A FRAMEWORK TO INTEGRATE SYSTEM SPECIFICATION TOOLS

A. Di Stefano and M. Malgeri

Istituto di Informatica e Tele comunicazioni
Facolta' di Ingegneria - Universita' di Catania
Viale A. Doria 6 - 95125 Catania - ITALY

ABSTRACT

The design of communication protocols is very important today. Many Formal Description Techniques have been developed which present useful features assuring to the designer completeness, uniqueness, unambiguity and readability of the specification but many FDTs lack in readability. Then several automatic tools have been realized to assure an easy use of FDTs in order to simulate and verify the specifications. Any tool developed to aid the designers should offer a powerful user interface in order to avoid readability troubles. The interface should also handle different tools performing different tasks belonging to same FDT's class. This paper presents an architecture of an environment able to manage many tools based on behavioural FDTs, focusing its attention mainly on user interface. An analysis will be presented of the functions that general purpose interface should offer, and the problems and solutions relevant with the merging of a tool inside this framework will be discussed.

1. INTRODUCTION

The spreading of computer networks is determining the development of tools improving the capability to define standards (for example networks components, communication protocols, etc.).

The increasing of computation power of the hardware devices coupled with the decreasing of their cost gives raise to a wide distribution of the computers. Infact now each single machine is able to perform the amount of calculus performed by one large computer of ten years ago. This points out the need of communication among computers.

The problem connected with the computer networks consists mainly in the way to exchange information among different machines often built by different farmers. Therefore the standardization of communication protocols is very important today.

In order to give a standard relevant with an object a specification technique must be chosen suitable to express the object behaviour. Many Formal Description Techniques (FDTs) can be used. The main features an FDT should present are:

- completeness
- uniqueness
- unambiguity
- readability

Many research groups have investigated about one FDT presenting all the above mentioned properties. The FDTs belonging to behavioural models appear suitable to well cover the aspects of completeness, unambiguity, and uniqueness, but it happens they lack in readability.

In addition to completeness, uniqueness, and unambiguity a good specification must assure the correct behaviour (e.g. behaviour without deadlock and livelock). The correctness verification is needed in order to guarantee a well designed object. For example a standard organization should always check its proposals before releasing them. But, the verification of the specification is not a simple job.

As said above, the specification done by using an FDT presents the properties of completeness, uniqueness, and unambiguity, but the designer is not assured about correctness. Therefore several automatic tools are developing (and many are jet prototypes) to aid the designer to check his work [1][2].

Any tool developed to aid designers should offer a powerful interface with user in order to avoid readability trouble.

The interface should handle different kinds of tools belonging to same FDT's class. This paper presents an architecture of user interface able to manage any tool based on behavioural FDTs (e.g. CSP, CCS, LOTOS, LIPS [3][4][5][6]).
Since the interface must interact with different tools (e.g., simulator, performance evaluator, verification tools, etc.) the data flow is very critical. Therefore the interface needs a communication policy allowing disomogeneous tools to work together. The authors discuss some strategies to support the communication between tools and interface.

An analysis will be presented of the functions that general purpose interface should offer. These functions are collected in two sets: the first oriented to give an user friendly interface, the second to tackle both user and tools needs.

We discuss the problems and their solutions relevant with the merging of a tool inside this framework. As a case study, we consider a behaviour simulator [171]. The object to be simulated is specified by using the behavioural FDT LOTOS [151] based on Concurrent Communicating Systems (CCS) [4]. Presently this language is under standardization by ISO.

LOTOS models the object as a black box synchronizing itself with the environment through some access points, called gates, in order to exchange information. Therefore LOTOS specification consists in describing the sequences of all interactions which can take place at any gate. LOTOS has a derivational semantics which can be well mapped onto a tree representation.

The paper consists of the following main parts:
- sect. 2 showing the architecture
- sect. 3 describing the communication policies
- sect. 4 suggesting the functionalities the interface must offer.

2. ARCHITECTURE

The architecture we propose is modular since it must be quite general in order to match with several different tools. The characteristic of modularity allows us to modify the framework when a new tool is added.

At first view-depth we can individuate three large blocks (fig.1):
- database concerning with specification and framework configuration information. This database is common to all tools.
- tools (e.g. simulator)
- user interface which offers an unique approach to the use of each tool.

2.1 USER INTERFACE

This subsection deals with user interface that consists in three modules (fig.2):
- Command Manager
- Communication Handler
- Screen Manager

Command Manager (CM) is the module which processes the information provided by the user, either commands or data (fig.3).

The user gives commands by using either strings or menus and forms.

The command-string must be verified in order to check its syntax. Moreover it is checked in order to evaluate if the command can be applied in the current context.

![Figure 1. Framework Structure](image-url)
The module Syntax Analyzer (SA) performs all the above mentioned command-string analysis operations.

The commands provided by this framework must be also interpreted in order to be "translated" in the set of commands recognized by the appropriate tool. For this purpose the module Command Translator (CT) is provided. All the command so "translated" CT are collected into packets and they are given to proper module (one or more than one module together can perform the desired function).

![Figure 2. Interface Structure](image)

The Response Handler (RH) module coordinates the results given by the activated module or tool in order to provide the user with them. This module performs the dual function with respect to the CT, i.e. it transforms the results coming from any tool in the format handled by SM.

The Command Manager module provides user with an unique virtual environment overcoming the different tool approach.

For example let us examine the analysis of a LOTOS specification by using a simulator able to perform only an action at time.

Let us suppose the user asks for the set of actions rooted at the same node. He gives only a single command-string; SA checks if command string is meaningful and, if successful, activates CT. CT transforms the commands into the set of simulator commands needed to drive the tool. At last he RH collects the intermediate data in order to extract only significant results.

![Figure 3. Command Manager](image)

Moreover information on what tool should do (e.g. commands) should be exchanged by using an intermediate storage different from that one used for information to be displayed.

The main problem of batch approach is the low capability to interact with the tool. Infact information must be fully processed either by graphic user interface or by tool before they can be exchanged.

CH allows tool and interface to interact in another mode, called in the following "direct mode", i.e. CH drives tool interacting with it so they run together.
CH drives the tool according to the set of commands provided by the CT. This method permits the user to have a faster response and he also may change behavior of tool interactively, depending on partial results he can know.

A third method is "concurrent run", that is CH and tool proceed independently synchronizing themselves by exchanging primitives needed to continue.

Also this method gives results to the user in a short time (depending on tool capability).

The above presented three methods have different features: the second and third have similar features, but they mainly differ for architectural policy. The second method follows a master-slave approach, the third one is more suitable for distributed and etherogeneous architecture.

Moreover the second method generally is realized as a simple task, whilst the third one is realized implementing different tasks.

The first method has its mainly applications when the tool needs a long time to achieve the desired results.

Still referring to the LOTOS simulator example, a batch approach may be useful in such cases as an exhaustive simulation is required. Whilst the direct mode and concurrent run may be suitable to investigate the immediate result of a single action.

4. FUNCTIONS

The main purpose of an interface is to simplify the user access to all capability of the tools. Moreover the interface should permit access to the tool also to an user that does not have practice with it. In this way the tool becomes really available to a large amount of people.

The interface should be able both in interpreting data provided by the tool and in aiding the user to give his questions.

The former function of the interface presents two aspects:
- data give algebraic information (e.g. the result of an equation giving performance measures)
- data give behavioural or logic information.

The data belonging to the first item should be presented always in graphic format, for example bar diagram, pie diagram or cartesian diagram. In fact these formats allow the user to immediate understand.

Data belonging to the second item generally can be well given by using a tree picture. This kind of representation should be structured in different levels of detail in order to allow user to choose the view depth more suitable to his investigation.

Obviously, some comments should be associated to the tree picture clarifying the meaning of each element of the picture. For example, in a LOTOS tree representation the names of the actions or the set of their possible values should be provided.

The latter function, i.e. to help user to input data and command, is well realized using either menus or forms. Menus should be always structured with more then one abstraction level. Each level should provide help information as detailed as depth.

The forms should give to the user the default values valid for the selected command.

4.1 Configuration

Another feature the framework should offer to the user is the "configurability". Since this framework represents an environment for different tools, it should be able to work on different computer machines, interfacing with tools written in any programming language, and also with different operating system. Moreover the framework should allow any user to set the environment characteristics tackling with his taste.

The framework should be implemented by using a diffused programming language as "C" to permit the most portability. In fact several C compilers are available on a lot amount of computer and they are supported by the more diffused operating system. Therefore the framework is easily portable on different computer running operating system, for example UNIX, if it developed in C, but it is also portable on other computer simply recompiling it.

Moreover, since the framework allows user different ways to synchronize itself with tool, depending on host computer and tool characteristics, the best way to merge them may be chosen.

The way to exchange information may be modified during life of framework. To help in doing it, a "setup menu" should be provided which allows user to change the default "synchronization mode", or work context, and so on.
The framework should also provide user with a facility to create own menus and forms, in order to better merge the tools inside the environment.

When the tool is installed the designer must define what are the most common question the user may ask, and prepare a menu to aid him. To do it, the installer should simply list the command that will belong to menu into file in ASCII format. He must also define the table that permits the conversion between user commands and tool commands.

During the run of a tool the user may also change some characteristics, mainly those concerning with framework appearance. For example the user may chose the color of the screen, the default menu format, some default strictly regarding tool behaviour. For example, taking into account the simulator mentioned in the paper, the user may set the default "simulation depth", the dimension of the inferential tree path to be represented as tool response, and other characteristics.

REFERENCES


