Generating Abstract User Interfaces from an Informal Design

Adrien Coyette *Jean Vanderdonckt* *Stéphane Faulkner* *Manuel Kolp*

Université Catholique de Louvain, School of Management (IAG)
Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)
{Coyette, Vanderdonckt, Faulkner, Kolp}@isys.ucl.ac.be
www.isys.ucl.ac.be/staff

Abstract. Sketching activities are widely adopted during early design phases of user interface development to convey informal specifications of the interface presentation and dialog. Designers or even end users can sketch parts or whole of the future interface they want. With the ever increasing availability of different computing platforms, a need arises to continuously support sketching across these platforms having various programming languages, interface development environments and operating systems. To address the needs along these dimensions that pose new challenges to user interface sketching tools, SketchiXML is a multi-platform multi-agent interactive application enabling designers and end users to sketch user interfaces with different levels of details and support for different contexts of use. The results of the sketching are then analyzed to produce interface specifications independently of any context, including user and platform. These specifications are exploited to progressively produce one or many interfaces, for one or many users, platforms, and environments.

1 Introduction

Designing the right User Interface (UI) the first time is very unlikely to occur. Instead, UI design is recognized as a process that is [12] eminently *open* (new considerations may appear at any time), *iterative* (several cycles are needed to reach an acceptable result), and *incomplete* (not all required considerations are available at design time). Consequently, means to support early design of UI has been extensively researched [13] to identify appropriate techniques such as paper sketching, prototypes, mock-ups, diagrams, etc. Most designers consider hand sketches on paper as one of the most effective ways to represent the first drafts of the future UI [1,8]. Indeed, this kind of unconstrained approach presents many advantages: sketches can be drawn during any design stage, it is fast to learn and quick to produce, it allows the sketcher to focus on basic structural issues instead of unimportant details (e.g., exact alignment, typography, and colors), it is very appropriate to convey ongoing, unfinished designs, it encourages creativity, sketches can be performed collaboratively between designers and end-users. Even more, the end user may herself produce some sketches to initiate the development process and when the sketch is close enough to the expected UI, an agreement can be signed between the designer and the end user, thus facilitating the contract and validation. Van Duyne *et al.* [13] reported that creating a low-fidelity UI prototype (such as UI sketches) is at least 10 to 20 times easier and faster than its equivalent with a high-fidelity prototype (such as produced in UI builders). The idea of developing a computer-based tool for sketching UIs naturally emerged from these observations [6,11]. Such tools would add on top of the advantages provided by sketching techniques a wide range of advantages: easily creating, deleting, updating or moving UI elements, thus encouraging checking and revision, typical activities in the design process [12]. Some research was carried out in order to propose a hybrid approach taking the best of the hand-sketching and computer assisted interface design, but this wedding made some shortcomings preeminent:

- Some sketching tools only support the sketching activities without producing any output: when the designer and the end user agreed upon a sketch, a contract can be signed between them and the development phase can start from the early design phase, but when the sketch is not transformed, the effort is lost.
- Sketching tools that recognize the drawing do produce some output, but not in a reusable format: the design output is not necessarily in a format that is directly reusable as development input, thus forbidding reusability.
- Sketching tools are bound to a particular programming language, a particular UI type, a particular computing platform or operating system: when an output is produced, it is usually bound to one particular environment, therefore preventing developers

to reuse sketches from one case to another, such as for various platforms.

- Sketching tools do not take into account the sketcher's preferences: as they impose the same sketching scheme, the same gestures for all types of sketchers, a learning curve may prevent these users to learn the tool and efficiently use it.
- Sketching tools do not allow a lot of flexibility in the sketch recognition: the user cannot choose when recognition will occur, thus contradicting the openness [12] and when this occurred, it is difficult to come back to a previous state.

To unleash the power of informal UI design based on sketches, there is a need to address the above shortcomings observed on existing UI sketching tools. It is expected that in this way, UI sketching will be lead to its full potential. SketchiXML consists of a new informal prototyping tool solving *all* these shortcomings, allowing the designer to sketch the user interfaces as easily as on paper. In addition, SketchiXML provides the designer with on-demand design critique and assistance during early design. Instead of producing code that is peculiar to a particular case or environment, SketchiXML generates UI specifications written in UsiXML (User Interface eXtensible Markup Language – www.usixml.org), a platform-independent User Interface Description Language (UIDL) that will be in turn exploited to produce code for one or several UIs, for one or many contexts of use simultaneously.

The structure of the paper is the following: section 2 proves that state-of-the-art UI sketching tools all suffer from some of the above shortcomings. Section 3 reports on an experimental study conducted to identify the sketchers' preferences, such as the most preferred and appropriate UI representations. These results feed the development of SketchiXML in Section 4, where these widgets are recognized on demand. The multi-agent architecture of SketchiXML is outlined to support various scenarios in different contexts of use with examples. Section 5 discusses some future work and concludes.

2 Related work

UI prototypes usually fall into three categories depending on their degree of fidelity, that is the precision to which they reproduce the reality of the desired UI.

The *high-fidelity* (Hi-Fi) prototyping tools denote software allowing to build a UI that looks complete, and might be usable. Moreover, this kind of software is equipped with a wide range of editing functions for all UI widgets: erase, undo, move, specify physical attributes, etc… This software allows the designer to build a complete GUI from which is produced an accurate image (e.g., Adobe Photoshop, PowerPoint) or the code in a determined programming language (e.g., Visual Basic, DreamWeaver). Even if the final result is not executable, it can also be considered as a high fidelity tool given that the result provided looks complete.

The *medium-fidelity* (Me-Fi) consists in building a UI mock-up giving importance to the content, but keeping secondary all the information regarding the typography, color scheme or others minor details. A typical example is Microsoft Visio where only the type, the size and the contents of UI widgets can be specified graphically.

The *low-fidelity* (Lo-Fi) drafting tools are used to capture the general information needed to obtain a global comprehension of what is desired, keeping all the unnecessary details out of the process. The most standard approaches for Lo-Fi prototyping are the "paper and pencil technique", the "whiteboard/blackboard and post-its approach" [13]. Such approaches provide access to all the components, and prevent the designer from being distracted from the primary task of design. Research shows that designers who work out conceptual ideas on paper tend to iterate more and explore the design space more broadly, whereas designers using computer-based tools tend to take only one idea and work it out in detail [6,11,12]. Many designers have reported that the quality of the discussion when people are presented with a Hi-Fi prototype was different than when they are presented with a Lo-Fi mock up. When using Lo-Fi prototyping, the users tend to focus on the interaction or on the overall site structure rather than on the color scheme or others details irrelevant at this level [13].

Consequently, Lo-Fi prototyping offers a clear set of advantages compared to the Hi-Fi perspective, but at the same time suffers from a lack of assistance. For instance, if several screens have a lot in common, it could be profitable to use copy and paste instead of rewriting the whole screen each time. The combination between these approaches appears to make sense, as long as the Lo-Fi advantages are maintained. This consideration basically initiated two software families: tools allowing to sketch the UI and to represent the scenarios between them, with or without any code generation.

DENIM [6] helps web site designers during early design by sketching information at different refinement levels, such as site map, story board and individual page, and unifies the levels through zooming views. DEMAIS [1] is similar in principle, but aimed at prototyping interactive multimedia applications. It is made up of an interactive multimedia storyboard tool that uses a designer's ink strokes and textual annotations as an input design vocabulary. Both DENIM and DEMAIS use pen input as a natural way to sketch on

screen, but do not produce any final code or other output.

In contrast, SILK [8], JavaSketchIt [2] and Freeform [10,11] are major applications for pen-input based interface design supporting code generation. SILK uses pen-input to draw GUIs and produce code for OpenLook operating system. JavaSketchIt proceeds in a slightly different way than Freeform, as it displays the shapes recognized in real time, and generates Java UI code. JavaSketchIt uses the CALI library [6] for the shape recognition, and widgets are formed on basis of a combination of vectorial shapes. The recognition rate of the CALI library is very high and thus makes JavaSketchIt easy to use, even for a novice user. Freeform only displays the shapes recognized once the design of the whole interface is completed, and produces Visual Basic 6 code. The technique used to identify the widgets is the same than JavaSketchIt, but with a slightly lower recognition rate. Freeform also supports scenario management thanks to a basic storyboard view similar to that provided in DENIM.

SketchiXML's main goal is to combine in a flexible way the advantages of the tools just presented into a single application, but also to add new features for this kind of application. Thus SketchiXML should: produce UI specifications and generate from them several UI codes to avoid binding with a particular environment and to foster reusability; support UI sketching with recognition and translation of this sketching into UI specifications in order not to loose the design effort; support sketching for any context of use instead of only one platform, one context; be based on UI widget representations that are significant for the designer and/or the end-user; and perform sketch recognition at different moments, instead of at an imposed moment.

Others vital facilities to be provided by SketchiXML are the possibility to handle input from different sources, such as direct sketching on a tablet or a paper scan, and the possibility to receive real time advice on two types of issues, if desired: the first type of advice occurs in a post-sketching phase, and provides a set of usability advice based on the UI drawn. For the second type of advice, the system operates in real time, looking for possible patterns, or similarities with previously drawn UIs. The objective of such an analysis is to supplement the sketching for the designer when a pattern is detected. Since the goal of SketchiXML is to incite designers to be creative and to express evaluative judgments, we infer the rules enunciated in [12] to the global architecture, and let the designer parameterizes the behavior of the whole system through a set of parameters (Section 3).

3 SketchiXML Development

In the previous sections, we have introduced the different requirements to be met in SketchiXML. The application has to, amongst other things, carry out shape recognition, provide spatial shape interpretation, provide usability advice, handle several kinds of inputs, generate UsiXML specifications, and operate in a flexible way.

On basis of these requirements, we have considered that a BDI (Belief-Desire-Intention) agent-oriented architecture was particularly judicious. Indeed, such architectures allow to build robust and flexible applications by distributing the responsibilities among autonomous and cooperating agents. In that situation each of the agents is in charge of a specific part of the process, and cooperate together in order to provide the service required according to the designer's preferences. This kind of approach presents the advantage of being more flexible, modular and robust than traditional architecture including object-oriented ones [5].

3.1 SketchiXML Architecture

Fig. 1 models the SketchiXML architecture using *i** [14]. *i** is a graph, where each node represents an *actor* (or system component) and each link between two actors indicates that one actor depends on the other for some goal to be attained. A dependency describes an "agreement" (called *dependum*) between two actors: the *depender* and the *dependee*. The *depender* is the depending actor, and the *dependee,* the actor who is depended upon. The type of the dependency describes the nature of the agreement. *Goal* dependencies represent delegation of responsibility for fulfilling a goal; *softgoal* dependencies are similar to goal dependencies, but their fulfillment cannot be defined precisely; task dependencies are used in situations where the dependee is required.

When a user wishes to create a new project, he contacts the *Broker* agent, which serves as an intermediary between the external actor and the organizational system. The *Broker* queries the user for all the relevant information needed for the process, such as the target platform, the input type, the intervention strategy of the *Adviser* agent,... According to the criteria entered, the coordinator chooses the most suitable handling and coordinates all the agents participating in the process in order to meet the objectives determined by the user.

For clearness, the following section only considers a situation where the user has selected real time recognition, and pen-input device as input. So, the *Data Editor* agent then displays a white board allowing the user to draw his hand-sketch interface. All the strokes are collected and then transmitted to the *Shape Recognizer*

agent for recognition. The recognition engine of this agent is based on the CALI library [5], a recognition engine able to identify shapes of different sizes, rotated at arbitrary angles, drawn with dashed, continuous strokes or overlapping lines. Subsequently, the *Shape Recognizer* agent provides all the vectorial shapes identified with relevant information such as location, dimension or degree of certainty associated to the *Interpreter* agent. Based on these shape sets, the *Interpreter* agent attempts to create a component layout. The technique used for the creation of this layout takes advantage of the knowledge capacity of agents. The agent stores all the shapes identified is his belief, and each time a new shape is received all the potential candidates for association are extracted. Using its set of patterns the agent then evaluates if the shape couples forms a widget or a sub-widget. The conditions to be tested are based on a set of fuzzy spatial relations allowing to deal with imprecise spatial combinations of geometric shapes and to fluctuate with user preferences.

Based on the widgets identified by the *Interpreter*, the *Adviser* agent assists the designer with the conception of the UIs in two different ways. Firstly, by providing real-time assistance to the designer by attempting to detect UI patterns in the current sketch in order to complete the sketch automatically. Secondly in a post operational mode, the usability adviser provides usability advice on the interface sketched.

If the *Interpreter* fails to identify all the components or to apply all the usability rules, then the *Ambiguity Solver* agent is invoked. This agent evaluates how to optimally solve the problem according to the initial parameters entered by the user. The agent can either attempt to solve the ambiguity itself by using its set of disambiguation algorithms, or to delegate it to a third agent, the *Graphical Editor* agent. The *Graphical Editor* displays all the widget recognized at this point, as a classical element-approach software, and highlights all the components with a low degree of certainty for confirmation. Once one of the last three agents evoked considers the degree of certainty associated to the overall widget layout sufficient, the user interface is transmitted to the *XML Parser* agent for UsiXML generation

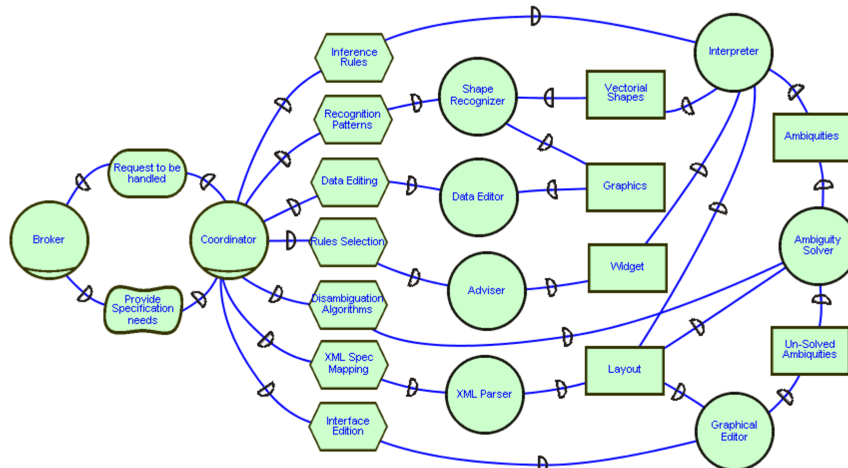

Fig. 1. i* representation of SketchiXML architecture

3.2 SketchiXML prototype

As described in the previous section, the first step of the process is the gathering of all the information needed for the process.

Fig. 2. Settings interface.

Fig. 2 displays the settings interface where the designer has to provide all the parameters for the instance. Here, Fig. 2 depicts a situation where the designer wants to obtain all the advice during the process, but does not want the recognition engine to disturb him with real-time recognition. The UsiXML parsing is set on fully manual mode, and the output quality is set on medium quality. The quality level

affects the way the agents consider a widget layout to be acceptable, or the constraints used for the pattern matching between vectorial shapes. The sketching phase in that situation is thus very similar to the sketching process of an application such as Freeform. Of course, the designer is always free to re-parameterize the system while the process is running, or to execute it manually.

Figure 3 illustrates the SketchiXML workspace configured for designing a user interface for a standard personal computer. On the first figure we can observe that shape recognition is disabled as none of the sketches is interpreted, and the widget layout generated by the *Interpreter* agent remains empty. The second figure represents the same user interface with shape recognition and interpretation.

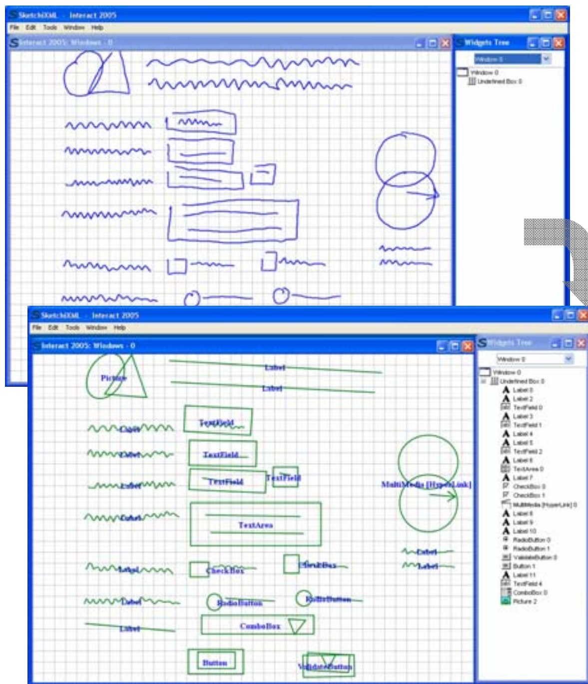

Fig. 3. SketchiXML workspace

Figure 4 depicts SketchiXML parameterized for another context of use, a pda, and its importation to GrafiXML[9]. We can observe that shape recognition is activated, each time a new widget is identified the color of the shapes turns to green, and the widget tree generated by the *Interpreter* is updated.

Changing the context has a deep impact on the way the system operates. As an example, when a user builds a user interface for one platform or another, adaptations need to be reflected on the design knowledge that should be used for evaluation, by selecting and prioritizing rule sets [12], and on the set of available widgets. As the size of the drawing area is changing, the set of constraints used for the interpretation needs to be tailored too, indeed if the average size of the strokes drawn is much smaller than on a standard display, the imprecision associated with each stroke follows the same trend. We can thus strengthen the constraints in order to avoid confusion.

Once the design phase is complete, SketchiXML parses the informal design to UsiXML specification [3]. Each widget is represented with standard values for its attributes, as SketchiXML is only aimed at capturing the core properties of the interface. Additionally, the UsiXML specification integrates all the information related to the context. As UsiXML allows to define a set of transformation rules for switching from one of the UsiXML models to another, or to adapt a model for another context, such information is thus required.

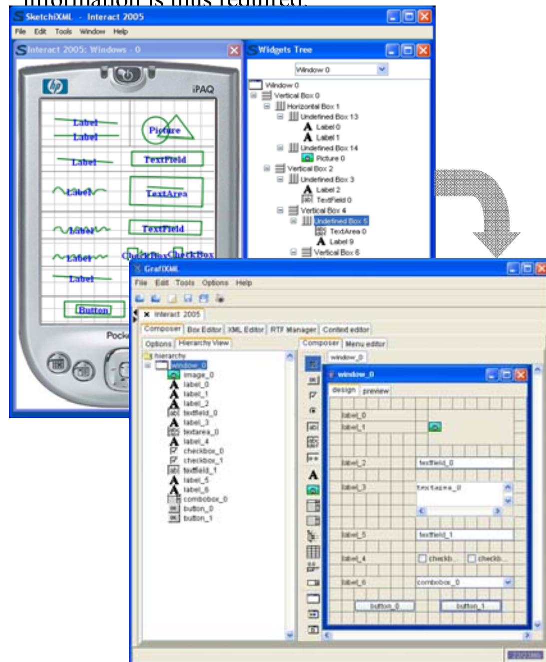

Fig.4. SketchiXML workspace configured for a PDA and its importation to GrafiXML

Figure 4 illustrates the SketchiXML output imported in GrafiXML, a high fidelity user interface graphical editor. On basis of the informal design provided during the early design, a programmer can reuse the output without any loss of time in order to provide a complete revised version of the user interface with all the characteristic that cannot and must not be defined during the early design phase. This contrasts with a traditional approach, where a programmer had to implement the user interfaces on basis of a set of blackboard photographs or sheets of paper, and thus start the implementation process from the beginning.

4 Future Work and Conclusion

Even if the SketchiXML prototype integrates a wide set of features, many evolutions have to be done. Out of many ideas, three major ones retain our attention. One of the biggest drawbacks of the current version of SketchiXML is the lack of scenario editor. Capturing such information is very profitable, and is quite natural to represent, even for a novice designer. Moreover such information can be directly stored in the UsiXML model, and can be re-used just as easily as the code generated for each user interface. A second high potential evolution consists in developing an evolutionary recognition engine. SketchiXML uses the CALI library and a set of spatial constraints between the vectorial shapes recognized to build the widget. Even if the recognition rate is very high, the insertion of new widget representation is restricted to a combination of the set of vectorial shape supported. To this aim, research in the biometric domain such as handwriting recognition [2], could provide valuable answers, taking full advantage of the multi-agent architecture.

During the sketching process, the possibility to have a runable overview of the current project would be useful. Extensions could be developed in order to invoke external interpreters directly from SketchiXML. Interpreters already exist for Flash, Java, XHTML and Tcl-Tk.

So, with SketchiXML we have introduced a new and innovative tool. Firstly, SketchiXML is the first informal design tool that generates a platform and environment independent output and thus provides a solution to the language neutrality weakness of existing approaches. Secondly, the application is based on a BDI multi-agent architecture where each requirement is assumed by an autonomous and collaborative agent part of an organizational system. Based on the criteria provided by the designer at the beginning of the process, the experts (agents) adapt the way they act and interact with the designer and the other agents in order to meet the global objectives.

Eventually, SketchiXML extends a set of tools allowing to start the design process from the early design phase to the final concrete user interface, with tools supporting every stage. The complete widgets catalogue, screen shots, demonstration of SketchiXML and implementation are available at http://www.usixml.org/. SketchiXML is developed in Java, on top SKwyRL-framework [7] and JACK Agent platform, with recognition based on CALI library [5].

Acknowledgements

We gratefully acknowledge the support of the Request research project under the umbrella of the WIST (Wallonie Information Société Technologies) program under convention n°031/5592 RW REQUEST). We also warmly thank J. A. Jorge, Filipe M. G. Pereira and A. Caetano for allowing us to use JavaSketchIt in our research.

References

1 Bailey, B.P., Konstan, J.A.: Are Informal Tools Better? Comparing DEMAIS, Pencil and Paper, and Authorware for Early Multimedia Design. In: Proc. of the ACM Conf. on Human Factors in Computing Systems CHI'2003. ACM Press, NY (2003) 313–320

2 Caetano, A., Goulart, N., Fonseca, M., Jorge, J.: JavaSketchIt: Issues in Sketching the Look of User Interfaces. In: Proc. of the 2002 AAAI Spring Symposium - Sketch Understanding (Palo Alto, March 2002). AAAI Press (2002) 9–14

3 Coyette, A., Faulkner S., Kolp, M., Vanderdonckt, J., Limbourg, Q.: SketchiXML: Towards a Multi-Agent Design Tool for Sketching User Interfaces Based on USIXML. In: Proc. of the 3rd Int. Workshop on TAsk MOdels and DIAgrams for user interface design TAMODIA'2004 (Prague, November 2004). ACM Press, New York (2004) 75–82

4 Faulkner, S.: An Architectural Framework for Describing BDI Multi-Agent Information Systems. Ph.D. Thesis, UCL-IAG, Louvain-la-Neuve (May 2004)

5 Fonseca, M.J., Jorge, J.A.: Using Fuzzy Logic to Recognize Geometric Shapes Interactively. In: Proc. of the 9th Int. Conf. on Fuzzy Systems FUZZ-IEEE'00 (San Antonio, 2000). IEEE Computer Society Press, Los Alamitos (2000) 191–196

6 Hong, J.I., Li, F.C., Lin, J., Landay, J.A.: End-User Perceptions of Formal and Informal Representations of Web Sites. In: Extended Abstracts of CHI'2001, 385–386

7 Kolp, M., Giorgini, P., Mylopoulos, J.: An Organizational Perspective on Multi-agent Architectures. In: Proc. of the 8th Int. Workshop on Agent Theories, Architectures, and Languages ATAL'01 (Seattle, 2001).

8 Landay, J., Myers, B.A.: Sketching Interfaces: Toward More Human Interface Design. IEEE Computer 34, 3 (March 2001) 56–64

9 Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Lopez-Jaquero, V. USIXML: a Language Supporting Multi-Path Development of User Interfaces. In: Proc. of 9th IFIP Working Conf. on Engineering for Human-Computer Interaction EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004). Kluwer Academics, Dordrecht (2004)

10 Plimmer, B.E., Apperley, M. Software for Students to Sketch Interface Designs. In: Proc. of IFIP Conf. on Human-Computer Interaction INTERACT'2003. IOS Press (2003) 73–80

11 Plimmer, B.E., Apperley, M.: Interacting with Sketched Interface Designs: An Evaluation Study. In: Proc. of CHI'04. ACM Press, New York (2004) 1337–1340

12 Sumner, T., Bonnardel, N., Kallag-Harstad, B., The Cognitive Ergonomics of Knowledge-based Design Support Systems. In: Proc. of CHI'97. ACM Press, New York (1997) 83–90
13 van Duyne, D.K., J.A. Landay, and J.I. Hong, The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience. Addison-Wesley (2002).
14 Yu, E.: Modeling Strategic Relationships for Process Reengineering. Ph.D. thesis. Department of Computer Science, University of Toronto, Toronto (1995)