Sophtalk tutorials

Ian Jacobs, Janet Bertot

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Rapports Techniques

N°149

Programme 2

Calcul symbolique, Programmation et Génie logiciel

SOPHTALK TUTORIALS

Ian JACOBS
Janet BERTOT

Février 1993

Institut National
de Recherche
en Informatique
et en Automatique

2004 route des Lucioles
B.P. 93
06902 Sophia-Antipolis
France

Sophtalk tutorials

Ian Jacobs Janet Bertot

February 17, 1993

Sophtalk tutorials

Abstract

This paper presents the Sophtalk system through two tutorials that describe salient features of the system and explore appropriate design methods. Sophtalk implements an event model of communication. System objects, called *stnodes*, emit messages when significant events occur, such as the termination of a computation, a request for a service from another object, error conditions, etc. Messages circulate asynchronously in a network of stnodes. An stnode's type determines which messages it will receive; upon reception, an action corresponding to the message and stnode instance is triggered.

In the first tutorial, a small network in which a calculator —possibly in a separate process— serves several clients, we introduce the essential Sophtalk functionalities. In the second tutorial, a news network, we concentrate on design aspects and illustrate traps to be avoided.

Une introduction au système Sophtalk

Résumé

Dans ce rapport, nous présentons le système Sophtalk par le truchement de deux exemples. Sophtalk est un ensemble de fonctionalités pour décrire l'interaction d'objets selon un modèle de communication basé sur des événements. Les objets primaires du système, dits *stnodes*, émettent des messages lors d'événements significatifs, tels que la terminaison d'un calcul, une requête pour un service fourni par un autre objet, ou encore une erreur, etc. Les messages circulent de façon asynchrone dans un "réseau" de stnodes. Le type d'un stnode détermine l'ensemble des messages qu'il écoute. Lors de la reception, une action qui est fonction du message et du stnode est déclenchée.

Dans le premier exemple —un petit réseau contenant une calculatrice (éventuellement un processus indépendant) qui sert de multiples clients— nous considérons les fonctionalités essentielles de Sophtalk. Dans le deuxième exemple, un réseau d'informations, nous nous concentrons sur les aspects de conception ainsi que les façons d'éviter quelques pièges fréquents.

Contents

1 Introduction **1**
1.1 Tool encapsulation 1
1.2 Nested stnodes 1
1.3 Communication protocol 2

2 Tutorial prologue **5**
2.1 Network principles 5
2.1.1 Networks 5
2.1.2 Events 6

3 Tutorial I : A calculator **7**
3.1 Designing the calculator network 7
3.2 Declaring network stnodes 7
3.3 Creating atomic stnodes 9
3.4 Atomic stnode input actions 10
3.5 Emitting messages 11
3.6 Inclusion in a network 12
3.7 Multiple clients 13
3.8 Interprocess communication 15
3.8.1 Stio 15
3.8.2 Stservice 16
3.8.3 Connecting an external process to a network 16
3.9 Pretty printing 19
3.10 Tracing signals with spies 20

4 Tutorial II : A news network **24**
4.1 Design choices 24
4.2 Functional specifications 28
4.2.1 News functions 29
4.2.2 Reader functions 29
4.3 Communication 31
4.3.1 News network ports 31
4.3.2 News manager ports 34
4.3.3 Reader ports 36
4.3.4 Newsgroup ports 37
4.3.5 Newsgroup manager ports 37
4.3.6 Subscriber ports 39

4.4 Demonstration 41

5 Bibliography **47**

6 News network implementation **48**

1 Introduction

Recent progress in software engineering techniques has consolidated the idea that any tool that performs a task (editor, button, typechecker, etc.) should be able to operate in isolation, without having to worry about sources of information it requires or recipients of information it generates [1, 2]. Rather than converse directly with its comrades, a tool announces an important event by broadcasting an appropriate message "into the world." The term *multicast* is more appropriate since a message only reaches the ears of those tools that have already declared their interest in such a message. Listening tools may react to a message, perform some activity, and emit their own messages. If no tools listen to a given message, it falls on deaf ears.

This event model of communication has been widely used to coordinate communication between several processes, but the principle of separating a tool's task from its communication needs also applies to single process tool interaction. A tool emits a message whenever it requires information, or when it accomplishes (or fails to accomplish) an important task. "I have finished compiling," "Someone has modified this file," or "The current selection has changed" are examples of significant events. Events may occur at any moment, so this model of communication is naturally asynchronous, but one may impose synchronization through design choices.

With Sophtalk, one may design a system as a network of event-conscious tools. This kind of design allows developers to add, modify, substitute, or remove tools from the system without disturbing global behavior.

1.1 Tool encapsulation

To ease the integration of tools into the system, each tool is *encapsulated* in a structure which we called an *stnode* (sophtalk node). The type of the stnode specifies how the tool communicates with the outside world. An stnode type is defined by its input and output *ports*, i.e., its communication interface.[1]

1.2 Nested stnodes

An atomic stnode encapsulates a tool (editor, button, etc.). A non-atomic stnode encapsulates other stnodes. Grouping stnodes together within a non-

[1] Ports are statically declared as simple names; in Sophtalk, it is not possible to specify the type of a value received or sent on a port.

atomic stnode gives rise to *bus lines* that link matching sibling input and output ports, allowing them to exchange messages. These bus lines may be viewed as wires over which messages circulate. When one adds an stnode to a group of siblings, input (resp., output) ports are connected to output (resp., input) bus lines with the same name. It is possible to rename a port in order to connect it to some other bus line. Thus, there is no problem connecting semantically compatible ports that differ only in name.

Encapsulation has the effect of hiding the details of an stnode's behavior behind an encapsulating interface (see figure 1). For example, a calculator stnode may have a simple interface: one input port for incoming expressions to be evaluated and one output port for the result. The calculator may be implemented by an stnode that contains other atomic (or non-atomic) components such as a memory, stack, registers, and a processor.

Each stnode instance contains only one tool, but a tool may be encapsulated by more than one stnode. In this way, one may change a tool's communication interface according to context.

1.3 Communication protocol

Informally, messages circulate in a tree of non-atomic and atomic stnodes as follows:

- An stnode may emit a message on one of its output ports **O**. The message circulates between siblings on the imaginary bus line to which **O** has been connected (renaming allows one to choose any bus line). If the encapsulating stnode has an output port connected to this bus line, it propagates this message outward.

- An stnode may receive a message on one of its input ports **I**. If the stnode is atomic, it calls a user-defined function, allowing the encapsulated tool to react to the incoming information. If the stnode is non-atomic, it propagates the message along a bus line that connects its descendents.

- An encapsulated tool may initiate communication when important events occur by sending a message over one of its container stnode's output ports. The object does not participate directly in the network, but speaks via its encapsulating components.