RC24873 (W0908-052) August 14, 2009
Other

IBM Research Report

Flexible Modeling Tools for Pre-requirements Analysis: Conceptual Architecture and Research Challenges

Harold Ossher[1], Rachel Bellamy[1], Ian Simmonds[1], David Amid[2], Ateret Anaby-Tavor[2], Matthew Callery[1], Michael Desmond[1], Jacqueline de Vries[1], Amit Fisher[2], Sophia Krasikov[1]

[1]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

[2]IBM Research Division
Haifa Research Laboratory
Mt. Carmel 31905
Haifa, Israel

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

LIMITED DISTRIBUTION NOTICE: This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties). Copies may be requested from IBM T. J. Watson Research Center , P. O. Box 218, Yorktown Heights, NY 10598 USA (email: reports@us.ibm.com). Some reports are available on the internet at http://domino.watson.ibm.com/library/CyberDig.nsf/home .

Flexible modeling tools for pre-requirements analysis: conceptual architecture and research challenges

Harold Ossher[1], Rachel Bellamy[1], Ian Simmonds[1], David Amid[2], Ateret Anaby-Tavor[2], Matthew Callery[1], Michael Desmond[1], Jacqueline de Vries[1], Amit Fisher[2], Sophia Krasikov[1]

IBM [1]T.J. Watson and [2]Haifa Research Centers

{ossher,rachel,simmonds,mcallery,mdesmond,devries,kras}@us.ibm.com
{davida,atereta,amitf}@il.ibm.com

ABSTRACT

There is a serious tool gap at the very start of the software lifecycle, before requirements are formulated. Pre-requirements analysts gather information, organize it to gain insight, envision alternative possible futures, and present insights and recommendations to stakeholders. They typically use office tools, which give them great freedom, but no help with consistency management, change propagation or migration of information to downstream modeling tools. Despite these downsides to office tools, they are still used in preference to modeling tools, which are too hard to learn and too constraining. This paper introduces the notion of *flexible modeling tools,* which blend the advantages of office and modeling tools. We propose a conceptual architecture for such tools, and outline a series of research challenges to be met in the course of realizing them. We also briefly describe the Business Insight Toolkit (BITKit), a prototype tool that embodies this architecture.

1. INTRODUCTION

Few business software projects truly begin with requirements engineering. Before a decision is even made to embark on a software development project, some form of *business analysis* is conducted. The primary activities at this stage, before requirements are formulated, include gathering information, organizing it to gain insight, envisioning alternative possible futures, and presenting insights and recommendations to stakeholders (when the need for a software system is assumed up front, these activities might be performed as part of requirements engineering). Software engineers and business stakeholders encounter a serious tool problem at this early stage of the software lifecycle.

Office tools are usually used, and for many good reasons, as discussed later. However, these tools are semantics free, are limited in organizing information, and do not have an underlying domain model. Consistency can only be maintained manually, and even small changes can take a lot of effort to propagate. Practitioners reported that entire days are often spent on the mechanics of maintaining consistency, which is not only time-consuming, but also disrupts flow. Migrating results from these tools to downstream modeling tools, which are founded on underlying domain models, is also a manual process, and traceability is usually lost.

Modeling tools, such as for Business Process Models (BPM)[1], are sometimes used even at this early stage. They have the opposite problem to office tools, however: they require conformance to a metamodel, a level of precision typically premature at this stage. A pre-sales consultant told us that almost as much time is spent making up stuff just to remove the error messages as mapping out the process, with no real improvement to the substance.

We argue that pre-requirements analysts need what we call *flexible modeling tools:* tools that blend the advantages of office and modeling tools. Such a tool would allow users to work freely and easily with visual elements, yet be able to attribute semantics to visual characteristics when necessary, enabling automatic construction and maintenance of an underlying model and management of consistency. Given suitable domain-specific definitions, the tool would provide guidance and checking, but without requiring strict conformance to a rigid metamodel. Users would thus be able to move smoothly between informal exploration and modeling with varying degrees of formality. This paper proposes a conceptual architecture for such tools, and outlines a series of research challenges to be met in the course of realizing them.

This architecture arose from our work on the Business Insight Toolkit (BITKit) [19], a prototype tool for business users involved in pre-requirements analysis. We designed and evolved it based on observations of and feedback from such target users. However, we believe that flexible modeling tools have broader applicability, since similar considerations apply in many other domains, especially for users engaged in exploratory activities. We also believe that it would be valuable to incorporate some of the features we describe into standard modeling tools, such as tools for UML-based design, so that the user can work at their desired level of formality. Especially since there are times when less formality is desired for example, during the early, exploratory phase of design.

The contributions of this paper are: (a) identification of a new, open area for tool research, (b) identification of requirements, derived from user studies, for tools supporting pre-requirements analysis, (c) a conceptual architecture for tools that integrates key benefits of office and modeling tools, enabling users to move smoothly between exploration and modeling (the novelty lies primarily in the way known architectural elements are brought

[1] http://www.bpmn.org/

together to achieve this), and (d) identification of research challenges in this area.

Section 2 briefly explores the tooling challenges faced by pre-requirements analysts, and the extent to which these are met by office and modeling tools, respectively. We highlight the features of office tools that need to be introduced into modeling tools to provide the requisite flexibility. Section 3 describes our conceptual architecture for flexible modeling tools, showing how the advantages of office and modeling tools can be blended. Section 4 briefly describes the BitKit prototype and our experience with it, and section 5 discusses related work.

2. PRE-REQUIREMENTS ANALYSIS

To understand the work of pre-requirements analysts we held a three-day workshop in June 2007 for 12 IBM Business Architects. It was supplemented by an email "shadowing" of a 3-month data governance engagement with a Senior Business Analyst, as well as an on-site observation of a one month engagement. At the workshop, attendees were asked to recount three types of pre-requirements engagements: one that had been a success resulting in insights for the customer, one that had failed, and one that had been a near-miss or last-minute save. By asking Business Architects to relate stories from actual engagements, the workshop focused on their life in the field—rather than 'by the book' versions.

We found that pre-requirements analysts typically work alone or in small teams, at the site of the business being analyzed. In the case of teams, the members work closely together to reach a shared understanding of the situation. They often use ad hoc concepts, diagrams and notations that they explain to one another, and that are defined, polished or changed before presentation to broader audiences. The analysts themselves are typically business people, with a deep understanding of business issues and varying degrees of technical expertise. They are usually not the same people who elicit detailed requirements or architect or design IT systems, and hence they are not used to using modeling and development tools designed to support the software lifecycle.

A central finding of this study is that pre-requirements analysts work with a holistic perspective including ideas from business and, to a lesser extent, from IT. They gather information relevant to a business or pre-identified business problem, and then organize and make sense of it to identify business issues and potential solutions. A presentation is created that is used to communicate to a client proposed solutions and their value to the client's business. Thus pre-requirements analysts help their clients frame business problems and explore a variety of feasible solutions. Their role is one of *envisioning* that often results in business *transformation.* The process is *exploratory.*

We also learned about the central role of narratives in structuring the work of pre-requirements analysts. The final outcome of their work is a presentation or report, containing a variety of tables, business process diagrams, organizational diagrams, as-is system diagrams, etc. While the creation of this narrative is an exploratory process, consultants like to be presentation-ready as soon as possible, even if that presentation is just a temporary state in their work towards the final version. Template reuse is a common working style, especially for consultants who apply their accumulated experience and expertise to address similar issues for a series of clients. Templates help organize findings into pre-defined categories and representations that have been found to be helpful in past engagements; and also as a way of reusing narrative structure in presentations. Often the template will be adapted to fit the needs of a particular engagement, such as by changing styles and vocabulary to fit the storyline, taste and culture of the client.

Without exception, the analysts prefer to use office tools rather than modeling tools for their work. To understand more about why, we conducted an additional study focusing on the user experience provided by these tools. Using a customized version of the Cognitive Dimensions questionnaire [5], and follow-up phone interviews, we interviewed five business consultants who use these tools for their day-to-day work. All but one reported on more than one tool. Two of the consultants reported on a modeling tool, two reported on a diagramming tool (an office tool supporting structured graphics, e.g. Visio) and four reported on a presentation tool. Table 1 summarizes the advantages of each type of tool. Given the small number of participants, these results are suggestive rather than conclusive.

All the interviewees prefer to use office tools where possible, even when creating technical diagrams. For example, depictions of business models are produced using a special-purpose diagramming tool in preference to using a modeling tool. One interviewee commented this was because their immediate need was communication and not detailed modeling: "*you got lots of details in the modeler model, and you just got the pictorial presentation with* [a diagramming tool]*, but that was sufficient for communicating the business process. You didn't need all the details that were in the* [modeling tool] *model."* Two interviewees commented that they did not want to take time to learn the features of the modeling tool. In comparison, the "*learning curve was minimal for getting to the ability to diagram*" using the diagramming tool.

The other major finding is that modeling tools cause premature commitment as they do not allow people to work either at the level of detail or in the order they choose: "... [it] *makes me be too rigorous, when I want to be sloppy."* Due to enforcement of the metamodel, people are forced to take care of details before they are ready: *"It tends to force compliance with standards, driving you into the detail before the bigger picture is sorted out."*

Office tools on the other hand have no metamodel, in fact they do not support semantics at all: *"..it* [diagramming tool] *is really agnostic about what terms 'mean'. They are just elements on a page and carry no metadata apart from style."* It is this lack of semantics that makes them easy to learn, and enables their broad applicability. It also means that they do not constrain what the user can express nor do they force a particular order of development on the user. The user is able to evolve the diagram or presentation at the level of detail, and in the order they choose, as their understanding of what they need to represent evolves: "[in office tools] *you have to put something in 'a' format or place, it has to go somewhere to begin with. I've never found that much of a problem because I can always move it or format it how I want when I have decided what the overall structure or content is going to be."*

Office Tools	Modeling Tools
• Have broad applicability • Are easy to learn • Presentation tools provide narrative structuring in the same medium as final presentation • Don't constrain development order • Provide multiple stylistic cues • Provide just syntax	• Support multiple views on the same model • Facilitates consistency management across multiple views • Provide domain-specific guidance • Provide syntax, semantic model and semantic mapping

Table 1: Advantages of office vs. modeling tools

Working in the medium used to present is important, and is one reason interviewees reported that they like to do even their preliminary work using presentation tools. Being in the presentation medium helps them think through the final form as they are creating the story: "*There is an element of rehearsal ... as you are selecting the elements and the sequence*...[I ask myself] *what level of detail do I need to go into, am I conveying the message? Am I teaching, or persuading?*"

However, analysts also mentioned downsides to office tools. One interviewee talked about how lack of a detailed model meant that there was a disconnect between the pre-requirements business case and the detailed models used for defining the system downstream. This causes severe problems when changes downstream need to be propagated back upstream, as there is no easy way to update the presentations, reports and diagrams produced using office tools, so they are not updated, and get out-of-date: *"...at that point you are beyond the* [diagramming tool] *and you don't go back and change the model. Then the docs get out of sync..."*

Office tools are *viscous*: a single change to the style or terminology of an element may require many actions to maintain consistency throughout the document. Managing consistency across documents and within large documents was a problem raised by every person interviewed: *"Difficult to maintain continuity of thought and to ensure that if an element occurs many times in the document they are all kept in sync."* A lot of time was wasted making low-level changes, and ensuring they were appropriately propagated throughout a presentation and the multiple files associated with it: *"Time-consuming to arrange all the different elements and ensuring consistent styles are used throughout."*

Such consistency issues are typically dealt with in modeling tools by supporting semantics. The visual elements in both an office tool and a modeling tool are syntactic, and combine to form a visual language supported by the tool. Semantics involves mapping of the syntactic elements to elements of a semantic domain [12]. In the case of office tools, the *tool* supports no such mapping; the *user* might have one in mind, promoting disciplined, consistent usage, but might not adhere to it strictly and might change it at will. Modeling tools do map syntactic elements to elements of the underlying model, which is the semantic domain. This allows such tools to leverage the underlying model to provide domain-specific guidance and multiple views of a single model, and for structure-aware operations like query and refactoring, rather than operating at the purely syntactic level.

Clearly, neither office tools nor modeling tools are ideal for the exploration, sense-making and communication tasks of pre-requirements analysts. These user studies show that there are clear trade-offs between the added power and functionality enabled by semantics support, and the increased burden it brings to the user experience. For the pre-requirements analysts we studied, the user experience of office tools has clear advantages. They have broad applicability, are easy to learn, and give users the freedom to choose the order of development, allowing the semantics to evolve as content is created. However, if such a user experience could be maintained, pre-requirements analysts would clearly welcome the advantages that tool support for semantics provides.

3. CONCEPTUAL ARCHITECTURE

In this section, we motivate and describe architectural elements that work together to blend the advantages of office tools and modeling tools so as to support pre-requirements analysts' work and practices, as described in Section 2, more effectively. The architectural elements are illustrated in Figure 1, and are:

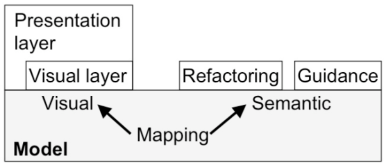

Figure 1: Architectural Elements

- A *visual layer* providing multiple views, in which the user can work with much of the freedom of office tools.
- An underlying model consisting of *related visual and semantic sub-models*, enabling visual cues to be given semantics.
- A forgiving approach to domain-specific *guidance,* with *structure definitions* specifying structural constraints used to check for structural violations and provide assistance. When provided as a package, they effectively define a meta-model. Models that violate them can, however, be created, manipulated and saved.
- *Refactoring* support to allow convenient reorganization.
- A *presentation layer* supporting the synthesis of presentations from working views.

These architectural elements provide the considerable flexibility in usage required by our target users. In contexts where individuals or small groups with shared understanding are engaged in exploratory activities, they do not need support for precision. Their work is better facilitated by automatically mapping visual entities to *generic* semantic entities: untyped entities and relationships that can be refined and constrained as work progresses. This provides coordination of multiple views without requiring premature commitment or constraining the structure that can be expressed. This usage also requires minimal learning. Structure definitions might not be used; or predefined packages of them designed for the domain can be used, to provide assistance; or generic entities can be refined and structure definitions introduced incrementally to record evolving understanding. Violations might be ignored at first. If and when greater precision is required, such as for communication in larger groups or with other tools, (predefined packages of) structure definitions can be introduced and violations dealt with. This can be done incrementally as the need for precision, and the understanding needed to provide it, increase.

The architectural elements presented do not constitute a complete tool architecture. A full tool requires many other elements, such as support for persistence, versioning, atomic model updates, undo, and collaboration. Issues such as these are not specific to flexible modeling tools, and hence are outside the scope of this paper. It is worth noting, however, that the unique elements of this architecture can have profound implications for them. For example, flexible support for visual cues with underlying meaning can be exploited to show collaboration dynamically (e.g., distinctive decorators associated with entities others are working on [13]).

We aim to support organization and communication of information during pre-requirements analysis in a business context. The content of this information is greatly variable, depending on the project, but typically concerns entities (people, organizations, etc.) and their interactions and relationships. Our users are not primarily concerned with numeric data. The entity-relationship approach is therefore a natural fit. We henceforth assume that the information we are dealing with consists of entities, which can have properties, and relations between them.

3.1 Visual Layer

The visual layer is responsible for displaying visual elements and allowing the user to interact with and manipulate them. Our target users told us that it is important to allow a great deal of flexibility in this layer, so that they can explore visually, creating and manipulating visual elements as they focus on the content of their analysis. Office tools provide such flexibility, which is one of the reasons they are popular in this domain. Modeling tools typically have a visual layer that is separate from the model, such as view and controller in the MVC pattern [9]. If the model constrains the information and its structure, as is usually the case, the constraints show through in the visual layer, limiting freedom: the visual layer cannot support anything that the underlying model cannot represent. Flexible modeling tools therefore require particularly rich underlying models, as described in section 3.4.

An important aspect of visual flexibility is the ability to select from, and even improvise, a variety of views. We learned that analysts often use combinations of diagrams, tables and text, combined into presentations or other documents, and a flexible modeling tool could offer an even richer variety. Multiple views give users the option of working with their information in many different ways, facilitating exploration and presentation.

Office tools do offer a variety of document and diagram types, but because they maintain no relationship between them, consistency management is a manual task. To reduce this burden, flexible modeling tools must use the same underlying model for multiple views. Furthermore, in contrast to typical visualizations, the views must allow editing and manipulation of the information they show, with changes made in one being reflected in all. This has been common practice in programming environments and modeling tools for decades [21], and is important in this context also, with a wider variety of views over more diverse information.

Some views, such as hierarchy charts and process diagrams, are designed for information with specific structure. Because of the flexibility inherent in the underlying model, and the fact that the model can be changed through other views, it is possible that the user might try to apply such a view to inappropriate information. An important special case occurs where some of the expected structure is missing because the user has not provided it yet. Flexible modeling tools must be tolerant: rather than refusing to show the information at all, the view should show as much as possible and use the guidance mechanism (section 3.5) to indicate what is incomplete or incorrect and how the user might fix it.

3.2 Visual Organization

Analysts frequently work with information visually when exploring it to gain understanding and again when communicating that understanding. They rely on a variety of visual cues, including:

- *Style*. Different shapes, colors, line styles, etc. are used to highlight distinctions and commonalities. Entities are often drawn as shapes, and relationships as lines or arrows connecting them.
- *Positioning*. Related entities are often positioned in a way that indicates the relationship, such as nearby, side-by-side, stacked or layered, or arranged in a structure such as a list or table. A common case is containment, which is often used to indicate a containment or subsumption relationship.
- Other dimensions, such as time and animation, which remain research challenges from a flexible modeling perspective.

These sorts of cues provide what we call *visual organization*. They are important aids to the cognitive processes involved in organizing and understanding information [6]. They also have a significant effect on communication: if they are used in a consistent and evocative manner, they help to make or subtly reinforce important points being communicated, whereas if they are used haphazardly or inconsistently, they can easily create confusion [15]. For this reason, as we learned in our user studies, good presenters put a good deal of effort into the visual cues.

Office tools offer a wide variety of visual cues, and a lot of freedom to make good (or bad) use of them. The user can move things around freely and experiment with different organizations during the exploratory phase, and can craft beautiful presentations to communicate key points once they have been conceptualized. Unfortunately, the relationship between the visual cues and what they mean is not understood by the tool, since office tools have no notion of underlying meaning. For example, one cannot pose a query to an office tool asking for all entities related to some entity, and expect it to use the visual cues to find them. One cannot even make a query on the style, e.g., to find all boxes that are green. Flexible modeling must enable such queries, but not at the expense of hindering the visual organization capabilities.

3.3 Semantic Organization

To create a coherent analysis and presentation of business issues and potential solutions, organization of the underlying concepts and data, not merely the visual elements, was important to our users. We call this *semantic organization.* Typical elements of semantic organization include relationships, grouping, distinctions and classification. We observed great variability in kinds of semantic organization used across, and even within, projects.

In our user studies, we discovered that users evolve the concepts that go into the final presentation by organizing snippets of information from interviews, prior engagements and client documents. *Sets* provide a well-understood means of dealing uniformly with diverse semantic organizations. In office tools, these typically start as lists. Flexible modeling tools treat lists as enumerated (extensional) sets. During exploration, users need to reorganize the content within and between these lists as they attempt to find the appropriate structuring of their information. Once a user has many snippets, being able to regroup these using query-based (intensional) sets is another important aspect of flexible modeling, minimizing the viscosity inherent in manual re-categorization.

We have found that another key form of organization important to our users is making distinctions between information snippets, often leading to classification. Our users told us that after creating lists, they start to see that there are distinct sets within the items listed. They then create lists within lists, i.e. subsets, to represent such distinctions. When done recursively, this leads to a classification hierarchy. Classification can be done by first defining the categories and then categorizing entities, but during the kinds of exploratory activities of our users it is often done the other way around: they group entities together based on a sense that they are similar in some way, but before they can articulate just how, then define the categories later, once their understanding has grown.

Groupings are not enough to allow users to make sense of their information, and are typically accompanied by drawing arrows to show relationships, for example that an item 'Shipping' in the list of 'Departments' 'reported' the 'fulfillment issue' found in the list of 'Issues.' Users then want to be able to group all items that reported fulfillment issues. Such relationships can be captured by means of query-based sets, such as "all entities related to entity *e* by relationship *r*," where *r* denotes a particular kind of relationship, such as "reported" in the previous example, or perhaps a wildcard representing many or all kinds of relationships.

Our users often viewed the same set of information snippets from different perspectives. So the same entities often need to be organized simultaneously along multiple *organizational dimensions*, to reflect different relationships, groupings or distinctions that are important for the users' understanding. For example, employees might be organized by department, by job description, by experience level, by the customers they interact with, etc. An organizational dimension is a pair (s, {s_1, s_2, …, s_n}), where s is the set of entities being organized according to this dimension, and each s_i is a subset of s. For example, s might be the set of all employees in a company, and each s_i might be the set of all employees in department$_i$ or interacting with customer$_i$, etc.

During exploration, users would like to be able to try a variety of organizational dimensions before settling on a few that prove valuable. This is difficult in office tools. Many users do this using stickies and a wall, and are then frustrated by the effort needed to get their work into electronic form in order to share it with their team. Also, many users curtail their exploration due to the insurmountable difficulties of reorganizing their information snippets. It is important for flexible modeling tools to support multiple organizational dimensions. The implications for classification are that multiple, simultaneous classifications must be supported, allowing the same entity to be classified in multiple different ways; each is an organizational dimension.

A flexible modeling tool must allow users to capture the organizations that emerge rather than have to force-fit them to limited organizations provided. This implies that the tool should provide not only a rich collection of useful kinds of sets and organizational dimensions, but also allow the user to define or modify custom sets and organizational dimensions. Given that our target users are not usually computer scientists, providing a suitable user experience remains a significant research challenge.

3.4 Related Visual and Semantic Models

In our architecture, as in many modeling tools, the underlying model is split into two related sub-models: the visual model and the semantic model. Figure 2 shows this, and other model details that will shortly be explained. The model is an entity-relationship model, suitable for representing the kind and variety of information described above. Both entities and relationships may need to be organized. For example, relationships can be classified according to their *kind*, such as "depends on" or "interacts with." To allow uniform treatment of entities and relationships when appropriate, relationships are themselves considered to be entities. In fact, everything in Figure 2 specializes Entity, but this is not shown, to reduce the complexity of the figure.

The *visual model* captures the visual cues and their properties, such as style and position. Because of their importance to our users, presentations are explicitly represented—as collections of views and visual elements. The *semantic model* captures the underlying information, and the sets representing their semantic organization; structure definitions and violation sets are described below. Entities in the visual model *depict* entities or relationships in the semantic model, shown as dotted arrows in Figure 2.

As we learned from our users, during exploratory activities users experiment visually as a way to gain understanding, thinking more about visual elements than their underlying meaning. During this phase, the model elements being depicted must be generic: entities

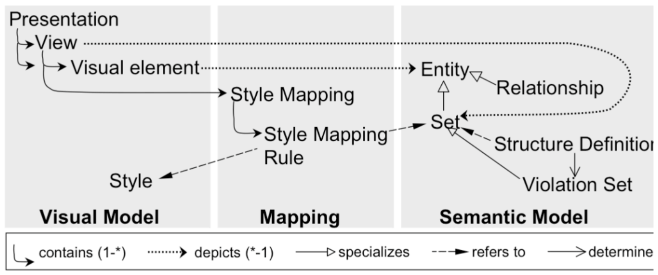

Figure 2: Related Visual and Semantic Models

or relationships with little or nothing known about them. As understanding is gained and distinctions emerge, the user must be able to express them, and the underlying model must evolve accordingly. This is an important reason to avoid rigid typing, where it is hard or impossible to change the type of an entity once created. Conventional typing has two elements: *classification* of entities according to type, and *structural properties*, specifying properties that entities of a particular type are guaranteed to have. The flexible approach to semantic organization described above deals with classification in a less restrictive way: entities and relationships can be classified in multiple ways simultaneously, and their categories can be changed at will. Structural properties can be specified by means of *structure definitions* (right of Figure 2), perhaps in terms of categories, using the guidance component described in section 3.5. Reclassifying an entity or relationship might result in guidance violations, since the structure might be suitable for the old category but not the new one. It remains possible, however, and the guidance component will guide the user through the process of fixing the structure.

The relationship between the models is deeper than just depictions, however: it also provides a mapping between semantic and visual organization. This provides an important capability not present in typical modeling tools: the ability to specify the visual cues to be used uniformly to manifest aspects of semantic organization, as described below. This enhances the user experience by reducing viscosity at a whole different level. For example, correcting a classification error, such as all 'Sales employees' should be 'located in' 'AL' not 'NY,' when all NY employees are colored blue, can be done with a single action, rather than by having to color every depiction of Sales Employee by hand, as is the case with office tools. This should ameliorate the time-consuming and distracting editing reported by the business users we interviewed.

3.4.1 Style Mapping

Style is one of the key visual cues providing visual organization. When using office tools, analysts in our user studies told us they used styling to capture various semantic dimensions in their information. However, they also said that having to set every style on every element manually was time-consuming, and error-prone. Flexible modeling provides an explicit mapping between style and underlying semantic organization, as shown in the center of Figure 2. This mapping allows tools to provide support that can reduce user errors and effort, while allowing them to explore visually as they uncover semantic dimensions. Such support also reduces the effort required when users want to visually represent semantic distinctions they have realized in their head.

A *style*, following common practice such as CSS[2], is a set of *attribute:value* pairs. We interpret style attributes liberally, to in-

[2] http://www.w3.org/TR/CSS2/

clude typical ones like fill color and line thickness as well as shape, decorations and any other visual cues provided by a view. For example, a view showing issues of various kinds might depict an important personnel issue using a red stick figure style:

redFigureStyle = { shape:stickFigure, fillColor:red }

A *style mapping rule* is an ordered pair *s* → *y*, where *s* is a set and *y* a style, specifying that the depiction of any member of *s* should be styled according to *y*. For example, both style mapping rules

r_1: all issues classified as "personnel" → { shape:stickFigure }

r_2: all entities classified as "important" → { fillColor:red }

apply to all issues that are classified as both "personnel" and "important," causing their depictions to have redFigureStyle (i.e., to have at least the style attributes specified by redFigureStyle).

In our architecture, each view has a *style mapping* made up of such style mapping rules. The styles used in these rules must be such that the view can render them; for example, list or table views might not support variations in shape. The style mapping could be manifested to the user, as a *legend*. Not only does this allow all users to see what the styles mean, it can also be the place where style mappings are edited. Changes immediately affect the styling of all appropriate visual elements in the view.

Many style attributes, such as fill color, shape and a variety of decorators, are independent: any visual element can have arbitrary combinations of values for them. We call such a set a *style space*, and each independent attribute a *style dimension*. An appealing structure for the style mapping is to associate each style dimension in a style space with an organizational dimension, mapping each subset in the organizational dimension to a different value in the style dimension. This allows the user to see the organizational dimensions visually and to control which to show, and clearly separates distinctions being made by the different dimensions. The example above was based on mapping the organizational dimensions "issue kind" (with values such as "personnel," "accounting," etc.) and "importance" (with values such as "important" and "insignificant") to shape and fill color, respectively, so that different issue kind values are displayed as different shapes and different importance values are displayed as different colors. This kind of use of styles was typical in the diagrams created by our target users as they did their work.

Our users reported sophisticated use of styles across sets of views. For example, consistent styling is often desired across a presentation, as an aid to understanding. However, in subsections of the presentation users may purposely change a style to more appropriately convey their point. In one example, a consultant included diagrams done by another team that had shared semantics with the rest of the presentation and, to remain true to the originals, she used the original styles for just this set of diagrams. Several views should therefore be able to share a base style mapping, with each view specializing or overriding it if desired. If the user changes a style mapping rule inherited from a shared base, the question arises of whether the user intends to change the local view only, or the base. The user experience remains a research challenge.

It is possible for multiple style mapping rules within a style mapping, whether local or inherited from a shared base, to apply to a single entity, as for r_1 and r_2 above. A *style clash* occurs if these mappings specify different values for the same style attributes. A related problem is *style ambiguity,* where the same style attribute is mapped to different sets, leaving the user in doubt as to what the style means in any specific case. Implementations based on this architecture must cope with clashes and ambiguity in a non-intrusive manner. Occasionally, multiple clashing style attributes can be blended; for example, two fill colors can be rendered as a transition from one to the other. Otherwise, the guidance component described below can be used to indicate clashes and ambiguity; its non-intrusive approach avoids interrupting the user's flow, especially important during creative, exploratory activities.

If style spaces are used, with different style dimensions being associated with different organizational dimensions, as described above, style ambiguity cannot occur. If the subsets of the organizational dimensions are disjoint, style clashes cannot occur either.

3.4.2 Position Mapping

Position mapping is conceptually similar to style mapping, except that mapping rules specify positioning (e.g., placement or containment) instead of style. This requires a model of positioning instead of the simple *attribute:value* pairs in the case of styles. Some views offer continuous positioning, such as diagrams that allow arbitrary placement, whereas others offer discrete alternatives, such as tables that allow placement in specific rows and columns. Such models do exist, e.g. [10], but their integration into the flexible modeling architecture requires further research.

3.5 Guidance

Thus far we have focused on flexibility, and on providing consistent appearance in the face of flexibility. The flexibility is primarily afforded by the general underlying model, capable of representing arbitrary, unconstrained entities and relationships. This is unlike most modeling tools, for which data that does not conform to the metamodel is effectively corrupt, and that therefore restrict user interactions to those that respect the metamodel. While helping users get things right, such enforcement puts them in a straitjacket. We aim to provide guidance without confinement.

During early exploration, and in unfamiliar domains, there might be no known guidance to give. However, once a structure has emerged or is known up-front, as when analysts are working within an analysis framework that has been developed on previous engagements, guidance can help the user to work within it and conform to it. Our architecture therefore contains a guidance component. At the conceptual architecture level, we identify and justify some of its properties, leaving many options for specific designs and implementations. Some concrete details of guidance in the BITKit prototype are given in section 4.

3.5.1 Structure definitions

Guidance is specified by means of *structure definitions,* which can specify a variety of structural constraints on the model. For example: every item classified as an issue should also be classified along the "importance" dimension. Structure definitions about domain entities and relationships, such as these, are specified in terms of the semantic model, not the visual elements in the visual model. However, structure definitions can also cover the mapping between semantic and visual models, such as specifying that style mappings should be structured according to style spaces; e.g., items in subsets of 'Issue' should be styled using a color set.

Our architecture does not support typing of entities as such, because typing usually connotes a degree of viscosity: changing an entity's type can be difficult or even impossible in some models. Abstraction can also be compromised, because an entity can usually have just a single type. Structure definitions in combination with semantic organization, however, provide key benefits offered by typing, as noted above. Structure definitions effectively cause referenced categories to be like types, but with greater flexibility.

The specific nature of the structure definitions and the language used to express them are not laid down by this architecture. A flexible modeling tool might use an existing, general approach, such as xlinkit [16] or Crocopat [4], or implement specific kinds of high-level, parameterized structure definitions that provide convenient abstractions embodying many low-level rules [14].

3.5.2 Positive and negative guidance

The obvious use of structure definitions is to detect and report violations. Architects' Workbench AWB did this in a non-intrusive way by means of *reminders* shown in a separate view [1]. This *negative guidance* is important, alerting users to the fact that their model violates the specified structure.

However, as AWB users confirmed, it is more helpful to provide *positive guidance* by doing things automatically for the user whenever possible, or leading the user to a correct result by presenting a restricted set of choices. For example, if the user draws a line in a diagram between the boxes depicting two entities, a relation is created between those entities. In the absence of guidance, the relation might be given a generic kind like "related to." The user would then have to edit the relation to change its kind once it becomes clear. This generally avoids premature commitment: the user can draw a line without first having to decide what kind of relation it depicts. If, as in some of the cases reported by our users, however, the analyst is using a framework defined by a senior analyst, such as a version of issue-based consulting, a structure definition might already have been specified. For example, an item classified as 'Hypothesis' can only be related to an item classified as 'Issue' by the relation 'for.' Positive guidance would reassign the relation kind accordingly. The user might still change it, if desired, but that is expected to be a rare exception.

An intermediate form of positive guidance is where the structure definitions cannot determine a unique correct result, but instead can offer a restricted set of alternatives. This facilitates autocompletion, a feature of tools that shows a restricted menu of valid possibilities based on context and user input.

The manner in which guidance is actually given to the user through the user interface greatly affects the user experience. In the case of pure positive guidance, the tool can just do the single, right thing, but in all the other cases some guidance-specific interaction is required. In accordance with our users' desire for tools that provide flexibility, we avoid the more stringent approaches, such as bringing up a dialog forcing the user to correct the violation immediately. The flexible modeling architecture uses an approach to guidance with low invasiveness and integrated with other architectural elements: visual cues controlled by the style mapping. The sets used in style mapping rules can include *violation sets* (bottom-right of Figure 2) of items or relationships involved in guidance violations, resulting in visual cues indicating the violations. Like all style mapping rules, these will be manifested to the user in the legend, making the meaning of the cues clear, and also allowing the user to change them, thereby allowing user-control of their obtrusiveness, and even their presence.

Different degrees of flexibility may be appropriate at different times and in different settings. For example, users told us that in some teams when junior people are collaborating within a structure defined by more experienced people, they may extend, but not change, the structure. The level of freedom allowed by this architecture might seem undesirable, or even dangerous, in such contexts. It is certainly possible to provide levels of guidance enforcement, and to use policies based on attributes such as user role to control their use. However, it is an open question, whether such measures are necessary or desirable.

When exporting a model from a flexible modeling tool to a standard modeling tool, one must ensure that it conforms to that tool's metamodel. Suitable structure definitions allow the guidance component to check for this, alerting the user to violations.

3.5.3 Guidance checker

The *guidance checker* is responsible for evaluating the (relevant) structure definitions when changes occur or are underway, to determine both violations and what positive guidance to give. The requirement for positive guidance places greater demands on the checker than is typical, and so it, and the language used to express structure definitions, should be designed with the goal of providing user-friendly positive guidance. Nentwich, Emmerich and Finkelstein show how to compute repair actions for first-order logic constraints [17]. Remaining research challenges include user experience issues for positive guidance and for end-user manipulation of structure definitions, and interaction protocols between the guidance checker and views to allow timely computation and display of positive guidance in response to user gestures.

3.6 Refactoring

During the kinds of exploratory work being done by our target users, it is important to be able to try things without worrying about whether they are exactly right, and then be able to change them later. This would reduce premature commitment by making change, and thus iteration, easy. Rich views with an underlying flexible model support many kinds of changes, but there are some structural changes that require refactoring support. For example, when extracting important statements from interview transcripts, one might quickly categorize them based on who made the statements (among other criteria). Later on, however, it might be beneficial to refactor to use "said by" relationships between statements and interviewees instead. This makes the nature of the relationships explicit, allowing for other relationships between statements and stakeholders. This change would be tedious and error-prone to do by hand, illustrating the importance of refactoring support.

Some refactorings involve just the primitives in the underlying model, such as categories and relations in the example above. The tool can include built-in support for these. In other cases, however, refactoring will involve transforming to or from structures defined by structure definitions. Though refactoring has been offered by tools for some time now, many research challenges remain in the area of extensible support for new refactorings, as well as determining appropriate refactorings for the domain of flexible modeling (e.g., refactorings that both transform data and create structure definitions). It is particularly challenging to enable stakeholders with domain understanding who are not expert software engineers to define new refactorings.

3.7 Presentation layer

A key focus of pre-requirements work is the creation of *presentations*: coherent expositions, usually of understanding gained about the business situation together with implications and recommendations. Primary user challenges are to determine a good narrative structure, and to ensure that each key point is backed by appropriate supporting material.

The presentation layer provides *presentation views* for working with presentations. Any views used for exploration are candidates for incorporation into presentations. Each presentation unit (e.g., slide) may be an assembly of one or more views together with titles and other supporting visual elements. The incorporation of large exploratory views into presentations may require user-guided pagination or refactoring to ensure legibility, conformance to form factors, and that material is revealed in appropriate and useful chunks. In general, considerable work may be needed to polish presentations, in addition to assembly of the narrative.

The act of constructing a presentation brings a fresh perspective to the exploratory and envisioning role of the user, frequently causing them to revise and enrich their understanding. The presentation layer therefore allows full manipulation of the views within presentations. This further justifies presentations being assemblies of views, as well as our overall goal of blending modeling and presentation features into a single tool. User interaction techniques enabling gradual yet fluid transitions between modeling, preparation and presentation are a topic for further research.

Explicit style mappings play additional roles within presentations. They form the basis for legends that accompany each view. Also, in interactive presentations, the reader may be presented with controls that enable or disable specific mappings, or select one or more entries to highlight from a style dimension. Decisions about what to place under reader control are made by the author.

Suitable presentation views could be used for actual presentation, but in many contexts it is necessary to export presentations in standard forms, such as slide decks or web sites. This is the task of *presentation exporters*.

4. BITKIT

BIKit is an early prototype tool for business stakeholders involved in pre-requirements analysis [19]. We followed the approach of starting with minimal capabilities and adding enhancements only when clearly necessary. Several senior business analysts provided feedback on design sketches and test-drove very early versions of BITKit. We evolved it based on observations of and feedback from them. Our work on BITKit inspired the architecture described above. In formulating the architecture, however, we engaged in extension and generalization beyond BITKit's current capabilities. This section briefly describes BITKit as a concrete, and still fairly limited, instance of the architecture.

BITKit is built as a Rich Internet Application using Adobe® Flex®[3]. Persistence is provided by streaming XML to a REST server, or storing it on the local disk (when running under Adobe® AIR™). The underlying model is founded on entities, called *items*, and directed, binary relationships, which are themselves considered entities. Sets are items, and both enumerated and query-based sets are supported. BITKit does not include a general-purpose query language, however, but rather specific kinds of parameterized sets supporting a standard set interface. For example, an "AllTargetsSet" is the set of all target items reachable by following a specified relationship from a specified source item. These various kinds of sets, along with standard set operations like intersection and union, effectively provide an open, extensible means of expressing queries. This approach enabled us to get started quickly and to grow the query capability as needed.

We chose to use *tagging* as our means of classification. Tagging is a flexible, informal approach to organization that has become popular, especially on the Internet (e.g., Delicious[4], flikr[5]). It is lightweight, allowing users to attach tags to any entities, and we believe that it is particularly effective during early exploration. Multiple classification is naturally supported by attaching multiple tags to entities. Although tags might be used with little thought initially, they can be defined and organized as understanding is gained [20]. We found that the business analysts we were working with were mostly not used to tagging, and their understanding and appreciation increased when we pointed out that they were effectively classifying things in a lightweight way.

Early feedback led to the introduction of *tag groups:* collections of related tags, such as tags representing degrees of importance or kinds of issues. Tag groups serve as organizational dimensions in multi-dimensional classifications, allowing simultaneous classification by such criteria as importance and issue kind. Later feedback revealed (as we expected) the need for *implication* relationships among tags, which would support classification hierarchies.

BITKit provides three primary kinds of views: diagrams, lists and tables [20]. A diagram depicts items as shapes and relationships between them as lines; the items are explicitly placed in the diagram by the user. A list is founded on a set, and displays all items in that set. Like all BITKit views, lists can be changed by the user, and this causes the underlying set to be updated automatically. In some cases there is not a unique way to do this, such as when an item is removed from an intersection, and the user needs to be consulted. Determining exactly what the user experience should be remains a research challenge.

Tables are also set-based. The user selects row and column sets, determining the headers. The contents of a cell, at the intersection of a row and column, is computed based on the headers of that row and column. For example, if the rows are "all tags in group Issues" and the columns are "all tags in group Importance," then each cell will contain entries that are tagged with a particular kind of issue and degree of importance. Changes the user makes in the table cause appropriate changes to the underlying tagging.

All BITKit views have legends, showing their style mappings. Diagrams currently support the richest variety of styles: shape, background color, line color and line thickness. Lists and tables do not support variation in shape, as this is not a style typically used by business analysts in these views. Experience has revealed the need for mapping organizational dimensions to style dimensions, as described in section 3.4, so that the user can comprehend organizational dimensions at a glance. We plan to add support for such mapping, and for other style attributes, such as decorators.

As with sets, BITKit offers an initially small but extensible set of structure definitions in the model. Currently, it supports implications involving sets (every item in s_1 must also be in s_2, e.g., every item tagged with a tag in tag group "Issues" must also be tagged with a tag in tag group "Importance"), and constraints on the endpoints of relations (e.g., both the source and target of a "sub-issue" relation must be tagged with "Issue"). We have given priority to supporting positive guidance. For example, given the structure definition above, if the user creates a "sub-issue" relation anywhere, the source and target are automatically given "Issue" tags, if they do not have them already. Violations are currently

[3] http://www.adobe.com/products/flex/. Adobe, Flex and AIR are trademarks of Adobe Systems Incorporated.

[4] http://delicious.com/

[5] http://www.flickr.com/

shown as decorators on shapes and lines in diagrams only. We plan to introduce violation sets, as described in section 3.5.2, which would then allow style mappings to be used to indicate how violations should be depicted. Violation decorators can be opened to see details, including a list of possible repairs.

BITKit's presentation layer is still primitive. Based on user feedback, two separate aggregations of views are provided, *workbooks* and *presentations*. This allows users to explore using workbooks, and then to copy views into presentations and polish them. As practice areas mature, we expect workbooks to become fairly stable templates, guiding analysts in there work, but presentations to be tailored differently for different audiences. Supporting smooth transitions between them remains an area for future work.

The need for refactoring soon emerged as we showed analysts early versions of BITKit. Because tagging is so lightweight, we found it being used early on in situations where it later turned out relationships were more appropriate. We also found that it is not always clear upon creation whether something should be an item, a tag or a tag group. Wand and Weber discuss another situation where it is hard for users to make correct choices up front [22]. Avoidance of premature commitment requires that the user can pick quickly, and change the decision easily later.

Our experience with BITKit and feedback from business analysts thus led to the insights behind the architecture described in this paper. We are currently preparing for pilot use of BITKit in a real industrial context, to support consultants in evaluating software development organizations in order to recommend practices (e.g., requirements management or change management) that would improve them. This has already led to the need to support *collateral material*: standard material, perhaps from prior engagements, that can be adapted and reused. Users want to include (copies of) parts of it in their models and make local changes, and also parameterize it, effectively creating templates, whose importance was noted in Section 2. We have begun experimenting with approaches to abstracting from and parameterizing existing models.

5. RELATED WORK

Requirements modeling tools, such as GRAIL/KAOS [3] and Rational Requirements Composer[6], embody specific metamodels. Extensible modeling tools, such as RSM[7], also embody specific metamodels, but allow some degree of flexibility, such as user-defined stereotypes and profiles, and sometimes the ability to add annotations. The need to conform with the metamodels, however, prevents the degree of flexibility needed in our domain.

Model Integrated Program Synthesis [18], and other environments supporting metamodeling, allow domain experts to define metamodels from which domain-specific modeling tools can be generated. Each such tool supports manipulation of models that conform to its metamodel. These systems provide valuable time savings and flexibility to tool builders, but the tools produced do not reflect it: each supports the metamodel from which it was generated, and changes require regeneration. Flexible modeling tools, in contrast, allow tool *users* to evolve metamodels and to manipulate models that do not conform to specific metamodels.

Architects' Workbench (AWB) [1] served as an inspiration for our work, though it was developed for IT architects rather than business stakeholders. It also provides a flexible underlying model with guidance, but is configured with specific metamodels. The underlying model allows change-type refactorings and graceful handling of incompleteness and metamodel evolution, which result in *reminders* indicating what is incomplete or incorrect. AWB's metamodels include catch-all types (RawNode, Basic-Relation and Note), which allow free-form, untyped capture of data, but with no way of introducing new categories or kinds of relations except by changing the metamodel and reconfiguring. AWB also does not support general style mapping.

Telelogic Systems Architect[8] is a modeling tool for enterprise architecture, and hence closer to the domain of business analysis. It supports a large corpus of diagrams, methods and notations, based on underlying metamodels that can be tailored by expert administrators. The user thus has a wide choice of metamodels, but not the flexibility to deviate from or extend the one chosen.

Our approach to guidance relates to Balzer's landmark work on tolerating inconsistency [2]. His approach involved introducing *pollution markers* to indicate the violation of specific constraints: instead of requiring a constraint to hold always, one instead requires a pollution marker to be present if and only if the constraint does not hold. The presence of the marker thus indicates the violation of a specific constraint, and, as he noted, can be used to provide a visual cue to the user, such as coloring inconsistent spreadsheet cells. The markers can also be used as guards, preventing the execution of code that depends on the constraints being satisfied. Balzer's motivation was facilitating complex updates involving multiple parties, where consistency would only be restored once all parties had completed their parts of the updates. The inconsistencies were thus expected to be temporary. Our motivation is to allow users to be guided by domain-specific structure definitions, but with the freedom to deviate from them. Violations thus represent cases where the semantic model violates whatever structure definitions are in use, and not necessarily inherent inconsistencies within the models. These violations might be repaired as work proceeds, and must be if the models are to be passed to downstream modeling tools. Sometimes, however, the violations are deemed acceptable; they then persist as such, or the structure definitions themselves are modified to reflect new understanding or a difference in domain.

Another important area of inconsistency handling in requirements tools is that of representing and working with mutually-inconsistent information provided by multiple stakeholders. This must be addressed by a model that reifies multiple perspectives, allows them to be inconsistent and allows their relationships to be explored, such as the Viewpoints model [7]. Support for multiple stakeholder perspectives, as well as for alternative possible futures, which might also be mutually inconsistent, are important in our domain also. They are, however, largely independent of the focus of this paper, and are therefore not addressed here.

Another largely-independent area not addressed in this paper is sketch-based user interfaces. Grundy and Hosking describe generic support for adding sketch-based input to modeling tools [11]: drawn shapes are recognized as diagram elements, at times that are customizable by the user. They argue that this approach improves tools along several cognitive dimensions [5]: premature commitment, viscosity, progressive evaluation, secondary notation, closeness of mapping and error proneness. Since most of these are important in our domain too, we postulate that sketch-based interfaces would be a valuable addition to pre-requirements

[6] http://www-01.ibm.com/software/rational/announce/rrc/

[7] www.ibm.com/developerworks/rational/products/rsm/

[8] http://www.telelogic.com/Products/systemarchitect

tools, but how they would be received by business stakeholders, and how much they would help, remain issues for further study. An interesting research challenge when adding sketch-based interfaces to flexible modeling tools is that the user might draw new shapes, not yet known to the tool. The sort of support Grundy and Hosking provide for coexistence of sketches and computer-drawn shapes can be expected to be helpful here. The user will also need means to select or define new computer-drawn shapes and associate them with sketched shapes, causing the recognizer to be updated dynamically.

6. CONCLUSION

This paper introduced the notion of *flexible modeling tools,* and described key architectural elements needed to blend the advantages of office tools and modeling tools. Pre-requirements analysts, and others engaged in exploratory work, currently find themselves forced to use either office tools or modeling tools. Input we received from many practitioners clearly indicated that neither is ideal. Through a visual layer allowing the user great freedom, an underlying model able to represent any information, mapping from semantic characteristics to visual cues, forgiving guidance and refactoring and presentation support, flexible modeling tools bring together many of the key advantages of office tools and modeling tools, allowing users to move smoothly between informal exploration and modeling with varying degrees of formality and precision. They therefore have the potential to fill an important gap in the current tool spectrum.

We also identified a number of research challenges remaining in the area of flexible modeling tools. Several involve infrastructure, but many involve the user experience: providing an experience that gives power and flexibility to business users without being too complex, confusing or intrusive is particularly challenging. We hope that this paper will spark research on these challenges, and provide a framework within which it can be positioned.

7. ACKNOWLEDGMENTS

We thank Barthélémy Dagenais for many helpful comments.

8. REFERENCES

[1] S. Abrams, *et. al.*, “Architectural thinking and modeling with the Architects' Workbench.” *IBM Systems Journal 45(3),* July, 2006.

[2] R. Balzer, “Tolerating inconsistency.” In *Proceedings of the 13th International Conference on Software Engineering (ICSE 1991)*, IEEE, pp. 158 – 165, 1991.

[3] P. Bertrand, et. al., “GRAIL/KAOS: An Environment for Goal-Driven Requirements Engineering”, In *Proceedings of the 19th international conference on Software Engineering (ICSE 1997),* pp. 612—613, 1997.

[4] D. Beyer, “Relational programming with CrocoPat.” In *Proceedings of the 28th International Conference on Software Engineering (ICSE 2006),* pp. 807-810, 2006.

[5] A. F. Blackwell and T.R.G Green, “A Cognitive Dimensions questionnaire optimised for users.” In A.F. Blackwell & E. Bilotta (Eds.), In *Proceedings of the Twelfth Annual Meeting of the Psychology of Programming Interest Group*, pp. 137-152, 2000.

[6] S.K. Card, J.D. Mackinlay and B. Shneiderman (Eds.), *Readings in information visualization: Using vision to think.* Morgan Kaufmann, 1999.

[7] A. Finkelstein, *et. al*., “Viewpoints: A framework for integrating multiple perspectives in system development.” *International Journal of Software Engineering and Knowledge Engineering 2(1),* pp. 31-58, 1992.

[8] A. C. W. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh, "Inconsistency Handling in Multi-perspective Specifications", *IEEE TSE 20(8),* pp. 569-578, 1994.

[9] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, April 2005.

[10] E. R. Gansner and S. C. North, “An Open Graph Visualization System and its Applications to Software Engineering.” *Software—Practice and Experience, 30(11),* pp. 1203-1233, 2000.

[11] J. Grundy and J. Hosking, “Supporting generic sketching-based input of diagrams in a domain-specific visual language meta-tool.” In *Proceedings of the 29th international conference on Software Engineering (ICSE 2007),* pp. 282-291, 2007.

[12] D. Harel, and B. Rumpe, “Meaningful Modeling: What's the Semantics of "Semantics"?” *IEEE Computer 37(10)*, pp. 64-71, 2004.

[13] S. Hupfer, L-T. Cheng, S. Ross and J. F. Patterson, “Introducing collaboration into an application development environment.” In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, pp. 21-24, 2004.

[14] H. Kilov and J. Ross, *Information Modeling: an Object-Oriented Approach*. Prentice Hall, 1994.

[15] J.H. Larkin and H.A. Simon, “Why a Diagram is (Sometimes) Worth Ten Thousand Words.” *Cognitive Science 11(1),* pp. 65-99, 1987.

[16] C. Nentwich, L. Capra, W. Emmerich and A. Finkelstein, “xlinkit: a consistency checking and smart link generation service.” *ACM Transactions on Internet Technology 2(2)*, pp. 151-185, 2002.

[17] C. Nentwich, W. Emmerich and A. Finkelstein, “Consistency management with repair actions.” In *Proceedings of the 25th International Conference on Software Engineering (ICSE 2003),* pp. 455-464, 2003.

[18] . Nordstrom, J. Sztipanovits, G. Karsai, and A. Ledeczi, "Metamodeling — rapid design and evolution of domain-specific modeling environments." In *Proceedings of the IEEE ECBS '99 Conference*, Nashville, Tennessee, pp. 68—74, April, 1999.

[19] H. Ossher *et. al*., “Business Insight Toolkit: Flexible pre-requirements modeling.” Informal demonstration paper in *ICSE 2009 Proceedings Companion*, May 2009.

[20] H. Ossher *et. al*., “Using tagging to identify and organize concerns during pre-requirements analysis.” Workshop paper in *ICSE 2009 Proceedings Companion*, May 2009.

[21] S. P. Reiss, “PECAN: Program Development Systems that Support Multiple Views.” *IEEE TSE 11(3)*, pp. 276-285, 1985.

[22] Y. Wand and R.A. Weber, “Research commentary: information systems and conceptual modelling—a research agenda.” *Information Systems Research 13(4),* pp. 363–376, 2002.