

**INFORMATION SOCIETY TECHNOLOGIES  
(IST)  
PROGRAMME**



*SmartSketches* 

***Demonstrator I (upholstery) - Technical Report***

Project acronym: **SmartSketches**

Project full title: **SmartSketches: A Multimodal Approach to Improve Usability in the Early States of Product Design**

Contract no.: **IST-2000-28169**

DELIVERABLE: D26a

VERSION: 1.1

AUTHORS: J. Calvário, B. Mendes (MIND)

PAGES: 13

KEYWORDS: Car Seat Upholstery Design, Technical Report, Integration

## **Executive Summary**

---

This technical report describes how integration between this demonstrator and SmartSketches framework and other partners specialized components was made. The demonstrator heavily relies on a MIND's product targeted at the car seat upholstery industry. The integration procedures and methods followed closely the specification described in deliverable 7. The scope of the integration was defined jointly by MIND and INESC-ID where common ground was acquired to demonstrate some of the functionality already developed inside INESC-ID prototypes. The work done in this scope includes making both components, either product and prototype sub-components, compliant to the framework to enable process collaboration scenario as stated in the SmartSketches System Framework deliverable.

# Table of Contents

<b>1. INTRODUCTION .....</b>	<b>5</b>
<b>2. SOLUTION OVERVIEW.....</b>	<b>6</b>
<b>3. SMARTSKETCHES RESULTS INTEGRATED IN THE PROTOTYPE.....</b>	<b>7</b>
3.1 BASIC PRIMITIVE CREATION .....	7
3.2 EDITING INTERACTIONS.....	8
<b>4. INTEGRATION TECHNICAL DETAILS .....</b>	<b>9</b>
4.1 GENERAL TOPOLOGY.....	9
4.2 MESSAGE FLOW.....	10
4.3 MESSAGE FORMATS.....	10
4.3.1 <i>Notification of strokes from the client application.....</i>	<i>11</i>
4.3.2 <i>Notification of recognitions from the CALI service .....</i>	<i>11</i>
4.3.3 <i>Notification of recognitions from the Stroke Recognition service .....</i>	<i>13</i>
<b>5. CONCLUSIONS.....</b>	<b>13</b>

## List of Figures

---

Figure 1 - activation of the sketching tool.....	7
Figure 2 - hand stroke of expected circle.....	7
Figure 3 - single circular shape being recognized and displayed to end user.....	8
Figure 4 - target primitive and stroke.....	9
Figure 5 - recognized gesture as delete operation .....	9

## 1. Introduction

Part of the process of making car seats involves drawing the seat design lines directly on a mould, made from a foam or rigid material. These lines will define the part that will be placed on a specific region of the seat. The set of those parts, connected or not, will form the intended design. After setting those parts as the design aspects of the seat, industrial operation like sewing margins and markers could be added as helpers for the assembly process. Because of the inevitable deformations to materials induced by the mould surfaces, an precise/empiric flattening process is needed that will take in account material stretch factor and orientation. Current mapping processes are manual and will depend deeply on operator experience. This mapping process is then used to calculate the 3D lines counterparts on the flattening space. Parts margins to enable consistent assembly are then added to the 2D flattening parts. The parts will then be sent to a cutting machine which will cut the patterns over selected materials, defined on the design stage.

As a motivation for targeting this industry, is the importance of having a precise digital mockup, before starting the pre-assembly procedures done solely to the parts taking in account the full pattern (i.e. where they stitch together). Having a correct mapping between 3D space and flattening space is critical. Sometimes, the only way to verify the correctness of the cut pattern is by assembling it. Any correction will require cutting of new parts, therefore wasting of materials that in some cases could be expensive.

As an overall objective for this application should allow the user to draw the lines directly on top of a digital 3D version of the seat mould. This will enable the user to see a 3D version of the final seat, before sending the resulting 2D lines to the next production step.

## 2. Solution overview

In order to enable a realistic preview of the final car seat, we use a 3D digital mold of the seat. The application then allows for line drawing over the mold surface, generating the final seat's surfaces from those lines. The user can manipulate those surfaces changing its colors, textures, making holes to the surface, adding stitch lines, and also making some deformations to simulate padded surfaces and other 3D deformations. It's also possible to add other type of 3D objects to the scene, like pipes and accessories, which can be modeled by any general 3D modeling application.

The rendering in "CarSeat Design" is done using industry standard OpenGL. The use of 3D visual effects like "Bump Mapping" and "Environment Mapping" allow a more realistic preview of the car seat.

To make the drawing possible on top of the seat mold, and later send the design lines to the cutting machine, we have to "flatten" the 3D lines to a 2D plane. The flattening is the most important part of all the process, and it uses state of the art algorithms to evaluate and manipulate the 3D surface and "map" it to a 2D plane. Exact operation like offsets, margins, distribution of features are always based on flattening space.

To facilitate the drawing of lines and quicken the operations and changes over the seat surfaces, we use a "stroke tool". This tool is integrated with the SmartSketches System Framework. The particular setup of this system to enable shape and gesture recognition involves two components, an helper service and a regular service component as defined in the "SmartSketches System Framework – API Specification" document that constitute the core work under WP2. With it the user can quickly draw, create, edit, delete and manipulate the objects that make the car seat.

### 3. SmartSketches results integrated in the prototype

After some discussion with INESC-ID on what results should be integrated in our prototype a common platform was acquired. Using some of the results displayed on the Gides++ prototype we decided to integrate the stroke recognition feature and expectation lists. Creation of primitives and gesture recognition will be available only on activation of a specially made sketching tool.

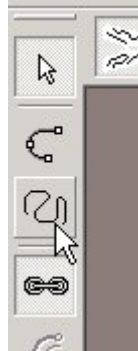


Figure 1 - activation of the sketching tool

#### 3.1 Basic primitive creation

The application will collect the stroke information and send a special formatted message to the recognizer helper service. Then it will expect a result within a certain time frame without locking user interactions.

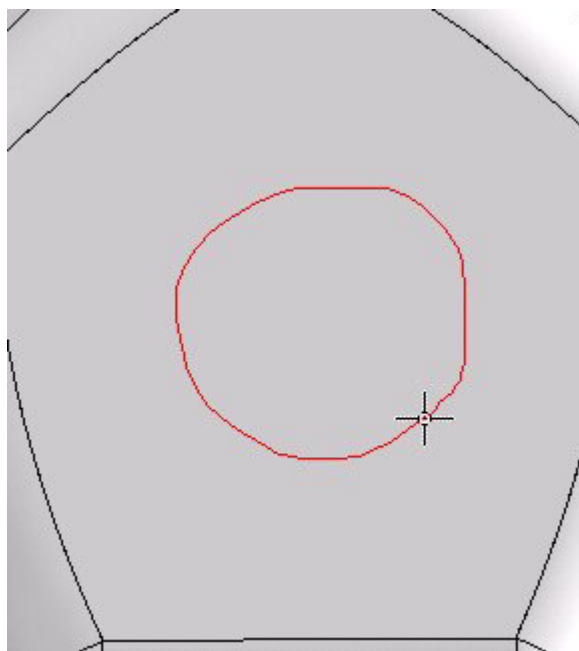
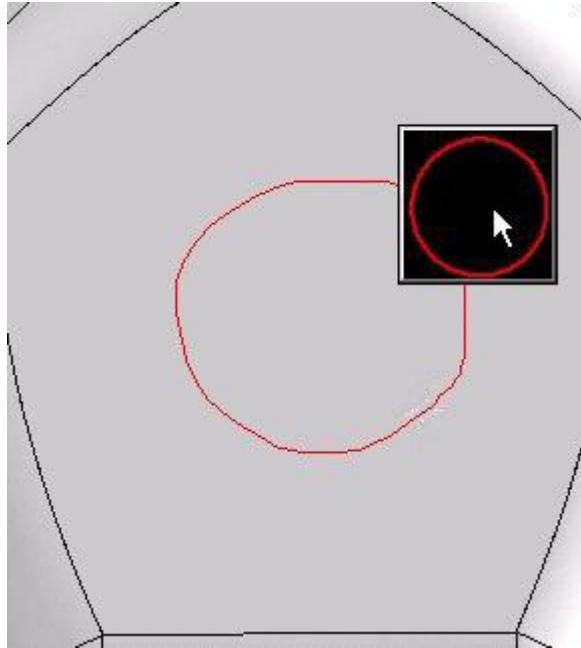


Figure 2 - hand stroke of expected circle

On arrival of a set of recognized shapes it will display the options to the user in the form of a expectation list. Then the user will select the option which best match his expectations.

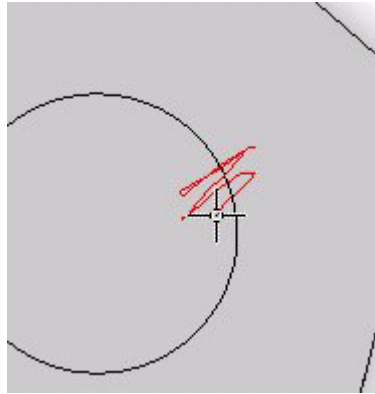


**Figure 3 - single circular shape being recognized and displayed to end user**

If no recognition set is received from the recognition service then it will disregard any further results related to the original stroke. This timeframe is needed to avoid long waits on an end user application basis. This will avoid pestering the user with no longer wanted recognitions.

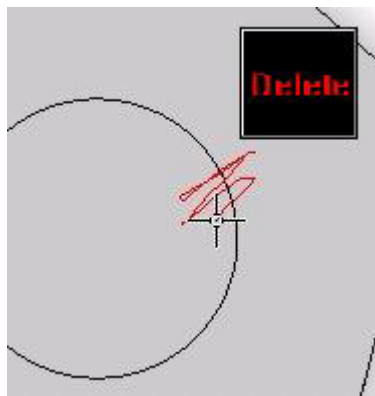
### ***3.2 Editing interactions***

The application will collect the stroke information and send a special formatted message to the recognizer helper service. Then it will expect a result within a certain time frame without locking user interactions.



**Figure 4 - target primitive and stroke**

On arrival of a recognized gesture it will display the options to the user in the form of an expectation list. In this case a single suggestion is displayed, the delete interaction. User will confirm or disregard by choosing one of the options displayed on the expectation list.



**Figure 5 - recognized gesture as delete operation**

## 4. Integration technical details

Integration

### 4.1 General topology

The application “CarSeat Upholstery Design” impersonate two major roles. As a subscriber consumer of framework events and as a input event provider. As a consumer in the sense that will require from the Stroke Recognition service the ability of proper stroke classification and identification to allow the end user to take appropriate action according to expectations. As an input event provider because it will capture strokes,

convert them to a context coordinate system and send them to the single threaded helper service specialized in recognition and classification. It will be referred from this point as the *application*. The helper service will be referred as *CALI service*. Following the best practices defined in D7, this service acts as an helper service to the full featured, subscription enabled, service that will expose the stroke recognition functionality to the rest of components resident in the framework system. The main logic specialized in stroke recognition is in fact implemented inside the *CALI service*.

The application is a subscriber of stroke recognition events with the *Stroke Recognition service*.

The application in this setup is responsible for the launching of the helper service. All the necessary subscriptions procedures and linkage between systems is preconfigured.

## 4.2 Message flow

Following the framework methodology, being the application responsible for capturing user interactions, it sends hand-drawn stroke events directly to the *CALI service* component for further processing.

The application will preprocess the stroke within a context before sending them for recognition. It will effectively project 2D strokes onto a context surface, convert the 3D stroke in the flattening coordinate space of the surface. Then it will send that stroke to the CALI component.

This call is asynchronous and no reply is expected besides successful delivery.

Being aware of those stroke events, *CALI service* will process them. If strokes were found that are recognizable, recognition events are generated. This component sends the recognition events to the *Stroke Recognition service* as notifications. This notification is done without losing context of the original stroke by an unique identifier.

The stroke recognition service will then process and forward appropriate events to all the subscribers, in this scenario, the application.

At that stage, because the process is asynchronous, it will be aware of triggered subscriptions related to the generated stroke by the original unique identifier. If in a case of successful recognition the application will receive information. This information will be used if filtering conditions based on stroke identifier and timeout constrains, could be met. In the affirmative condition an expectation list is build for user selection.

*Stroke Recognition* events, like Delete and Copy are cross referenced with the initial stroke for additional context information.

## 4.3 Message formats

It was necessary to define, under the rules of the initial specification to proper integration, the messages that are specialized in stroke and gesture recognition. The relevant formats were defined as XML snippets to be introduced on the already well defined envelopes of the specification. All the methods defined in the framework accepts

two parameters. The first named *origin* is the full identification of the sender and the second named *eventinfo* includes specialized information related to service functionality. Because the first parameter will remain constant for all the calls, we will omit it in the following sections. For reference, a sample of the content of the parameter *origin* is provided below:

```
<subscriber url="http://localhost:9000/ClientLocation.rem" remtype="2" />
```

The *subscriber* tag used in this case will apply to the Subscribe and Unsubscribe. An *origin* tag should be used for the remaining methods defined as entry point in the framework.

This node has two attributes, *url* with the location of the invoker and *remtype* with the type of remoting technology used by it. At the moment the type of remoting available are Web Service interface and .NET Remoting interface, both using SOAP envelopes.

#### 4.3.1 Notification of strokes from the client application

The application calls the notify method to report strokes to the CALI service. A sample of the content of the parameter *eventinfo* is defined below:

```
<eventinfo>
  <event>stroke</event>
  <parameter>
    <name>pointlist</name>
    <value>
      <point x="2.34" y="9.22" />
      <point x="2.33" y="8.11" />
      <point x="2.31" y="7.77" />
      <point x="2.27" y="6.38" />
      <point x="2.22" y="5.12" />
      <point x="2.14" y="4.92" />
      <point x="2.03" y="3.72" />
      <point x="1.94" y="2.66" />
    </value>
  </parameter>
</eventinfo>
```

The *event* tag defines the primitive being sent and *parameter* tag *pointlist* will provide the detail.

#### 4.3.2 Notification of recognitions from the CALI service

The *CALI service* application calls the notify method to report recognitions to the *Stroke Recogniton* service. A sample of the content of the parameter *eventinfo* is defined below.

```
<eventinfo>
  <event>recognition</event>
```

```
<parameter>
  <name>gesture</name>
  <value>
    <gesture>
      <gesture_param>
        <name>type</name>
        <value>shape</value>
      </gesture_param>
      <gesture_param>
        <name>name</name>
        <value>circle</value>
      </gesture_param>
      <gesture_param>
        <name>confidence</name>
        <value>.33</value>
      </gesture_param>
      <gesture_param>
        <name>centre</name>
        <value>
          <point x="-99" y="144" />
        </value>
      </gesture_param>
      <gesture_param>
        <name>radius</name>
        <value>5</value>
      </gesture_param>
    </gesture>
    <gesture>
      <gesture_param>
        <name>type</name>
        <value>shape</value>
      </gesture_param>
      <gesture_param>
        <name>name</name>
        <value>line</value>
      </gesture_param>
      <gesture_param>
        <name>confidence</name>
        <value>.99</value>
      </gesture_param>
      <gesture_param>
        <name>points</name>
        <value>
          <point x="-99" y="144" />
          <point x="-90" y="146" />
          <point x="-85" y="145" />
          <point x="-90" y="150" />
        </value>
      </gesture_param>
    </gesture>
  </value>
</parameter>
</eventinfo>
```

In this snippet two shapes were recognized with different confidence factors. A circle and a line that matches the original stroke. The *Stroke Recognition* service will forward this notify event to the subscribers.

### 4.3.3 Notification of recognitions from the Stroke Recognition service

No extra information is added at the current stage of development by the *Stroke Recognition* service to the original notify message coming from the *CALI service*. In the future subscription filtering could be done by simply implementing a filter sensitive to subscription parameters where for example, recognition events within a certain confidence level range could be excluded or accepted.

## 5. Conclusions

By using the SmartSketches System Framework, we were able to integrate functionality from other project partners working in an distinct platforms. We hope to seamlessly integrate sketching plugins and others developed by third parties. Those new interactions will simplify the learning and use of our application by the designers.