Evaluation of User Interface Transcoding Systems

Guido Menkhaus and Sebastian Fischmeister

Software Research Lab, Department of Computer Science
University of Salzburg, A-5020 Salzburg, Austria
{lastname}@SoftwareResearch.net

Abstract. Users access the Web through an increasingly diverse set of devices, such as mobile telephones, personal digital assistants, notebooks and tablet PCs. Services and content need to be accessible by anyone, anywhere, anytime, and anyhow. Device independent authoring is the goal of service providers. This paper presents an abstract model in form of a finite state machine (FSM). The FSM is the basis for the comparison of user interface transcoding systems (UITS). Based on this abstract model we identify the strengths and weaknesses of these approaches on an architectural level.

1 Introduction

The success of small gadget devices has lead to a diversification and multiplication of markup language [1]). Thus, a large amount of recent work in human computer interaction has been devoted to the establishment of principles for device independent user interface development and device independent content delivery; especially for the Web [2–4]. The explosion of the variety of devices for Internet access with widely different properties in UI capacity directed research on transcoding systems that focused on "1-to-n" relation and "n-to-n" relations between design time and presentation time description. The aim of these systems is to create a single description to serve a multitude of platforms or to make existing descriptions available to more than the originally intended class of devices.

The trend of systems using a single UI description languages targeting a single platform to a more extended interpretation has engendered a multitude of approaches to device independent UI description and device dependent user interface transcoding systems. Both type of systems fall in the category of UITSs. This paper proposes an abstract model for data conversion tailored to UITS. The abstract model tries to present a foundation for understanding, classifying and comparing UITSs.

The remainder of the paper is organized as follows: Section 2 discusses the context of this paper. A finite state machine (FSM) describing transcoding architectures is presented in Section 3. Section 4 evaluates systems of different transcoding configurations. A conclusion closes the paper after a discussion in Section 5.

2 Context and Motivation

The reference model for UI management systems that has been widely accepted is the Arch Slinky meta-model [5]. It proposes a structural and functional decomposition into the following five components: functional core, functional core adapter, dialog, logical interaction, and physical interaction component. UI management systems support two-way communication between the functional core and the physical interactor. Since transcoding architectures deal primarily with the communication interoperability from the functional core to the physical interactor, the discussion is focused on this aspect.

Considering the situation where there is a single class of physical interaction components and a single functional core i.e., a "1-to-1" relation, there is only need for a dialog component that controls task sequencing and rendering data for the unique physical interaction component. However, given the case of multiple physical interactors and a single functional core, i.e., an "n-to-1" relation, the dialog component becomes more complex.

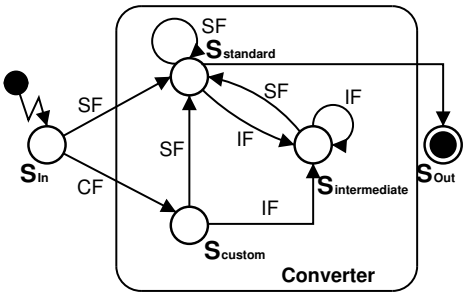
This situation is common in mobile computing: A growing number of networking enabled devices with different of UIs capabilities emerged on the market. Obviously, authors of Web-based applications cannot afford to develop content aiming at a single type of physical interactor. The objective is to create a single description to serve a multitude of platforms or to make existing descriptions available to more than the originally intended class of devices. The cardinal question that needs to be solved is: How to enable content to be converted into various physical interactor-dependent formats? It is obvious that the dialog component occupies a central role in content delivery to different devices. Thus, the dialog component becomes the application node that connects otherwise incompatible UI descriptions and physical interactors via transcoding. UI transcoding or conversion refers to the tasks of filtering and translating or converting content from one representation to another [2].

3 Abstract Model for UIT Systems

This section elaborates the architecture of transcoding systems using an FSM (Figure 1). Different transition paths through the FSM correspond to different architectural configurations.

3.1 Transcoding Finite State Machine

The FSM consists of five states. States are denoted as circles. S_{in} is the initial state and receives content data as input. The only success final state of the FSM is S_{out}; a sample component that is connected to the final state is a HTML browser. The remaining three states, S_{custom}, $S_{intermediate}$, and $S_{standard}$, reflect the internal state of the converter. The arcs denote state transitions. A transcoding action is triggered by incoming events after or before a transition



States $Q = \{S_{custom}, S_{intermediate}, S_{standard}, S_{in}, S_{out}\}$
Input lang. $\Sigma = \{x | x \in CF \cup SF \cup IF\}$
Transitions $\delta = Q \times \Sigma$
Start state $S = \{S_{in}\}$
End state $F = \{S_{out}\}$

Fig. 1. FSM for transcoding architectures.

from one state to the next. The identity transformation is the action associated in state $S_{standard}$ before the transition to the final state.

An event consists of the delivery of UI data. The set of user interface data consists of elements belonging to one of the following groups.

- **Custom Format.** The *custom format* (CF) comprises all formats that are proprietary formats and require at least one transformation to be displayed by widespread consumer devices.
- **Intermediate Format.** The *intermediate format* (IF) comprises all intermediate formats that allow transforming source formats into target formats.
- **Standard Format.** Finally, the *standard format* (SF) comprises all formats that widespread consumer devices can display without transformation (e.g., WML, HTML).

The states perform format conversions. For example, if the current state is S_{custom}, the current format f is an element of CF, and the target format is HTML (an element of SF), then S_{custom} transforms f into HTML and the FSM moves to state $S_{standard}$.

Transitions starting at S_{in}, require the developer of the converter to generate data formats of type CF or SF at *design time*. All transitions starting in the other part are handled by the FSM and generated at *run-time*. Furthermore, the transition from state $S_{standard}$ to state S_{out} is the identity transition, that means, successive formats are identical (for example, $L = \{\ldots, wml, wml, \ldots\}$). Two transitions in the FSM need further explanation. In state S_{custom}, the input of an element of CF leads to a failure. As the designer must generate CF, there is no need for transcoding between two CFs. The designer must do this at design time. In $S_{intermediate}$ the input of CF also leads to a failure. Formats belonging to CF are input formats and created at design time. Thus, there is no need for this transition.

An example is the conversion from WML to HTML using RXML as the intermediate data format IF. The input is $L = \{WML, RXML, HTML\}$. WML and HTML belong to the group of SFs. WML and HTML can be displayed by widespread consumer devices such as WAP enabled mobile telephones and Web browser, respectively. The transitions of the FSM for this conversion are depicted

in Figure 2 (for details about the semantic of this process see Section 4.3). The three-stacked boxes represent one state and the arrow between two states represents a transition. The top box shows the current state in the FSM, the second box shows the current format of the UI description, and the third box shows the output format (the converter will convert the format from the source format into the target format). The initial state in the example is S_{in}. The first transition is from S_{in} to $S_{standard}$ and the original content is transcoded into WML. In the second step the FSM converts WML into RXML, an IF; thus, getting into state $S_{intermediate}$. The remaining steps follow the same structure.

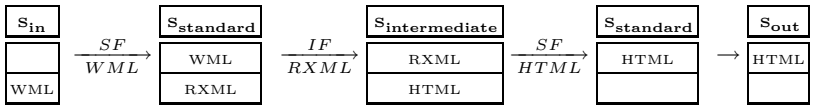


Fig. 2. Transition example.

3.2 Architectural Comparison of UIT Systems

The FSM provides the basic model that is mapped onto conceptual software architecture models. The concept of software architectures consists of three abstract building blocks [6, 7]. The following paragraph shows how the FSM can be mapped onto these building blocks (Figure 3):

- **Component.** Components are either processing elements or data elements:
  - **Processing elements.** The FSM differentiates broadly between three types of processing elements: (1) the functional core, represented by state S_{in}, (2) the physical interactor (S_{out}), and (3) the transcoding components. An example of a physical interactor is a WAP browser of a mobile telephone. It accepts and displays WML data. A sample transcoding component is "dvi2ps"; it converts DVI data into Postscript data.
  - **Data elements.** Data elements contain, among other information, UI data and are created, displayed, or transcoded in processing elements. For example, data elements contain HTML code for display on a browser.
- **Connector.** A connector acts as data channel between two components. The connector holds information about the two connected components and obeys to a specific protocol. A protocol is defined as a set of incoming and outgoing data types and the data exchange sequence [8]. An example connector is a UNIX pipe.
- **Configuration.** The configuration is the structural organization of components and connectors; It corresponds to a transition path in the FSM. Thus, the configuration provides a high-level overview of the transcoding architecture.

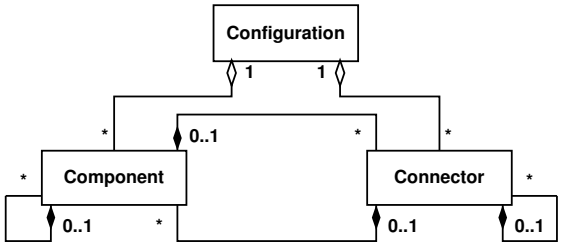


Fig. 3. Meta-model of conceptual architecture view. Adapted from [8]

As an example, a UI converter using indirect conversion to convert WML into HTML requires five components (creator, display, and three transcoding components) and four connectors (see Figure 2 on the facing page). Although the FSM shows all possible transitions within the given restrictions, an analysis of existing UITSs reveals four main transition paths:

- **Identity configuration.** Using identity configuration, the source and the target formats are equal. Thus, the transition path only consists of three states and two transitions (the process of creation and the identity transformation): $S_{in} \rightarrow S_{standard} \rightarrow S_{out}$ An example of an identity configuration is a developer creating content for just one device type (e.g., only for HTML capable devices).
- **Direct configuration.** Using the direct configuration the number of transitions equals three. Input and output format are elements of SF.The transition consists of $S_{in} \rightarrow S_{standard} \rightarrow S_{standard} \rightarrow S_{out}$ An example for the direct configuration is converting WML directly into HTML.
- **Indirect configuration.** Using the indirect configuration the number of transitions equals four and an IF is used, so $S_{intermediate}$ is reached. The path within the FSM is: $S_{in} \rightarrow S_{standard} \rightarrow S_{intermediate} \rightarrow S_{standard} \rightarrow S_{out}$
- **Hybrid configuration.** Finally, the hybrid configuration inherits one property of the direct and one property of the indirect configuration. The hybrid conversion consists of (1) two transitions and (2) the initial input format is CF (a non-SF)—thus, S_{custom} is reached: $S_{in} \rightarrow S_{custom} \rightarrow S_{standard} \rightarrow S_{out}$

4 Comparison of Existing UIT Systems

On the basis of the FSM, four transcoding configurations have been identified. This section evaluates the four categories and surveys representatives of each category.

4.1 Identity Configuration

Systems using the identity configuration employ the same data format at design and presentation time.

Traditional Approach The identity configuration reflects the traditional computing paradigm. Figure 4 illustrates the architectural configuration using the FSM. The designers of these solutions produce UI data for a single platform: For example, a highly optimized HTML-based application. The advantages of this configuration are clear: The knowledge of well-defined properties of the target platform at design-time enables the designer to optimize the user-interface data for visualization. This is possible, since the data format used at design time (S_{in}) is identical to the data format employed at presentation time (S_{out}). The main

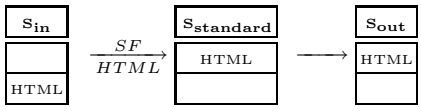


Fig. 4. Identity configuration.

disadvantage of this approach is the inflexibility. Since a pure "1-to-1"-relation is rare and merely stable over time, the inflexibility is a major concern when porting the application to new platforms if the set of possible target platforms is enlarged and the system needs to be accessed across a multitude type of devices. If the "1-to-1"-relation remains stable this approach is the best way to choose. Such optimization results in high quality rendered data, however, if the relation is likely to change, the inflexibility prevents easy adaptation to new requirements and environments.

4.2 Direct Configuration

Direct configuration is applied to systems that have been developed for a single platform. Due to a change of the systems requirements the application has to migrate from a "1-to-1" relation to a "n-to-1" relation or an "n-to-n" relation. The key challenge is to modify the application enabling it to use multiple platforms.

IBM Websphere IBM Webshpere supports "n-to-n" relations between design formats and the formats used at presentation time [2]. Websphere comes with a set of transcoding plug-ins for a wide range of SFs. It allows having a multitude of data formats at design time. A set of transformations converts one SF into another SF. Figure 5 shows the transition path through the FSM, which represents the "n-to-n" relation. The conversion from one format into another format cannot be done without loss of precision if,

- elements of the input format have no corresponding elements in the target format and cannot be mapped to other elements of that format or
- elements of the input format have no corresponding elements in the target format with exactly the same features.

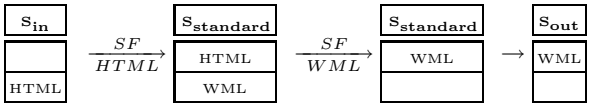


Fig. 5. Direct configuration.

Elements offering no corresponding elements in the target data format are omitted and non-convertible elements are reduced to elements that can be transformed to equivalent functionality, that means, their functional intent is preserved, although their functional presentation can be quite different.

The advantage of this approach is that existing "1-to-1" relation applications can be transformed into "n-to-1" relation solutions. The existence of a set of transformer suggests that existing solutions developed using a specific standard data format can be converted without loss of functionality to other device dependent formats. This is only true for those specific cases where the design format uses merely elements that can be converted to the target formats. However, the migration is problematic, if the existing solution has been developed given the "1-to-1" relation assumption. Therefore in most cases existing solutions have to be redesign and reengineered to make them accessible via different mechanisms. The fact that there are specific transformations from one SF into another has the advantage that the transformations can be highly optimized for exactly those data formats.

4.3 Indirect Configuration

The motivation for indirect configurations is very similar to the situation described in Section 4.2. The aim is to enable existing solutions that have been targeted for use on a specific target platform to be used through a wide variety of access mechanism. In order to reduce the creation of a multitude of transformations, an IF component layer is introduced. Data in SF is transformed to the IF, which is device independent and further transformed to an SF different from the SF used at design time. The introduction of an additional layer reduces the total amount of transformations required. The use of a common IF means that the input data is possibly filtered and elements of that format are omitted to map the input data to the IF.

Relational XML A representative of this category is relational XML (RXML) [9]. Figure 2 on page 4 sketches the high-level architecture of the indirect configuration. Although RXML pursues with the introduction of an IF a different concept as the direct configuration approaches both may be equivalent. It is sufficient to label an SF of the direct configuration as IF and apply two transformations: the first transformation from the SF to the SF, which is labeled as the IF and the second transformation from the IF to the target SF.

4.4 Hybrid Configuration

The key idea of representatives of this approach is the introduction of a UI description in CF that is shared among all target SFs. The shared data is in this respect SF independent. In the proposed FSM the use of the CF is different from the IF, although both formats share the feature of being SF independent. The IF is only temporarily present, whereas the CF data is created at design time. The intent of the proprietary data is to support multiple target platforms, i.e., to be transformable to a set of predefined SFs. Figure 6 illustrates an example of the hybrid configuration. This approach is attractive, since a single shared

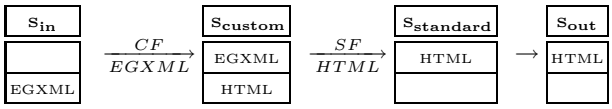


Fig. 6. Hybrid configuration.

format can serve a multitude of target formats. The shared format assures that the functional presentation in the CF also provides an adequate presentation of the same functionality in any other supported SF. The main disadvantage is the logical consequence of one of the main advantages. The least sophisticated format determines the features of the shared data format. This effect is also known as the *least denominator problem*. The shared data format incorporates only elements that can be transformed to equivalent elements in all target formats.

User Interface Markup Language The User Interface Markup Language (UIML) defines 28 generic elements that are common to UIs [10]. Transformations from UIML to different target SFs are done with rendering machines. UIs for various platforms can be designed with UIML. However, UIs for different platforms require usually the creation of separate and distinct UIML files. The specific requirements of the platforms need separate UIML vocabulary and therefore separate UIML files have to be written (one for each platform). The main advantage is the usage of a single UI language although the vocabulary can be very different for different platforms. Only recently, Farooq and Abrams tried to overcome this by extracting a generic UIML vocabulary [11].

.NET .NET offers the Compact framework and the Microsoft Mobile Internet Toolkit for UI generation for mobile devices [12, 4]. The Compact framework allows traditional graphical UI development via an API. The Compact framework has no built-in support for markup language UI creation. The resulting applications need the .NET framework running on the target device. This restricts the range of devices which allow an application to execute to the set of *smart*

devices. The Microsoft Mobile Internet Toolkit gives support for a wide range of markup languages. Device adapters allow the different rendering of controls to mobile device specific markup languages.

MUSA Multi User Interface, Single Application (MUSA) introduces the event handler graph (EGXML, Figure 6) that describes the common interaction of an application [13, 3]. The use of generic UI elements is replaced in support of the interaction description between interfaces and users by means of the event handler graphs. The event handler graph is an abstract description of a service, which is presented to the user who interacts with it through a UI. Event handlers in the event handler graph do not specifically define UI elements. However, event handlers are assigned and eventually mapped to UI elements that are able to trigger the event handlers. In addition to the event handler graph, whose description is shared by all target platforms, the global and local layout of the UI is described separate from the event handler graph.

5 Discussion

This section discusses the three transcoding configurations with respect to system development. The identity transformation category will not be considered here.

5.1 Proprietary vs. Standard Format

With respect to the transformation process the FSM differentiates broadly between two approaches to UI conversion. The direct configuration approach uses only standard format in the conversion process whereas the indirect and the hybrid approach define a proprietary or intermediate format. Standard formats have not been defined with respect to the current situation, where content authors need to develop content targeted for use via a variety of platforms. They do not incorporate the idea of platform independent authoring and are therefore not truly apt to be converted to different standard platforms. Since recently, HTML for example, did not even comply with the XML standard and there are still HTML editors, which produce non-XML compliant HTML code. As a consequence, tools like XSLT can merely be used in combination with HTML. Proprietary formats are especially designed to form a compact medium, which serve as an intermediate step towards presentation. The disadvantage of a proprietary format is the fact that designers have yet to learn another new language.

5.2 System Development

Development time is crucial to the success of a software system. Software systems have long life cycles, and attributes like modifiability gain more and more attention during development time. Modifiability subsumes activities like changing and extending a software system. This section discusses system development

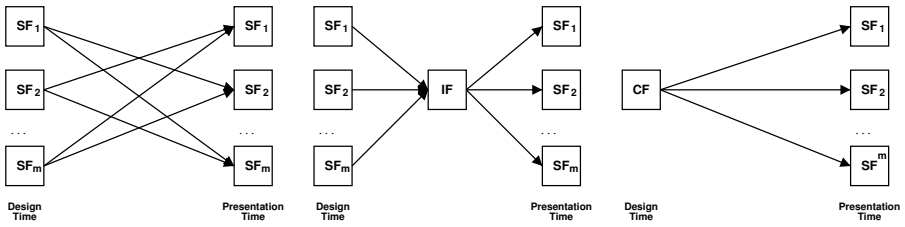


Fig. 7. System development effort: (a) Direct configuration (b) Indirect configuration (c) Hybrid configuration. The arrows indicate transformations and the rectangles data formats.

effort, change and extension development effort of the three classes of transcoding configuration. The identity transformation category will not be considered here.

Tool support is important during development and maintenance time and is considered crucial for the success of a project.

System Development Effort In this context, development effort is quantified as the creation of transformations and data formats. The direct configuration uses exclusively SFs and needs $m^2 - m$ transformations (T) to support m data formats (Figure 7(a)). The indirect configuration needs the creation of $2m$ transformations and an IF that is apt to work as an intermediate format between 2 transformations (Figure 7(b)). Only m transformations are required for the hybrid transformation configuration (Figure 7(c)). In addition to the creation of the transformation, a CF needs to be created.

The comparison returns advantageous results for the hybrid transformation. However, direct and indirect transformations start with SFs whereas the hybrid transformation configuration requires the input to be written in CF. If an existing application needs to be extended to allow additional devices to access it, the direct and indirect transformation give immediately results. As for the hybrid approach, the application needs to be migrated to the proprietary data format.

Modification Development Effort Modifiability is defined as the ability to make changes quickly and cost effectively [14]. This section elaborates on the impact of evolution of an SF on the three configurations. The discussion can broadly be categorized into changes of SFs and subsequent changes of proprietary or IF. However, the change of the latter category is induced by a change of an SF. The impact of a change is especially high for those configurations, which accept SFs as input. The change of an SF entails a modification of *m-1* transformations in the direct configuration: the transformations from the SF that has

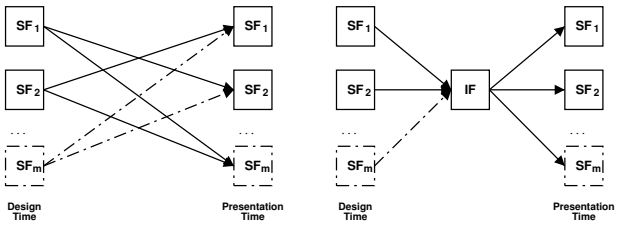


Fig. 8. Modification development effort: (a) Direct configuration (b) Indirect configuration. The dashed arrows and rectangles denote modified artifacts.

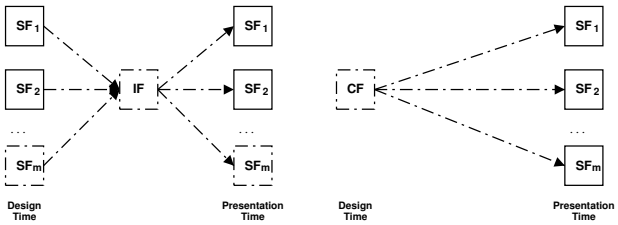


Fig. 9. Modification development effort: (a) Indirect configuration (b) Hybrid configuration. The dashed arrows and rectangles denote modified artifacts.

been changed to every other SF (Figure 8(a)). If a transformation from an SF wants to exploit new features of the changed SF, it must be changed as well. The indirect configuration requires changing only one transformation (from the modified format to the IF, Figure 8(b)). As a consequence the IF may need to be changed. In this case all $2m$ transformations change since the IF is the unique intermediate channel that connects two transformations (Figure 9(a)). Strictly speaking, the hybrid configuration may not require any changes. The transformations convert the CF into SFs and there is no need to support new features. However, if the CF changes it is mandatory that all other SFs support that specific feature as well and thus, all m transformations change (Figure 9(b)).

Extension Development Effort The number of formats is growing as the number of new Internet enabled devices is growing. There are even multiple formats for one class of devices. Extension development effort deals with the question: How capable are UITSs to integrate additional SFs? The direct transformation configuration requires the definition of $2m$ new transformations for each additional format: m transformations from the new format to the existing

formats and another m transformations from the existing formats to the new format (Figure 10(a)). The indirect transformation configuration only needs two additional transformations: one from the new format to the IF and one from the IF back to the new format (Figure 10(b)). The introduction of an IF as an intermediate layer is the reason for the constant number of additional transformations. Only one additional transformation is required using the hybrid configuration: the transformation from the CF to the new format (Figure 10(c)). Due to the use of a CF the number of additional transformations is minimal.

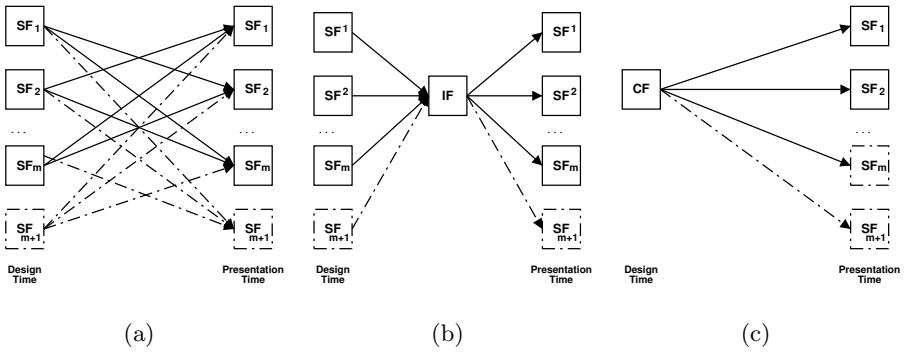


Fig. 10. Extension development effort: (a) Direct configuration (b) Indirect configuration (c) Hybrid configuration. The dashed arrows and rectangles denote new and modified artifacts.

6 Concluding Remarks

The paper presents an abstract model for user-interface transcoding systems. On the basis of an FSM, we identified four transcoding configurations and mapped them onto existing solutions. In the evaluation of these configurations, we concentrated on the development effort, i.e., the system, the extension, and the modification development effort. The results of the evaluation are summarized in Table 1.

From these results, we conclude that the hybrid configuration provides the best overall performance for these three kinds of efforts. Our experience with industry and their approaches [15] further support this conclusion as most recent products utilize hybrid configuration and proprietary base formats (sometimes closely related to standard formats).

As future work, we will research the differences between these three configurations in more detail; especially concentrating on software quality attributes such as throughput and capacity of adaptivity.

Configuration	System Development Effort	Change Development Effort	Extension Development Effort
Direct	$(m^2 - m)T$	$(m-1)T$	$2mT$
Indirect If IF changes	$2mT + 1IF$	$1T$ $2mT + 1IF$	2
Hybrid If CF changes	$mT + 1CF$	$0T$ $mT + 1CF$	1

Table 1. Development effort for direct, indirect, and hybrid configuration.

References

1. Goeschka, K., Smeikal, R.: Interaction Markup Language - An Open Interface for Device Independent Interaction with E-Commerce Applications. In: Proceedings of the 34th Hawaii International Conference on Systems Sciences, IEEE Computer Society (2001)
2. Britton, K., R.Case, A.Citron, Floyed, R., Li, Y., Seekamp, C., Topol, B., Tracey, K.: Transcoding. Extending e-business to new environments. IBM Systems Journal **40** (2001) 153–178
3. Menkhaus, G.: An Architecture for Supporting Multi-Device, Client-Adaptive Services. Special Volume of the Annals of Software Engineering Journal on OO Web-based Software Engineering (2002)
4. Microsoft: Microsoft Mobile Internet Toolkit (2002)
5. Workshop, U.T.D.: A Metamodel for the Runtime Architecture of an Interactive System. SIGCHI Bulletin **24** (1994) 32–37
6. Perry, D., Wolf, A.: Foundations for the Study of Software Architecture. ACM SIGSOFT Software Engineering Notes **17** (1992)
7. Shaw, M., Garlan, D.: An Introduction to Software Architecture. In Ambriola, V., Tortora, G., eds.: Advances in Software Engineering and Knowledge Engineering, River Edge, NJ: World Scientific Publishing Company (1993)
8. Hofmeister, C., Nord, R., Soni, D.: Applied Software Architecture. Addison-Wesley object technology series. Addison Wesley, Reading, Mass. (1999)
9. Saha, S., Jamtgaard, M., Villasenor, J.: Bringing the Wireless Internet to Mobile Devices. IEEE Computer **34** (2001) 54–58
10. Abrams, M., Phanouriou, C., Batongbacal, A., Williams, S., Shuster, J.: UIML: An Appliance-Independent XML User Interface Language. WWW8 / Computer Networks **31** (1999) 1695–1708
11. Farooq, M., Abrams, M.: Simplifying Construction of Multi-Platform User Interface Using UIML. In: UIML Europe Conference. (2001)
12. Microsoft .NET: The .NET Compact Framework-Overview (2002)
13. Fischmeister, S., Menkhaus, G., Pree, W.: MUSA-Shadow: Concepts, Implementation, and Sample Applications; A Location-Based Service Supporting Multiple Devices. In: Proceedings of Pacific TOOLS, Sydney, Australia (2002) 71–79
14. Bass, L., Clemens, P., Kazman, R.: Sofware Architecture in Practice. "Addison Wesley", Reading, Mass. (1998)
15. Mandyam, S., Vedati, K., Kuo, C., Wang, W.: User Interface Adaptations: Indispensible for Single Authoring. In: W3C Workshop on Device Independent Authoring Techniques, SAP University, St. Leon-Rot, Germany, W3C (2002)