applications mobiles pour Android et iOS

à partir d'un moule unique

Veikko Krypczyk (Allemagne)

De plus en plus d'applications (p. ex. autour de l'internet des objets) requièrent un accès par les appareils mobiles, *Android* et iOS. Diverses approches se disputent les faveurs du programmeur qui peut difficilement se soustraire à la demande multi-plate-forme et devra offrir une interface intuitive. L'environnement de développement *RAD Studio* – plus connu sous le nom de *Delphi* – semble bien approprié à ces exigences.

De plus en plus d'applications pour l'électronique reposent sur une interaction raffinée entre matériel et logiciel. Les applications pour appareils mobiles tels que tablettes et téléphones tactiles avec les systèmes d'exploitation *Android* et iOS sont courantes. Si la conception d'applications mobiles n'est pas votre pain quotidien, la création d'une application propre aux deux systèmes, adaptée parfaitement aux exigences du projet risque d'être un croûton sur lequel vous vous casserez les dents. Le démarrage ne sera pas sans surprises. Les approches en concurrence ont en commun des courbes d'apprentissage abruptes. Ainsi la programmation des petits appareils se change en grosse affaire, au point de devenir une pierre d'achoppement pour l'ensemble du projet. Un remède possible est une approche multi-plate-forme, résolument intuitive, de la conception de l'interface utilisateur.

Dans cet article, nous donnons un aperçu des possibilités de conception d'applications mobiles, avant d'examiner de plus près une approche intéressante. Avec *RAD Studio* – également connu par de nombreux praticiens sous le nom de *Delphi*, un outil efficace pour le développement d'applications de bureau - les applications pour *Android* et iOS peuvent désormais être créées à partir d'une base commune de code source, et d'un effort considérablement réduit. La programmation se fait avec *Delphi* (*Object Pascal*) et le support d'un outil graphique approprié. Un exemple montrera comment cela fonctionne.

Applications pour les appareils mobiles

Du point de vue de la technique, on peut distinguer les types applications, natives, web et hyrides.

Les applications natives sont créées exclusivement pour *Android* ou iOS. Vous utilisez directement l'API des systèmes. L'interface utilisateur s'intègre au mieux dans la plate-forme. L'absence de restriction dans l'accès à d'éventuels matériels spécifiques du téléphone ou de la tablette est un avantage : il est permis de s'adresser directement à tous les capteurs

de ces appareils. L'application est disponible dans les *app stores*. Si une application native est installée, elle peut, si cela a un sens, être exécutée sans accès à l'internet (hors ligne). Une synchronisation des données nécessaire peut avoir lieu automatiquement lors de la prochaine connexion en ligne. Les applications pour iOS (*Apple*) sont créées avec *Xcode* (environnement de développement) et *Swift* (langage de programmation). Pour *Android*, on utilise *Android Studio* et *Kotlin* ou *Java*.

Les applications web, en revanche, sont destinées aux appareils mobiles. Cela concerne principalement la conception de l'interface utilisateur. L'accès au matériel du système est limité. Certaines fonctions de base, comme le positionnement par GPS, sont toutefois possibles. Les API *JavaScript* correspondantes sont utilisées à cet effet. Les applications ne peuvent pas être proposées dans les *stores*. Elles fonctionnent sur un serveur et imposent donc la connexion permanente à l'internet. Une icône peut être installée sur l'écran d'accueil afin d'accélérer le démarrage de l'application. Le manque de capacité hors ligne peut être partiellement éliminé grâce à la technologie des applications web progressives (PWA). Une PWA met en symbiose un site web et une application. Par le biais d'un travailleur de service, une fonction de mise en cache peut être mise en œuvre. Ce travailleur de service est connecté entre serveur web et application sur l'appareil mobile. Pour créer une application web pour systèmes mobiles, toute la gamme des technologies web est disponible. En particulier, les bibliothèques et les cadres les plus courants peuvent être utilisés. En fin de compte, ces applications reposent sur les éléments HTML (structure), CSS (conception) et *JavaScript* (interaction, logique), puisque le navigateur ne peut qu'interpréter ces langages.

Les applications hybrides sont basées en interne sur les technologies du web. L'application web est exécutée dans un navigateur intégré, c'est-à-dire qu'elle fonctionne dans une sorte de «bac à sable» qui la rend indépendante de la plate-forme. Le système croit donc avoir une application native (le navigateur) devant lui. Il existe plusieurs approches pour les applications hybrides, p. ex. Cordova de l'*Apache Software Foundation*. Le principe est toujours similaire. L'application ouvre une fenêtre de navigateur en mode plein écran au démarrage, de sorte que le navigateur n'est pas identifiable en tant que tel et que l'adresse web ne peut pas être modifiée. L'application créée avec HTML, CSS et *JavaScript* est affichée dans le navigateur. Le cadre donne à l'application l'accès aux fonctions du système telles que caméra ou carnet d'adresses. Cela se fait à l'aide de greffons (*plug-ins*).

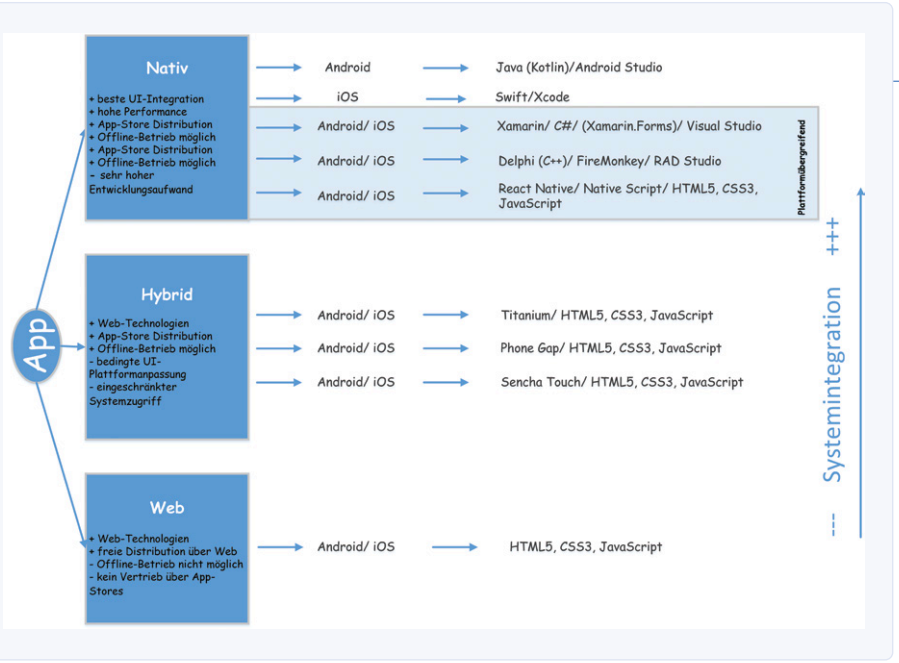

Figure 1. Types d'applications et moyens d'accès à l'application.

Les applications web et les applications hybrides ciblent *Android* et iOS simultanément. Cependant, elles présentent des inconvénients tels que des performances médiocres et un accès matériel limité par rapport aux applications natives. C'est pourquoi on essaie de relier les deux types de produits entre eux par des *approches multi-plates-formes.* L'objectif est de se rapprocher le plus possible du modèle natif (également pour l'aspect et pour la convivialité de l'interface utilisateur) et de créer l'application pour *Android* et iOS à partir d'une base de code source commune. Il existe un grand nombre d'approches multi-plates-formes, très différentes les unes des autres, notamment *Xamarin*, *RAD Studio*, *NativeScript* et *React Native* (**fig. 1**).

Dans cet article, nous allons examiner de plus près la procédure avec *RAD Studio (Delphi)*, bien adaptée, en particulier pour les projets électroniques et IdO. La procédure est familière à de nombreux praticiens. Avec l'aide d'un concepteur graphique, l'interface utilisateur peut être créée - comme pour le bureau de *Windows*. La logique est écrite dans le langage *Delphi* (*Object Pascal*), également intuitif et simple. Certains électroniciens connaîtront déjà *Pascal*, pour avoir p. ex. programmé des logiciels embarqués pour microcontrôleurs.

Installation et configuration du système

Nous installons maintenant *Delphi* sur *Windows* et le configurons pour le développement mobile. Téléchargez l'édition communautaire [1] après l'enregistrement et lancez l'installateur. P
endant l'installation, vous verrez une boîte de dialogue avec une option de sélection de la plate-forme (**fig. 2**).

RAD STUDIO, DELPHI ET C++-BUILDER

Delphi et *C++ Builder* sont les environnements intégrés d'*Embarcadero* pour la création d'applications pour différents systèmes d'exploitation. *Delphi* utilise le langage de programmation *Delphi* du même nom (une évolution d'Object Pascal), tandis que C++-*Builder* utilise le C++. Dans *RAD Studio*, les deux environnements sont combinés. Plusieurs éditions sont disponibles. Pour l'usage professionnel, il existe les produits commerciaux *Professionnel*, *Entreprise* et *Architecte*. Pour un usage commercial limité, pour tester, apprendre, pour des projets privés et open source, il existe l'édition communautaire gratuite. Nous l'utilisons également ici. La version actuelle est la 10.4 de *RAD Studio*.

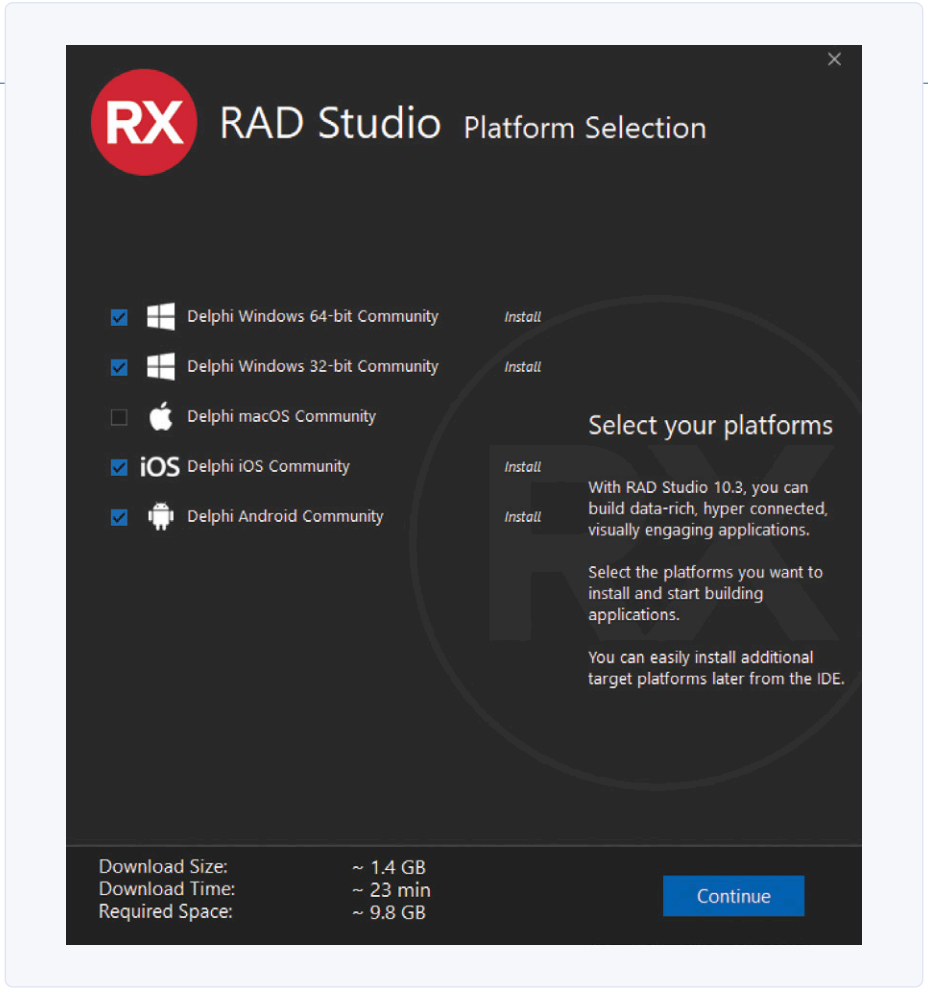

Figure 2. Sélection de la plate-forme.

En plus de *Windows*, sélectionnez *Android* et *iOS*. Ensuite, il vous sera demandé de sélectionner des options supplémentaires. Vous devriez faire cela pour le *SDK/NDK Android.* Après l'installation, ouvrez l'environnement de développement (**fig. 3**).

Pour le développement pour l'appareil mobile, des retouches sont encore nécessaires. Si vous voulez tester l'application sur iOS, vous devez avoir accès à un ordinateur *Apple* avec *Xcode* sur le réseau. Un simulateur pour iOS ne peut être exécuté que sous macOS. Même

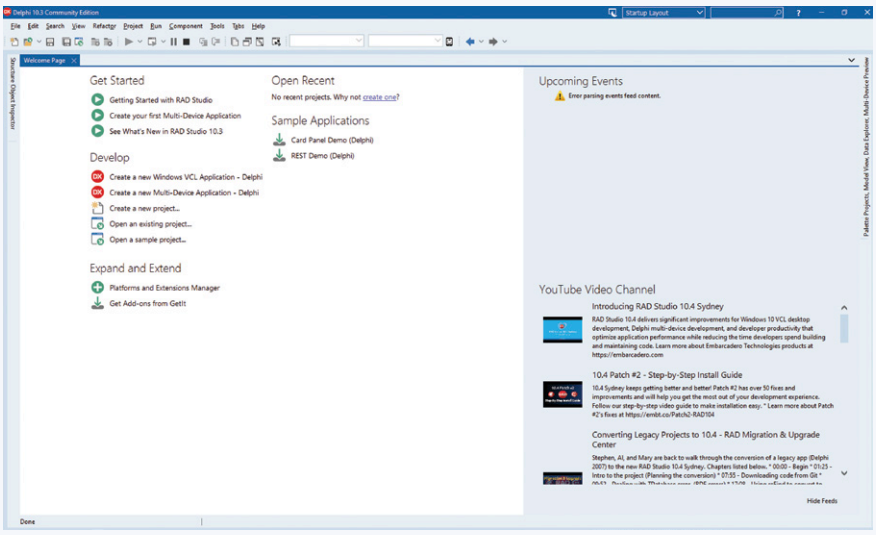

Figure 3. Écran de démarrage de l'environnement de développement intégré de Delphi.

un *iPhone* physique est directement couplé avec l'ordinateur Mac. Celui-ci est ensuite accessible sur le réseau à partir de *Windows* avec *RAD Studio* (**fig. 4**).

Pour ce faire, le logiciel *PAServer* doit être installé sur le Mac pour un accès à distance. Si vous travaillez sur un ordinateur Mac, vous pouvez exécuter *Windows* avec une machine virtuelle (*Parallels*, *Virtual Box*, etc.) et installer *RAD Studio* dans cette machine virtuelle à partir de laquelle on accède ensuite à l'ordinateur Mac, c'est-à-dire que tous les composants pour le développement sont installés sur un seul ordinateur. Le fait de devoir disposer d'un ordinateur Mac pour créer des applications pour iOS n'est pas propre à *RAD* Studio, mais s'applique à toutes les procédures.

Vous pouvez créer des applications pour *Android* directement à partir de *Windows*. Pour les tests, vous pouvez connecter un téléphone/tablette à votre ordinateur par l'USB ou (alternativement) installer un émulateur. Cependant, un «vrai» appareil est beaucoup plus rapide. Connectez votre téléphone à l'ordinateur avec le câble USB et configurez-le correctement, c'est-à-dire activez dans les paramètres les autorisations du développeur, y compris le débogage USB sur le téléphone. Vérifiez maintenant dans le gestionnaire de périphériques si votre appareil mobile a été correctement reconnu et configuré. La documentation officielle pour la mise en place de *Delphi* pour le développement mobile se trouve sous [3] et [4]. La configuration est terminée. Un premier test («*Hello World*») illustre la procédure de développement d'une application mobile.

« Hello Elektor »

Cette section concerne la création d'un premier projet. Dans l'environnement de développement *Delphi*, sélectionnez *File | New | Multi-Device Application - Delphi dans* le menu principal et cliquez sur *Blank Application* (**fig. 5**) dans la fenêtre, puis sur *OK*. Sauvegardez le projet. Grâce à la palette d'outils, nous pouvons créer l'interface utilisateur à l'aide de contrôles visuels. Pour ce faire, faites glisser un composant de *type TLabel dans* la boîte de dialogue pour un test. Vous pouvez changer l'aperçu pour les différents systèmes (*Windows*, *Android*, iOS).

Dans l'*inspecteur d'objets,* les propriétés des composants peuvent être configurées. Nous avons réglé la *propriété Text sur la* valeur

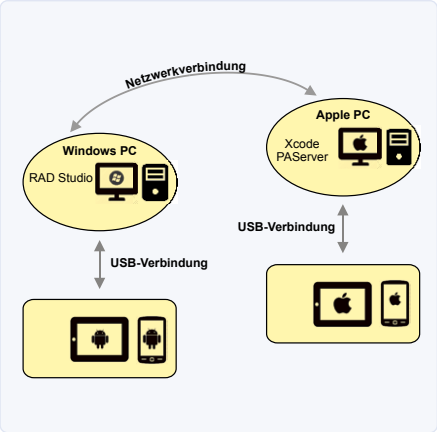

Figure 4. Configuration du système pour le développement d'applications pour iOS.

PAS D'IOS SANS MACOS ET XCODE

iOS est un système fermé, c'est-à-dire que pour créer les paquets d'applications et les distribuer aux appareils mobiles il faut impérativement Xcode (environnement de développement intégré) et un ordinateur Mac avec macOS. De même, le simulateur iOS ne peut être utilisé que sous macOS. Vous pouvez également utiliser un service de *cloud computing*, c'est-à-dire «louer» un ordinateur Mac hébergé avec tous les outils de développement nécessaires et le commander à travers une connexion à distance, comme le service MacInCloud [2]. L'écran est ensuite transféré sur l'ordinateur de développement propre. Cela résout le problème de sorte que vous pouvez créer le paquet d'applications et utiliser le simulateur iOS pendant la programmation. Dans le nuage, il n'est pas possible d'utiliser un appareil réel. L'auteur a fait de bonnes expériences avec le service mentionné. Le logiciel PAServer pour la commande à distance du Mac est préinstallé, vous pouvez donc démarrer immédiatement après l'enregistrement et la connexion. Pour commencer, il existe la formule flexible *Pay-as-You-Go :* vous ne payez que le temps d'utilisation du Mac dans le nuage.

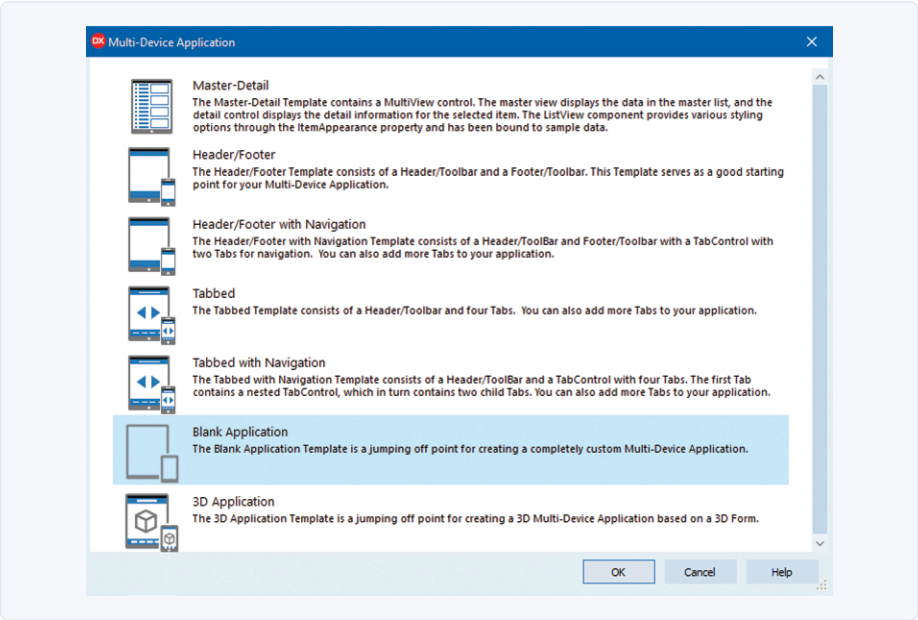

Figure 5. Création d'une nouvelle application multi-appareils.

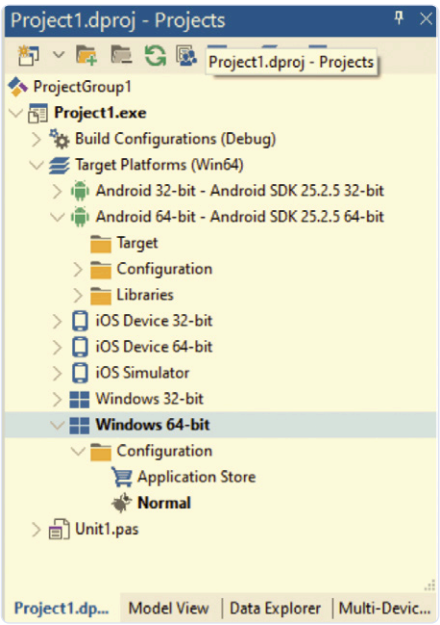

Figure 6. Les plates-formes cibles sont sélectionnées directement dans ***Delphi***

«*Hello Elektor*». Enregistrez le projet et les fichiers associés à l'aide du menu *Fichier*. Nous avons déjà un premier test sur les différentes plateformes. Ouvrez le nœud *Plates-formes cibles* dans le *gestionnaire de projet* et sélectionnez la plate-forme *Windows 32 bits* en double-cliquant dessus (**fig 6**). Démarrez l'application (flèche verte). Il fonctionnera comme une application *Windows* «normale» (**fig 7**).

Fermez maintenant l'application et retournez à l'environnement de développement. Sélectionnez *Android 64 bits* comme plate-forme cible. Si le téléphone/tablette a été correctement configuré ci-dessus, l'appareil connecté apparaît maintenant ici. Activez-le et redémarrez l'application. L'application est installée et exécutée sur l'appareil mobile. Vous pouvez également prévisualiser l'application pour un

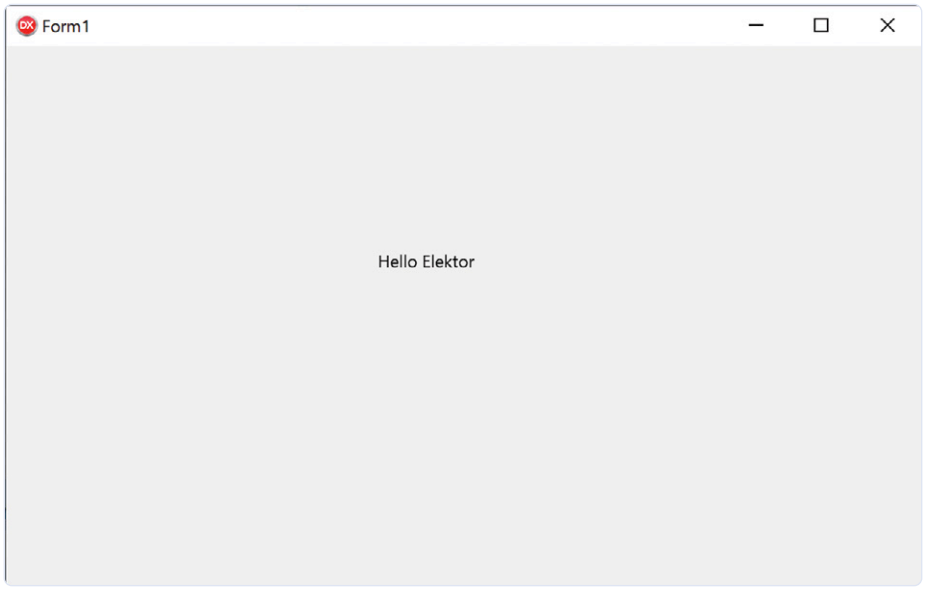

Figure 7. L'application de test en tant qu'application ***Windows****.*

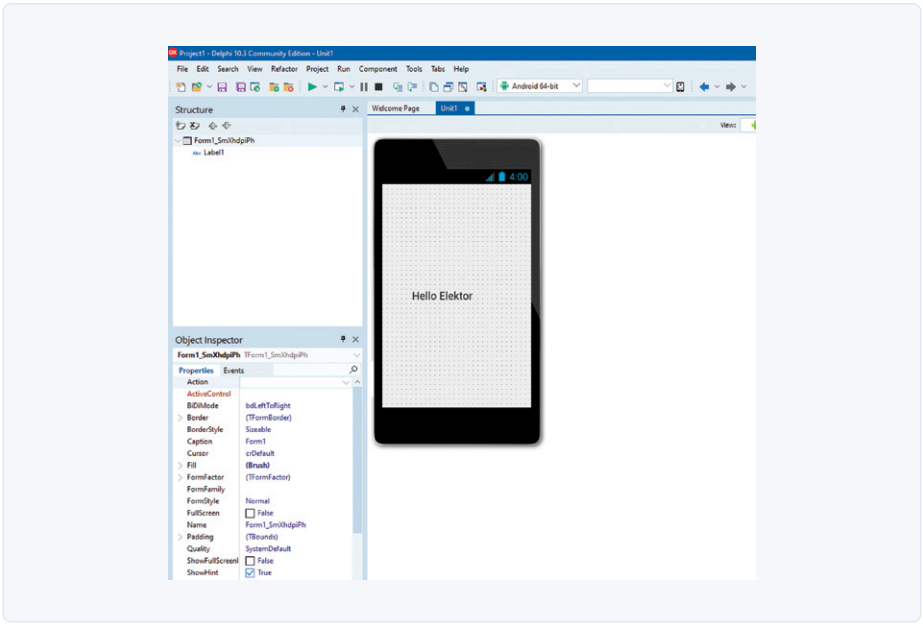

*Figure 8. L'aperçu de l'application pour **Android** dans le **RAD Studio Designer**.*

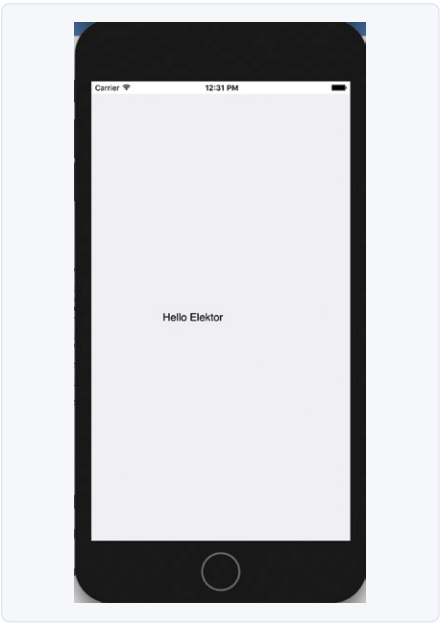

*Figure 9. «Hello Elektor» sur le simulateur de l'**iPhone**.*

appareil *Android* dans le *Designer* de l'environnement de développement (**fig. 8**).
Si vous avez également configuré le système pour iOS, vous pouvez vous connecter au Mac et, via le serveur PAS (voir ci-dessus), à *Xcode*, au simulateur et à un *iPhone* et y exécuter l'application (**fig. 9**). Après cette introduction succincte, nous voulons maintenant créer une application plus complète et nous familiariser davantage avec son fonctionnement.

Une application pour électroniciens

Dans cette section, nous décrivons la conception d'une application permettant de déterminer la résistance à partir des anneaux de couleur. Nous concevons d'abord l'interface dans l'outil graphique. Encore une fois, choisissez *Fichier* | *Nouveau* | *Application Multi-Dispositifs - Delphi* comme modèle et maintenant la sélection *Onglet*. On obtient une vue avec des onglets (dans *Android* en haut, dans iOS en bas). Nous devons maintenant remplir le contenu de ces onglets en conséquence. Le modèle contient quatre onglets (type *TTabItem*), que nous réduisons à deux, c'est-à-dire que nous supprimons deux onglets en utilisant la vue *Structure*. Dans l'onglet 1, nous sélectionnons les éléments du code couleur et dans l'onglet 2, nous plaçons les informations générales sur l'application. Il est judicieux de faire un croquis avant de mettre en place l'interface utilisateur et de préciser ainsi où les contrôles doivent être placés (du papier et un crayon suffisent). La disposition des éléments de contrôle ne se fait généralement pas avec des informations de position absolue, car il faut tenir compte du fait que nous lancerons l'application sur des appareils mobiles avec des tailles d'écran et des résolutions différentes. Une structure tabulaire peut être un bon point de départ pour une répartition approximative de l'espace. Les différents éléments sont ensuite disposés en lignes et en colonnes. Pour chaque ligne et colonne, nous pouvons à nouveau spécifier la taille (absolue, ou, mieux, relative). Pour notre application, nous avons choisi une grille de cinq lignes et quatre colonnes. Utilisez une commande du type *TGridPanelLayout*. Sélectionnez-le dans la palette et faites-le glisser jusqu'au premier onglet. Pour qu'il occupe tout l'espace, réglez le paramètre *Align* sur la valeur *Client*. En utilisant la *vue Structure,* ajoutez un total de cinq lignes et quatre colonnes à ce *TGridPanelLayout.* Vous pouvez diviser la taille des

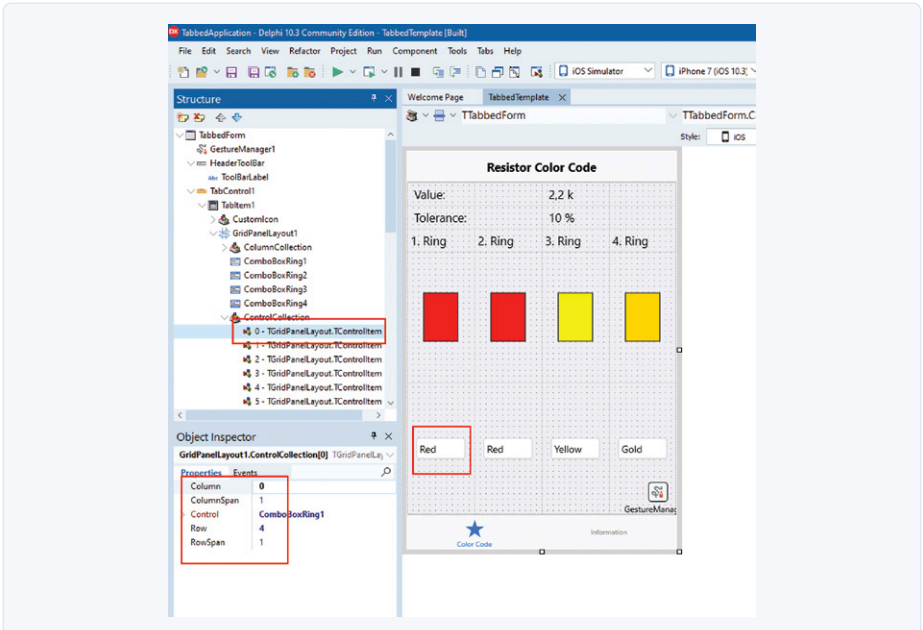

Figure 10. Le positionnement des éléments de contrôle se fait ici dans une grille de tableau.

couleur	anneau 1 position 1	index dans TComboBox	anneau 2 position 2	index dans TComboBox	anneau 3 multiplicateur	index dans TComboBox	anneau 4 tolérance	index dans TComboBox
noir	0	0	0	0	1	0	-	-
marron	1	1	1	1	10	1	1%	0
rouge	2	2	2	2	100	2	2%	1
orange	3	3	3	3	1 000	3	-	-
jaune	4	4	4	4	10 000	4	-	-
vert	5	5	5	5	100 000	5	0,5 %	2
bleu	6	6	6	6	1 000 0000	6	0,25 %	3
violet	7	7	7	7	10 000 000	7	0,1 %	4
gros	8	8	8	8	-	-	0,05 %	5
blanc	9	9	9	9	-	-	-	-
or					0,1	8	5 %	6
argent					0,01	9	10 %	7

Tableau 1 Signification des anneaux de couleur et position (index) dans la boîte de sélection respective (TComboBox).

lignes en pourcentage, par exemple, comme suit : 7 %, 7 %, 7 %, 40 % et 39 % (total = 100 %). Vous divisez les quatre colonnes en parts égales, c'est-à-dire avec une taille de 25 % chacune.

Il s'agit maintenant de placer les contrôles dans ce tableau. Sélectionnez un élément souhaité dans la palette et faites-le glisser vers l'onglet. Pour la sélection des couleurs, nous utilisons des contrôles de type *TComboBox*. Dans la vue *Structure,* nous pouvons alors définir la position exacte (ligne et colonne) de chaque élément sous le nœud *TabItem1 | Control Collection.* La **figure 10** illustre cette situation pour la sélection de la couleur du premier anneau.

Les positions *Column : 0* et *Row : 4* sont sélectionnées, puisque la case de sélection doit également être placée à cette position. Pour l'affichage du texte, nous utilisons des contrôles de type *TLabel,* par exemple pour les étiquettes ou les notes. Nous utilisons également *TLabel* pour les valeurs que nous voulons afficher plus tard, par exemple pour la valeur de résistance calculée. Nous simulons les anneaux colorés avec des rectangles colorés, c'est-à-dire des contrôles de type *TRectangle*. Nous voulons ajuster leur couleur plus tard à partir du code source. Dans les champs de sélection des anneaux de couleur (*TComboBox*), nous ajoutons les couleurs disponibles selon le modèle connu du **tableau 1.**

La mise en page de l'application dans l'outil graphique est illustrée à la **figure 11**. Après avoir lancé l'application (simulateur ou dispositif physique), nous voyons la disposition de l'onglet (**fig. 12**).
Nous avons initialement rempli l'onglet *Information* uniquement avec un logo (contrôle de type *TImage*) et un texte (**fig. 13**).
L'étape suivante consiste à mettre en place la logique de détermination de la valeur de résistance. Un nouveau calcul et une mise à jour de l'affichage doivent être effectués chaque fois que nous sélectionnons une nouvelle valeur de couleur. Pour faciliter l'identification des contrôles de l'interface dans le code source, nous donnons aux éléments des noms uniques. À cette fin, la propriété *Name de l'*élément est ajustée dans chaque cas, par exemple *LabelValue*, *RectangleRing1*, *ComboBoxRing1* etc.

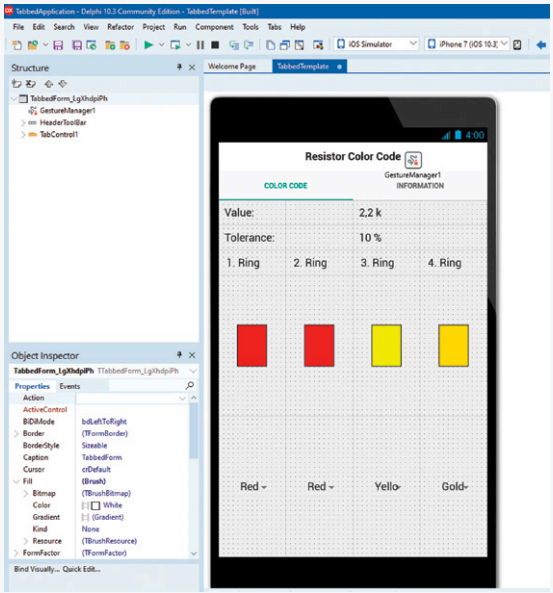

Figure 11. La présentation de l'application dans la vue Delphi - Android.

Liste 1. Affichage des couleurs selon la sélection.

```
case digit1 of
    0:
    begin
        RectangleRing1.Fill.Color := TAlphaColors.Black;
    end;
    1:
    begin
        RectangleRing1.Fill.Color := TAlphaColors.Brown;
    end;
    2:
    begin
        RectangleRing1.Fill.Color := TAlphaColors.Red;
    end;
    …
    …
    9:
    begin
        RectangleRing1.Fill.Color := TAlphaColors.White;
    end;
end;
```

Liste 2. Calcul de la valeur de résistance et formatage de l'affichage.

```
if digit3 < 8 then
    value := Power10((digit1 * 10 + digit2), digit3)
  else if digit3 = 8 then
    value := (digit1 * 10 + digit2) * 0.1
  else if digit3 = 9 then
    value := (digit1 * 10 + digit2) * 0.01;

result := FloatToStr(value) + ' Ohm';

if value > 1000 then
    result := FloatToStr((value / 1000)) + ' k Ohm';

if value > 1000000 then
    result := FloatToStr((value / 1000000)) + ' M Ohm';

LabelValue.Text := result;
```

Nous sélectionnons les cases de sélection pour les couleurs des anneaux tous ensemble et choisissons l'événement *OnChange*. Par un double-clic dans l'inspecteur d'objets, nous faisons en sorte que l'environnement de développement crée une procédure pour cet événement. Ce code source est donc appelé chaque fois que nous changeons la sélection de couleur.

Nous commençons par ajuster la couleur des rectangles en fonction de la sélection de l'anneau. Cela se fait, par exemple, par l'utilisation :

Anneau rectangulaire 1, couleur de remplissage : = Couleurs TAlpha, rouge

Le rectangle sera alors dessiné en rouge. Pour la première case de sélection, elle ressemblera à la **liste 1**.

Pour les autres anneaux de couleur, la sélection est ajustée en fonction de l'indice ci-dessus (voir **tableau 1).** Le calcul de la valeur de résistance suit, c'est-à-dire
(Valeur du chiffre 1 * 10 + valeur du chiffre 2) * 10^multiplicateur.

Le calcul est effectué en ohms. Pour les valeurs supérieures à 1.000, une conversion est effectuée en kΩ et pour les valeurs > 1.000.000 en MΩ. Pour les couleurs de bague or et argent, une multiplication avec les valeurs 0,1 et 0,01 est effectuée. Il ne manque que la valeur de la tolérance dans le tableau ci-dessus (**liste 2**).
Notre application est maintenant fonctionnelle : vous pouvez choisir n'importe quelle combinaison de couleurs des anneaux et la valeur correcte de la résistance est affichée. Vous pouvez maintenant créer les paquets d'application pour *Android* et/ou iOS et déployer l'application. Il est également possible de rendre l'application disponible dans *Google Play* ou l'*Apple Store* à partir de *Delphi*. Vous pouvez télécharger le code source de l'exemple de programme sur le site d'Elektor [5].

Autres possibilités

Dans l'exemple, nous avons déjà montré quelques possibilités de développement d'applications avec *Delphi*. Mais il y a bien plus que cela. Voici quelques exemples :

- ***Composants visuels*** : Pour créer des interfaces attrayantes, il existe divers composants (boutons, champs de texte, cases à cocher, onglets, etc.). Vous pouvez placer ces contrôles à l'aide du concepteur graphique et les configurer via les propriétés. Le système crée automatiquement les éléments appropriés pour la plate-forme cible (*Android*, iOS, *Windows*) lorsque vous compilez l'application.
- ***Composants non visuels*** : Ces derniers constituent des éléments de base importants pour la programmation, qui sont nécessaires à maintes reprises. Il s'agit p. ex. d'un composant de temporisation (*timer*) ou de composants destinés à faciliter l'accès aux bases de données.
- ***Spécificités des applications mobiles*** : des composants visuels et non visuels sont proposés, ce qui permet de résoudre plus rapidement les tâches typiques de la programmation d'applications mobiles, par exemple l'accès aux fonctions de suivi ou à la caméra du téléphone ou la gestion de l'échange de données avec les serveurs (*backends*) dans le nuage.

Ces possibilités de développement sont nécessaires successivement, si vous voulez créer une application mobile jusqu'à ce qu'elle soit prête pour une utilisation productive. Le développeur est également aidé à générer les paquets d'applications directement à partir de l'environnement de développement, qui sont nécessaires pour la distribution des applications par *Google* ou dans *l'Apple Store*.

Conclusion et perspectives

De telles applications « de poche » requièrent donc un travail qu'on aurait tort de sous-estimer. Grâce à un outil approprié, vous vous épargnerez bien des tracas et développer simultanément pour *Android* et iOS à partir d'un code source commun. *Delphi* offre une approche intuitive, qui devrait séduire de nombreux électroniciens familiers de *Visual Basic*, *Windows Forms*, etc.. L'interface utilisateur est conçue directement et assez rapidement dans l'éditeur graphique. Cette approche est également utile si vous devez écrire une application pour *Windows* ou *Linux*. De cette façon, vous pouvez réutiliser le code source plusieurs fois. En un clic, vous pouvez passer d'une plate-forme à l'autre. L'environnement de développement produit automatiquement les paquets d'applications nécessaires. Dans l'idéal, vous n'aurez rien d'autre à faire.

200265-02

Votre avis, s'il vous plaît...
Vous pouvez adresser vos commentaires ou vos questions à l'auteur (en anglais ou en allemand) : v.krypczyk@larinet.com.

Figure 12. L'application lancée - toujours sans fonction - dans le simulateur (iOS)

Figure 13. L'onglet Information.

Ont contribué à cet article :
Auteur : **Veikko Krypczyk** Rédacteur : **Jens Nickel** Traduction : **C. Zig** Maquette : **Giel Dols**

LIENS

[1] https://www.embarcadero.com/products/delphi/starter/free-download
[2] http://www.macincloud.com/
[3] http://docwiki.embarcadero.com/RADStudio/Rio/en/Android_Mobile_Application_Development
[4] http://docwiki.embarcadero.com/RADStudio/Rio/en/IOS_Mobile_Application_Development
[5] **La page de cet article sur le site d'Elektor** : www.elektormagazine.fr/200265-02