Inter-component communication as a vehicle towards end-user modeling

Manolis Koutlis (koutlis@cti.gr)
Petros Kourouniotis (P.Kourouniotis@asyk.ase.gr)
Kriton Kyrimis (kyrimis@cti.gr)
Nikolina Renieri (renieri@cti.gr)

Computer Technology Institute
Patras, Greece

Introduction

In this work we describe a mechanism for inter-component communication. This mechanism forms the basis of a broader environment designed to support the construction of educational applications. These applications are constructed by end-users and not by "programmers", by assembling high-level, domain-specific software components into functional wholes. The above mechanism arose from requirements put forward by the nature of the targeted application domain:

- It must be possible for teachers to prepare the required course material "on their own" (i.e., by altering predefined templates, or by constructing new templates from scratch), customizing it for the subjects on which they want to focus, the intended audience, their individual teaching style, etc. Thus, the software should be (re)constructable by the teacher rather than being provided as "hard-coded" pre-defined scenaria, created by a programmer.
- To do so, teachers would need to think and build using high-level entities, modeling concepts, processes and phenomena close to their domain of concern, instead of having to deal with the low level data-types and primitive operations that are typically provided by programming or scripting languages [1], [2], [3]. Using a "kit" of such building blocks, it should be possible for teachers to construct many different teaching scenaria (just like a small number of Lego brick types suffice to build arbitrary constructions).
- In addition, teachers would like to exchange educational materials and use constructions made by other colleagues or found in repositories like the web, thus making shareability and reusability of resources an additional requirement.
- Finally, the software should be usable by both students and teachers, and its use should not depend on the users' familiarity with computers. Thus, the user interface should be intuitive, based on familiar paradigms from the real world.

With these pursuits in mind we took the path of designing educational components, able to inter-operate in user-defined "editable applications"[1]. To achieve this inter-operation we designed and implemented the inter-component communication mechanism described below.

The communication mechanism

Components are connected together via the *plug* metaphor. Each component has a number of plugs, which can be likened to the sockets of a Hi-Fi component. To connect two components, they must each have an appropriate plug, corresponding to a plug on the other component. This is similar to a tape-deck's "output" socket, which is connected to the "input" socket of an amplifier. To expand on this metaphor, connecting a tape deck's "input" socket to the "input" socket of an amplifier does not work. Similarly, the interconnection mechanism does not allow the connection of unrelated plugs, which would have no meaning.

User-level description

Each component has a set of plugs, which can be displayed by clicking on the component with the right mouse button. Each plug has a name, suggesting the plug's function, and an icon. The icon is in the form of a jigsaw puzzle piece, with a certain shape and color. To connect two components, the user must select one of the plugs from the first component, and a matching plug from the second component. Two plugs are considered to match if their icons have the same color, and their shapes fit together (see Figure 1).

For example, consider a mathematics lesson on vectors. The teacher can set up an activity (scenario) where two building blocks (components) are used: an aeroplane and a vector editor. The aim of the activity is to study the

[1] In essence and with reference to OMT [5] terminology, we view this latter process as high-level "modeling" task in which end-users are designing the object modeling part of a targeted application, with the exception that they are constrained to do so with prefabricated entities (components), and define the interrelationships among them, guided (constrained) by their (pre-decided) connectivity capabilities. By doing so, they have finished developing the application, as the next two phases of the process (again in OMT terminology), dynamic and functional modeling, are already encapsulated in the components' behavior and interaction patterns and the supporting communication mechanism.

effect on the aeroplane's motion. As shown in Figure 2, the wind velocity can be specified by the user via a vector component. By connecting the two components, user-given values for wind velocity are passed to the aeroplane component, making it possible for the user to study the effect of wind velocity on the aeroplane's motion. The teacher has the flexibility to choose to display this motion on a map component, to display the aeroplane's ground velocity on another vector component, or to do both. By considering the aeroplane's air velocity either fixed or controllable by the student through yet another vector component, the teacher is given even more options in setting up activities.

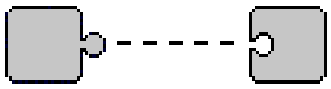

Figure 1: Two matching plugs

Notice in Figure 2 that the vector component has a generic plug, called "vector", and that the aeroplane has a matching plug called "wind velocity". The vector can be connected to any component that can accept vector data as input (e.g., a planar graph drawing component). Also notice, that the matching plugs have different names.

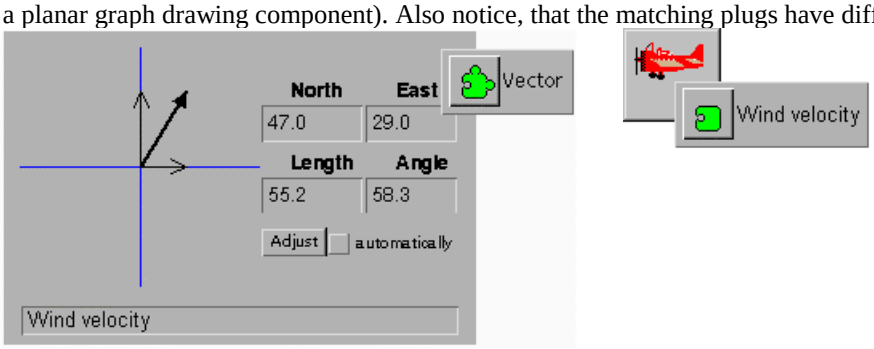

Figure 2: Connecting two components

Implementation-level description

Internally, plugs are implemented in two different ways: using *shared objects* and connection-specific *protocols* [4].

Shared objects are data that can notify all components participating in a connection whenever their value (e.g., "wind velocity" in Figure 2) changes. Thus, shared objects behave as if they were part of all components participating in a connection instead of only the component that created the object. To communicate via a shared object, a component sets the shared object's value. The shared object notifies all other connected components that the shared object's value has changed. They, in turn, read the shared object's value and perform the appropriate action. In other words, shared objects provide a data flow mechanism.

The plug of the component that creates the shared object is an *output plug*, while the plugs of the other components that connect to it are *input plugs*. This is reflected in the shape of the plug's icon (see Figure 1).

Protocols are an alternative way inter-component communication is implemented. A protocol is the specification of the set of methods that a component must implement in order to communicate with another component. E.g., consider a joystick component. Another component that can be connected to it (e.g., a spaceship), must implement a "fire" method, which is invoked by the joystick component whenever its fire button is hit.

These two mechanisms would appear to be redundant: shared objects could be implemented via protocols that specify the methods that read and write the shared object's data, and protocols can be implemented via shared objects by having a shared object called "command", where commands are passed as plain text (e.g., "fire"). However, both methods were implemented, because, depending on the nature of the plug and the programmer's perception, it is more appropriate to view some plugs conceptually as shared objects, and other plugs as implementing a protocol.

The plug mechanism can be described using the OMT [5] diagram in Figure 3.

Implementation and applications

Our work in educational software components began in 1993 and a first version of the mechanism was implemented in OpenDOC. Given that this platform stopped being supported in 1997, our additional requirements for web support, and the current turn of the programming world towards the Java platform, the current version of the mechanism was re-implemented in Java.

The mechanism is being used for the interconnection of educational software components which are being developed in the course of various projects[2].
The mechanism is not limited only to educational components. It is a general-purpose inter-component communication mechanism, and is available in the form of an application programming interface (API) for use by third parties.

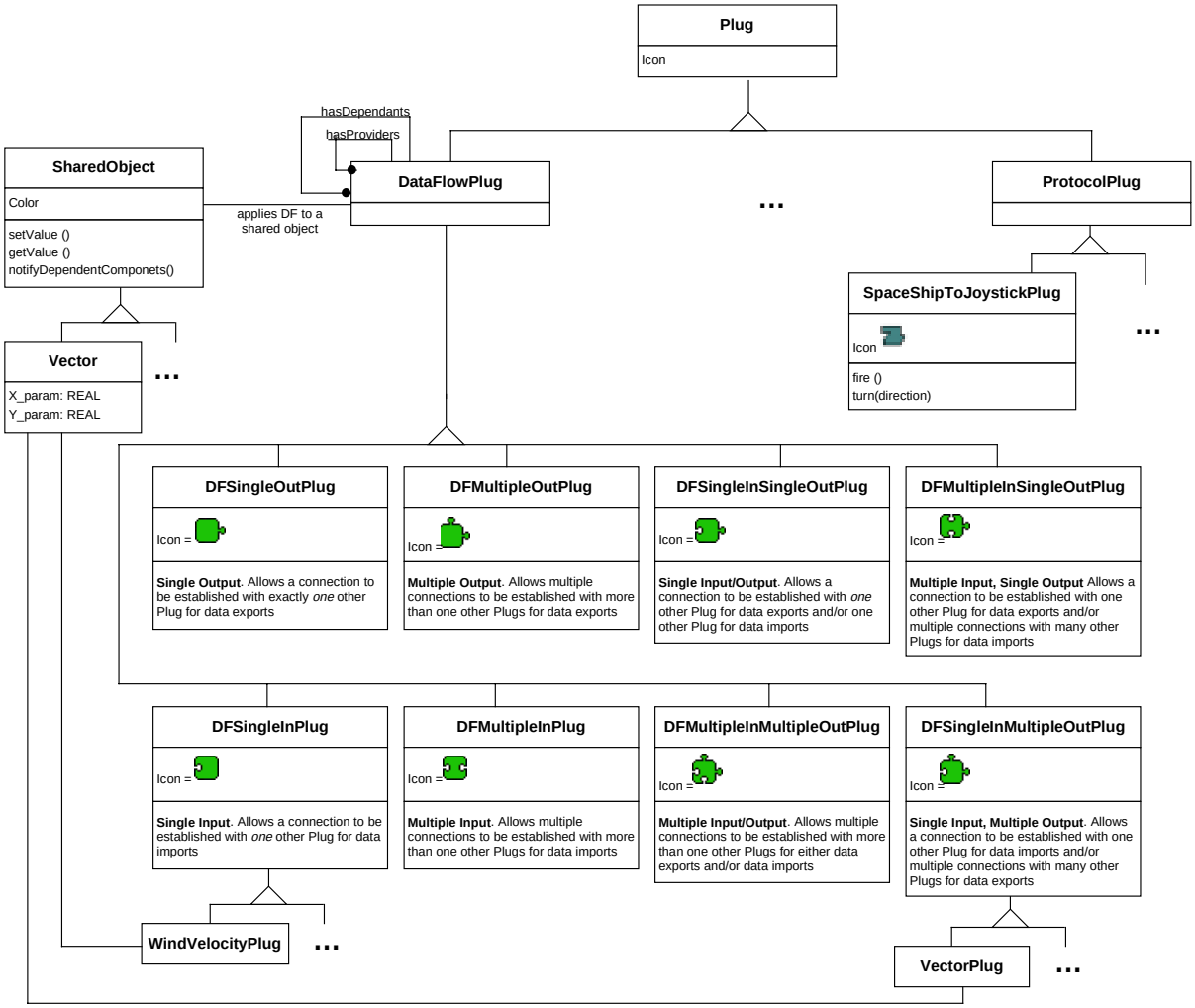

Figure 3: OMT description of the plug mechanism

References

[1] Roschelle, J. & Kaput, J. (1996). Educational software architecture and systemic impact: The promise of component software. Journal of Educational Computing Research, 14(3), 217–228.

[2] diSessa, A. (1997). Open toolsets: New ends and new means in learning mathematics and science with computers. (In press).

[3] Kynigos, C., Koutlis, M., Hadzilacos, T. (1997). "Mathematics with component-oriented exploratory software". To appear in: International Journal of Computers for Mathematical Learning.

[4] Pintado, X. (1995). "Gluons and the cooperation between software components", in Object-Oriented Software Composition, Eds Oscar Niersrtasz & Dennis Tsichritzis, Prentice Hall 1995, ISBM 0-13-220674-9, pp 322–349.

[5] Rumbaugh J. etal "Object Oriented Modeling and Design", Prentice Hall Int. Editions 1991, ISBN 0-13-630054-5.

2 The projects where this mechanism is being used are:

- Project "YDEES" ("The computer as a tool for exploration, expression of ideas and communication for everyone in the school", 1995–98, http://www.cti.gr/RD3/EduTech/ydees.html), funded by the European Community Support Framework II (Greek Ministry of Industry Energy and Technology, General secretariat for R&D, Measure 1.3, Project 726).
- Project IMEL ("Intercultural Microworld courseware for Exploratory Learning", 1996–98, http://www.cti.gr/RD3/EduTech/IMEL.htm) funded by the European Union's SOCRATES programme, ref # 25136-CP-1-96-1-GR-ODL).
- Project "ODYSSEAS" ("Integrated Network of School and Educational Regeneration in Achaia, Thrace and the Aegean", 1996–99, http://odysseia.cti.gr/odysseas/english/ukabout.html) funded by the European Community Support Framework II (Greek Ministry of National Education and Religious Affairs, Measures 1.1 and 1.4).