Japanese HWR

Steven B. Poggel
steven.poggel@gmail.com

January 27, 2010

Contents

Chapter 1

Technical Design of the Application

The focus of this chapter is on the general architectural choices made during the development of the system. In this chapter, the technical design aspects of the application are described. The general system architecture is layed out in section 1.1. It contains the global view on the software architecture in section 1.1.1, the data flow in within the system in section 1.1.2 and describes the design of the individual modules in section 1.1.3. Section 1.2 describes the technical set-up and framework choices. However, the handwriting recognition engine is described in detail in a separate section (see chapter **??**).

1.1 System Architecture

The system architecture of the Kanji Coach follows the requirements of an e-learning environment dealing with the specific difficulties for learners of the Japanese script (see chapter **??**) and those of an on-line handwriting recognition. Techniques of handwriting recognition are reviewed in chapter **??**. The general requirements of an e-learning application are presented in chapter **??**. The resulting specific conceptual design choices have been layed out in chapter **??**. This section deals with the technical aspects of the system design.

1.1.1 Global Architecture

The global architecture of the application follows the Model-View-Controller (MVC) design pattern. This paradigm is used as a general model, however, it is not designed the strict way proposed by (Krasner and Pope 1988). Figure (1.1) shows the general set-up of the MVC design pattern after (Krasner and Pope 1988). xxx: Also figure (1.2) - decide how it should be done and use the appropriate gfx. xxx! In the MVC paradigm the *model* is a domain-specific software, an implementation of the central structure of the system. It can be a simple integer, representing a counter or it could be a higly complex object structure, even a whole software module. The *view* represents anything graphical. It requests data from the model and displays the result. The *controller* is the interface between the model and the view. It controls and scheduls the interaction between the input devices, the model and the view (Krasner and Pope 1988).

A global overview of the system architecture can be seen in figure (1.3).

1.1.2 System Data Flow

The system data flow is shown in figure (1.4). The controller lies in the centre of the application, it runs on the stationary device. It contains a web service that is used as an interface to receive data from the handwriting input data view. The desktop view is the main interaction point for the user. The model contains the logic, while the data access layer provides a reusable interface for storing data. The details of figure (1.4) are described in subsequent sections 1.1.2.2 and 1.1.2.3.

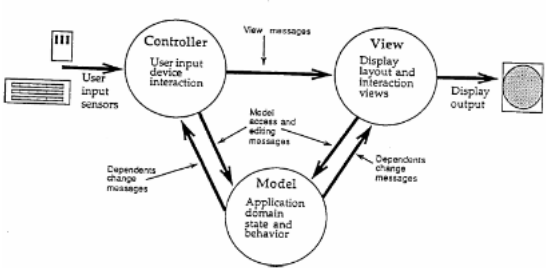

Figure 1.1: The Model-View-Controller paradigm

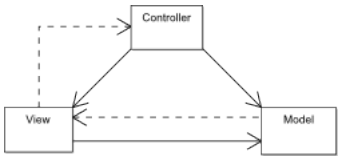

Figure 1.2: The Model-View-Controller paradigm AGAIN!

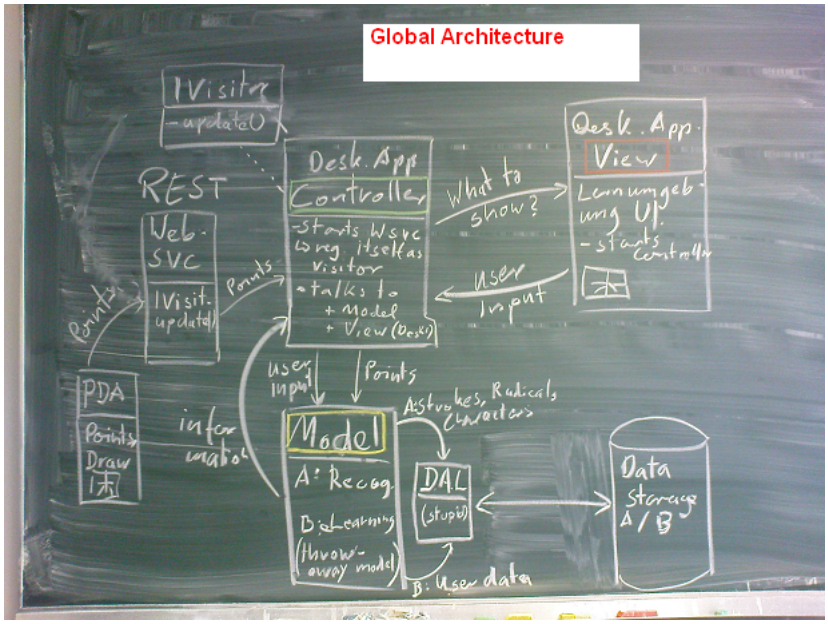

Figure 1.3: The global architecture of the software system

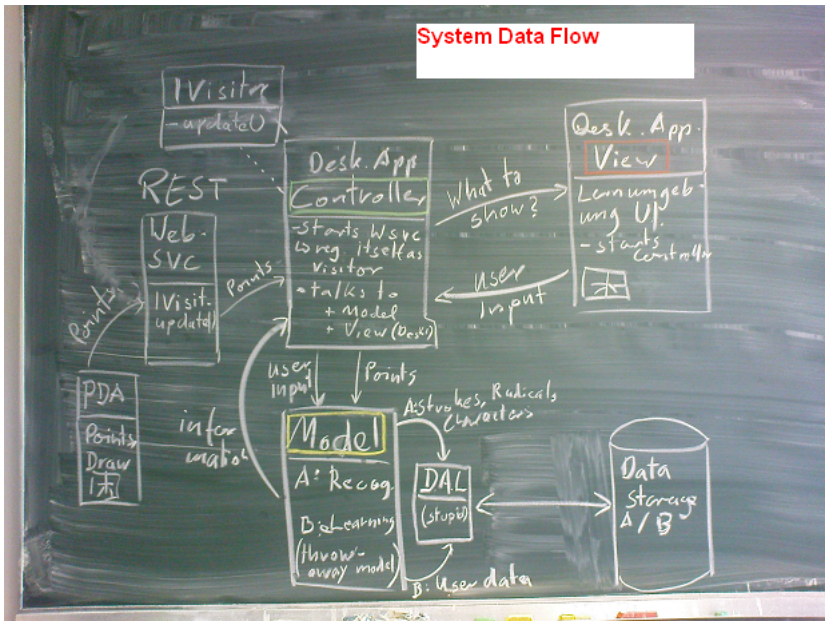

Figure 1.4: The data flow within the software system

1.1.2.1 Communication

The communication between the different parts of the application is realised in two independet and distinct ways. The communication between the modules running under the same process is realised via a messaging system. The communication between the modules that run on different devices or at least as a separate process is realised via a web service.

Loose coupling is used here as a term to emphasize that different modules in a larger system are only loosely connected and do not depend largely on each other. *Coupling* can be understood as the degree of knowledge a module or class have of each another. The lesser the knowledge of each other the software modules can manage with, the more loose the coupling.

In the course of the design and development cycles of the Kanji Coach, it became apparent that it has to be possible to attach different views, input devices and data storage systems to the controller. This is due to the distributed nature of the application. In order to ensure that the handwriting data input view, which currently runs on a mobile device, can run on a different device, it had to be loosly coupled to the main controller. Therefore, the communication between the handwriting input data view and the main controller is realised via a web service. Because of this communication structure, it is possible to exchange the handwriting data input view with a different one, for instance when running the application on a device like a tablet PC. The design of the communication structure within the web service is described in greater detail in section 1.1.3.3.

The messaging system that forms the communication structure between the software modules running as the same process is realised with a message class that can be manipulated by the modules using these messages. Technically, the web service and the controller both run as a subprocess of the main desktop view. When the web service receives a request and accompanying data from the handwriting input data view, both request and data are bundled into an encapsulated message and passed to the controller. A similar type of message is used for requests from the controller to the model, for instance a recognition task that needs be performed.

1.1.2.2 Recognition Data Flow

The recognition data flow is simply the flow of data occuring in a recognition scenario. It can be described in 6 steps as depicted in figure (1.4).

1. Clearing the input GUI screen [Controller ⇒ Web service ⇒ Handwriting data input view]
2. Transmitting user input data to the web service [Handwriting data input view ⇒ Web service]
3. Encapsulating data into a message and passing it to the controller [Web service ⇒ Controller]
4. Requesting recognition [Controller ⇒ Model]
5. Returning recognition result [Model ⇒ Controller]
6. Displaying recognition result [Controller ⇒ Desktop view]

When the desktop application requests the use to input a Kanji character, it sends a message to the web service advising the handwriting data input view to clear the screen (step 1). The handwriting data input view receives the information by polling the web service. When the user starts drawing on the surface of the handwriting data input view, the data is caputered and transmitted to the web service (step 2). Each stroke is captured and sent individually in order to ensure a faster recognition. That way, the recognition process is initiated before the user finishes writing a character. The web service receives the data and creates an encapsulated message. That message is passed to the controller (step 3). The controller initiates a request to the model, containing all strokes subsequent to the last clear screen event (step 4). The model performs the recognition of the set of strokes and returns the result or partial result to the controller (step 5). The controller advises the desktop view to display the resulting character (step 6).

1.1.2.3 Learning Data Flow

The learning data flow is the data flow between the learning module of the application, the recognition process and the user interaction. The learning data flow design can be summed up in 5 steps.

1. In learning mode or test mode: Asking user to draw a character, based on the current lesson data. The controller sends a display request to the desktop view. [Controller ⇒ Desktop view]
2. After recognition process: Request storing recognised character in learning profile. [Controller ⇒ Model]
3. Calculation of error points for a character (creation of new data) and storage. [Model ⇒ Data access layer]
4. Returning learning state of character [Model ⇒ Controller]
5. Displaying learning state [Controller ⇒ Desktop view

The learning data flow is described in the middle of a user interaction with the learning application as it illustrates a typical data flow most appropriately as shown in figure (1.4).

In step 1 of the learning data flow, the controller sends a message to the desktop view with a request to display an invitation to the user to draw a specific character. That step is a part of the general messaging system design that manages the communication between the controller and the other modules. When the recognition process is finished, the character needs to be stored. The controller requests the model to store the character (step 2). The logic layer then creates new data, namely the error points of a character that naturally become a part of the data flow. Both the character recognition result and the error points are stored using the data access layer (step 3). The resulting learning state defines which character will be displayed next. This calculation result is transmitted from the model to the controller after storage in the penultimate step (4). The last step in the learning data flow is the display of the resulting learning state to the user in the desktop view (step 5).

1.1.3 Software Modules

1.1.3.1 Handwriting Data Input View

The handwriting data input view is a graphical user interface. It is designed for simplicity and usability. It's main task is data capturing. It contains only an input area, with a cross in order for the user to better locate the strokes of the character. This follows a common practice in Kanji teaching - a cross departs the writing area in four areas.

There is no *commit* or *finish* button on the handwriting data input GUI, since the end of a stroke sequence is handled with a time out in the learning module. When the user needs too long to input a character there is a problem and help should be offered. Therefore, it is sufficient to only display a reset button on the writing area. Whenever the reset button is pressed, the controller is notified of that event. The current drawing will be removed from the screen. The next user input will be treated as a new character

In the background of the handwriting data input view the user input is sent to the controller module. Besides that, a polling mechanism ensures that any messages from the controller to the handwriting input data view are received. Whenever the user finishes a stroke, the module sends a message via a web service (see section 1.1.3.3) to the controller.

1.1.3.2 Main View

The main view of the system is a graphical user interface. It contains a simple learning environment, a display area for the characters that have been drawn in the handwriting input area an information area that displays addidional information about a character. In a configuration area the user can choose between the different lessons the system offers and between a training mode and a test mode.

1.1.3.3 Web Service

The main communication module between the handwriting data input view and the controller is a web service that runs on the main part of the application as a subprocess of the controller. When the desktop application requests the user to input a Kanji character, it sends a message to the web service advising the handwriting data input view to clear the screen. The handwriting data input view receives the information by polling the web service. However, the main task of the web service is to receive data from the handwriting input data view and forward that data to the controller of the system.

A web service is a standard means of interoperation between different software systems, possibly running on different platforms and frameworks. The web service as such is an abstract definition of an interface, it must be implemented by a concrete agent. This agent is a software that sends and receives messages (W3C Consortium 2004). The web service in the Kanji Coach system is not a web service in the strict sense as intended by the W3C Consortium (2004). It does not provide a service as such. The calling client does not receive a reply to a request, just an acknoledge message stating that the data has been received. A reply with more information is not necessary, since the handwriting data input view does not display any information to the user. In summary, the web service design for the Kanji Coach system is a web service in the technical sense, but not in the conceptual sense, as there is no real crosswise interaction between server and client. The concrete implementation of the web service is realised with the *Windows Communication Foundation* (WCF) (Smith 2007). The WCF uses a *SOAP protocol* (originally: *Simple Object Access Protocol*) internally, but the functionality can be accessed from within the .NET framework, the concrete SOAP XML messages will be created and unpacked into instances of Common Language Runtime (CLR) objects by the WCF framework (W3C Consortium 1997; Kennedy and Syme 2001; Smith 2007).

1.1.3.4 Recognition Module

1.1.3.5 Learning Module

1.2 Framework and Devices

1.2.1 Operating System

1.2.2 Framework

http://de.wikipedia.org/wiki/Windows$_C$ommunication$_F$oundation
.NET vs. Java etc.

1.2.3 Desktop Computer

1.2.4 Pen Input Device

warum mobil? - damit man sehen kann, was man macht!

1.2.4.1 Stylus Input

List of Figures

List of Tables

References

Kennedy, A. and D. Syme (2001). Design and implementation of generics for the .net common language runtime. In *PLDI '01: Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation*, New York, NY, USA, pp. 1--12. ACM.

Krasner, G. E. and S. T. Pope (1988). A cookbook for using the model-view controller user interface paradigm in smalltalk-80. *J. Object Oriented Program. 1*(3), 26--49.

Smith, J. (2007). *Inside Microsoft Windows Communication Foundation*. Redmond, USA: Microsoft Press.

W3C Consortium (1997). SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). Online. Retrieved from `http://www.w3.org/TR/2007/REC-soap12-part1-20070427/` on 2010-01-25.

W3C Consortium (2004). Web Services Architecture. Online. Retrieved from `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/` on 2010-01-25.