Interactive Data Units: A Framework to Support Rich Graphical Data Presentations on Heterogeneous Devices

Geert Houben Fabian Di Fiore Kris Luyten Frank Van Reeth Karin Coninx

Hasselt University
Expertise Centre for Digital Media
and
transnationale Universiteit Limburg
School of Information Technology
Wetenschapspark, 2
BE-3590 Diepenbeek (Belgium)

{geert.houben,fabian.difiore,kris.luyten,frank.vanreeth,karin.coninx}@uhasselt.be

Abstract

The use of mobile computing systems continues to increase among a wider diversity of end-users. On desktop computers, a lot of research in the area of interactive visualization of graphical information has been done, but there is a growing opportunity to have the possibility of creating scalable and animated graphical data presentations on heterogeneous mobile devices. Latest trends in the mobile phone and PDA community (e.g. games and MMS) emphasize the demand for a better support of such rich graphical data that is scalable over multiple platforms. In this paper we present mobile services for the automotive after sales market that support the user in coping with the increasing functionality and complexity of cars and their repair procedures. We introduce a generic framework to support the retrieval and visualization of operating instructions and car repair information according to the device, end-user (or consumer of the data) and car repair guidelines. As its core the framework provides the concept of Interactive Data Units (IDUs): graphical data blocks with a clean separation of structure, style and animation.

1 Introduction

At present, a vast amount of mobile devices with noticeable diversity in screen dimensions, resolutions and platforms are available as end-consumer devices on the market. The creation of a consistent presentation and visualization of a particular service on these heterogeneous systems is not a straightforward process. Scalability and platform independency are two major concerns when presenting such information.

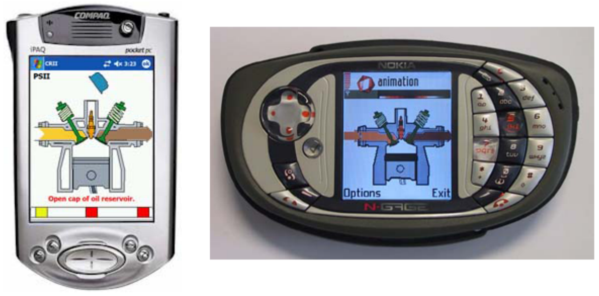

Figure 1. Use of Interactive Data Units on PDA and Nokia N-Gage

In this paper we present a technique to create scalable interactive visualizations on portable devices for the automotive after sales market. As part of the European MYCAREVENT (MCE) project, a service is created to support the user in a car repair process. Depending the role of the user, it can be the driver of the car, the road-side technician or even a worker in a service station, the service should present the information in an appropriate way. The service generates manufacturer specific operating instructions or car repair information according to the problems described by the user. Depending on the type of the

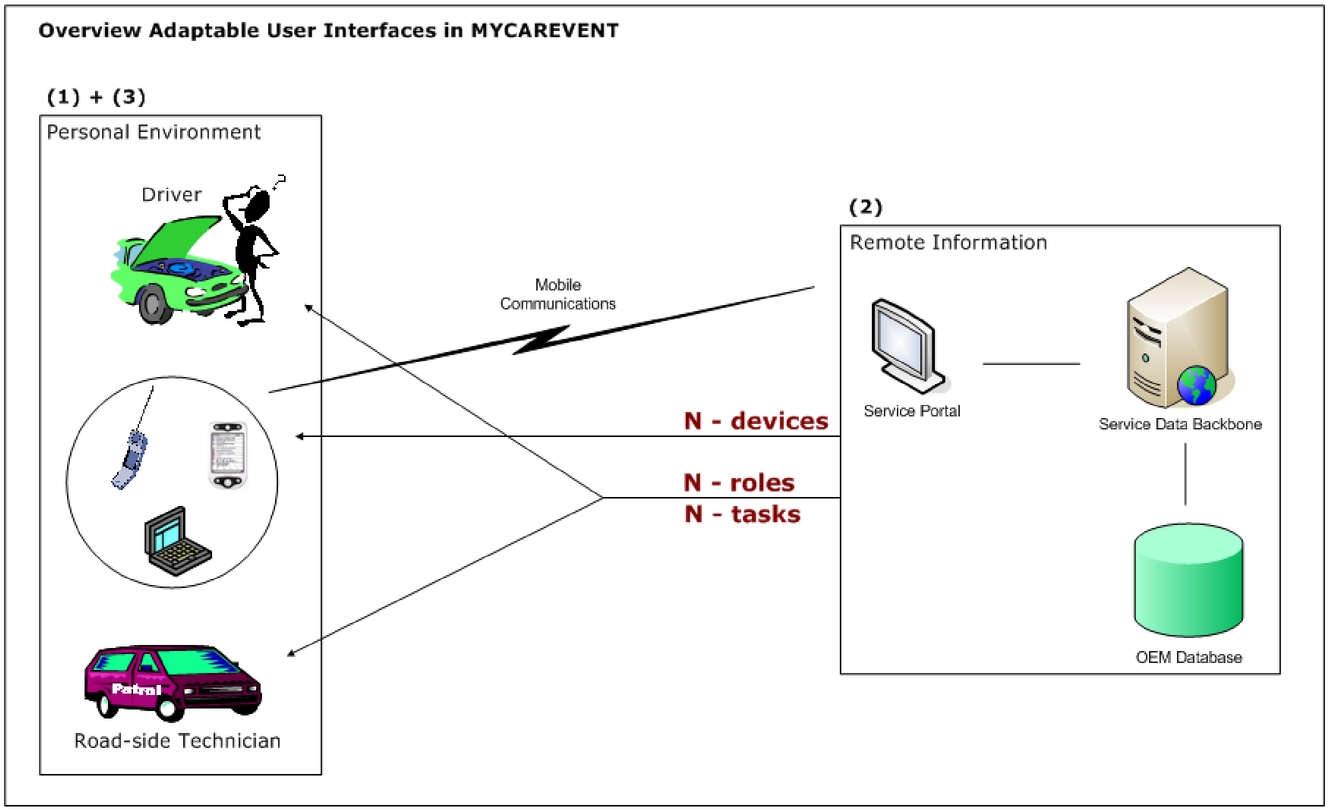

Figure 2. Overview of the scenario with different roles, devices and tasks

problem, the service can provide support for car drivers to help them to operate car functions or support for a road side technician to provide detailed repair information in case of a breakdown problem.

Based on the user profile the repair information is visualized with an interactive visualization engine that allows to use different animations and styles for the same content. The basic information is provided by a remote service and loaded dynamically into the application. An appropriate visualization of this data is of great importance to complete succesfully the support process. To meet the requirements we created a framework that supports *Interactive Data Units* (IDUs): interactive rich graphical objects that can be animated and used on different platforms and devices. The user can manipulate an IDU just like a widget of a traditional widget set can be manipulated. A clear separation between structure and presentation is applied. Figure 1 shows the use of IDUs on a PDA as well as on a smartphone. The framework that will be presented in this paper can be used in all application domains where rich graphical presentations of data are needed. The MYCAREVENT scenario in this paper is just fictive and is not connected with any necessary implementation during the project.

The remainder of this paper is organized as follows: section 2 discusses related work in this field of research and describes differences with other approaches. Section 3 describes the problem and the scenario where are working in. In section 4 we elaborate on the properties and usage of IDUs to create a scalable interface for a remote service. Next, section 5 describes the runtime animation and rendering of the framework. Finally, we elaborate on a case implementation in section 6 and end with conclusions.

2 Related Work

Content can have a surplus value when using a rich graphical representation, it could be made more clear and attractive to work with. An example of animated interfaces on mobile devices is the implementation of Zoomable GUIs [2]. Zoomable GUIs offer some kind of animation to support a better information visualization. Animated user interfaces are easier to understand and more pleasant to use, e.g. cartoon-style interfaces have proven to be as comprehensible as traditional interfaces while being even more enjoyable in usage [3].

One widely used toolkit for the creation of animated user interfaces is Macromedia Flash [5]. Although it is a commonly adopted ‘standard’ for web-based applications, some

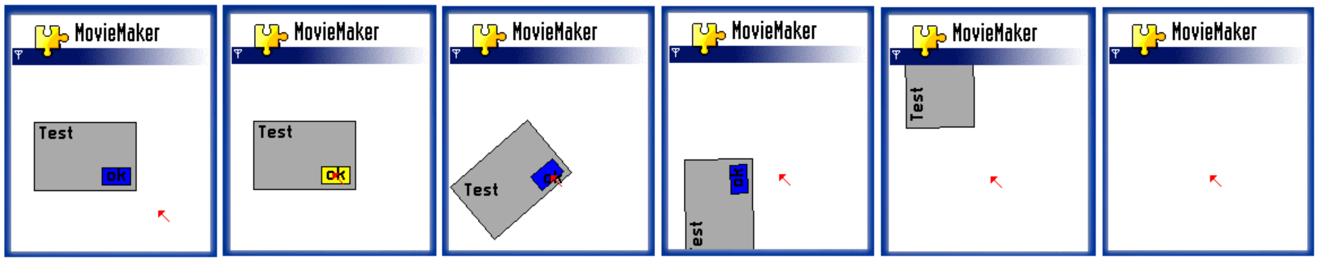

Figure 3. Example of an animation of an IDU triggered by a mouse event

main issues can be identified: there is no clear separation between content and interface, and it is not suited to write large programs due to its scripting language. However, in the recent releases of Flash this separation is one of the major points Macromedia has intended to overcome by loading external resources as XML-files, movie clips etc. Macromedia Flash is also available on mobile devices, but mentioned issues remain.

Another way to build graphical user interfaces is SVG (Scalable Vector Graphics [8]). SVG also allows the designer to animate objects, however, as its purpose is rather to represent graphical content, it is not satisfactory to build user interfaces pur sang. Furthermore, SVG uses, similar to Flash, scripting for defining animations (ECMA Script [1]) and has to cope with the same issues. Also transformation with tags is possible in SVG (e.g. ¡animateTransform¿), in our approach also such declarative animation is used.

Rendering and animation of the same data on diverse platforms is a very important contribution of our work. Several techniques for animation, scalable to other platforms, can be used, but in our work we focus on vector graphics because of low memory requirements, viewport independency and the support of basic drawing primitives. Furthermore, vector graphics can be described in XML and so can be interpreted on any platform.

3 Overview of the scenario

The core idea of the MCE project is the failure assistance of a user's vehicle. By setting up a car manufacturer database and a Service Portal to communicate with the user, failure related information, based on the given input, is supplied to the driver when an operating problem occurs. This information is also called an *information bundle*. Also roadside technicians or workshops can use this remote "repair information" system to get more information about car failures or breakdowns.

In the described scenario three types of variances exist: there are different *roles* (driver, technician etc.), they work with different *devices* (PDA, GSM etc.) and have to perform different *tasks* (operations). An overview of this situation is given in Figure 2. As stated in the first paragraph of this section, different types of users are defined in this scenario. We will focus on two profiles: the driver of the car and the road-side technician. Both of them can receive a different information bundle selected by the MYCAREVENT portal, because they have a different level of knowledge. Legal issues will be taken into account. A driver is able to fill the oil reservoir or change a tire, but repair the spark plug of the engine is probably too complicated. Adaptivity of the provided information is needed.

Few years ago, most mobile phones and smartphones had a low resolution screen and were only able to provide textual information. Due to the increasing resources of mobile systems, rich graphical presentations on these small devices are possible. Also the increasing interaction possibilities make these devices suitable for showing interactive content and interfaces. Because in a lot of cases repair information is in fact just convenient if it is combined with visual information, these graphical capabilities are necessary [4].

When using raster data (bitmaps) to visualize such repair information, two major shortcomings are identified. First of all they are not scalable without losing quality. When deploying the application on different devices, scalability to other screen resolutions is very important. A second problem with bitmaps is the fact they are not interactive and animatable. In addition to the data representing the real drawing, some sort of metadata is necessary, which will give us a lot of advantages. We can define different parts in the drawing, make them interactive, animate them separately or give parts other visualizations.

To anticipate on the issues cited in this section, Interactive Data Units are introduced. We will elaborate on them in the next section.

4 Properties of Interactive Data Units

An Interactive Data Unit (IDU) is a platform-independent graphical object consisting of a separated *style*, *structure* and *animation* description (see Figure 4). Furthermore, they are *scalable*, *animatable* and easily *adaptable*. The structure description of the IDU can be considered as a high-level specification (hierarchy), the style as low-level (appearance). In this section we will focus on its properties and internals. Each part is described in a distinct XML file, so there is a clear separation between them. These files make up the format *hXMLa*, which is an abstract XML-compliant format describing the graphical objects, making them independent of any platform. The different properties will be discussed more in detail in the following subsections. Figure 3 shows an example of an IDU, acting here as an interface element.

In the MYCAREVENT project, there is currently no process to transform standard repair information into IDUs. For this purpose more research is required and the feasibility of this approach can currently not be proved. This is one of the main problems when implementing such a system and it does not fall within the scope of the MYCAREVENT project.

Listing 1. Structure file of dialog example (fig. 3)

```
<structure id="st0" name="example">
 <object id="o0" name="box">
  <uitype type="dialogbox">
   <label>Test</label>
   <transformations>
    <translation x="20" y="50"/>
    <skinning skinref="s0"/>
   </transformations>
  </uitype>
  <children>
   <object id="o1" name="button">
    <uitype type="button">
     <label>ok</label>
     <event type="onclick" animation="a0"
       action="exit"/>
     <event type="onmouseover" animation="a1"/>
     <event type="onmouseout" animation="a2"/>
     <transformations>
      <translation x="60" y="40"/>
      <skinning skinref="s1"/>
     </transformations>
    </uitype>
   </object>
  </children>
 </object>
</structure>
```

4.1 Structure

The structure part of an IDU contains the high-level information, for instance the events and animations linked with the unit, as well as the name of the object and the corresponding skins (styles). The individual structure parts of the IDUs are bundled in a general structure file, in which also the hierarchy of the IDUs is represented. Transformations of a unit are recursively applied to its children. Listing 1 contains the structure file of the dialogbox described in this paragraph.

The structure file provides information which has nothing to do with the graphical presentation. There is a clear separation between functionality and style (visualization). This implicates that it is possible to choose a specific visualization or animation according to a certain context or role. A road-side technician gets a more detailed presentation of an object than a driver gets, or a visualization where other parts are emphasized.

Listing 2. Animation file of dialog example (fig. 3)

```
<animation id="a0" name="boxfly" dur="4"
loop="1">
 <frame id="f0" timing="0">
  <object idref="o0">
   <transformations>
    <translation x="20" y="50"/>
    <rotation angle="0"/>
    <scaling x="1" y="1"/>
    <skinning skinref="s0"/>
   </transformations>
  </object>
 </frame>
 <frame id="f1" timing="50">
  <object idref="o0">
   <transformations>
    <translation x="0" y="100"/>
    <rotation angle="-90"/>
    <scaling x="1" y="1"/>
    <skinning skinref="s0"/>
   </transformations>
  </object>
 </frame>
 <frame id="f2" timing="100">
  <object idref="o0">
   <transformations>
    <translation x="0" y="-100"/>
    <rotation angle="-90"/>
    <scaling x="1" y="1"/>
    <skinning skinref="s0"/>
   </transformations>
  </object>
 </frame>
</animation>
```

The events in Listing 1 have their own functionality. The first event closes the program after clicking the button (when the animation with id = a0 ends, i.e. box moves out of the screen). The second event changes the color of the button when the mouse cursor hovers about it (animation with id = a1). Event number three has the reverse effect (change the color to the initial color when the mouse leaves the button, animation with id = a2). The `transformation` tag within the `uitype` tag represents the initial situation of the object in the application.

4.2 Animation

As an example, take the scenario where the driver has to fill the oil reservoir. The user has to open the hood, remove the cap of the reservoir and fill it with oil. These steps, represented in Figure 9, have to be visualized in the user interface, and they are more clear when present them with animated graphics. The interface shows a user how to *do* something. When using animations properly, it is easier to understand the visualized information.

Also the animation part is separated from the structure and style part of an object. For example we can change an animation without changing the appearance. The animation file contains the information about the animations of the abstract objects defined in the structure file. Objects can be animated in two different manners: transformation and warping. These affine transformations in the framework include translation, rotation, scaling, skewing and shearing.

The animations are described with keyframes. For each keyframe the transformation of objects is specified. These objects match the objects defined in the structure file. Listing 2 gives an example of an animation file belonging to the dialog example.

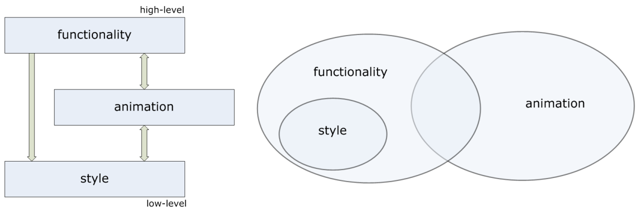

Figure 4. Relations between the properties of an IDU

4.3 Style

In the skin file the visible characteristics of an IDU are defined. It contains the primitives (in our case vector graphics) that determine the appearance of the object. The syntax of the description is based on SVG primitives to guarantee consistency with other SVG viewers and editors. Listing 3 gives an example of a skin file. This skin file belongs to the dialogbox example and represents the box. The primitives in Figure 5 can be used in the style description. Also properties of primitives can be adjusted: color, linewidth, fill etc.

Listing 3. Skin file of dialog example (fig. 3)

```
<skin id="s0" name="box">
 <svg width="90" height="60">
  <polygon fill="gray" stroke="black"
   points="0,0 0,60 90,60 90,0"/>
  <text x="5" y="15" font-color="red">
   &uitype;</text>
 </svg>
</skin>
```

The vector graphics we use in the framework are described in XML and consequently platform-independent. They are scalable and animatable due to the storage of the coordinate points making up the primitives. There are many applications suitable for drawing vector-based images. Even traditional drawing programs have facilities to export to SVG. This enables designers to create their own object layouts using their favorite external program (e.g. Adobe Illustrator), and to use them as skins for units in the framework.

5 Runtime Animation and Multi-Device Rendering of the IDUs

In this section we will discuss the runtime animation and multi-device rendering of created IDUs.

5.1 Runtime Animation

As we mentioned earlier, the data presentation is represented as a skeleton (hierarchy) of IDUs. An example of a specific interface is given in Figure 6. In a user interface every unit or widget has its own local coordinate system. The primitives making up the skin or style are defined related to this system. Every transformation is relative to the transformation of its parent. The screen of the target device is considered as the global coordinate system.

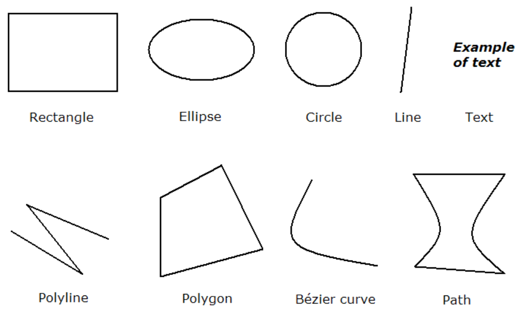

Figure 5. Primitives forming the basic layer of the framework

Forward Kinematics are used to calculate the global transformation of a child. The obtained matrix is multiplied with each point of the vector primitive to derive its global coordinates. Whenever an object has different skins in the keyframes of an animation, interpolation is applied between these skins.

The final presentation can hold several animations in which every animation consists of frames and keyframes. The latter are defined by the user, whereas the first are generated on-the-fly using interpolation between the keyframes [7]. Frames can be considered as particular configurations of object transformations. Figure 7 gives an overview of the IDU animation part of the framework. A `transform object`, as shown in Figure 7, is a transformation of the corresponding object in a specified frame.

5.2 Multi-Device Rendering Engine

The transferability to other platforms is handled by providing a basic, thin layer as the foundation of the framework. This layer is a set of drawing primitives and is ‘thin’ because we only have to use drawing functions of the target platform for these specific primitives. Hence, this means that every platform with support for 2D drawing is able to render the application. As a result we could also expand to other devices, or even to the Web.

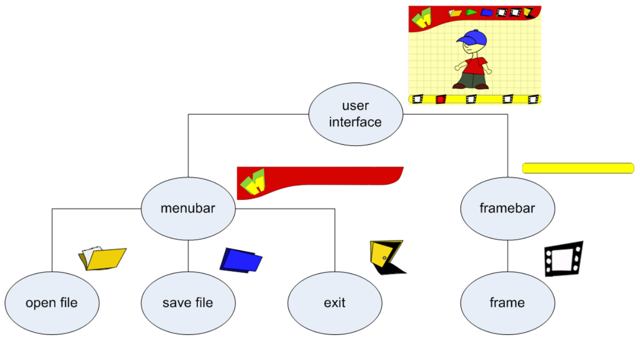

Figure 6. Hierarchy of a specific user interface

The thin layer consists of six functions: *DrawLine()*, *DrawPolygon()*, *DrawPolyLine()*, *DrawRect()*, *DrawEllipse()* and *DrawText()*. Since it is not possible to draw curves on every (mobile) platform, we implemented a recursive subdivision algorithm to approach curves by straight lines [6]. Paths, filled curves or filled paths can then be considered as polygons. The primitives used in the style description are mapped in the framework to one of the six drawing functions.

These functions, bundled in one abstract class, are the only points of contact between the framework and the platform. By subclassing this abstract class, the framework can be easily transferred to other devices or platforms. Figure 8 shows the architectural overview. The `Graphical Context` class contains 2D API drawing functions for primitives. It is possible to take a `drawPixel()` method as the basis, but we use the drawing functions for primitives, otherwise we had to implement algorithms to draw the primitives using the `drawPixel()` method.

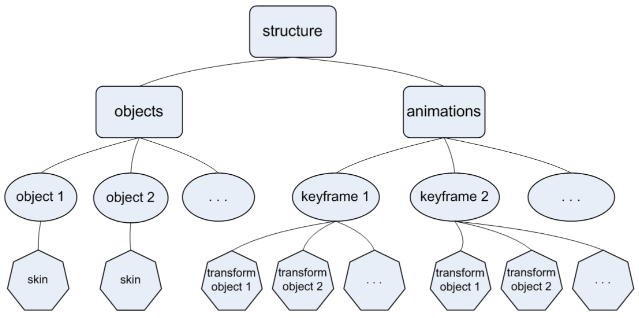

Figure 7. Overview of the object animation module

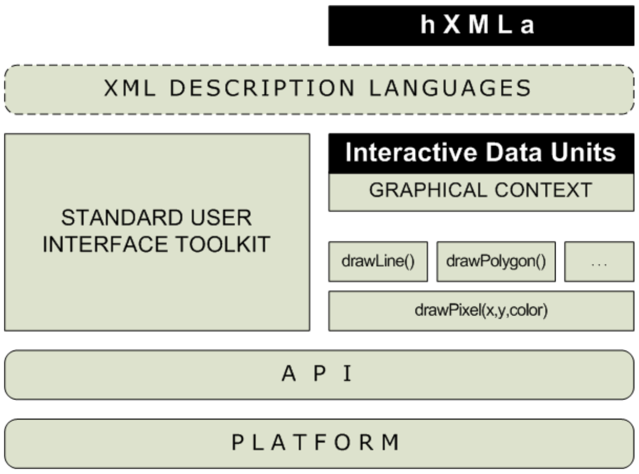

Figure 8. Architectural overview

Scaling of the data presentation is automatically done according to the target screen resolution without losing quality. Because of the limited computing power of mobile devices related to desktop PCs, it is not recommended to draw fancy and slow 3D graphics, but using fast and still attractive 2D graphics. After all, the user or artist can draw his widgets in his favorite drawing application, save the files as SVG and use them in the framework.

Regarding the animation it is also easy and time-saving to use vector graphics. Transformations only have to be applied to the points making up the primitive, contrary to the case of raster data (bitmaps) in which they have to be applied to all the pixels of the picture.

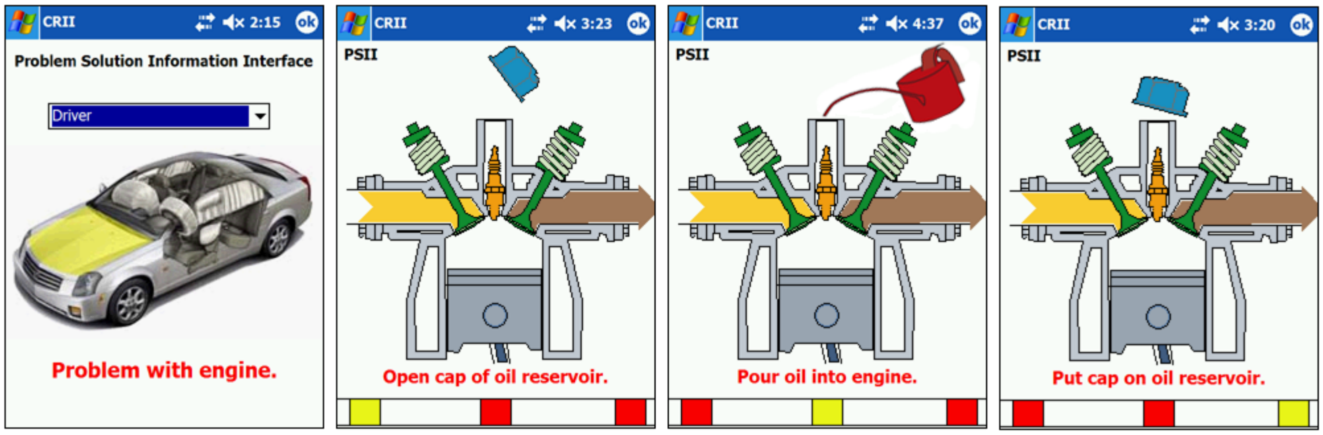

Figure 9. PSII application (role: driver, task: refill oil reservoir, device: PDA)

6 Case: PSII - Problem Solution Information Interface

In this section we elaborate on an implementation case; the *Problem Solution Information Interface* (PSII) that illustrates the use of IDUs. PSII is an application that helps the user to solve problems of her/his car. The graphical presentations are described in the hXMLa format. IDUs are used to guide the user through the repair process by providing interactive and animated schematical car information. Contrary to the use of straight-forward bitmap data, IDUs allow us to adapt the user interface according to context information.

The PSII application is used in different situations: on different devices, for different persons, and to complete different tasks. The interface of the application has to cope with these variances, also mentioned in Section 3: different roles, devices and tasks. When a car failure occurs, the corresponding repair instructions are provided by the MCE system as an information bundle via the Service Portal. To provide this information on different devices and for different types of persons, adaptation to the type of device and the profile of the user is necessary. IDUs are adaptable to different screen resolutions, and they are able to provide specific animations, interactions and appearances due to the user role.

The repair information consists of schematic drawings of repair instructions. These instructions are presented in a sequence of steps to lead the user through to process. The data is interactive and the specified action is emphasized by animating the corresponding object(s). The user can click on objects to trigger animations or to get extra information. Figure 9 gives an example of a simple process where the user has to fill the oil reservoir of the engine. Three steps have to be completed: open the cap of the reservoir, pour oil into it and put the cap back on the reservoir. By presenting this information with IDUs we can easily attach animations and different visualizations to parts of the engine. This step-by-step interface just exemplifies the use of IDUs.

The example in the previous paragraph is linked with the *driver* profile (as shown in the most left image of Figure 9). Figure 10 shows an example of the same content (engine), but used for another profile (*roadside technician*) to complete another task (change the spring of the left piston of the engine). We can use the same IDUs to visualize the process, only other animations (separated XML files) have to be applied to the objects. It is also easy to change the appearance of objects, e.g. the left spring in the second screenshot is visualized in another way compared to the last screenshot.

As stated in the second paragraph, three types of variances exist: there are different *roles*, different *devices* and different *tasks*. These issues are roughly covered by using IDUs.

Tasks

The tasks in the repair scenario depend on the role of the user, and are steps to reach a goal: to operate and to repair the car. To support these tasks, IDUs provide a rich graphical and interactive presentation of the different steps in the information process. Appropriate animations or visualizations can be selected specifically for a task. Tasks are closely connected with user roles. The task description itself is not included in the hXMLa language.

Roles

It is possible to link profiles with objects, events and animations in the structure file of the hXMLa language. The roadside technician for example needs probably other information than the driver needs. Due to a certain profile, an object can have another presentation, another event

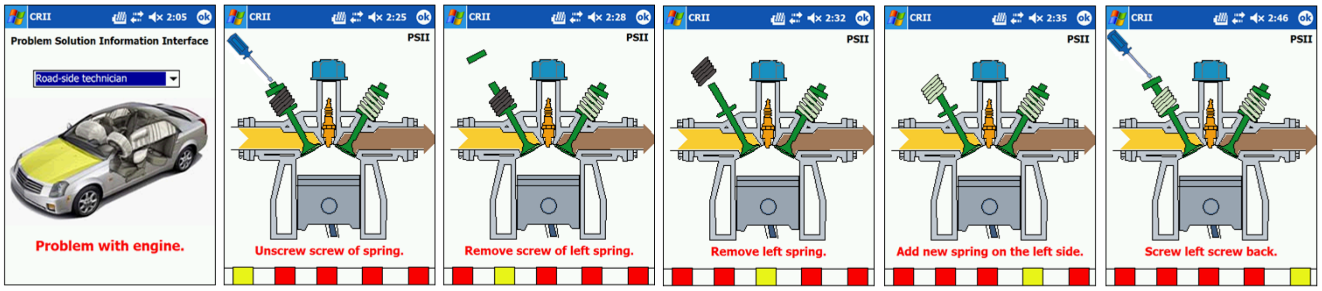

Figure 10. PSII application (role: roadside technician, task: change spring of piston, device: PDA)

can be linked with the object or another animation can be shown when interacting with the object. In the XML file we can link profiles as follows (profile 1 = driver, profile 2 = roadside technician, profile 3 = workstation worker):

```
<event type="onclick" animation="0"
       profiles="0,1"/>
```

In this example the **animation with id=0** is shown when **clicking** on the corresponding object in the **driver profile and the roadside technician profile**, but **not in the workstation worker profile**.

Devices

By using XML for the hXMLa description of the IDUs, platform-independency is guaranteed. The rendering engine of the framework only uses 2D drawing functionalities of the platform, so a thin layer forms the point of contact of the framework with the system where it is running on. The advantage of using vector graphics is the scalability without losing quality, so adaptation to other screen resolutions is no problem. The graphics are not yet being reduced automatically to device capabilities. Because a hierarchy of vector data skins is used, it is possible to represent less or more detail, according to the resources available on the device. Another advantage of using vector data is the possibility to change parameters on the fly. Figure 1 shows the PSII application running on a PDA (Pocket PC) and on a Symbian phone (Nokia N-Gage).

7 Conclusion

In this paper we presented a framework to support the creation of rich graphical data presentations. Although a lot of research in the field of interactive visualization of graphical information has been done, there is a growing opportunity to create scalable and animated graphical data presentations on heterogeneous mobile devices. We presented a scalable information visualization technique, based on the use of *Interactive Data Units*. These IDUs have to support the user when confronted with the complexity of car repair procedures.

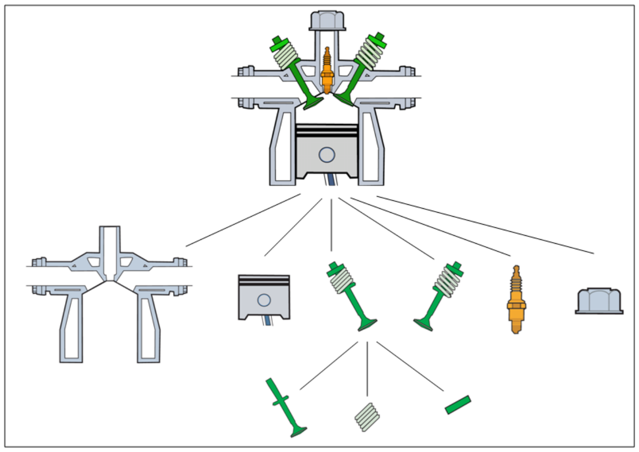

Figure 11. Hierarchy of objects of the engine IDUs

IDUs are scalable, animatable, well adaptable, and described in a platform-independent way. They consist of a separated style, structure and animation description. This separation has the advantage of giving an existing IDU a new appearance while preserving its functionality. The structure description of the IDU can be seen as a high-level specification the IDU (e.g. which animations or skins are linked with the object) whereas the skin or style description is rather low-level emphasizing the *appearance* of the IDU (e.g. color of the object).

Information is provided by a remote service and can be loaded dynamically into the application. Depending on the role of the user, the device he/she uses and the tasks he/she has to complete, an appropriate presentation of this data is provided. The information is used to lead the user through the problem solution process, as well as to provide more information in a more complex car failure. To cope with the different variances in the scenario (tasks, roles and devices), adaptivity of the user interface is necessary.

In Section 6 we presented an implementation using IDUs for a car reparation scenario. A sequence of steps is provided to the user, which helps him/her in the repair process. By using IDUs, issues about adaptivity are covered. In contrast with using bitmap drawings to present the data, IDUs allow more freedom due to the separation of structure, animation and style of visuzalized information.

8 Acknowledgments

The research at the Expertise Centre for Digital Media (UHasselt) is partly funded by the Flemish government, EFRO (European Fund for Regional Development), tUL (transnationale Universiteit Limburg) and the Flemish Interdisciplinary Institute for Broadband Technology (IBBT). The MYCAREVENT[1] project FP6-IST No. 004402 is an Integrated Project sponsored by the European Commission in support of the Strategic Objective "Information Society Technologies (IST)" in the Sixth Framework Program. The authors would also like to thank Jan Van den Bergh for providing valuable ideas and Bjorn Geuns for his appreciated graphical input and feedback.

References

[1] ECMA script. World Wide Web, `http://www.ecma.ch`, 2002.

[2] B. Bederson, A. Clamage, M. Czerwinski, and G. Robertson. Datelens: A fisheye calendar interface for PDA's. In *Transactions on Computer-Human Interaction*, volume 11, pages 90 – 119. ACM, 2004.

[3] B.-W. Chang and D. Ungar. Animation: From cartoons to the user interface. In *ACM Symposium on User Interface Software and Technology*, pages 45–55, 1993.

[4] G. Houben, J. V. den Bergh, K. Luyten, and K. Coninx. Interactive Systems on the Road: Development of Vehicle User Interfaces for Failure Assistance. In *Proceedings of First Workshop on Wireless Vehicular Communications and Services for Breakdown Support and Car Maintenance (W-CarsCare)*, pages 84–89, Nicosia, Cyprus, April 2005. European Wireless 2005.

[5] Macromedia. Flash. World Wide Web, `http://www.macromedia.com/`.

[6] R. J. Oddy and P. J. Willis. Rendering NURB regions for 2D animation. In *Computer Graphics Forum*, number 11(3), pages C–35 – C–44 and C–456, September 1992. Eurographics 92 Conference, Cambridge.

[7] M. Owen and P. Willis. Modelling and interpolation cartoon characters. In *Proceedings of Computer Animation '94*. IEEE, May 25-28th 1994.

[8] W3C. Scalable vector graphics. World Wide Web, `http://www.w3.org/Graphics/SVG/`.

[1]`http://www.mycarevent.com`