

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

Harold Ossher¹, Rachel Bellamy¹, Ian Simmonds¹, David Amid², Ateret Anaby-Tavor²,
Matthew Callery¹, Michael Desmond¹, Jacqueline de Vries¹, Amit Fisher², Sophia Krasikov¹

¹IBM T.J. Watson Research Center
Hawthorne, New York, USA

ossher,rachel,simmonds,mcallery,mdesmond,devries,kras@us.ibm.com

²IBM Haifa Research Center
Haifa, Israel

davida,atereta,amitf@il.ibm.com

Abstract

A serious tool gap exists at the start of the software lifecycle, before requirements formulation. Pre-requirements analysts gather information, organize it to gain insight, envision possible futures, and present insights and recommendations to stakeholders. They typically use office tools, which give great freedom, but no help with consistency management, change propagation, or information migration to downstream tools. Despite these downsides, office tools are still favored over modeling tools, which are constraining and difficult to use. We introduce 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 research challenges to be met in realizing them. We briefly describe the Business Insight Toolkit, a prototype tool embodying this architecture.

Categories and Subject Descriptors: D.2.1 [Software Engineering]: Requirements/Specifications – Tools.

General Terms: Design, Human Factors.

Keywords: Flexible modeling, modeling tools, pre-requirements analysis, business analysis.

1. Introduction

Few business software projects begin with requirements engineering. Before requirements some form of *business analysis* is conducted to determine whether new development is needed. At this stage, before requirements are formulated, activities include gathering information, organizing it to gain insight, envisioning alternative futures, and presenting insights and recommendations to stakeholders.

Currently available tooling for this early stage of the software lifecycle is inadequate.

For good reasons, discussed below, users prefer office tools. However, these tools are limited in organizing information, with no underlying content models; while representations of visual elements are persisted, there is no notion that those elements represent content in the user's domain. Consequently, consistency must be maintained manually; users spend valuable time propagating even small changes. Practitioners reported spending entire days on manually maintaining consistency, which not only consumes time but disrupts flow. Migrating results to downstream modeling tools, which are founded on domain-specific content models, is inevitably manual; traceability is usually lost.

Modeling tools, such as for Business Process Models (BPM)¹, may be used at this early stage. They have the opposite problem to office tools, however. They have underlying content models depicted by visual elements. Metamodels constrain the structure of the content, thereby restricting the usage of visual elements, typically with a level of precision premature at this stage. A pre-sales consultant reported spending as much time inventing details to dismiss irrelevant error messages as in mapping the process, and with no actual quality improvements.

We argue that pre-requirements analysts need *flexible modeling tools*: tools blending the advantages of office and modeling tools. Such a tool would allow users to work freely and easily with visual elements, attributing meaning to visual characteristics by associating them with elements of the underlying content model, thereby enabling consistency management. Given suitable domain-specific definitions, the tool would provide guidance and checking, without requiring strict conformance to a rigid metamodel. Users can thereby move smoothly between informal exploration and modeling with varying degrees of formality. We propose a conceptual architecture for such tools, and outline research challenges for them to be realized.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Onward! 2010 October 17–21, 2010, Reno/Tahoe, Nevada, USA.
Copyright © 2010 ACM 978-1-4503-0236-4/10/10...\$10.00.

¹ See, for example, <http://www.bpmn.org/>

This architecture arose from our work on Business Insight Toolkit (BITKit) [19], a prototype tool supporting pre-requirements analysis. We designed and evolved it by observing its target users and from their feedback. However, we believe that flexible modeling tools have broader applicability, since similar considerations apply in other domains, especially those involving exploration. We see value in incorporating many of the features we describe into standard modeling tools, including tools for UML-based design, allowing users to work at varying levels of formality. Less formality is desirable 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 tool architecture integrating key benefits of office and modeling tools, enabling users to move smoothly between exploration and modeling (primarily a novel combination of known architectural elements brought together to address specific requirements), and (d) identification of related research challenges.

Section 2 introduces the tooling challenges of pre-requirements analysts, before assessing the adequacy of office and modeling tools, respectively. We highlight features of office tools missing from modeling tools that would give them 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 describes the BitKit prototype and our experience with it. Section 5 discusses related work.

2. Pre-Requirements Analysis

To understand the work of pre-requirements analysts we held a three-day workshop for twelve IBM Business Architects. This was supplemented by an email “shadowing” of a three month consulting project, in which a Senior Business Analyst established data governance practices² within a customer organization, and with on-site observation of a one month project. Each workshop attendee described three pre-requirements projects: one that successfully resulted in insights for the customer, one that failed, and one that was a near-miss or last-minute save. Through these real stories from actual projects, workshop participants 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, members work closely together to reach shared understandings of the situation. Analysts 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. Analysts themselves are typically business people with 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 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 organize and make sense of it to identify business issues and potential solutions. A presentation is created to communicate 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*, which often results in business *transformation*. The process is *exploratory*.

Narratives play a central role in structuring the work of pre-requirements analysts. Their final output 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 almost immediately, even if that presentation is a rough draft of a final version. Template reuse is a common working style, especially for consultants applying their accumulated experience and expertise to similar issues for several clients. Templates help organize findings into pre-defined categories and representations that were helpful in past projects; and also as a way of reusing narrative structure in presentations. Often the template is adapted to fit the needs of a particular project, including changing styles and vocabulary for the storyline, taste and culture of the client.

Invariably, analysts prefer to work in office tools than modeling tools. To understand why, we conducted an addi-

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

Table 1: Advantages of office vs. modeling tools

² Data governance includes the establishment of standard data definitions to promote integration of and consistency between the many systems, organizations and processes of a company, together with processes for both maintaining and ensuring adherence to those standards.

tional study focusing on the users' experience of 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 reported on a modeling tool, two reported on a diagramming tool (an office tool supporting structured graphics) and four reported on a presentation tool. Table 1 summarizes advantages of each type of tool. Given the small number of participants, these results are suggestive rather than conclusive.

All interviewees prefer to use office tools where possible, even for technical diagrams. One interviewee commented, their immediate need was communication, 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 it would take too long to learn the modeling tool's features. In comparison, the *"learning curve was minimal for getting to the ability to diagram"* using the diagramming tool. Equally importantly, modeling tools cause premature commitment by imposing inappropriate work order and levels of detail: *"[it] makes me be too rigorous, when I want to be sloppy."* Metamodel enforcement forces users to provide details before they are ready: *"It tends to force compliance with standards, driving you into the detail before the bigger picture is sorted out."*

With no underlying content model, office tools are free of constraints imposed by metamodels: *"[the diagramming tool] is really agnostic about what terms 'mean'. They are just elements on a page and carry no metadata apart from style."* This makes them easy to learn, and ensures broad applicability. Work order is unconstrained, as is level of detail. The user can evolve the diagram or presentation 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."*

Working in the medium used to present is important, and is one reason interviewees like to do even preliminary work using presentation tools. Being in the presentation medium helps them think through the final form as they create 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 recounted how the lack of a detailed model led to a disconnect between the pre-requirements business case and the detailed models that defined the system downstream. Changes downstream need to be propagated back upstream, but there is no easy way to update the

presentations, reports and diagrams produced using office tools. 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 consistent change in style or terminology of an element may require many actions throughout a document. All interviewees complained of managing consistency across documents and within large documents: *"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 and propagating low-level changes: *"Time-consuming to arrange all the different elements and ensuring consistent styles are used throughout."*

Consistency issues are typically handled well in modeling tools. Visual elements in both an office tool and a modeling tool are syntactic, and combine to form a visual language supported by the tool. However, modeling tools map syntactic elements to elements of an underlying *content model* (or abstract syntax) [12]. With office tools, the *tool* supports no such mapping; the *user* might have one in mind, promoting disciplined, consistent usage, but might deviate from it or change it at will. Modeling tools can leverage the content model to provide domain-specific guidance and multiple views of the model, and structure-aware operations like query and refactoring.

Neither office nor modeling tools alone are ideal for the exploration, sense-making and communication tasks of pre-requirements analysts. These user studies show clear trade-offs between the added power and functionality enabled by an underlying content model, and the increased burden it brings to the user experience. For the studied pre-requirements analysts, the office user experience has clear advantages. Office tools have broad applicability, are easy to learn, and allow the user to choose the order of development, with meaning evolving as content is created. However, if such a user experience could be maintained, pre-requirements analysts would clearly welcome the advantages of an underlying content model.

3. Conceptual Architecture

As introduced above, flexible modeling tools blend the advantages of office and modeling tools, thereby more effectively supporting pre-requirements analysts' work. In particular, they provide the free-form, relatively-unconstrained user experience of office tools blended with the support for structure provided by modeling tools, which facilitates multiple views, consistency management and domain-specific support. This structural support is often loosely termed "semantics," to reflect the fact that the visual elements are not just drawings, as they are in office tools, but have underlying "meaning" that is represented in the tool (in the underlying content model or abstract syntax) and used by the tool to help the user. We avoid the term "semantics" in this paper, however, because some

respected authorities deem it to be incorrect in this context [12].

A flexible modeling tool might be built as a modeling tool with added flexibility. There are many possible forms of flexibility that would be beneficial, but the primary focus of our work is freedom from the rigid metamodel that typically underlies a modeling tool: the flexibility to do, and to save, free-form work that does not conform to the metamodel, to be shown violations with customizable degrees of intrusiveness, and to evolve the metamodel while modeling, if appropriate. A flexible modeling tool might also be built as an office tool with added support for representing meaning associated with visual elements—a “smart office tool.” A third possibility is for a flexible modeling tool to be a new kind of tool entirely, built afresh to realize the desired blend of capabilities.

We present a conceptual architecture that, we argue, is key to accomplishing the blend of office- and modeling-tool capabilities whichever of the above paths is followed. It is illustrated in Figure 1, and consists of the following elements:

- A *visual layer* providing multiple views, giving the user much of the freedom of office tools.
- An underlying model comprising *related visual and content sub-models*, mapping visual cues to content items. It represents arbitrary, unconstrained information; an entity-relationship model is suitable.
- A forgiving approach to domain-specific *guidance*, with *structure definitions*. These specify structural constraints, e.g., restricting the entities that can be related by a particular kind of relationship.³ Structure definitions are used to identify structural violations and provide assistance. Models violating structure definitions can, however, be created, manipulated and saved, as the underlying model is unconstrained.
- *Refactoring* support allowing convenient reorganization.
- A *presentation layer* supporting synthesis of presentations from working views.

These architectural elements provide the considerable flexibility required by our users. Individuals or small groups with shared understanding engaged in exploratory activities do not need precision. Their work is better facilitated by automatically mapping visual entities to *generic* content entities—untyped entities and relationships that can be refined and constrained as work progresses. Multiple views can be coordinated without demanding premature commitment or constraining which structures can be expressed. Structure definitions might not be used; or predefined packages designed for a domain can be reused; or generic entities can be refined and structure definitions introduced incrementally as understanding evolves. Violations might be ignored.

³ We do not consider behavioral constraints, since models produced during pre-requirements analysis are typically not executable.

If and when greater precision is required, such as for communication with other tools, predefined structure definitions can be introduced and violations dealt with.

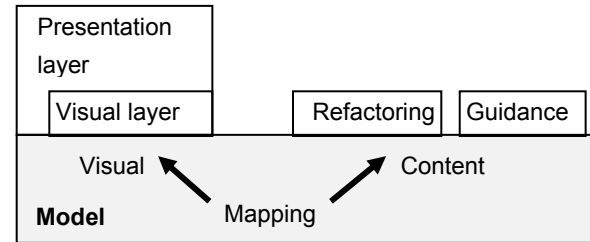


Figure 1: Top-level conceptual architecture for flexible modeling tools

Packages of structure definitions together with additional material to be discussed, such as classification hierarchies, effectively constitute metamodels. This is key to providing domain-specific assistance. However, the forgiving approach to guidance ensures that the metamodel acts as an aid rather than a straitjacket, allowing the user to deviate from it when appropriate. It is also possible to evolve the metamodel along with the model when appropriate, such as during early exploration of new domains. These issues, including the desirability and safety of this degree of freedom, are discussed further in section 3.8.

The architectural elements presented do not constitute a complete tool architecture. A full tool requires many other elements, such as persistence, versioning, atomic model updates, undo, and collaboration. These are not specific to flexible modeling tools and, therefore, are beyond the scope of this paper. However, unique elements of this architecture can have profound implications elsewhere. For example, our flexible support for visual cues with underlying meaning might be used to visualize collaboration, highlighting entities that 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, rather than large quantities of numeric data. The entity-relationship approach is therefore a natural fit. We henceforth assume that information consists of entities with properties, and relations between entities.

Example. Throughout this section, we will illustrate concepts using a running example, based on *issue based consulting*. This approach is used by many business analysts. They conduct interviews and examine material to determine key issues affecting a business, then make recommendations for improvement. In our running example, we

follow a business analyst exploring issues at the Useless-Gadget manufacturing company.

3.1 Visual Layer

This layer is responsible for displaying visual elements and allowing users to manipulate them. Analysts told us that this layer should allow a great deal of flexibility in visually exploring, creating and manipulating visual elements as they focus on the content of their analysis. Office tools provide such flexibility, contributing to their popularity in this domain. Modeling tools typically have a visual layer separate from the model, as in the view and controller of the MVC pattern [9]. If, as usual, the model constrains the information and its structure, constraints show through in the visual layer, limiting freedom: the visual layer is limited to what the underlying model can represent. Flexible modeling tools therefore require particularly rich underlying models, as described in section 3.4.

Visual flexibility depends upon the ability to select from and improvise a variety of views. Analysts often combine diagrams, tables and text into presentations or other documents, and a flexible modeling tool can offer an even richer variety. Multiple views allow users to work with their information in many different ways, facilitating exploration and presentation.

Office tools do offer a variety of document and diagram types, but maintain no relationship between them. Consistency management is, therefore, manual. To reduce this burden, flexible modeling tools must use the same underlying model for multiple views. Furthermore, unlike typical visualizations, all views must allow editing and manipulation of the information they show, with changes propagated accordingly. This has been common practice in programming environments and modeling tools for decades [21], but here applies to a wider and open-ended 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 flexibility inherent in the underlying model, and the fact that the model can be changed through other views, the user might try to apply such a specialized 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, both exploring it to gain understanding and again when communicating that understanding. They make use of a variety of visual cues, including:

Departments	Issues
• Sales	• High employee turnover
• Manufacturing	• High peaks in parts inventory
• Human Resources	• Low summer throughput
	• Hiring skilled seasonal workers
	• Seasonal quality problems
	• Relatively few repeat customers
	• Late fulfillment due to missing parts

Figure 2. Two lists that were assembled from inputs gathered in a series of interviews

Style. Different shapes, colors, line styles, etc, used to highlight distinctions and commonalities. Entities are often drawn as shapes, and relationships as lines or arrows connecting them.

Position. Related entities are often positioned to indicate relationships: nearby, side-by-side, stacked or layered, or arranged in a list or table. A common case is visual containment, typically used to indicate a containment or subsumption relationship.

Other dimensions, such as time and animation remain research challenges from a flexible modeling perspective.

Example. Figure 2 shows lists of departments and issues mentioned during interviews with employees in those departments. Simple lists are typical in very early stages of projects, as information is gathered. Some of the issues are shown with red text, because they were stated as being important.

These sorts of cues provide what we call *visual organization*. They aid cognitive processes involved in organizing and understanding information [6]. They also facilitate effective communication. When used in a consistent and evocative manner, they make, communicate and subtly reinforce important points. When used haphazardly or inconsistently, they can create confusion [15]. As our user studies emphasized, good presenters put a good deal of effort into visual cues.

Office tools offer a wide variety of visual cues, and freedom to make good (or bad) use of them. Users 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 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 asking for all issues, and expect it to use the list shown in Figure 2 to find them. One cannot even query on style, e.g., find all entities colored red. Flexible modeling must enable such queries, without limiting capabilities for visual organization.

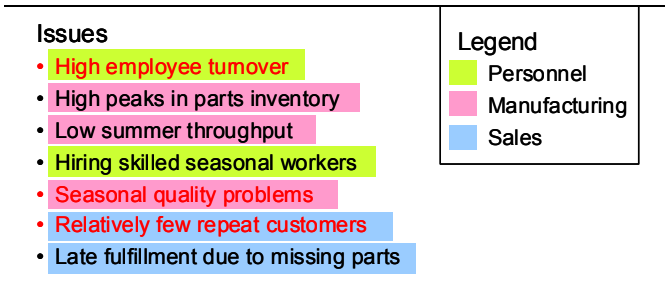


Figure 3. A legend which shows how background colors, used in a list, map to various named categories.

3.3 Content Organization

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

We discovered that users evolve the concepts that go into the final presentation by organizing snippets of information from interviews, prior projects and client documents. *Sets* provide a well-understood means of dealing uniformly with diverse content organization. In office tools, these typically start as lists. Flexible modeling tools treat lists as enumerated (extensional) sets. During exploration, users reorganize the content within and between lists as they attempt to seek appropriate structure. Once a user has organized many snippets, regrouping them using query-based (intensional) sets offers the user further flexibility, reducing the viscosity inherent in manual re-categorization.

Example. Underlying the lists shown in Figure 2 are three sets:

- Departments, consisting of 3 departments.
- Issues, consisting of 7 issues.
- An anonymous set, consisting of the 3 issues shown with red text.

These sets are conceptually distinct from the lists or styles that depict them in Figure 2. Each set could be depicted in many other ways, such as nodes in a diagram or rows or columns in a table. The names of the first two sets can be derived from the list titles. Naming the third set sensibly, e.g., “Important,” requires user intervention to specify the intended meaning of the red text style. The user should be able to name it when ready, but leave it anonymous until then.

Another key form of organization is making distinctions between information snippets, leading to classification. After creating lists, our users notice distinct sets within the

lists. They then create lists within lists – subsets capturing the distinctions. When done recursively this leads to a hierarchy. Classification can be done by first defining categories and then categorizing entities. However, in exploratory activities it is often done the other way around: users often group entities together based on intuitions before they can articulate them, then define the categories as their understanding deepens.

Example. The analyst categorizes issues into Personnel Issues, Manufacturing Issues and Sales Issues. Figure 3 shows these categories in a legend, associating a unique style to each one. These styles overlay the issues list, showing the categorization visually. Any sets, like Departments and Issues, can be used as categories too; given these sets, users naturally talk about “an issue” or say that “Human Resources is a Department.”

Grouping is insufficient for allowing users to make sense of all information and must be complemented by relationships. These too can be captured by query-based sets, such as “all entities related to entity e by relationship r ,” where r denotes a particular kind of relationship.

Example. The user captures which department(s) reported each of the issues. Figure 4 shows this as a diagram with each ‘reported’ relation shown as an arrow. The underlying relationships support queries, such as “What issues did the Manufacturing Department report?,” “Which department(s) reported the ‘High Employee Turn over’ issue?” and “What issues were reported by more than one department?”

Our users often viewed the same set of information snippets from different perspectives. The same entities are often organized simultaneously along multiple *organizational dimensions*, to reflect different relationships noticed by the

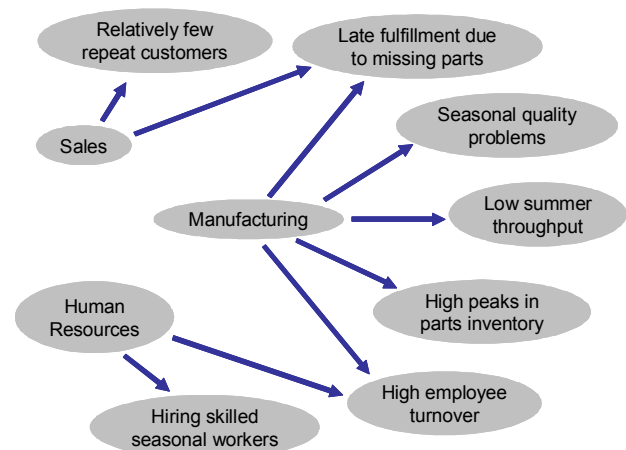


Figure 4. A simple diagram in which ‘reported’ relations between departments and issues are shown as arrows. Note that departments and issues are only distinguished here by their role in these relations

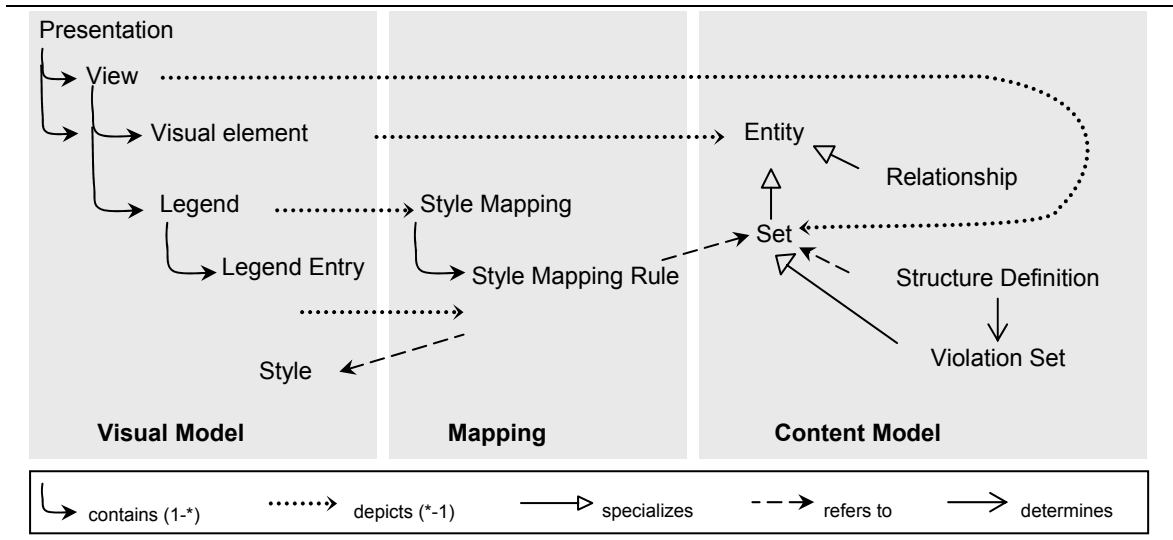


Figure 5. The various types of entity comprising the visual model, content model and mapping.

users. 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 $(s, \{s_1, s_2, \dots, s_n\})$, where s is the set of entities 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 try various organizational dimensions before settling on a few that prove valuable. This is difficult in office tools. Many users do this using stickies and a large wall, and then face the tedious task of getting their work into electronic form for sharing with colleagues. Some users curtail exploration due to the insurmountable difficulty of reorganizing the snippets. Flexible modeling tools must support multiple organizational dimensions. Multiple, simultaneous classifications must be supported, allowing the same entity to be classified in multiple different ways.

A flexible modeling tool must allow users to capture organizations that emerge rather than force-fit content into limited predefined organizations. While the tool may provide a starter collection of useful sets and organizational dimensions, it must also allow the user to define 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 Content Models

As in many modeling tools, our architecture's underlying model is split into two related sub-models: the visual model and the content model (Figure 5). The model is an entity-relationship model, suitable for representing the kind and

variety of information described above. Both entities and relationships can be organized. For example, relationships can be classified according to their *kind*, such as “reported” 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 5 specializes Entity, but this is not shown, to simplify the figure.

The *visual model* captures visual cues and their properties, including style and position. Presentations are explicitly represented—as collections of views and visual elements. The *content model* captures underlying information, with sets representing its organization; structure definitions and violation sets are described below. Entities in the visual model *depict* entities or relationships in the content model, shown as dotted arrows in Figure 5.

Example. Figure 6 shows the visual and content models underlying the view of the two lists in Figure 2.

As reported above, users experiment visually during exploratory activities as a way to gain understanding, thinking more about visual elements than their underlying meaning. During this phase, model elements being depicted are generic: entities or relationships with little or nothing known about them. As user understanding is gained and distinctions emerge, the needs to express them, with the underlying model evolving 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 must have. The flexible approach to content organization deals with classification in a less restrictive way: entities and relationships can be classified in multiple ways simultaneously, and be reclassified at

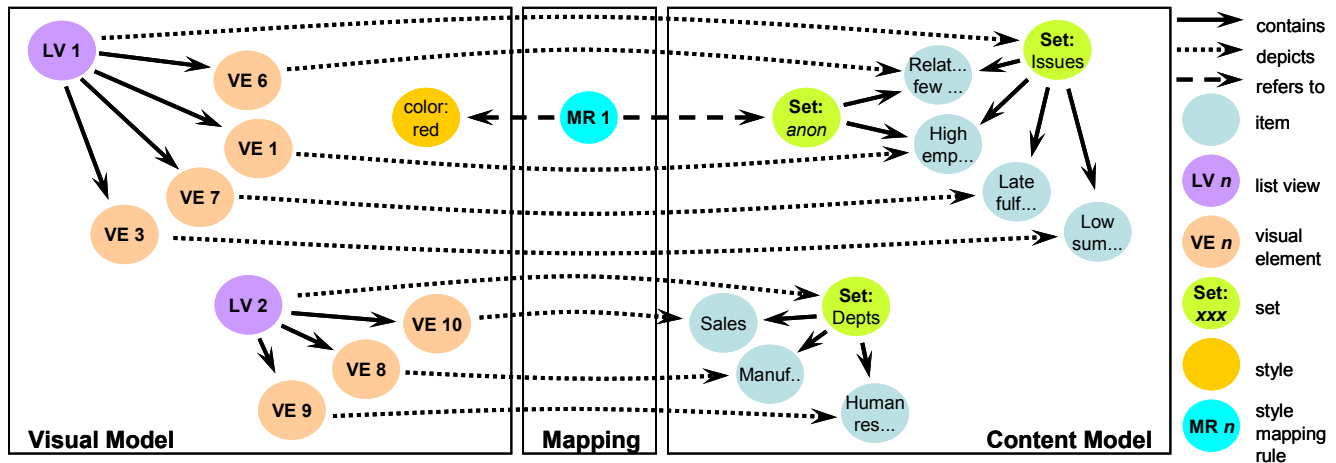


Figure 6. Visual and content models underlying the lists shown in Figure 2.

will. Structural properties are specified using *structure definitions* (Figure 5, right), 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 its structure might be suitable for the old category but not the new one. It remains valid content, however, and the guidance component will guide the user through the process of fixing the structure.

Example. Initially, the entities in the issue list were just generic entities. After some categorization, they became Personnel Issues, Manufacturing Issues and Sales Issues. Later, a category-specific guidance rule was added stating that any Sales Issue should have an “affected region” relationship to an item classified as Sales Region. Sales Issue then behaves rather like a type, in that it describes expected structure. However, the analyst can still work with a model that violates such rules, and can later decide that Sales Issue is not a useful category after all and should be dropped.

The relationship between content and visual models is deeper than just depictions, however: it also provides a mapping between content and visual organization. This allows users to specify visual cues to be used uniformly in manifesting aspects of content organization – a capability missing from many modeling tools. This reduces interface 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 and color will be propagated automatically.

3.4.1 Style Mapping

Style is a key part of visual organization. Analysts reported using office tools’ styling to capture various content dimensions of their information. However, manually setting and maintaining correct styles was time-consuming and error-prone. Flexible modeling explicitly maps style to underlying content organization (Figure 5, center). This mapping enables tool support that reduces user errors and effort, while enabling visual exploration to uncover the dimensions along which content is organized. Such support also reduces the effort required to visually represent distinctions users have noticed but not recorded.

A *style*, following common practice such as CSS⁴, is a set of *attribute:value* pairs. We interpret style attributes liberally, complementing typical ones like fill color with shape, decorations and any other visual cues provided by a view.

Example. Figure 7 shows issues of various kinds. Important personnel issues are depicted using the red stick figure style:

```
redFigureStyle = { shape:stickFigure, fillColor:red }
```

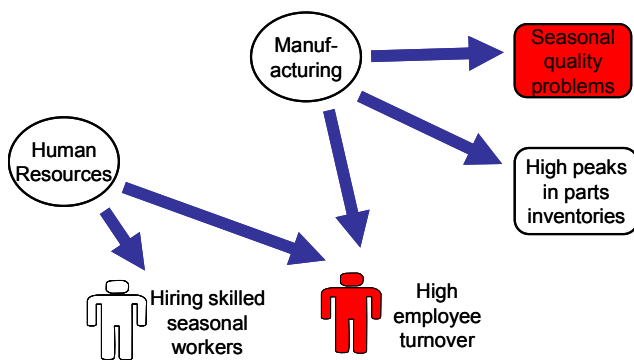


Figure 7. A simple diagram in which the styling of a visual element as a red stick figure indicates that it is an important personnel issue.

⁴ For example, see <http://www.w3.org/TR/CSS2/>

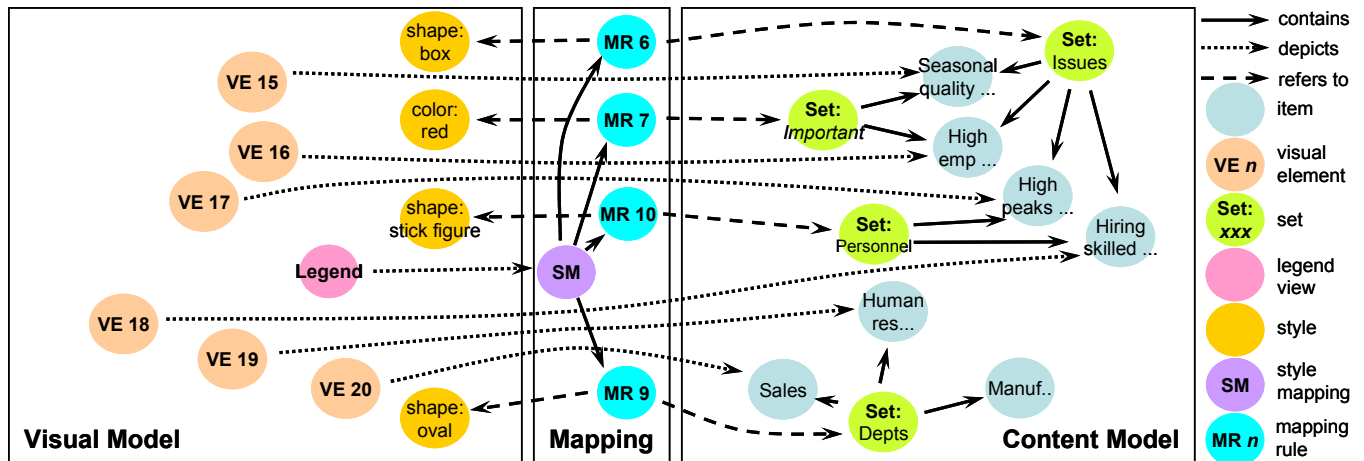
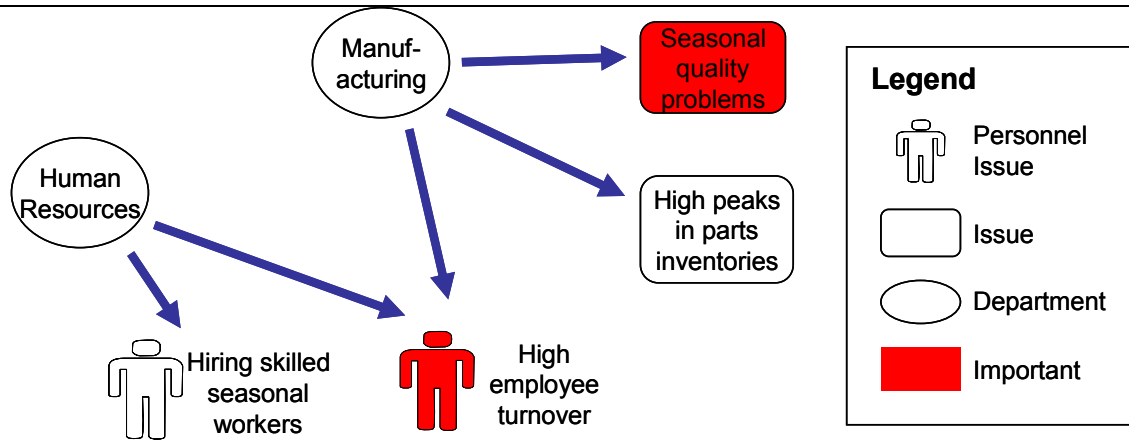


Figure 8. The style mappings that were implicit in figure 7 are revealed in the form of a legend, both visually (above) and as represented in the underlying model (below).

A *style mapping rule* is an ordered pair $s \rightarrow y$, where s is a set and y a style, specifying that depictions of members of s should be styled according to y . For example, both style mapping rules

- r_1 : all issues classified as “personnel” $\rightarrow$ { shape:stickFigure }
- r_2 : all entities classified as “important” $\rightarrow$ { fillColor:red }

apply to 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 style mapping rules. Styles used in these rules must be ones that the view can render; for example, list or table views might not support shape variations. The style mapping could be manifested to the user, as a *legend*. This al-

lows users to see what the styles mean, and can be a place for editing style mappings. Changes immediately affect the styling of all appropriate visual elements in the view.

Example. Figure 8 shows the diagram in Figure 7 along with an accompanying legend and some of the underlying model items, including the style mapping. Note that the legend is a depiction of the style mapping, with each legend entry depicting a single style mapping rule.

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 a set of such style attributes 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.

Example. The diagram in figures 7 and 8 was based on mapping the organizational dimensions “issue kind” (with values “personnel,” “sales,” etc.) and “importance” (with values “important” and “insignificant”) to certain shapes and fill colors, respectively, so that different issue kind values are displayed as different shapes and different importance values are shown as different colors. This use of styles is typical in diagrams created everyday by our users.

Our users reported sophisticated style coordination across sets of views. For example, consistent styling is often desired across a presentation, as it aids understanding. However, subsections may purposely change a style to emphasize some point. One user included diagrams done by another team that had shared meanings with the rest of the presentation. To remain true to the originals, she used original styles for these diagrams only. 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 an inherited style mapping rule, the question arises whether the user intends to change the local view only, or the base. User experience remains a research challenge.

Multiple style mapping rules within a style mapping may 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. *Style ambiguity* occurs when the same style attribute is mapped to different sets, leaving the user to guess what each occurrence means. Implementations based on this architecture must cope with clashes and ambiguity in a non-intrusive manner, ideally using the guidance mechanism (section 3.5).

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, with mapping rules specifying position (e.g., placement or containment) instead of style. This requires a model of position instead of simple *attribute:value* pairs. Some views offer continuous positioning, such as diagrams allowing arbitrary placement, whereas others offer discrete alternatives, such as tables allowing 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, contrasting this to tools that restrict user interactions to those that respect a metamodel, a form of enforcement that puts users in

a straitjacket. In contrast, 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 work within an analysis framework developed on previous projects, guidance can help the user work within it and conform to it. Our architecture therefore contains a guidance component for which we identify and justify some 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 in *structure definitions*, which specify a variety of structural constraints on the model. For example, every item classified as an issue should be classified along the “importance” dimension. Structure definitions about content entities and relationships, like this, are specified in terms of the content model, not the visual model. However, structure definitions can also cover the mapping between content 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. Typing usually brings a degree of viscosity: changing an entity’s type is difficult or impossible in some models. Abstraction can also be compromised, because an entity can usually have just a single type. Structure definitions in combination with content organization, however, provide key benefits of typing, as noted above. Structure definitions effectively cause referenced categories to be like types, but with greater flexibility.

Example. Figure 4 showed ‘reported’ relations between departments and issues. Quite early on in the project, the analyst registered this structure as a structure definition:

reporting: Every issue should be the target of a ‘reported’ relation from a department.

The specific nature of 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 with *reminders* shown in a separate view [1]. This *negative guidance* alerts users when their model violates specified structures.

Example. Given the *reporting* structure definition above, a reminder is shown for any issue with no incoming ‘re-

ported' relation, reminding the user to provide one in due course. A reminder is also shown for any 'reported' relation between the wrong kinds of items, i.e., not from a department to an issue.

However, as AWB users confirmed, it is equally 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 avoids premature commitment: the user can draw a line without first deciding what relation it depicts. It is cumbersome, however, when the kind of relation is clear—it is more helpful for the tool to create the right kind automatically. The user might still change it, if desired, but that is expected to be an exception.

Example. Given the *reporting* structure definition above, the tool automatically creates 'reported' relations whenever an arrow is drawn from a department to an issue. If the analyst decides that a particular arrow depicts a different relation, e.g., 'addresses', s/he can change it to that. This will cause a temporary violation, which might be ignored or might lead to another structure definition.

If, as our users reported, the analyst uses a framework defined by a senior analyst, useful structure definitions might already have been specified, and can be imported. These would provide useful guidance based on past experience.

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

Example. If the reporting structure definition is complemented with an 'addresses' relation, the analyst can be offered a choice between these two when creating a relation between a department and an issue.

The manner in which guidance is actually given to the user through the user interface can significantly affect 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. Given our users' desire for flexibility, we avoid more stringent approaches, such as forcing the user to correct each 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* (Figure 5, bottom-right) of items or relationships involved in guidance violations, resulting in visual cues indicating the violations. Like all style mapping rules, these are manifested in the legend, making the meaning of the cues clear, and allowing the

user to change them, to control their obtrusiveness, and even their presence.

Different degrees of flexibility are appropriate at different times and in different settings. For example, 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, with policies based on attributes such as user role. However, it is an open question whether such measures are necessary or desirable.

Material developed during pre-requirements analysis is critical to downstream activities, such as requirements engineering and project planning and management. Structure definitions provide the key to integration between flexible modeling tools and other, traditional modeling tools used in a project. When exporting a model fragment from a flexible modeling tool to a standard modeling tool, it must conform to the target metamodel. Structure definitions effectively defining that metamodel can be loaded, allowing the guidance component to check for conformance and helping the user to achieve it. Once all violations have been removed, the model is suitably structured for direct export to the standard modeling tool.

3.5.3 Guidance checker

The *guidance checker* is responsible for evaluating (relevant) structure definitions when changes occur or are underway, to determine both violations and what positive guidance to give. Positive guidance places greater demands on a checker than is typical. Both 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 exploratory work, users must be able to try things without worrying about whether they are exactly right, and then change them later. This reduces premature commitment by making change, and thus iteration, easy. Rich views with an underlying flexible model support many kinds of changes, but some structural changes require refactoring support. For example, when extracting important statements from interview transcripts, one might quickly categorize them based on who made the statements. Later on, however, it might be preferable to use 'said by' relationships between statements and interviewees instead, or even 'reported' relationships. This makes the nature of

the relationships explicit, allowing other relationships between statements and stakeholders. This refactoring would be tedious and error-prone to do by hand: explicit refactoring support is preferable.

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 of understanding of the business situation together with implications and recommendations. User challenges are to determine a good narrative structure and ensure that each key point is backed by appropriate supporting material.

The presentation layer provides *presentation views* for working with presentations. All views are candidates for incorporation into presentations. Each presentation unit (e.g., slide) is 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.

As noted in section 2, pre-requirements analysts (in common with many other users) often use *templates* when constructing presentations. A template in this context is a guide or pattern that enables the creation of presentation elements of some specific form. They range from the simple, such as slides containing bullet lists, to the complex, such as organization charts. Such templates often derive from prior projects. Though sophisticated users might engage in template definition using whatever means their presentation tools offer, it is more common to pick up an example, remove the old (and probably confidential) data, and use the resulting skeleton as a template.

The presentation layer of a flexible modeling tool should therefore provide a means to create templates from existing presentation fragments, and to use them conveniently in new presentations. The templates are primarily artifacts in the visual layer, represented and stored in the visual model. However, it is also important that a template record what is to be done in the content model as it is filled in. For exam-

ple, an organization chart template would typically be tied to an underlying model of the organization and its employees, and this model would be updated as the template is filled in.

Constructing a presentation brings a fresh perspective to the exploratory and envisioning role of the user, causing them to revise and enrich their understanding. The presentation layer therefore allows full manipulation of viewed content. 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 require further research.

Explicit style mappings play additional roles within presentations. Legends may accompany each view, and in interactive presentations, may present the reader with controls to 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*.

3.8 Metamodel Analogs and Evolution

Now that all the elements of our flexible modeling conceptual architecture have been presented, it is worth revisiting the issue of metamodels. The function of a metamodel in a traditional modeling tool is to define the permissible structure of models, which is then enforced by the tool. The metamodel embodies much domain-specific understanding, allowing the tool to provide domain-specific assistance and functionality. The metamodel often ensures conformance to industry standards, and facilitates interoperation with other tools. In the case of flexible modeling tools, we clearly want the enforcement to be replaced by forgiving guidance. It is still useful in many situations, however, to help the user build models that have expected, and perhaps standard, model structures in a particular domain.

Several of the architectural elements described above work together to enable this:

- Classifications specify domain concepts and (some of) their relationships. They serve the role of types.
- Structure definitions specify expectations and constraints about model structure.
- Sets and organizational dimensions embody useful ways of organizing content in the domain.
- Templates specify useful ways of organizing and presenting content visually.
- Style mappings manifested as legends specify useful visual conventions for displaying content.

We call a package made up of a coherent collection of these items a *thought framework*, because it supports work following a particular way of thinking (such as issue based consulting, used in the running example earlier). A thought framework effectively constitutes a metamodel. When loaded into a flexible modeling tool, it enables the tool to help the user work in the supported fashion. It allows the tool to present tailored diagramming views, with appropriate palettes and visual conventions. It allows the tool to show the user where expectations are violated, and often to help the user avoid or remove violations. It allows the tool to export models that conform to the metamodels of standard modeling tools.

Users working in a new domain can start from scratch and just do free-form modeling. An initial thought framework will naturally evolve as they proceed, for example as they create classifications and sets, and structure definitions aimed at helping them with their work. They might import (fragments of) existing thought frameworks along the way, as they see fit. As their understanding of the domain matures, or at the point that it would be useful to another project, they can do some explicit work to harden the thought framework and make it suitable for sharing. The process of supporting non-expert users in creating thought frameworks, both by deducing as much as possible from their regular modeling activities and by providing convenient means for them to explicitly examine and manipulate the thought framework, remain research challenges.

Users working in an established domain would generally be expected to start out by loading domain-specific thought frameworks. These might have been produced as just described, or be provided together with a flexible modeling tool, providing a good out-of-the-box experience for the domain, or be obtained from domain experts, or developed within an organization for use in multiple projects, etc.

Users working within an existing thought framework might still need to deviate from it at times, for example to introduce new relationships that capture connections not foreseen by the thought framework authors. Thanks to the forgiving guidance, they can do this, resolving violations later or not at all. If they determine that the thought framework is deficient, they can enhance it, and potentially share the enhancements. This ability to evolve the metamodel during and as part of modeling is an important property of flexible modeling tools. In contrast, changing the metamodel of a standard modeling tool generally requires an expert working in a different language and environment, and requires the tool to be regenerated, or at least restarted, and data to be migrated.

This ability to violate, and especially to change, a metamodel is valuable in some contexts and detrimental in others. It tends to be valuable during exploratory activities, especially in new domains, where supporting free-wheeling thinking and exploration is especially important. It tends to be detrimental in cases where work is being done on a shared, established model, such as an enterprise architec-

ture. Changing the metamodel on the fly could lead to lack of understanding or misunderstanding by other stakeholders and failure of other tools that rely on the model structure. Even in such cases, however, some free-form exploration is often valuable, such as when considering possible futures based on an enterprise architecture; it is only if and when it comes time to change the architecture itself that rigid conformance is needed.

Discretion is thus needed in the use of flexibility. Such discretion could be put in the hands of individual users, with purely cultural pressure to do the right thing. In more controlled environments, the discretion could be put in the hands of senior stakeholders, projects leaders or managers, who would specify role-based restrictions to be enforced by the tool. The important thing is that a flexible modeling tool provide a range of possibilities to suit different scenarios.

4. The BITKit Prototype

BITKit is a prototype tool for business stakeholders involved in pre-requirements analysis [19]. We started by building minimal capabilities, adding enhancements only when necessary, including based on observations of and feedback from senior business analysts who test-drove early versions. Our work on BITKit inspired the architecture described above, but we extended and generalized beyond BITKit's current capabilities. This section briefly describes BITKit as a concrete yet limited instance of the architecture.

BITKit, shown in figure 9, is built as a Rich Internet Application using Adobe Flex⁵. Persistence is provided by streaming XML to a REST server, or storing it on the local disk (when running under Adobe AIRTM). The underlying model is founded on entities, called *items*, and directed, binary relationships, which are themselves items. 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 sets, along with standard set operations like intersection and union, provide an open, extensible means of expressing queries. This approach enabled us to start quickly and fill holes as needed.

We chose *tagging* as our means of classification. Tagging is a flexible, informal approach to organization and popular on the Internet (e.g., Delicious⁶, flickr⁷). It is lightweight, allowing users to attach tags to any entities. It appears to be particularly effective during early exploration.

⁵ <http://www.adobe.com/products/flex/>. Adobe, Flex and AIR are trademarks of Adobe Systems Incorporated.

⁶ <http://delicious.com/>

⁷ <http://www.flickr.com/>

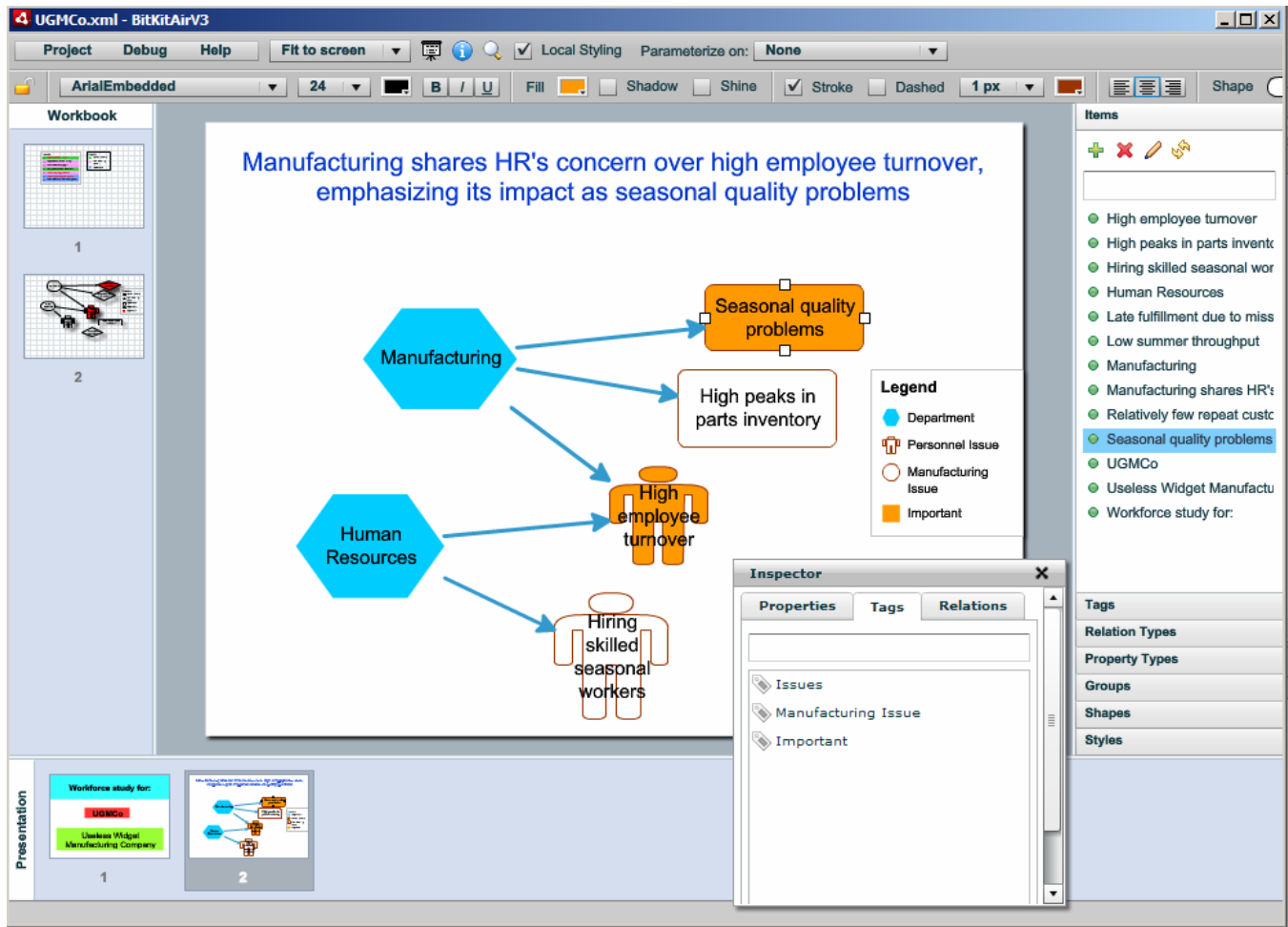


Figure 9. A screenshot of the BITKit prototype, showing a simple project based upon the running example of this paper. The screen consists of a simple workbook (left) and presentation (bottom left), containing two worksheets and two slides respectively. Of the various model views shown on the right, the items view is currently open. At center is a presentation slide displaying several issues and departments and their relationships, all styled by the style mappings shown in the slide’s legend. “Seasonal quality problems” is currently selected and its tags are being viewed in the “inspector.”

Multiple classification is supported by attaching multiple tags to entities. Although tags might be used with little thought initially, they can be defined and organized as understanding increases [20]. The business analysts we worked with were mostly not used to tagging, and their understanding and appreciation increased when we re-explained them as lightweight classifications.

Early feedback led to *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. Not surprisingly, feedback revealed the need for *implication* relationships among tags, which would support classification hierarchies.

BITKit provides three kinds of view: diagrams, lists and tables [20]. Diagrams depict items as shapes and relationships as lines; items are manually placed on diagrams. A list is founded on a set, displaying all items in that set. Like all BITKit views, lists can be changed by the user, causing the underlying set to be updated. 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 an optimal user experience 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 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 naturally support shape variation. Experience 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 additional style attributes including decorators.

As with sets, BITKit offers a small but extensible set of structure definitions. Currently, it supports implications involving sets (every item in s_1 must also be in s_2), 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, if the user creates a “sub-issue” relation, the source and target are automatically given “Issue” tags, if they do not have them already. Violations are currently 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 allow violations to be depicted via style mappings. 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, views are aggregated into both *workbooks* and *presentations*. Users explore using workbooks, and then copy views into presentations. As practice areas mature, we expect workbooks to become fairly stable templates, guiding analysts in their 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 analysts used BITKit. Because tagging is so lightweight, it is used early on where relationships are more appropriate. Also, it is not always clear 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 easily change decisions later.

Earlier preparations for pilot use in a real industrial context – to support consultants in evaluating software development organizations and recommend improved practices (e.g., requirements management or change management) – led to the need to support *collateral material*: standard material, perhaps from prior projects, largely visual but perhaps also with underlying content, that can be adapted and reused. Users want to copy parts of it into their models and make local changes, and to parameterize it, effectively creating templates. We have begun experimenting with approaches to abstracting and parameterizing existing models.

5. Related Work

Requirements modeling tools, such as GRAIL/KAOS [3] and Rational Requirements Composer⁸, embody specific metamodels. Extensible modeling tools, such as RSM⁹, also embody specific metamodels, but allow some degree of flexibility, through user-defined stereotypes and profiles. 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 Architect¹⁰ 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 introduced *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.

⁸ <http://www-01.ibm.com/software/rational/announce/rrc/>

⁹ www.ibm.com/developerworks/rational/products/rsm/

¹⁰ <http://www.telelogic.com/Products/systemarchitect>

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 guide users with domain-specific structure definitions, but with the freedom to deviate from them. Violations thus represent cases where the content model violates whatever structure definitions are in use, and not necessarily inherent inconsistencies within the models. These violations might be repaired as work proceeds, especially if the models are to be passed to downstream tools. Sometimes, however, 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 area of inconsistency handling in requirements tools is to represent and work with mutually-inconsistent information from multiple stakeholders. This requires a model that reifies multiple perspectives, allowing them to be inconsistent and their relationships explored, such as the Viewpoints model [7]. Support for multiple stakeholder perspectives, and 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 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. Although it is unclear how these sketch-based interfaces would be received by business stakeholders, and how much they would help, we postulate that would be a valuable addition to pre-requirements tools and to flexible modeling tools in general.

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 model 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 a 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.

Acknowledgements

We thank Barthélemy Dagenais and the anonymous reviewers for many helpful comments.

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 vis-*

- ual 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] G. Nordstrom, J. Sztiapanovits, 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