal,giovanni.martini,corrado.moiso}@cselt.it

Abstract

*The objective of the paper is to present an innovative environment (**A**pplication **C**onstruction **E**nvironment - ACE) for the specification, the development and the generation of Telecommunication applications according to the TINA and OMG standards. The functionalities offered by the environment are outlined in the following sections, in addition to experiences of usage of ACE for specification activities.*

The work presented in this paper has been partially funded by the European Union in the context of the ACTS ReTINA project (AC048).

1. Introduction

The motivations for developing ACE (Application Construction Environment) were to allow the specification, development and the generation of applications according to the computational model of TINA, with a particular emphasis on the aspects which nowadays are more difficult to be found in commercial tools.

In the IT world the available object-oriented CASE tools for distributed applications currently are more oriented to code generation than to support the specification process. However there is some attempt at improvement due to *defacto* standard architectures such as CORBA [6] and MS OLE (distributed OLE) [13] 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); finally they also generate the stubs that deal with the distribution mechanics. This approach offers a good starting point for later design/development phases, however it represents only a small step further, since it still lacks the expression of constructs or "services" such as transactions, persistence, 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, usually, not addressed in a integrated way. Notably, only SDL-based CASE environments, approximate this level of requirement coverage.

However standard languages familiar to the telecommunication community such as SDL or LOTOS and their latest evolution (that at least addresses some of the behavioural description requirements) do not have sufficiently matched the object-oriented models and behavioural concepts inherent 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 suitable 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, SDL, CORBA IDL and TINA ODL etc. to distributed computational architectures.

ACE therefore aims at providing functionalities to support main application lifecycle phases, and to represent, in terms of development, a nut-shell for a number of tools, specification editors, development/compiler tools, performance evaluation tools and code generators, etc., from within the analysts/designers/developers all together orchestrate.

2. ACE Functionalities

In this section the main functionalities supported by ACE are outlined. 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 the ACE editors it is possible to produce code both for the simulation environment and for the real platform. The specifications (aligned with the TINA Computational Model [1]) can be executed in a simulated environment, for a first functional validation of the service logics, that can be traced directly on the specification editors in order to achieve a better comprehension of their execution.

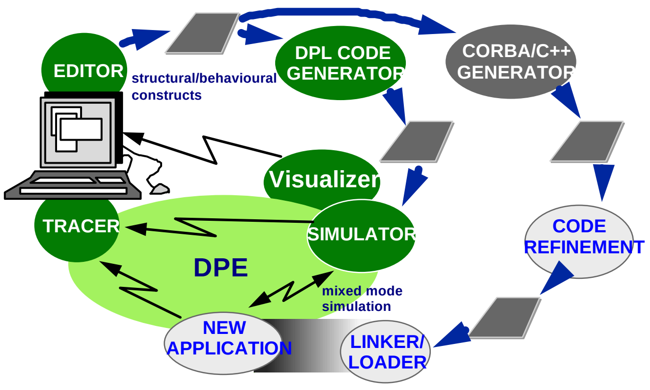

Figure 1: overview of the development cycle

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. In the next sections the main components and features of the environment are described.

2.1 ACE Computational Model

As far as conformance objectives towards TINA Computational Model [1][8], the following computational model is realised in ACE:

- objects have multiple interfaces;
- creation/deletion of objects and object-groups;
- interactions among objects, through interfaces:

  - invocations of operations: invocations can be either blocking or non-blocking;

  - accesses (read/write) to attributes;
- concurrence control in multi-threaded objects: guards and semaphores.

2.2 Editing Component

The editing component consists of a suite of graphical editors. They support the detailed specification of templates for object-groups and objects (including their behavioural parts) and interface templates

Editors for the analysis and design offer support for the standard OMT object model notation, in addition to the possibility of devising object interaction diagrams. Previously defined components in TINA ODL [7] and CORBA IDL [6] files can be imported in ACE, and hereby used and extended.

Each graphical editor is associated to a (graphical) language, consisting of a set of dedicated and customized icons and rules on their composition and interconnection through (labelled) links. 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.

Objects' methods are specified by means of a behavioural editor in 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. if-then-else constructs) and synchronisation constructs.

2.3 Syntactic and Semantic Checking

ACE performs global syntax and type checks of the specified templates (e.g. all the constructs and declarations are type-checked, the syntax of the attributes of the icons is controlled). The global checks 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, that holds all the information concerning the semantics and the typing of the templates, and is used as input to all the final code generators.

2.4 Simulation and Visualization of Specifications

The user specification can be automatically translated into a multiparadigm language that can be used to execute them in a (sequential) simulated environment.

In this way, it is possible 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.

This issue has been tackled devising a mixed mode simulation extension, that allows simulated components to interact with applications and services executing on the real CORBA platform. The interaction of simulated and real world allows the possibility of validating more complex service logics specified by the user (involving interactions with already deployed services), by executing them in a controlled and secure environment.

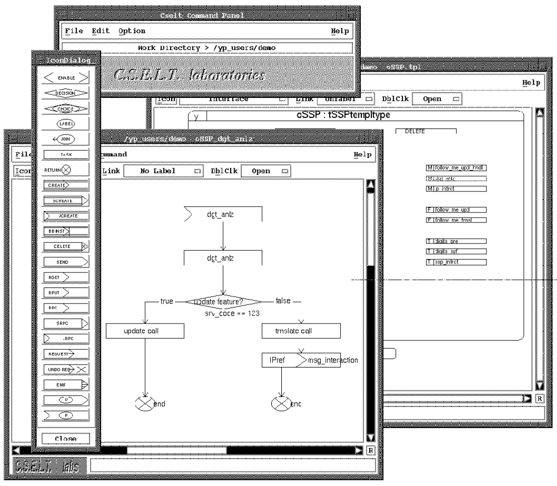

Figure 2: behavioural and object template editors

Visualisation and tracing of executions

The execution of specifications during a simulation 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 scenarios that represent the agents and the artefacts involved in the application (e.g., telephones, switches,...).

The model oriented visualisation is based on the user selectable trace of behaviour diagrams and of object states. These functionalities are tightly integrated with the editing component, in order to trace the executions on the same graphical representation of specifications. The users can interact with the execution by invoking announcements of the objects, define break-points or inspect objects' state

The application-oriented visualisation (explicitly programmed by the specifier) is based on an interface through which the users can introduce inputs to the application (e.g. by means of dialogue-boxes) and the application can provide outputs in a graphical way (e.g. through animations).

An additional output resulting from a simulation session consists of the automatic production of object interaction diagrams. They ease the task of comparing required output with the effective implemented results.

2.5 Validation and Performance Evaluation of Specifications

At present, functional validation of specification is assumed to be carried on in a user-based interactive way (see 2.4), although a more automated approach has also been devised. It uses technology for exhaustive state space exploration based on SPIN+[2], an extended model checker based on a timed version of PROMELA.

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.:

- duration 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 object-group is "deployed" on a physical processor;
- weighting of non-deterministic choices.

The same compiler used for the validation is currently used for performance evaluation. A number of executions of the PROMELA 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.

2.6 Code Generation for Real Distributed Execution

ACE provides a code generator to produce CORBA/IDL and C++ for deploying and running the applications on real CORBA platforms (Iona Orbix [4] and Chorus COOL Orb [3]).

The actual mapping tackles and solves the differences between the object model of the TINA computational model and the one supported by the CORBA [6] binding with C++ language. One of this differences concerns the support for objects having multiple dynamic interfaces, as prescribed by the TINA computational model and not equally present in the CORBA C++ mapping. It is in the process of the code generation that 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, and generate the corresponding graphical specification, in order to be later re-used in the development of a component). Likewise, starting from the object specifications (structural and behavioural sections), skeletons C++ implementations are generated automatically from the SDL-like behavioural notation.

3. Experiences in using ACE

ACE is the reference environment within CSELT to specify object-oriented distributed systems for the control and the management of services and network resources. In particular, it has been exploited, in traditional IN and TINA contexts, just as a specification environment or also as a development environment.

In the context of IN-like service specification, ACE is used as a tool to specify the components of services allocated on the different IN systems (e.g., SSP, SCP, SMS) and their interactions. In addition, in order to visualise the behaviour of a service from the point of view of the users, it was exploited the animation tools provided by the environment.

The TINA components that, at the moment, have been specified through ACE are:

- the session control, in particular the GSC component;
- the access session (i.e., the objects that perform user registration, service browsing, service invocation and user invitation;
- some modules of the specifications of the Connection Management, developed in the context of the TINA World Wide Demo[11].

The specifications of these components refined the definition provided by TINA Core Team and extended it by providing the behaviour in a formal language. During their development several feature of the environment were successfully exploited; in particular:

- the import in the environment (textual) IDL specification;
- the SDL-like formalism to express the behaviour of the system;
- the simulator to verify the correctness of the specifications and the tracer to perform their "debugging";
- the code generator to produce a first version of the implementation on the Iona/Orbix CORBA platform, to be refined to obtain the final code.

The possibility to formally define the behavioural aspects of the specification pointed out several aspects that were not properly covered by a specification of structural aspects and a description of the behaviour in natural language. In fact, the use of SDL-like formalism, aligned with the computational concepts, and forces the production of easily implementable specifications on a distributed platform. A possible example concerns the configuration of a set of objects: ACE forces to specify the actions and the invocations necessary to pass the necessary interface references to all the instances.

The production of computationally complete specifications is stressed also by the possibility to run the specifications in the simulated environment which covers the main aspects of a distributed platform. It was experimented the possibility to describe the behaviour in ACE at different level of detail: for instance, the specification of TINA GSC component includes objects whose behaviour was defined in a very detailed way (similar to the detail level of traditional implementation language), while the specification of TINA access session was produced through a process of incremental refinements to reach the satisfactory level of detail.

The analysis phase was performed on the design editor that provides a superset of the functionalities of analysis one. In this phase was detected the lacking of a support to the definition of object interaction diagrams.

The previously described specifications were mainly written by persons which were not involved in the development of ACE. The training period was about one week, provide that they have some knowledge of the computational model concepts and constructs underlying ACE. This training period does not consider the tools supporting the definition of animation scenarios: at the moment the training period of these tools is tightly related to that required by Metacard. In addition to available documentation/manuals the training required some interactions with the designers of the environment. The development of TINA specifications of session control and access session considered in this section took about two months each.

These experiences detected some weak points of the current version of ACE. The principal ones concern the simulation and the code generation. In particular, it was detected some limits in the simulation of complex systems with very detailed object behaviour; for instance it was not possible to simulate some objects of TINA GSC, whose methods were defined at a C-like level of details. While the code generation is satisfactory for the aspects concerning the structure of the applications (e.g., the production of CORBA servers from BB specifications and the handling of objects with multiple interfaces), at the moment it presents some limits in the generation of C++ methods, that require a general revision of the produced code. Some improvements are also required to the generation of documentation, mainly related to the aspects concerning the documentation of object behaviour.

As mentioned before, strong points that were confirmed are the possibility to specify both structural and behavioural aspects and to simulate the specifications. The simulation was exploited both to "debug" the specifications and to demonstrate the evolution of the specified systems; at this aim, the experiences showed the benefits of having both the tracing of specifications and the animation of application scenarios. Another strong point is the possibility to generate code for a real distributed platform, also with the current limitations, that contributes to speed up the implementation of specifications.

ACE is also used within the TINA Core Team for the validation [10] of some of the Reference Points, providing support for rapid prototyping and specifications in terms of IDL, ODL files and OMT diagrams.

Finally, in the lifetime of the ACTS ReTINA project[12], ACE is the core environment providing support functionalities for the definition, validation, and code generation of service components.

4. Conclusions

ACE is still evolving, and in the short term extensions are expected, in order to increase the offered functionalities. Among them, the support for the Java platform, at language and infrastructural level. The usage of alternative platforms (e.g. DCOM) for the distribution aspects will also be object of extensional activities.

At the present stage ACE is however an advanced integrated environment for distributed object oriented applications compliant to TINA computational model, and its support has proved of value in the development of new applications.

A public domain version of ACE can be obtained from the authors: `http:// andromeda.cselt.it/ace/ ACE.html.`

5. References

[1] TINA-C deliverable: *Computational modelling concepts, vers. 3.2*, TINA Consortium, 1996.

[2] E.Chiocchetti; R.Manione, P.Renditore: *Specification based performance evaluation of distributed systems for telecommunications*, Tools'94, 1994.

[3] Chorus Systeme: *COOL-ORB Programming Model*, Technical Report, 1994.

[4] Iona: *Orbix 2 Programmer Guide*, 1995.

[5] J.Rumbaugh et al.: *Object-Oriented Modeling and Design*, Prentice Hall, 1991.

[6] OMG: *The common object request broker architecture and specification revision 2.0*, OMG Document Number 95-07.01, 1995.

[7] TINA-C deliverable, *TINA Object Definition Language*, TINA Consortium, 1995.

[8] P.G.Bosco; G.Giandonato; C.Moiso: *A distributed processing model for telecommunication services and operations software*, Proc. TINA'93, 1993.

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

[10] R.Westerga, M.Hedberg, E.Darmois: *Using ACE for the validation of TINA Reference Points*, Service Creation Workshop,Como, 1997.

[11] G.Spinelli, W.Takita, G.Martini, S.Chikara: *TINA Connection Management Implementation over Distributed Processing Platforms*, Proc. NOMS '96, 1996.

[12] ReTINA: *http://www.chorus.com/Documentation/retina.html*.

[13] K.Brockschmidt, *Inside OLE 2*, MS Press, 1995