Interactive Human Interfaces with Engineering Software

Gonzalo Farias ∗ Sebastián Dormido ∗ Francisco Esquembre ∗∗

∗ Departamento de Informática y Automática, UNED, Madrid 28040, Spain (e-mails: gfarias@bec.uned.es, sdormido@dia.uned.es).
∗∗ Departamento de Matemática, Universidad de Murcia, Murcia, Spain, (e-mail: fem@um.es)

Abstract: Building interactive simulations with standard engineering software is normally a hard task for many non-programming instructors. Maybe the reason is that software for technical computing comes full of toolboxes for rapid-prototyping of engineering systems, but it frequently lacks tools to create interactive simulations. The paper introduces the interoperate approach, which aims to split the development of an interactive engineering simulation in two separate tasks. On the one hand, the interactive human interface is created by using specialized tools. On the other hand, the engineering simulation is built by using the preferred engineering software. The necessary API of the protocol to communicate both parts of the simulation is described in detail here, and some examples can be found at http://lab.dia.uned.es/rmatlab/.

Keywords: Control Education, Virtual Labs, Simulations, MATLAB, EJS.

1. INTRODUCTION

Information and communication technologies provide great new opportunities for education. In the last years, the scientific community has devoted great efforts to taking advantage of these new possibilities, applying them to various fields of engineering education, including control engineering (Heck, 1999; Dormido, 2002). Among all the opportunities that these technologies offer, three present features of special interest to teaching engineering: network communications, visualization, and interactivity.

The connectivity between many different computational devices is increasing every year. Networks such as the Internet are widely distributed in society, connecting people across of the world. This situation gives educators the opportunity to offer their students new ways to access learning resources without time and location constraints.

On the other hand, visualization and interactivity have proved to be crucial aspects when designing virtual labs, i.e. simulations that are to be used for pedagogical purposes, in the field of control engineering. The graphical capabilities of computers, using images or animations, can help students to understand more easily the key concepts of the system under study. Moreover, interactivity allows students to simultaneously see the response of the systems to any change introduced by the user (Dormido et al., 2005a; Sánchez et al., 2005). These features add to engineering simulations rich visual content and the possibility of an immediate observation of system response, which turns a simulation into a natural and human-friendly way to learn, helping the student to get useful practical insight into engineering systems fundamentals.

Despite the educational importance of the three mentioned features, most of the engineering software currently used by teachers and students is far from the mentioned recommended way to teach and learn. Software for technical computing comes full of toolboxes for the design and rapid-prototyping of an engineering system, but it frequently lacks tools to facilitate adding user interaction and rich visualization capabilities to the simulations.

Another problem arises when an educator finally achieves (sometimes with great effort) to build an interactive simulation by using the facilities of a particular engineering software. Instructor then realizes that the graphical user interface thus created can not be used with a similar simulation developed using a different engineering software, which is somewhat frustrating. User interfaces of standard simulations usually display common features, and it is reasonable to assume that they should be reused, independently of the software selected to create the model of the engineering simulation.

The purpose of this paper is to describe the interoperate approach, which is a novel approach, based on previous work (Sánchez et al., 2002; Dormido et al., 2005b), that defines a simple, but at the same time generic, protocol to help add human interfaces to engineering simulations. This approach splits the development of an interactive simulation for engineering in two separate tasks.

On the one hand, the model of the engineering simulation is created using a standard simulation software for engineering. On the other hand, the interactive user interface of the simulation is created using any high-level computer language or software tool specialized in the design and implementation of graphical user interfaces. Finally, these two separate components are connected using a communication protocol in a clean, effective, and reusable way.

An advantage of the interoperate approach is the possibility of reusing a given user interface to manipulate different
Fig. 1. The **interoperate approach** allows to use the same user interface with different engineering simulations. The figures correspond to: a) A Simulink model of a bouncing ball, b) A Scicos model of a bouncing ball, c) An interactive human interface.

engineering simulations which model essentially the same system. This feature of the approach helps reduce the development time of interactive simulations. For instance, an existing human interface that students are familiar with, and can use comfortably, can be used again to control a new simulation with a totally different engineering software. Fig. 1 visualizes this idea. In it, the same user interface can be selected to control one of two distinct engineering simulations. Note that one of these engineering simulations is located on a remote computer.

The paper is organized as follows. Section 2 describes briefly the creation of standard engineering simulations. The **interoperate approach** is introduced in Section 3. In Section 4 a communication protocol to connect engineering software with an interactive human interface is presented. Section 5 adapts the communication protocol to perform remote operation of engineering simulations. Finally main conclusions are discuss.

2. STANDARD ENGINEERING SIMULATIONS

There are currently many tools that can help build a simulation of a large class of systems in control engineering. Most of these tools are extraordinarily useful to study the behaviour of the systems under different scenarios. Nowadays MATLAB (The MathWorks, 2010) is the de facto standard software tool in control engineering.

MATLAB is a technical and numerical computing environment. The software allows matrix manipulation, plotting of functions and data, implementation of algorithms, etc. The capabilities of the environment are extended by a family of add-on application-specific solutions called toolboxes. MATLAB environment allows the development of user interfaces using the set of tools named GUIDE.

Although instructors can use GUIDE to create advanced user interfaces, this is not the common situation they build engineering simulations. Maybe the main reason is that GUIDE helps to build easy simple control panels to manipulate static simulations. However, dynamic simulations require more advanced mechanisms to take into account any user interaction while the simulation is being performed. The implementation of these mechanisms requires a greater effort from the programming point of view.

MATLAB also provides other sophisticated tools to create simple user interfaces. For example, Simulink can be used to simulate dynamic systems by connecting pre-defined blocks of this toolbox. Scope and Slider Gain blocks can be used to support user interaction in the Simulink simulations. However, the user interfaces obtained are not good enough in terms of visualization and interactivity.

A typical example of a Simulink simulation is displayed in Fig. 2. The Simulink model (Fig. 2a) simulates a non elastic bouncing ball under the effects of gravity. The common way to learn using this simulation is that users analyse the bouncing ball model by simply modifying the block parameters and running the simulation as many times as they need to.

Inspired by MATLAB, many other engineering tools have appeared in the last years. Many of these tools have a programming language compatible with MATLAB such as Sysquake (Calerga, 2010) or the open source solutions Scilab (Scilab Consortium, 2010) and Octave (Eaton, 2010). However these tools do not offer all the functionality that MATLAB provides. For example, neither Scilab nor Octave have a feature similar to GUIDE that allows authors to generate automatically graphical user interfaces. Moreover, Scilab offers a nice way to create interactive user interfaces, but does not provide the wide set of libraries included in powerful MATLAB toolboxes.

3. THE INTEROPERATE APPROACH

The simulation displayed in Fig. 2 models very well the system under study. But it clearly lacks a sufficiently rich level of interaction and visualization. In particular, there is no way to interact with the model while the simulation is running. An alternative simulation of the bouncing ball with a higher degree of interactivity and visualization is displayed in Fig. 3. In Fig. 3a, a more suitable graphical user interface (GUI) is displayed. This GUI shows a ball (the blue particle) bouncing on the floor. The parameters, such as gravity and restitution coefficient, can be modified by using sliders, and in addition, the vertical position of the ball can be changed by dragging up or down the particle in run time. Observing the plots in Fig. 3b, it is...
Fig. 3. An interactive simulation of a bouncing ball. a) Interactive human interface, b) Plots of the speed and vertical position of the ball.

easy to appreciate the effect of the change of the elasticity coefficient of the ball at t=5s to -1.0, and at 10s back to the previous value of -0.8.

The alternative simulation shown in Fig. 3 was created reusing the Simulink model of Fig. 2a, by adding to it a graphical user interface created in Java.

In this work, the claim is that this combined use of different tools, each suited best for a given task (engineering software specialized in modeling engineering systems for the model of the simulation, and authoring tools suited for the design and implementation of user interfaces), makes a perfect combination that brings the best of both worlds to educators in order to develop their learning simulations.

Alternatively, instructors can always try to add the required interactivity and visualization to an existing engineering simulation using the same engineering software they used to create the model. However, this approach usually has the following shortcomings:

- An interactive simulation created with a given engineering software requires that the engineering software be installed on the user’s computer.
- Different engineering software require different ways to create similar user interfaces.
- Engineering software does not always provide tools to ease the building of interactive simulations.

On the contrary, since the interoperate approach splits the interactive simulation into two components, instructors can take advantage of high-level programming languages to develop the user interface independently of the engineering software used. Instructors can even use specialized authoring tools, such as Easy Java Simulations (Esquembre, 2010), to facilitate the creation of the user interface of the interactive simulation. This approach also facilitates the reuse of components, either using the same engineering simulation with a more elaborate graphical user interface or using the same user interface with a simulation described in a different engineering software. Finally, the approach can benefit from networks to interoperate the human interface and the engineering software on different computers (see Fig. 1), eliminating the need to install the engineering software on the student’s computer.

4. THE COMMUNICATION PROTOCOL

Conceptually, in any interactive simulation that uses the model of an existing engineering software, two separate components that interoperate can be distinguished: the client application and the external application.

The client application represents the human interface, i.e., the graphical interface that the user observes and manipulates. The external application runs the engineering simulation, i.e., the simulation created with the standard engineering software that models the process of interest. The external application is controlled by the client application and provides it data for visualization. The client application adds to the underlying (external) engineering application an upper layer that allows students to manipulate and visualize the system’s response interactively.

For this scheme to work correctly, both client and external applications must exchange data continuously as the simulation is running. Thus a communication protocol to perform the interactive simulation is required. For example, the external application should be correctly configured and launched at start-up. It should also provide a way for the client to read, in run-time, the value of variables of the model, for visualization. The external application should also allow the client to start the simulation, pause it to modify any parameter or state variable of the system, and resume the simulation, as required by the user.

For its part, the client should be configured to control, query, and send data to the external application using a protocol that the external application can understand, without interrupting the execution of the external application and keeping perfect synchronization between the visualization offered to the user and the model of the external application.

The communication protocol described here represents a simple, but powerful application programming interface (API) that any engineering software must conform to, in order to provide all the features required to effectively implement the interoperate approach.

The protocol comes in two levels: low and high. A low-level protocol has been defined that lists all the required communication mechanisms that will offer the client application a complete and flexible control of the external application. Any external application must follow this low-level protocol in order to provide all the features required to effectively implement the interoperate approach.

The high-level protocol has been defined that lists all the required communication mechanisms that will offer the client application a complete and flexible control of the external application. Any external application must follow this high-level protocol in order to successfully implement the interoperate approach. But a high-level protocol has also been defined. This high-level protocol offers a simplified list of communication instructions that can, in most practical circumstances, help authors successfully implement the interoperate approach with a minimum of programming effort. The high-level protocol is based on the low-level protocol, in the sense that it ultimately uses the API provided by the external application, and offers the client application the minimum, but sufficient, set of rules to manipulate the exchange of data.

This two-level communication protocol scheme allows authors the choice to manipulate the engineering simulation in two different ways. On one hand, the high-level protocol gives authors the control of the engineering simulation at a simple, high level of abstraction, hiding a number of de-
tails, but still providing an effective link between the non-
interactive model and the interactive user interface of the
simulation. On the other hand, the low-level protocol gives
authors total control of the external simulation, providing
an enhanced link between both applications, bringing a
higher level of interaction and visualization.

Usually, the high-level protocol is all that most authors
will need to fulfill their interaction requirements, it being so
therefore the recommended entry level for authors who are
not expert programmers or do not need a very detailed
control of the communication between client and external
applications. The low-level protocol is the choice for au-
thors that need full control of the engineering simulation
and the communication mechanism. Needless to say, a cor-
rect use of the low-level protocol to design an interactive
simulation requires some more programming effort than
that of the high-level protocol. But it should also be noted
that it can also result in a more efficient simulation in terms of communication traffic and execution times.

Java is used here to exemplify the implementation of the
communication protocol and to provide some sample code
with examples of use. The choice of this language allows to
make the protocol explicit in a way that any programmer
of a different language can understand and adapt.

4.1 Low-level protocol

To manipulate and visualize the system’s response in-
teractively, client and external applications must keep a
continuous flow of information composed of data and com-
mands. The data is provided by the external application
as the simulation runs. The commands are used by the
client application to control the execution of and query
the external application.

The first command required in this communication flow is
a request for connection. Some engineering programs may
require an initialization command, for instance when some
code has to be executed in order to prepare the engineering
simulation before it is actually run. Once the connection
is established, the external application is manipulated by
the client application executing commands and retrieving
information about the results of the executed commands.

There are, therefore, three blocks of communication meth-
ods that an external application must support:

**Connection and configuration** These are methods that
initialize the external application, prepare it for opera-
tion, and eventually quit it in an orderly way.

**Setting and getting values** These methods allow the
client to set (write) and get (read) the value of any
(accessible) variable of the external application.

**Control commands** These methods allow the client ap-
lication to execute code in the external application.

These actions are supported by a number of methods
defined in the Java interface `ExternalApp`. Listing 1
shows an example of use of a given external application
(MyExternalApp) that implements the `ExternalApp`
interface. Note how the methods `connect`, `setValue`, `eval` and
`getDouble` are used to initiate the connection; to write the
external variable `t` and `f`; to compute a sinusoidal function;
and to read the external variable variable `y` respectively.

```
import es.uned.dia.interoperate.ExternalApp;
public class LowLevelExample {
    public static void main (String[] args) {
        // Declare local variables
        double time=0, frequency=1, value=0;
        // Create an ExternalApp instance
        ExternalApp externalApp = new MyExternalApp();
        // Start the connection with the external app
        externalApp.connect();
        // Set external variable f to frequency’s value
        externalApp.setValue("f", frequency);
        // Perform the simulation
        do {
            externalApp.setValue("t", time);
            externalApp.eval("y=sin(2*pi+f*t)+cos(t)");
            value=externalApp.getDouble("y");
            System.out.println("time: "+time+" value: "+value);
            time=time+0.1;
        } while (time<=10);
        // Finish the connection
        externalApp.disconnect();
    }
}
```

Listing 1. Sample code of use of the low-level protocol.

When run, the program produces the following output:

```
time:0.000 value: 0.000
time:0.100 value: 0.585
time:0.200 value: 0.932
...
```

4.2 High-level protocol

As the sample code above shows, the low-level protocol
provides all that is needed by a programmer to control
an external application. However, as simulations grow in
size and sophistication, it becomes more and more difficult
for a not-so-expert programmer to keep control of all
the method invocations required to keep the client and
external applications perfectly synchronized.

The utility of high-level protocol comes from the identifi-
cation of the basic mechanism of communication between
client and external application. In a large percentage of
simulations, the author just needs to connect and configure
the external application and then make sure that a number of
variables of the user interface are linked to correspond-
ing variables of the external application (e.g. `time` and `t`,
or `value` and `y` in Listing 1). Interactive communication
between both applications requires that any change in
the user interface of the client application immediately be
reflected by the external application and vice versa. That
two variables are linked means that both must hold the
same value, and that changes occurring in one of them
must be automatically propagated to the other one. The
high-level protocol must then assume the responsibility of
this synchronization, not the author of the simulation.

The high-level protocol is conceived as an extension of
the methods of the low-level protocol, and therefore offers
all the previous functionality, and is included in the
`ExternalApp` interface. The new methods added by the
high-level protocol can be divided in two main groups:
The linking methods and the control methods. The first
group of methods is used to link variables (by using a Java
mechanism called Reflection) of the client to variables of
the external application. The second group of methods is
used mainly to update the linked variables.
Listing 2 shows the high-level protocol in action. This is a version of the same program in Listing 1, but now using the high-level protocol (obviously same output of previous version is obtained). Note how the methods connect, linkVariables, setCommand and step are used to initiate the connection; to link the client and external variables; to define a sinusoidal function to be evaluated; and to step the interactive simulation.

Successive calls to the step method handle automatically all required calls to methods of the low-level protocol. The step command performs sequentially three actions:

- Sets external variables to the corresponding client values.
- Runs the engineering simulation (e.g. executing a given command) as many times as dt indicates.
- Gets external values and sets the corresponding client variables.

```java
import es.uned.dia.interoperate.ExternalApp;

public class HighLevelExample {
  // Declare local variables
  public double time=0, frequency=1, value=0;
  public static void main (String [] [] args) {
    new HighLevelExample();
  }

  public HighLevelExample () {
    // Create an ExternalApp instance
    ExternalApp externalApp = new MyExternalApp();
    // Set the client application
    externalApp.setClient (this);
    // Link variables with the external app
    externalApp.linkVariables("time", "t");
    externalApp.linkVariables("frequency", "f");
    externalApp.linkVariables("value", "y");
    // Configure the external app
    externalApp.setCommand("y=sin(2*pi*f*t)+cos(t)");
    // Start the connection
    externalApp.connect();
    // Perform the simulation
    do {
      externalApp.step(1); // step once
      System.out.println("time="+time+"value="+value);
      time=time+0.1;
    } while (time<=10);
    // Finish the connection
    externalApp.disconnect();
  }
}
```

Listing 2. Sample code of use of the high-level protocol.

After the call to the step method, therefore, the variables of the client which have a link will hold the same values as those of the external application linked to them. The client can then use them for whatever visualization is required. If the client changes any of these variables (for instance, due to user interaction), the step method will take care of updating the external variables prior to stepping the external application.

5. REMOTE INTERACTION

The possibility of using a network connection between both applications introduces a new factor that differentiates between two types of connections: local links and remote links. If the client application and the external application are located on the same computer, then the connection is called a local link. This is the situation that has been described so far. If, on the contrary, the client application and the external application are located on two different computers and communicate through a network connection, then the connection is called a remote link.

Performing engineering simulations over networks can be especially useful when hardware or software resources are limited. On some occasions, the engineering software is not installed on the students’ computers, or the simulation needs a high level of computational power, or it requires special hardware to be performed.

The scheme of such a remote link is presented in Fig. 4. Notice that the communication protocol is implemented on both the client and server sides. The approach encapsulates the network in such a way that it becomes transparent to the end user.

In practise the possibility of network delays introduces additional requirements for the protocol. If network delays are negligible, authors can use the interoperable approach indistinguishably of the type of the link selected to deploy the interactive simulation. However, if network delays are noticeable, the simulation will give a poor performance. For this reason, and in order to minimize this undesired effect, a modified version of the communication protocol is required. This new version is termed asynchronous to distinguish it from the standard one, that will be referred to as synchronous.

The introduction of synchronous/asynchronous links has effect only when authors use the high-level protocol, because the flow of information between client and external application will be accomplished differently depending on the version used. Because the low-level protocol provides authors with a direct control over the exchange of information among the applications, it is the responsibility of the programmer to decide how to handle this flow of information, taking into account or not the possible network delays.

In order to explain the difference in the execution of synchronous and asynchronous links, Table 1 lists all individual phases (depicted also in Fig. 4) of the stepping of a simulation using the remote link.

The synchronous link between client and external application is bidirectional. By this, it means that in every remote simulation step, all phases of Table 1 are executed. Hence, information flows constantly in both directions. Fig. 5a shows a chronograph of the synchronous link in action. Note that, in the remote execution of a synchronous link, possible delays may be caused not only by the network, but also when the client or external application are computing.

![Fig. 4. Phases of a remote engineering simulation.](image-url)
Table 1. Remote stepping operation.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Client side gets the values of the client variables.</td>
</tr>
<tr>
<td>2</td>
<td>Client side sends to server side the values obtained.</td>
</tr>
<tr>
<td>3</td>
<td>Server side sets the external variables.</td>
</tr>
<tr>
<td>4</td>
<td>External application runs the simulation once.</td>
</tr>
<tr>
<td>5</td>
<td>Server side gets the values of the external variables.</td>
</tr>
<tr>
<td>6</td>
<td>Server side sends these values back to the client side.</td>
</tr>
<tr>
<td>7</td>
<td>Client side sets the corresponding client variables.</td>
</tr>
<tr>
<td>8</td>
<td>Client application updates the user interface.</td>
</tr>
</tbody>
</table>

Despite the problem of an increased delay, the synchronous link presents some advantages, since it is similar to the local connection between client and external applications. It actually looks the same from the author’s point of view. This means that a simulation created originally to work using the local link can be easily transformed into a simulation that uses the synchronous remote link. The single change required to transform the original simulation into a remote one is to indicate the network location of the computer where the engineering software is running.

To reduce these execution delays, the asynchronous link intended does not keep a continuous synchronization between both applications. This carelessness is based on the key idea that the end user interacts with the simulation sporadically, spending most of the time precisely observing the system response after any modification in the parameters that govern the system.

Based on this idea, the communication between the applications is not bidirectional as in the case of the synchronous link, but unidirectional for most remote steps, showing bidirectionality only whenever the end user interacts with the user interface of the simulation.

Hence, most of the time, the external application will just be continuously running the engineering simulation, sending back the values of external variables to the client application. The client will spend most of its time just receiving and displaying those values. This situation will be interrupted only if and when the end user introduces a change in the values of the client variables that has to be reported to the external application.

Fig. 5. Remote links in action, a) Synchronous b) Asynchronous.

The synchronize method of the ExternalApp interface must be used then to inform the external application that a user interaction has taken place. The method will accomplish phases 1 and 2 of Table 1 updating the external application with all the values of linked client’s variables. Fig. 5b shows a chronograph of the asynchronous link in action.

A full documented Java classes, with examples of the use, that implements the ExternalApp interface for MATLAB/Simulink with support of local and remote links can be found at http://lab.dia.uned.es/rmatlab/.

6. CONCLUSIONS

This article introduces the interoperable approach, which aims to split the development of an interactive engineering simulation for pedagogical purposes in two separate tasks. On the one hand, the interactive human interface is created by using high-level languages or specialized authoring tools. On the other hand, the engineering simulation is built by using some standard engineering software. A necessary API of the protocol to communicate both parts of the interactive simulation is described in detail.

Advanced programming users can create Java classes that implements the API for supporting their preferred engineering software. Instructors can find some examples and full documented Java classes that manipulate local and remotely MATLAB and Simulink simulations at http://lab.dia.uned.es/rmatlab/.

REFERENCES