User Interface Plasticity: Model Driven Engineering to the Limit!

Joëlle Coutaz
Grenoble Informatics Laboratory (LIG)
Grenoble University
BP 53, 38041 Grenoble Cedex 9
+33 4 76 51 48 54
joelle.coutaz@imag.fr

ABSTRACT
Ten years ago, I introduced the notion of user interface plasticity to denote the capacity of user interfaces to adapt, or to be adapted, to the context of use while preserving usability. The Model Driven Engineering (MDE) approach, which was used for user interface generation since the early eighties in HCI, has recently been revived to address this complex problem. Although MDE has resulted in interesting and convincing results for conventional WIMP user interfaces, it has not fully demonstrated its theoretical promises yet. In this paper, we discuss how to push MDE to the limit in order to reconcile high-level modeling techniques with low-level programming in order to go beyond WIMP user interfaces.

Author Keywords
User interface plasticity, user interface adaptation, user interface generation, run time adaptation, user interface composition, dynamic service composition, model driven engineering (MDE), service-oriented architecture (SOA).

ACM Classification Keywords
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *User interfaces*.

General Terms
Design, Human Factors.

INTRODUCTION
Ten years ago, I introduced the notion of "user interface plasticity" to denote the capacity of user interfaces to adapt, or to be adapted, to the context of use while preserving usability [35]. My proposal was motivated by the emergence of ubiquitous computing with the need for accommodating a large degree of variability in terms of heterogeneity, dynamicity, and scalability. Although these challenges have permeated the whole ICT community, they have not been addressed in a holistic, systemic manner.

Typically, virtualization, as developed for cloud computing, does not cover that of interaction resources. SLA (Service Level Agreement) developed for dynamic service composition, does not cover any of the HCI-centered concerns. Research in autonomic systems conveniently keeps humans out of the loop. Service composition as supported by mash-up tools, boils down to the assembly of ready-for-use services whose UI's, which are tightly coupled with business code, cannot be "plastified". Unfortunately, during this period, the HCI community has not been any better, developing a variety of disjoint tools and concepts.

HCI researchers have addressed user interface plasticity from different starting points, depending on their "credo": at the toolkit level by those who advocate "hard core development" and fine grained control of user interfaces [15, 16], at the infrastructure level with the development of dedicated middleware by those who strive for generic computational substrates [1, 25, 29, 34, 36], to task level modeling by those who believe in the top-down development of user interfaces [4, 26, 32]. Principles and concepts from Model Driven Engineering have however brought some hope into a unifying and systemic approach to the problem of UI plasticity. But is MDE good enough and/or used appropriately?

In this article, we will analyze the contribution of Model-Driven Engineering to HCI as well as its limitations in the light of UI plasticity. From our early experience with MDE and UI plasticity, we will show how to exploit models at run time to obtain maximum flexibility and quality. We will conclude with recommendations for a research agenda.

CONTRIBUTIONS OF MDE TO HCI
The primary contributions of MDE to HCI are two simple notions – that of model and meta-model, which, combined with transformations and mappings, constitute a powerful framework for knowledge sharing and technical integration.

A *model* is a representation of a thing, with a specific purpose. It is "able to answer specific questions in place of

EICS'10, June 19–23, 2010, Berlin, Germany.

the actual thing under study" [5]. A *meta-model* sets the rules for producing models. A *transformation* is the production of a set of target models from a set of source models, according to a transformation definition. In turn, a transformation definition is a set of transformation rules that together describe how source models are transformed into target models [21]. A transformation expresses an overall dependency between source and target models. However, experience shows that a finer grain of correspondence needs to be expressed. Typically, the incremental modification of one source element should be propagated easily into the corresponding target element(s) and vice versa. The need for traceability between source and target models is expressed as *mappings* between source and target elements of these models[1].

The HCI community has a long experience with models and meta-models, long before MDE existed as a field. In the 1980's, grammars (meta-models) were the formal basis for generating textual and graphical user interfaces [19]. Until recently, transformation rules were implemented as code within UI generators offering very little to no control over the resulting user interface. In addition, mappings were limited to the expression of correspondence (bindings) between elements of the user interface with the API of the functional core (i.e. the business code).

MDE has helped the HCI community to promote transformation rules as models. "Transformations as models" has three notable advantages – which, so far, has not been fully exploited by the HCI community:

1. It opens the way to knowledge capitalization and reuse: frequent transformations can serve as patterns in libraries, which in turn, provide handles for intra- and inter- UI consistency.
2. Comparative evaluations of UI's can be performed in a controlled way, and UI's can be (re)targeted for different contexts of use using different transformations.
3. Most notably, transformations can be transformed, offering a powerful formal recursive mechanism for supporting UI plasticity.

To our best knowledge, no research has been conducted on transforming transformations for UI plasticity. On the other hand, patterns are emerging [35] and early work has been initiated on UI's generated with different sets of transformation rules to support different usability criteria [18, 33].

[1] In mathematics, a mapping is "a rule of correspondence established between two sets that associates each member of the first set with a single member of the second" [The American Heritage Dictionary of the English Language, 1970, p. 797]

Considering the big picture, MDE has provided the HCI community with a vocabulary and a framework to express its own conceptual generic framework for the development of plastic user interfaces: the CAMELEON reference framework [9]. As shown in Figure 1, the CAMELEON reference framework makes explicit a set of models (e.g., task model, Abstract UI, Concrete UI, Final UI) that serves as a common vocabulary within the HCI community to discuss and express different perspectives on a user interface. Again, the notion of transformation borrowed from MDE, is used to combine these models into distinct development processes. For example, conventional UI generation operates by the way of top-down vertical transformations. Typically, an abstract UI (AUI) is derived from the domain-dependent concepts and task models. In turn, the AUI is transformed into a concrete UI (CUI), followed by the final executable UI (FUI). At the opposite, a reverse engineering process infers abstract models from more concrete ones using vertical bottom-up transformations. Translations are horizontal transformations that maintain the same level of abstraction between the source and target models.

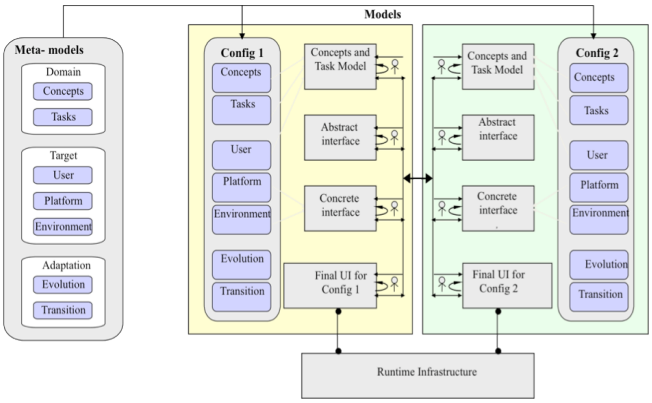

Figure 1. The CAMELEON reference framework for the development of plastic user interfaces (adapted from [9]).

Unlike the process initiated in the 1980's, which contained one entry point only at a high level of abstraction, the CAMELEON framework authorizes entry points at any level of abstraction from which any combination of horizontal and vertical bottom-up and top-down transformations can be applied. This theoretical flexibility means that the stakeholders involved in the development of an interactive system can use the development process that best suits their practice or the case at handIn short, the CAMELEON reference framework is an MDE-compliant conceptual generic structuring tool for the development of plastic UI's:

- As a structuring reference framework, it federates the HCI community around a consensus.
- As a conceptual generic tool, it sets a vast agenda for technical research.
- As an MDE-compliant framework, it is still unclear in practice that modeling is the only way to go in HCI. This issue is discussed next.

MDE AND HCI IN PRACTICE

The CAMELEON reference framework brings together the "right models" but the HCI community is far from having the "models right". The profusion of initiatives and User Interface Description Languages (UIDL) is symptomatic of the need – and difficulty, to define a coherent set of non-ambiguous and easy to understand meta-models capable of covering the problem space of plastic UI's. The UsiXML[2] consortium is putting significant effort in this direction, but has not reached its objectives yet. In my opinion, two meta-models (at least) deserve particular attention: transformations and Concrete UI's.

As stated above, transformations offer an elegant mechanism for full flexibility and technical integration. However, *transformations are hard to express* (QVT and ATL are not languages for naïve developers). In addition, usability rules are even harder to convey formally [33]. More importantly, *inverse transformations cannot be automatically derived* for any source transformations. This is a fundamental flaw that may result in inconsistent models as transformations are performed up and down iteratively during the life cycle of a system, breaking down the flexibility of the solution space envisioned by the CAMELEON reference framework.

At the CUI level, meta-modeling, not only lags behind innovation, but bridles creativity. UIDL's for the expression of concrete user interfaces are technology-driven instead of leaving rooms for new forms of interaction techniques. Although the CARE properties [12] have been devised 15 years ago, CUI languages have hardly scratched the surface of multimodal interaction. We are still unable to generate the paradigmatic "put-that-there" multimodal user interface introduced more than 25 years ago [6]. We do however generate simplistic multimodal UI's based on XHTML+VoiceXML but with very limited micro-dialogues for interaction repair [4]. Actually, CUI-level UIDL's are still struggling with the description of conventional GUI's for desktop computing. Meanwhile:

- New forms of "constructable" computers such as the MIT shiftables[3] and the CMU toy blocks[4] are put on the market;
- Novel interaction techniques are proliferating whether it be for supporting mobility (e.g., SixSense [22]), for 3D interaction (where gesture and 3D screens are becoming predominant), or even for graphical tabletops and multi-surface interaction [3];
- New requirements are emerging: design is switching from the development of useful and usable systems for people with precise goals, to engaging and inspired interaction spaces whose users can easily switch from a consumer mind to the creator mode.

In short, CUI meta-models need to capture the unbound vibrant convergence of physicality with "digitality". Perhaps, meta-modeling is, by essence, the wrong approach to CUI's: a model, which represents a thing, is necessarily a simplification, therefore a reduction, of the real thing. In these conditions, the subtle aspects of interaction, which make all the differences between constrained and inspired design, are better expressed using code directly in place of an abstraction of this code. However this assertion should be mitigated by the following findings: designers excel at sketching pictures to specify concrete rendering. On the other hand, they find it difficult to express the dynamics, forcing them to use natural language [24]. One way to fill the gap between designers'practice and productive models is to revive work à-la-Peridot [23] such as SketchiXML [14] where drawings are retro-engineered into machine-computable rendering. As for inferring behavior from examples, the promising "Watch What I Do" paradigm initiated in the late 1970's (cf. Dave Smith's PYGMALION system [31]) is still an opened question.

Model Driven Engineering, as a software development methodology, has favored the dichotomy between the design stages and the run time phase, resulting in three major drawbacks:

- Over time, models may get out of sync with the running code.
- Design tools are intended for software professionals, not for "the people". As a result, end-users are doomed to consume what software designers have decided to be good for their hypothetic target users.
- Run time adaptation is limited to the changes of context identified as key by the developers. Again, the envelope for end-users' activities is constrained by design.

Applied to UI development, the dichotomy between design and run time phases means that UI generation from a task model cannot cope with ambient computing where task arrangement may be highly opportunistic and unpredictable. On the other hand, because the task model is not available at run time, the links between the FUI and its original task model are lost. It is then difficult, not to say impossible, to articulate run-time adaptation based on semantically rich design-time descriptions. As a result, a FUI cannot be remolded beyond its cosmetic surface as supported by the CSS.

Blurring the distinction between the design stage and the run time phase is a promising approach. This idea is emerging in main stream middleware [17] as well as in HCI. The middleware community, however, does not necessarily address end-user concerns. Typically, a

[2] http://www.usixml.org

[3] http://sifteo.com/

[4] http://www.modrobotics.com/

"sloppy" dynamic reconfiguration at the middleware level is good enough if it preserves system autonomy. It is not "observable" to the end-user whereas UI re-molding and UI redistribution are! Thus, UI plasticity puts additional constraints. In particular, it becomes necessary to make explicit the transition between the source and the target UI's so that, in Norman's terms, end-users can evaluate the new state. We need to pay attention to transition UI's in generic terms, not on a case per case basis.

In the following section, I illustrate the combination of models and code at run time with the work we have done for addressing the problem of plastic UI's at run time.

MODELS AT RUN TIME

Early experience in the development of plastic UI's can be summarized as "one does not fit all". The following three principles show why.

Principle #1: Close-adaptiveness must cooperate with open-adaptiveness [27]. As discussed above, by design, an interactive system has an "innate domain of plasticity": it is close-adaptive for the set of contexts of use for which this system/component can adapt on its own. For unplanned contexts of use, the system is forced to go beyond its domain of plasticity. It must be open-adaptive so that a tier infrastructure (a middleware) can take over the adaptation process.

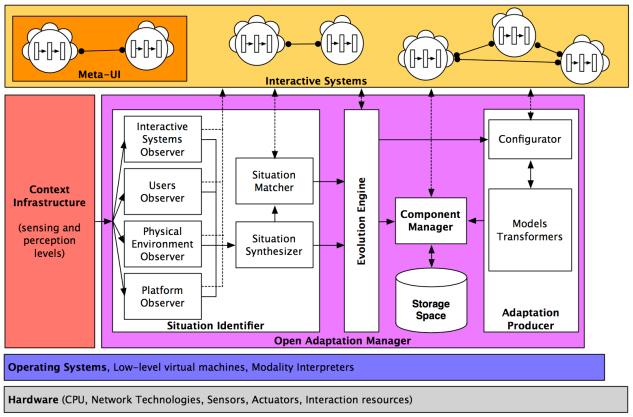

Figure 2. A typical functional decomposition for Ambient interactive spaces [2].

As shown in Figure 2, the functional decomposition of the middleware that supports open adaptation includes:

- A context infrastructure that builds and maintains a model of the context of use [30].
- A situation synthesizer that computes the situation and possibly informs an evolution engine of the occurrence of a new situation.
- An evolution engine that elaborates a reaction in response to the new situation.
- An Adaptation producer that implements the adaptation plan produced by the evolution engine.

Such a decomposition is commonly used for the development of autonomic systems. To adapt this decomposition for plastic UI's, we propose the following improvements:

- The end-user is kept in the loop: the reaction to a new situation may be a mix of specifications provided by developers or learnt by the evolution engine. In addition, it may call upon end-users' advice by the way of a meta-UI [13]. I see this meta-UI as an end-user development environment.
- The components referred to in the action plan do not necessarily exist as executable code. This is where Principle #2 comes into play.

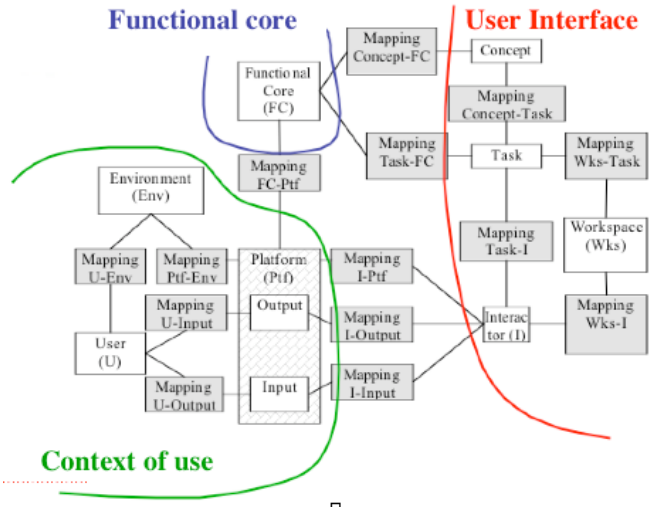

Figure 3. An interactive system as a graph of models available at run time. These models are related by mappings and transformations.

Principle #2: At run time, an interactive system is a set of graphs of models that express different aspects of the system at multiple levels of abstraction. As advocated by the CAMELEON framework, these models are related by mappings and transformations. As a result, an interactive system is not limited to a set of linked pieces of code. Models developed at design-time, which convey high-level design decision, are still available at runtime for performing rational adaptation beyond cosmetic changes. When a component retrieved by the component manager is a high-level description such as a task model, the configurator relies on reificators to produce executable code as in Digymes [11] and iCrafter [29]. A retrieved component may be executable, but may not fit the requirements. Ideally, it can be reversed-engineered through abstractors, then transformed by translators and reified again into executable code [7].

Principle #3: *By analogy with the slinky meta-model of Arch, the software developer can play with principles #1 and #2*. At one extreme, the interactive system may exist as

one single task model linked to one single AUI graph, linked to a single CUI graph, etc. (see Figure 3). This application of Principle #1 does not indeed leave much flexibility to cope with unpredictable situations unless it relies completely on the tier middleware infrastructure that can modify any of these models on the fly, then triggers the appropriate transformations to update the Final UI.

Alternatively, the various perspectives of the system (task models, AUI, FUI, context model, etc.) as well as the adaptation mechanisms of the tier infrastructure are distributed across distinct UI service-oriented components, each one covering a small task grain that can be run in different contexts of use. We have adopted this approach to implement the Comet toolkit [16].

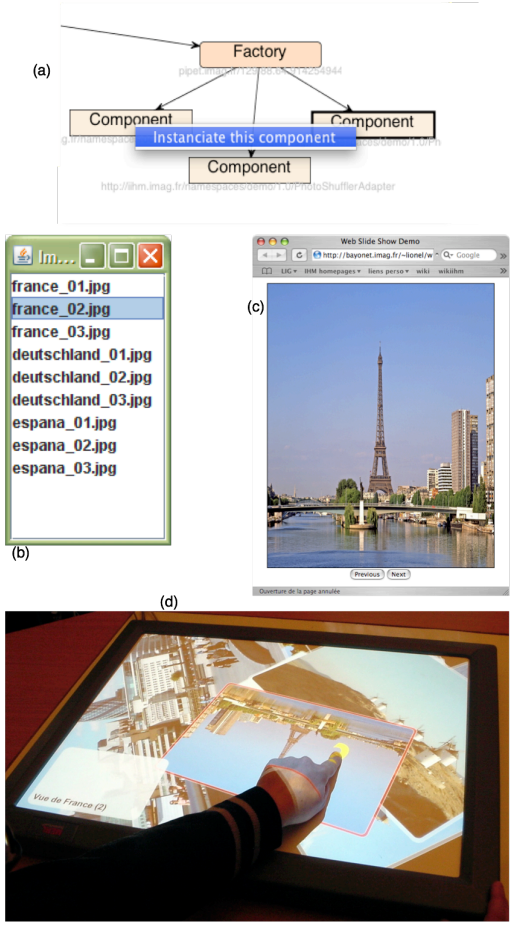

Figure 4. The Photo-browser application [1]: a dynamic composition of executable and transformable components, managed by a dynamic set of interconnected factories running on different platforms (Windows, MacOS X, and Android).

Basically, a Comet is a plastic micro-interactive system whose architecture pushes forward the separation of concerns advocated by PAC and MVC. The functional coverage of a comet is left open (from a plastic widget such as a control panel, to a complete system such as a powerpoint-like slide viewer). Each Comet embeds its own task model, its own adaptation algorithm, as well as multiple CUI's and FUI's, each one adapted to a particular context of use. FUI's are hand-coded possibly using different toolkits to satisfy our requirements for fine-grained personalization and heterogeneity. From the infrastructure point of view, a Comet is a service that can be discovered, deployed and integrated dynamically into the configuration that constitutes an interactive environment. The COTS [8], whose executable UI code is meta-described with the task they support, are based on similar ideas.

Figures 4 and 5 show another application of principles #1 and #2 for the implementation of Photo-browser. The FUI of Photo-browser is dynamically composed of:

- a Tcl-Tk component running on a multi-point interactive surface (Fig. 4-d),
- a Java component that shows a list of the image names (Fig. 4-b),
- and an HTML-based browser to navigate through the images set (Fig. 4-c).

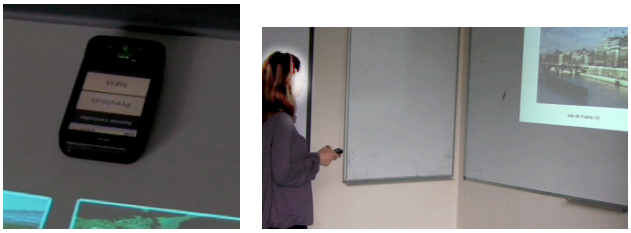

Figure 5. (Left) Connecting a Gphone to the interactive space by laying it down on the interactive table. (Right) Using the Gphone as a remote-controller to browse photos displayed by the HTML UI component of fig. 3c and video-projected on the wall. (In the current implementation, the contact of the Gphone with the Diamond Touch is simulated as a new situation event for interpretation by Ethylene).

Photo-browser is implemented on top of a tier middleware infrastructure (called Ethylene) that covers the evolution engine, the component manager as well as the adaptation producer of Figure 2 [1]. Ethylene is a distributed system composed of Ethylene factories each one running on possibly different processors (IP devices). The role of an Ethylene factory is to manage the life cycle of a set of components that reside on the same IP device as this factory and that have been registered to this factory. When residing on storage space, a component is meta-described using EthylenXML, an extension of WSDL. This meta-description includes the human task that the component supports, the resources it requires, and whether it is executable code or transformable code. In the latter case, it may be a task model, an AUI, a CUI, or even a graph of these models. For example, the HTML-based component (Fig 4.c) is a CUI expressed in a variation of HTML. It must be transformed on the fly to be interpreted by an HTML renderer. The Tcl-Tk multi-point UI and the Java list are executable code. Their EthyleneXML meta-description specifies that they support image browsing and

image selection tasks, that they need such and such interaction resources (e.g., a Tcl-Tk interpreter and a Diamond Touch) for proper execution, and that they require such and such communication protocol to be interconnected with other components. The GPhone UI component of Figure 5 is an executable Gphone app that supports the next-previous browsing tasks. Interconnection between components is initiated by the factories.

As these examples show, the engineering community of HCI has focused its attention on run time adaptation of the UI portion of an interactive system, not on the dynamic adaptation of the interactive system as a whole. The software engineering community is developing several approaches to enable dynamic bindings for service-oriented architectures. For example, Canfora et al. propose the dynamic composition of web services based on BPEL4People (that expresses a task-like model) as well as an extension of WSDL to meta-describe the services and using these two descriptions to generate the corresponding user interface [10]. Although bindings can be performed at run time, users are confined within the workflow designed by the software developers. In addition, the generated UI's are limited to conventional GUI.

One promising approach to support flexibility at run time, is to consider the functional core components as well as UI components as services. In Ethylene, UI components adhere to this philosophy. They can be implemented in very different technologies, they can be discovered and recruited on the fly based on their meta-description, they can be transformed on the fly. On the other hand, the business logic side of interactive systems is left opened. CRUISe [28] aims at supporting both sides in a uniform way, but applies to the dynamic composition of web services and UI composition for the web [38].

CONCLUSION

Model-Driven Engineering has provided the HCI community with useful concepts for framing its own research agenda. Additional research is required for the definition of meta-models, transformations and mappings provided that high-level descriptions can take full advantage of the latest innovations at the FUI level. Models at design time should not disappear at run time, but should be available to go beyond cosmetic adaptation. Design phase and run-time phase equal "même combat!"

Maximum flexibility and quality should be attainable by modeling the business logic as well as the user interface as services with their own domain of plasticity. UI components should not be pure executable code. They have to be meta-described to express their exact nature and contracts with a human-centered perspective. They can be retrieved, transformed, and recomposed on the fly thanks to a tier middleware infrastructure. This middleware, which supports context, dynamic discovery as well as the dynamic (re)composition of business logic and of transformable UI components, will permit interactive systems to go beyond their domain of plasticity. However, we must be careful at keeping the user in the loop while being able to produce transition user interfaces automatically.

The risk is that this wonderful apparatus will be designed for the specialists. We need to put the power in the people's hands and explore the potential from social programming. The success of the Apple App Store is a good indication for this. Mash-up tools have also started this trend for composing web-based applications (e.g., Google Gadgets or Yahoo! Widgets). More collaboration should be developed with the "cloud computing crowd". After all, an interactive space is a mini-cloud. If interaction resources were virtualized as memory, network and computing resources are currently envisioned by the "systemers", then this would simplify enormously the development of user interfaces. IAM [19] was an early attempt in this direction.

In short, MDE is an important tool for adaptation as long as it does not block creativity.

ACKNOWLEDGMENTS

I thank the ANR CONTINUUM project (ANR-08-VERS-005) as well as the ITEA 08026 UsiXML project for supporting this work. Special thanks to Alexandre Demeure for the implementation of the Comets, and to Lionel Balme for the implementation of Ethylene.

REFERENCES

1. Balme, L. Interfaces homme-machine plastiques : une approche par composants dynamiques. Thèse de doctorat Informatique préparée au Laboratoire d'Informatique de Grenoble (LIG), Université Joseph Fourier, 20 juin 2008, 240 pages
2. Balme, L., Demeure, A., Barralon, N., Coutaz, J., Calvary, G. CAMELEON-RT: a Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces, second European Symposium on Ambient Intelligence, EUSAI 04, EUSAI 2004, LNCS 3295, Markopoulos et al. Eds, 291-302
3. Balakrishnan, R., Baudisch, P. Special Issue on Ubiquitous Multi-Display Environments, Human-Computer Interaction, 2009, Vol. 24, Taylor and Francis publ.
4. Berti, S. & Paternò F. (2005). Migratory multimodal interfaces in multidevice environments. In Proc. International Conference on Multimodal Interfaces (ICMI 05), ACM Publ., 92-99
5. Bézivin, J., Dupé, G., Jouault, F., Pitette, G. & Rougui, J. First Experiments with the ATL Transformation Language: Transforming XSLT into Xquery. OOPSLA Workshop, Anaheim California USA (2003)
6. Bolt, R. "Put That There": Voice and gesture at the graphics interface. In Proc. of the 7th International Conf.

on Computer Graphics and Interactive techniques, ACM Publ. (1980), 262-270

7. Bouillon, L., Vanderdonckt, J., Souchon, N., "Recovering Alternatives Presentation Models of a Web Page with Vaquita", In: Proceedings of 4th International Conference on Computer-Aided Design of User Interfaces, CADUI 2002, Kluwer Academics Pub., p. 311-322, Valenciennes, France, May 2002
8. Bourguin, G., Lewandowski , A., Tarby, J.-C. Defining Task Oriented Component. In Proc. TAMODIA 2007, Lecture Notes in Computer Science 4849 Springer 2007, ISBN 978-3-540-77221-7, (2007) 170-183
9. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Souchon, N., Bouillon, L., Vanderdonckt, J.: Plasticity of User Interfaces: A Revised Reference Framework. 1rst International Workshop on Task Models and Diagrams for User Interface Design TAMODIA'2002, (2002) 127-134
10. Canfora, G., Di Penta, M., Lombardi, P., Villani, M.L. Dynamic Composition of Web Applications in Human centered Processes. IEEE PESOS'09, May 18-19, (2009)
11. Coninx, K., Luyten, K., Vandervelpen, C., Van den Bergh, J. & Creemers, B. (2003). Dygimes: Dynamically Generating Interfaces for Mobile Computing Devices and Embedded Systems. In Proc. Mobile HCI, 256-270.
12. Coutaz, J., Nigay, L., Salbert, D., Blandford, A., May, J., "Four easy pieces for assessing the usability of multimodal interaction: the CARE properties", In: Proceedings of the IFIP Interantional Conference on Human-Computer Interaction, INTERACT 1995, p. 115-120, Lillehammer, Norway, (1995)
13. Coutaz, J. Meta User Interfaces for Ambient Spaces. In Proc. TAMODIA 2006, 5th International Workshop on Task Models and Diagrams for User Interface Design TAMODIA'2006, (2006), Springer Verlag publ.
14. Coyette, A., Faulkner, S., Kolp, M., Limbourg, Q., Vanderdonckt, J., "SketchiXML: towards a multi-agent design tool for sketching user interfaces based on USIXML", In: Proceedings of the 3rd Annual Conference on Task Models and Diagrams, TAMODIA 2004, Prague, Czech Republic, (2004)
15. Crease, M., Brewster, S.A., Gray, P., "Caring, Sharing Widgets: a toolkit of sensitive widgets", In: Proceedings of the 14th Annual Conference of the British HCI Group, BCS-HCI 2000, Springer, pp 257-270, Sunderland, UK, September 2000ß Past, Present, and Future of User Interface Software Tools. Transactions on Computer-Human Interaction (TOCHI), ACM Publ., Vol 7(1), (2000) 3-28
16. Demeure, A., Calvary, G., Koninx, K. COMET(s), a Software Architecture Style and an Interactors Toolkit for Plastic user Interfaces. In Proc. 15th International Workshop, DSV-IS 2008, T.C.N. Graham & P. Palanque (Eds), Lecture Notes in Computer Science 5136, Springer Berlin / Heidelberg, Kingston, Canada, (2008), 225-237
17. Ferry, N. Hourdin, G., Lavirotte, S., Rey, G., Tigli, J.-Y., Riveill, M. Models at Runtime: Service for Device Composition and Adaptation. In 4th International Workshop Models@run.time, Models 2009 (MRT'09), (2009)
18. Gajos, K., Wobbrock, J., and Weld, D. Improving the performance of motor-impaired users with automatically-generated, ability-based interfaces. In CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems, pages 1257-1266, New York, NY, USA, ACM (2008)
19. Hayes, P.J., Szekely, P. & Lerner, R.A. Design alternatives for user interface management systems based on experience with COUSIN. In Proc. Of the ACM Conf. on Human Factors in Computing Systems (CHI'85, San Francisco, CA, Apr. 14-18), (1985) 169-175
20. Lachenal, C., Rey, G., Barralon, N. MUSICAE, an infrastructure for MUlti-Surface Interaction in Context Aware Environment. In Proc. HCI International, Crete, (2003), 125-126
21. Mens, T., Czarnecki, K. & Van Gorp, P. A taxonomy or Model Transformations. Dagstuhl Seminar Proc04101. (2005) http://drops.dagstuhl.de/opus/volltexte/2005/11
22. Mistry, P., Maes, P. SixthSense – A Wearable Gestural Interface. In Proc. SIGGRAPH Asia 2009, Emerging Technologies, Yokohama, Japan (2009)
23. Myers, B. Creating User Interfaces using programming by example, visual programming, and constraints. ACM Transaction on Programming Languages and Systems (TOPLAS), Vol. 12 (2) (1990), ACM Publ., 143-177
24. Myers, B., Park, S.Y., Nakano, Y., Mueller, G., Ko, A. How designers Design and Program Interactive Behaviors. In Proc. IEEE Symposium on Visual Languages and Human Centric Computing (VL/HCC) (2008) 177-184
25. Newman, M. W., Sedivy, J. Z., Neuwirth, C. M., Edwards, W. K., Hong, J. I., Izadi, S., Marcelo, K., Smith, T. F., "Designing for Serendipity: Supporting End-User Configuration of Ubiquitous Computing Environments", In: Proceedings of Designing Interactive Systems, DIS 2002, p. 147-156, London, UK, (2002).
26. Nichols, J., Rothrock, B., Chau, D. H., Myers, B. A., "Huddle: Automatically Generating Interfaces for Systems of Multiple Connected Appliances," In: Proceedings of the 19th Annual ACM Symposium on User interface Software and Technology, UIST 2006, p. 279-288, Montreux, Switzerland, (2006).

27. Oreizy, P., Gorlick, M., Taylor, R., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D., Wolf, A.: An Architecture-Based Approach to Self-Adaptive Software", IEEE Intelligent Systems, May/June (1999) 54-62

28. Pietschmann, S., Voigt, M., MeiBner, K. Dynamic Composition of Service-Oriented Web User Interfaces. Proc. of the 4th International Conf. on Internet and Web Applications and Services, ICIW 2009, IEEE CPS, ISBN 9780769536132, (2009)

29. Ponnekanti, S., Lee, B., Fox, A., Hanrahan, P. & Winograd, T. Icrafter: a Service Framework for Ubiquitous Computing Environments. In Proc. Ubicomp 2001, G. Abowd, B. Brumitt, S. Shafer Eds., Springer Publ., LNCS 2201, (2001) 57-75

30. Reignier, P. Brdiczka, O., Vaufreydaz, D., Crowley, J.L., Maisonnasse, J. Contexte-Aware Environments: from Specification to Implementation. Expert Systems: The Journal of Knowledge Engineering (2007)

31. Smith, D. C. Pygmalion: An executable Electronic Blackboard. Chapter1 In "Watch What I Do", A. Cypher ed., The MIT Press (1993)

32. Sottet, J.-S., Calvary, G., Favre, J.-M.: Towards Model Driven Engineering of Plastic User Interfaces. International workshop on Model Driven Development of Advanced User Interfaces (MDDAUI), MoDELS 05 (2005)

33. Sottet, J.-S., Calvary, G., Coutaz, J., Favre, J.-M. A Model-Driven Engineering Approach for the Usability of User Interfaces. In Proc. Engineering Interactive Systems (EIS2007), J. Gulliksen et al. (eds), LNCS 4940, (2007), 140-157

34. Sousa, J.P., Garlan, D., "The Aura Software Architecture: an Infrastructure for Ubiquitous Computing", In: Carnegie Mellon Technical Report, CMU-CS-03-183, (2003)

35. Taleb, M., Seffah, A., Abran, A. Interactive Systems Engineering: A Pattern-Oriented and Model-Driven Architecture. In Software Engineering Research and Practice (2009), 636-642.

36. Tandler, P., "Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogeneous Devices", In: Proceedings of UBICOMP 2001, LNCS 2201, p. 96-115, Atlanta, GA, USA, (2001).

37. Thevenin, D., Coutaz, J.: Plasticity of User Interfaces: Framework and Research Agenda. In Proc. Interact99, Edinburgh, A. Sasse & C. Johnson Eds, IFIP IOS Press, (1999) 110-1172002.

38. Yu, J. Benatallah, B., Saint-Paul, R., Casati, F., Daniel, F. M., Matera. A Framework for rapid Integration of Presentation Components. In WWW'07 Proc. of the 16th International Conf. on World Wide Web (2007) 923-932