



University of
Zurich^{UZH}

*Martin Glinz
Dustin Wüest*

A Vision of an Ultralightweight Requirements Modeling Language

TECHNICAL REPORT – No. IFI-2010.0006

July 2010

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland



A Vision of an Ultralightweight Requirements Modeling Language

Martin Glinz, Dustin Wüest
Department of Informatics
University of Zurich, Switzerland
{glinz, wueest} @ ifi.uzh.ch

Technical Report 2010.IFI-2010.0006 – July 2010

Abstract—Despite all efforts in creating and disseminating requirements modeling languages, natural language is still the dominant language for writing requirements specifications in practice. Furthermore, when documenting early requirements, natural language (in combination with pictures) outperforms today’s requirements modeling languages.

In this paper, we present a vision and research roadmap for an ultralightweight requirements modeling language which can be used as easily as natural language with pictures, but has a visual structure with some lightweight semantics that allow the visual expression of hierarchical structure, context, general relationship, flow, and influence, while all the details are specified in natural language, both form-based and free-text. Furthermore, it should be possible to evolve parts of such an ultralightweight model into classic models of structure and behavior with full-fledged semantics by incrementally adding more formal model elements and tightening the meaning of the already existing ones.

We envisage that such a modeling language – when supported by appropriate tools – will (1) outperform natural language requirements specifications with respect to comprehensibility, changeability, analyzability and internal traceability, (2) be simpler and more straightforward to read and create than today’s heavyweight modeling languages, (3) provide an efficient and effective means for expressing requirements at an early stage.

Keywords—*requirements modeling language; ultralightweight; natural language specification;*

I. INTRODUCTION

The idea of describing requirements in a requirements modeling language has been around for more than thirty years now [24]. Since then, a plethora of modeling languages has been developed and used for modeling requirements, including the ubiquitous UML [17]. In our own work in Zurich, we have more than ten years experience with the development of the requirements modeling language ADORA [7][20].

Despite all effort that went into the development of requirements modeling languages, the vast majority of requirements specifications created today in industry are still written in natural language, augmented with tables, pictures, and, increasingly, some isolated model diagrams. Standards and templates such as the IEEE guideline for software requirements specifications [13] or the Volere template [21] are also based on natural language. This situation is not just due to the inability of industry to adopt the existing modeling technology. It is a strong indicator

that heavyweight modeling languages such as UML don’t fit the needs of industrial requirements engineers [6].

Another strong indicator of the requirements modeling malaise was the “Next Top Model” contest at RE’09 [12], where the combined power of natural language and rich pictures outperformed all modeling approaches in the task of specifying a requirements problem [10] at an early stage.

Furthermore, a lot of the so-called non-functional requirements [8], in particular quality requirements and constraints, can’t be expressed as models and have to be written in natural language.

This situation motivates us to propose the creation and use of an ultralightweight modeling language (or ULM, for short), a small language with little formal expressive power, but one that easily integrates with natural language, helps structure a natural language requirements specification and provides some simple modeling constructs for those things typical requirements engineers and stakeholders hate to express textually: structure, relationships, influence, and flow. On the other hand, the envisaged language shall be constructed such that powerful tool support for editing, navigating, and analyzing specifications is possible.

We don’t aim at replacing heavyweight modeling languages such as UML or ADORA, but at improving that huge number of textual requirements specifications that have no structure beyond a chapter-section-subsection classification and are not analyzable beyond careful reading.

In this paper, we make a case for such an ultralightweight modeling language, sketch how it could look, describe potential tool support and present a roadmap towards such a language and tool. A condensed version of the ideas laid out in this paper will appear in [10].

II. CORE REQUIREMENTS FOR A ULM

Requirements specifications at an early stage are by their very nature mainly narrative and pictorial: most of the information transmitted from stakeholders to requirements engineers comes in this form. Hence, we have a first core requirement for a ULM:

R1. A ULM shall provide strong support for writing textual requirements and drawing pictures.

On the other hand, we want to harness the power of modeling for overcoming the greatest weakness of text and pictures: their unstructuredness and total informality. Hence we have two further core requirements:

R2. A ULM shall provide model elements for structuring text.

R3. A ULM shall provide model elements for drawing pictures as diagrams, with individually identifiable elements and some very lightweight semantics.

Visual languages are strong in providing an overview of a problem with little cognitive effort, while for the description of details, textual notations are superior to visual ones:

R4. A ULM shall be visual in the large and textual in the small.

One of the biggest advantages of modeling languages over textual ones is that the syntax and semantics of models can be exploited by tools:

R5. A ULM shall enable the creation of tools that provide powerful means for editing, navigation, selective visualization and analysis of specifications.

When competing with natural language and pictures, both readability and writability of models is of paramount importance:

R6. A ULM shall be easy to read, to write and to learn.

As a corollary, we can state that a ULM needs to be a small language in terms of modeling elements.

Many of today’s modeling languages are visually ill-designed [16]. In order not to repeat old mistakes, we state:

R7. The visual syntax of a ULM shall be well-designed with respect to the design principles for visual notations.

III. DESIGNING A ULM

A. Design Considerations

Based on the requirements presented in the previous section and on our own experience with visual modeling language design, we now discuss design considerations for a ULM that go beyond the general principles stated in [16].

Hierarchical structure. Requirements specifications are organized in hierarchies: document hierarchies (section-subsection-paragraph), component hierarchies (system-subsystem), classification hierarchies, etc. There are two intuitive visual structures for expressing hierarchy: nesting and trees, the latter coming in two flavors (Fig. 1).

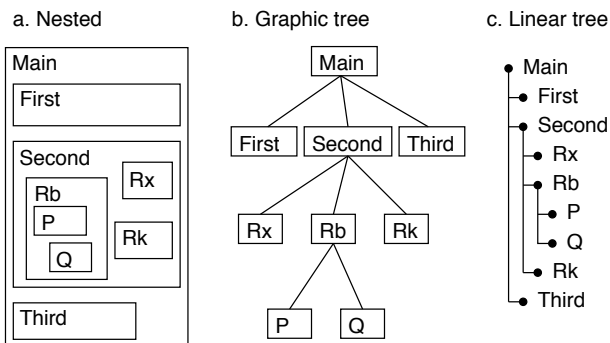


Figure 1. Options for intuitive visualization of hierarchy

Both tree options have an implicit connotation of elements on the same hierarchical level being ordered according to their visual presentation, which is frequently not the case and thus misleading. Furthermore, graphic trees need more space than a nested notation and linear trees are visually inflexible. On the other hand, tool support for viewing only selected parts of a hierarchy is trivial for linear trees and rather easy for graphic trees, while it is hard for nested structures with an arbitrary layout. How-

ever, this problem has been solved [20]. Hence, we prefer nesting for a ULM.

Contextualization. Whenever only a part of a model is visualized, this part needs to be *contextualized*, i.e. the surrounding context is required for understanding this part of the model [16]. Some existing modeling languages, UML being the most prominent example, just ignore this problem, thus forcing readers to reconstruct the context by mentally merging various diagrams. Fisheye views [5] [2] [20] are a proven technique for contextualizing partially visualized models. From a modeling language point of view, nested hierarchical structure suffices for contextualization, as such structures allow the application of fisheye views. However, fisheye views can become clumsy when viewing inner parts of large models and they can’t visualize the full context when an element is part of more than one hierarchy. Hence, for a ULM it is recommendable to choose a design that additionally provides the option of explicitly stating one or more hierarchical paths for a given model element, thus explicitly defining the context of this element.

Small visual vocabulary. Any ULM design will use a small visual vocabulary, otherwise one would eventually end up with a new normal-weight modeling language instead of a lightweight one. The requirement that the language needs to be easy to learn also leads to a language which is small with respect to the number of its visual constructs. Using graphic modifiers for the objects of the language (see Table I in next subsection) can further reduce the number of concepts to be learned. Despite the small vocabulary, the expressiveness of a specification written in a ULM must be ensured. This can be accomplished in two complementary ways that both are based on the principle of textual enrichment: (i) embedding natural language text in the language (e.g., as a part of an object) provides the full expressive power of natural language, but at the expense of analyzability; (ii) textual attributes similar to UML’s tagged values [17] or ADORA’s standardized properties [7] provide analyzable detail information, but with limited expressive power.













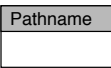



Naming. A major drawback of natural language specifications is that natural language does not distinguish nouns that denote an identifiable concept from other nouns that don’t. As a consequence, any phrase that shall be identifiable and traceable must be assigned an explicit identifier by the engineer who writes the specification. A ULM can (and must) add value here by providing a flexible naming concept that allows to tag a word or phrase to be a name and to create a traceability link just by referring to such a name elsewhere. Even when the person who writes a specification does not set any explicit traceability links, such a naming concept improves traceability: having a set of known names improves both the precision and recall of automatic traceability retrieval in comparison to a situation where names need to be guessed/inferred by the retrieval algorithm. Furthermore, a good ULM design makes it possible to annotate icons in a picture with traceable names.

Semantic enrichment. A ULM model will be more valuable (and, hence, more worthwhile to create) when the ULM provides an evolution path to full-scale models by incrementally enriching ULM models with additional syntax elements and semantics of the target modeling

language or by providing a straightforward transformation from a ULM model into a model expressed in the target modeling language.

Lightweight analysis. The additional effort required for modeling a problem in a ULM instead of using plain natural language and pictures is only justifiable when there is a clear benefit in terms of better analyzability of the ULM models (and, as a consequence, also better comprehensibility). Some analysis capabilities, e.g. the analysis of hierarchical structure and context, just pay off by enabling powerful tool capabilities for editing, navigation, tracing and selective visualization support. Further, a ULM design needs to support model validation. For example, one will choose a design that makes incompleteness analysis possible to some extent (e.g. names that are used but not defined) or exploit the semantics of relationships.

TABLE I. A PRELIMINARY SET OF LANGUAGE ELEMENTS

Objects	
	Technical item (object, component, activity, function, service, goal, system, device, ...)
	Living item (Person, actor,...)
	Connector
	Picture
Relations	
	Static relationship
	Influence (affects, uses,...)
	Flow (of information, data, control,...)
	Hierarchical structure
Modifiers	
	External
	Fuzzy
	Multiple
	Boundary
Context	
	Pathname provides the context for the element the grey box is attached to
Missing / hidden information	
Name...	Incompletely specified object
[...]	Object contains information not shown on this diagram
	A relation having annotations not shown on this diagram
Enrichments	
	AND-Connector
	OR-Connector

B. A Preliminary Language Design

In this subsection we sketch a concrete ULM based on the design considerations and core requirements given above. Table I summarizes the visual syntax. The speci-

cation in the appendix gives an impression of the look and feel of the language. This is *not* meant to be a complete and polished language design. Our intention is to make our vision of a ULM more concrete, tangible and criticizable.

Objects. A model in this language is a set of *objects* that may be specialized by *modifiers* and can have *relations* among each other. Technical items can have a *hierarchical* inner structure. The *context* of an object is given by its embedding in a hierarchical structure or an explicit context name path. An object may have multiple contexts.

Names. Objects may have a name and an additional shorthand identifier. Names, when prefixed with any of their context paths, must be unique. Fig. 2 illustrates the naming concept.

Object content. The content of an object can consist of

- other objects, possibly linked by relations, and
- text in natural language, possibly with links to other objects.

Missing / hidden information. Incompletely specified objects are marked with an ellipsis after their names. If some content of an object is hidden from a diagram (e.g. because it is specified separately or because a diagram is intended to give an overview only), this is indicated by a '[...]' symbol. Suppressed annotations on relations (see below and Fig. 3) are marked by a pull down handle symbol.

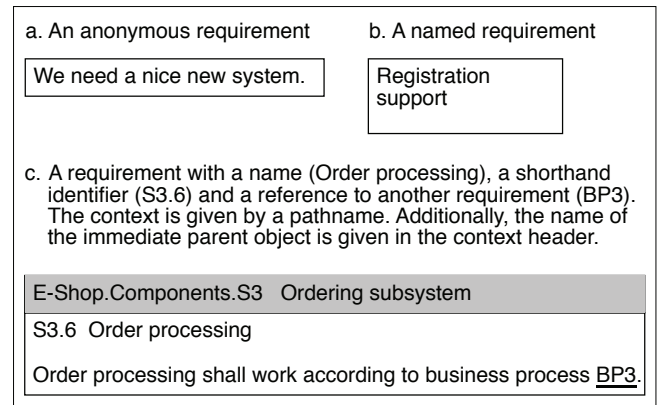


Figure 2. A naming concept for a ULM

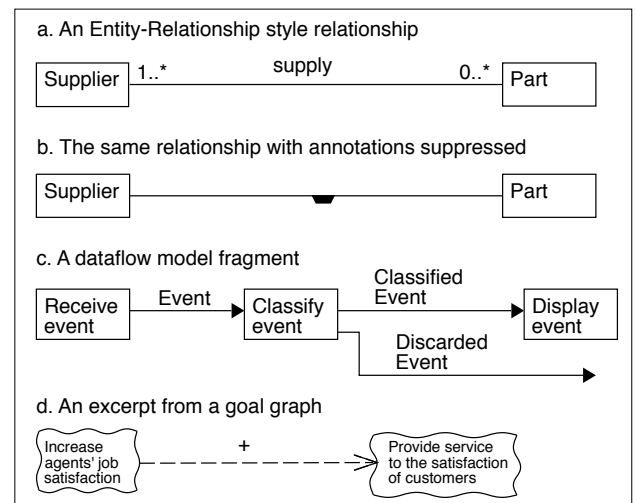


Figure 3. Examples of relation annotations

Enrichments. The basic objects and relations may be enriched with additional semantics. Table I specifies two enrichments for connectors that are useful for describing flow. Other enrichments may be needed.

Annotations. Relations may be annotated in the middle and at both ends (Fig. 3). In order to keep models readable, the display of annotations can be suppressed.

Attributes. Any object or relation may have metadata attached, such as author, status, date created, etc. Furthermore, modelers can define typed attributes similar to UML’s tagged values or ADORA’s standardized properties. Fig. 4 illustrates the concept. As described in the previous subsection, typed attributes provide information in a form that allows analysis by tools.

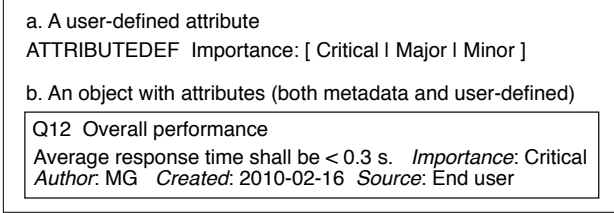


Figure 4. Attaching attributes to an object

C. Examples

The model in the appendix provides examples: GC FEDS-Spec is an object representing a document. Its top-level nested objects serve as an organizational structure.

On lower levels, for example in the Business goals object, nested objects are arranged in a diagram, in this case a goal graph. The ‘fuzzy’ modifier is used to denote soft goals.

The details of the Core requirements object are hidden from the overview diagram. The ‘[...]’ marker indicates that more details are available. On the other hand, the Glossary of terms object is incomplete. It is displayed in full (no ‘[...]’ marker), but more needs to be specified (name followed by ‘...’).

The underlined word incident in the definition of Dispatcher in the glossary is a reference to an object with that name.

In the objects contained in the Stakeholders object, *Importance* and *Goals* are typeset in italics, thus indicating that these are attributes that are followed by values.

The object BP6, the Emergency Call Process, is part of two hierarchies. By its visual embedding it is part of the Core requirements within the GC-FEDS-Spec. On the other hand, an explicit context path states that this object is also part of a Business processes object within an object called FDGC.

Core requirement R4 contains a named, but uninterpreted picture.

On the second page of the model, the Call processing object is shown in isolation. In order to contextualize it, the context paths of the two hierarchies that this object is embedded in are given.

In the System context object, the external and boundary modifiers are used to mark the context boundary of the GC-FEDS system and indicate which elements are external to it.

IV. TOOL SUPPORT

A software tool that supports modeling in a ULM is an indispensable part of our vision. Requirements engineering specifications can become very large artifacts over time. Editing and maintaining such artifacts is cumbersome without appropriate tool support. A tool can help in editing the requirements, it can ease and speed up navigation, and it can provide different views on the specification, thus providing abstractions and reducing complexity. Furthermore, a tool can provide support for requirements traceability and model validation.

When supporting a ULM, we need tool capabilities for handling textual specifications way beyond the basic editing, searching and configuration support that text processing software and classic requirements management tools provide today. Equally, when handling pictures, we need tool capabilities beyond those of graphic editors. Furthermore, when eliciting requirements at an early stage, both engineers and stakeholders love sketching requirements on whiteboards or blank sheets of paper [19], a procedure that goes well together with using a ULM and also needs to be tool-supported, hence.

A. The Tool Metaphor: a Large Canvas

A tool supporting a ULM should have the same flavor as the supported ULM itself: it should be lightweight and provide a simple and intuitive user interface.

We envisage that a tool presents a large 2D canvas to the users, mimicking a classic whiteboard, but with infinite space. The users can create and freely arrange the objects of the requirements specification in that 2D space, using a pointing device (e.g., a mouse) for drag-and-drop manipulation, a keyboard for entering/editing text and a pen or fingers for sketching.

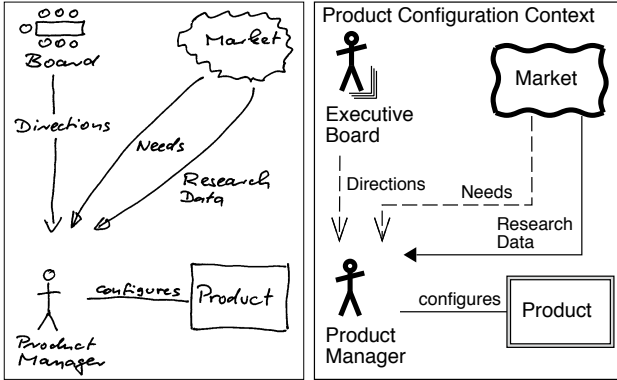
B. Editing

Mouse and Keyboard. The tool can display a drag-and-drop interface for a fast construction of diagrams and the structure of the specification. Natural language is heavily used in requirements specifications. Therefore keyboard input allows for a fast creation of textual elements. Standard editing features like an undo option can further facilitate the editing of documents.

Natural drawing. A ULM allows efficient requirements modeling at an early stage, and a tool has to support this. People like to use whiteboards and to sketch by hand at early stages, because freehand sketching gives them a great deal of freedom. (Creating a diagram with drag-and-drop can be easy and efficient, but this kind of interaction limits the drawing possibilities.) Thus a tool must provide a form of freehand sketching in order to be suitable for expressing requirements at an early stage. Furthermore it must be possible to import pictures.

The value of these drawings and pictures is increased when the tool recognizes them in a way such that they can be converted to model elements that have semantics. Sezgin et al. [23] discuss the technical details of online sketch recognition. Another problem is the analysis of imported images. Individual symbols can be recognized easier while a person draws them than when given as a static image. In the latter case, the recognition algorithm must be able to separate the elements of a drawing into individual symbols. Ouyang et al. investigate symbol recognition in

drawings [18]. Because a ULM uses few distinctive symbols, the recognition of hand-drawn diagrams should be easier for a ULM than for a heavyweight language such as UML. While fully automatic recognition of some parts of drawings should be feasible, other parts will definitively require human guidance (Fig. 5).



Automatic inference of the Product Manager actor and the Product and Market objects should be possible, while recognizing the Board as a group of actors and distinguishing influence from flow will require human guidance.

Figure 5. Converting a drawing into a ULM model

C. Navigation and Views

As requirements specifications can be large artifacts, a tool must provide help both for navigating and selectively viewing the specification. Navigation support eases orientation in large models. Selective views provide abstractions or present details about a part of the model in isolation from the rest. View generation must be versatile, because different user groups (e.g., novice vs. expert users) as well as different purposes require specifically adapted views. Below we summarize some proven techniques for navigation and view generation.

Overview map. Computer screens and projected displays are not able to show all parts of an advanced specification document at the same time. A simple way to give the users a better overview over the structure of a document is to display an overview map. Additionally, a rectangle on the map indicates the part of the document that is currently visible on the screen. The overview map can be extended with more features. For example, clicking an area in the map could automatically display that part of the specification.

Expand and collapse hierarchies. Hierarchies are a crucial structuring concept in a ULM. A tool should exploit these hierarchies for selectively displaying the parts of current interest in detail while abstracting or hiding others. Fisheye views [5] [2] are a proven technique for achieving this. However, as users can freely arrange the elements of the specification in the 2D space, a tool needs to preserve the user-created layout when parts of the specification are expanded or collapsed. This is a non-trivial problem that has impeded the practical use of fisheye views. In our previous work, we have contributed solutions to this problem [20]; so fisheye view based visualization of hierarchical models can now be used practically in a tool for a ULM.

Keyword search and filtering. As in text processing programs, a user can search for a certain term. All occurrences of the term will be highlighted in the normal view

as well as in the overview map (the text in the map is too small to be read, but the highlighting can give visual cues on where the term can be found). To further improve the keyword search, objects that contain the keyword can automatically be expanded, and other objects can be collapsed.

Filtering uses a slightly different concept. Filtering allows users to generate views where only those model elements are shown that match given filtering criteria. For example, in the model given in the appendix, a user might want to create a view that shows only the important stakeholders and their goals. He or she can achieve this by filtering the model with the criteria ‘objects contained in the Stakeholders object’ and ‘value of attribute Importance is Major or Critical’.

D. Analysis and Requirements Traceability

Apart from editing and navigation support, a tool can also help in requirements analysis and traceability tasks. The semantics of relations and user-defined attributes can be exploited for tool-based analyses. Traceability links created by referring to names (see ‘Naming’ in Sect. III) can also be exploited. Alternatively, a tool also can compute links semi-automatically, e.g. by analyzing the similarity of terms in different parts of the specification [4]. A related idea is a tool-assisted selection and verification of terms relevant to the project [15]. As natural language is ambiguous, a tool shall also support the identification of synonyms and words with the same stem. For relevant terms that are missing in the glossary, the tool can add entries automatically and notify the users about missing definitions.

When the software tool manages requirements traceability links, it is possible to immediately highlight the requirements and entities that get influenced by changes in the specification. This makes users aware of the effects caused by their changes. In addition, the tool can construct and present a special view of a traceability tree to the user.

The stronger the semantics of a modeling language, the more and stronger analyses are possible. However, the limited semantics of a ULM suffice to support the basic analysis and traceability tasks described above.

E. Simulation and Model Validation

Tool support enables the analysis of diagrams by simulation. Simulation can be used for analyzing dependencies between requirements and for the validation of the model – even when a model is not fully formalized. The missing formality is compensated by interaction with the modeler during a simulation run [22][9]. Even the weak semantics of a ULM allow some form of interactive simulation.

For example, a business process (such as the Emergency Call Process in the specification given in the appendix) can be simulated by stepping through the model along the flow paths. In each step of the simulation, the tool highlights active objects in the diagram. At the same time, all stakeholders and objects (requirements, goals, etc.) that are associated with the active step could also be visually emphasized. The next step is either determined by the model or the tool inquires it interactively from the person who runs and observes the simulation. The observer thus perceives relationships between the individual process steps and the involved requirements.

Such an animated simulation also helps validate requirements with stakeholders: stakeholders can be demonstrated the animated flow of information and control when a requested task is performed, which gives them a better understanding of the requirements than static reading of models would.

V. DISCUSSION

A. Related Work

We are not aware of any other work specifically directed at creating a ULM or using ultralightweight modeling in requirements engineering. Jackson’s work on Alloy [14] is called ‘lightweight’, but – in contrast to our approach – refers to lightweight *formal* modeling. Moody’s work is on general design criteria for visual notations in modeling [16].

In our own previous work, we have investigated tool support for modeling languages with a structure of nested hierarchical objects (see [20] and work cited there). We also have investigated the analyzability of semi-formal models by simulation [22] [9].

[3] and [23] investigate online recognition of model elements when drawing, while [18] investigates the extraction of model elements from pictures. Work on analyzing natural language requirements [1] and on identifying names in natural language text [15] also becomes relevant in the context of creating analysis tools for a ULM.

B. Where We Are: Achievements and Limitations

In this paper, we have developed a vision for an ultralightweight modeling language and presented its design rationale. Further we have presented a draft design for a ULM and shown that a typical problem can be adequately modeled with this language. We have also investigated the problem of supporting a ULM with an adequate tool.

With respect to our initial seven core requirements, we claim that a ULM carefully crafted along the lines given by our draft language will easily meet R1-R5. Checking for R6 requires empirical validation. Except our experience when writing the model given in the appendix, we have no empirical evidence yet concerning ease of writing, reading and learning. So this is subject to future work. With respect to R7 (that the language meets the design principles for visual notations [16]), we claim that our draft language already scores at least better than UML. Again, an in-depth analysis is subject to future work.

We have not yet investigated some important issues: for example, the role of color, the inclusion of rich media such as video and the question whether we need user/domain specific languages (or at least language dialects).

C. Where to Go: A Research Roadmap

Having made a case for an ultralightweight requirements modeling language, we subsequently present a research roadmap towards creating a fully developed, industrial-strength ULM.

Language design. The next step towards a real ULM will be designing a carefully crafted language. The language draft presented in this paper may serve as an inspiration. The design should be guided by (i) analytical considerations (for example, semiotic clarity [16] or orthogonality and minimality of language constructs), (ii) experi-

ments (for example, about intuitive understanding and ease of learning), (iii) benchmarking (specify a problem both in a conventional language and in the new language and compare the results), and (iv) empirical work (try the language on real-world problems). Design and preliminary validation should be closely intertwined. Design trade-offs will have to be made between simplicity and (formal) semantic expressivity. (Informal expressivity is secured by making unrestricted natural language text a part of the language.)

Another criterion that should be assessed is the effort required to transform a model written in the new language into a conventional model (class models, activity models, state machine models, etc.).

In a first step, the design should focus on simplicity and domain independence, but also on extensibility. As soon as a stable, high-quality language core is achieved, extensions can be considered, for example the inclusion of media beyond text and pictures, or domain-specific language dialects. Support for specific problem areas such as software product line requirements modeling or software product management modeling could also be considered in further steps.

Tool design. The challenges in tool design are primarily of technical nature. Existing algorithms for smart editing and navigation in hierarchical structures need to be drawn together and adapted. Mechanisms for selective visualization and smart report generation need to be adapted to the needs of a ULM. New concepts and algorithms for handling multi-hierarchies and for semi-formal analyses need to be developed. On the basis of existing work on understanding sketches, mechanisms for evolving pictures into models have to be developed.

Some basic tool support is required already in the early stages of language design: creating specifications with general-purpose drawing tools impedes the empirical tasks in language design.

Method/process considerations. Using a ULM doesn’t a priori require methods or processes different from those we already have. Nevertheless, it would be worthwhile to investigate whether using a ULM enables methods and/or document structures different from what we have today.

Validation. Ultimately, the ‘grand’ research challenge in validating a ULM is to test the validity of the two basic hypotheses that form the basis and motivation for developing a ULM:

(H1) A well-designed ULM outperforms both plain natural language with pictures and classic modeling languages when specifying requirements at an early stage.

(H2) Average requirements engineers will prefer using a ULM over plain natural language with pictures when given a choice.

However, testing these hypotheses empirically can only be the last step, when a fully developed ULM, well-supported by a tool, is available. On the way to that goal, intermediate validation work will be needed that addresses more specific topics, such as language design quality, language expressivity, suitability and ease of use. Suitability can be subdivided into suitability for (i) creating and communicating requirements, (ii) analyzing and verifying requirements, (iii) transforming requirements written in a ULM into more formal requirements models, and (iv)

using a specification written in a ULM as a basis for architecture and implementation.

VI. CONCLUSIONS

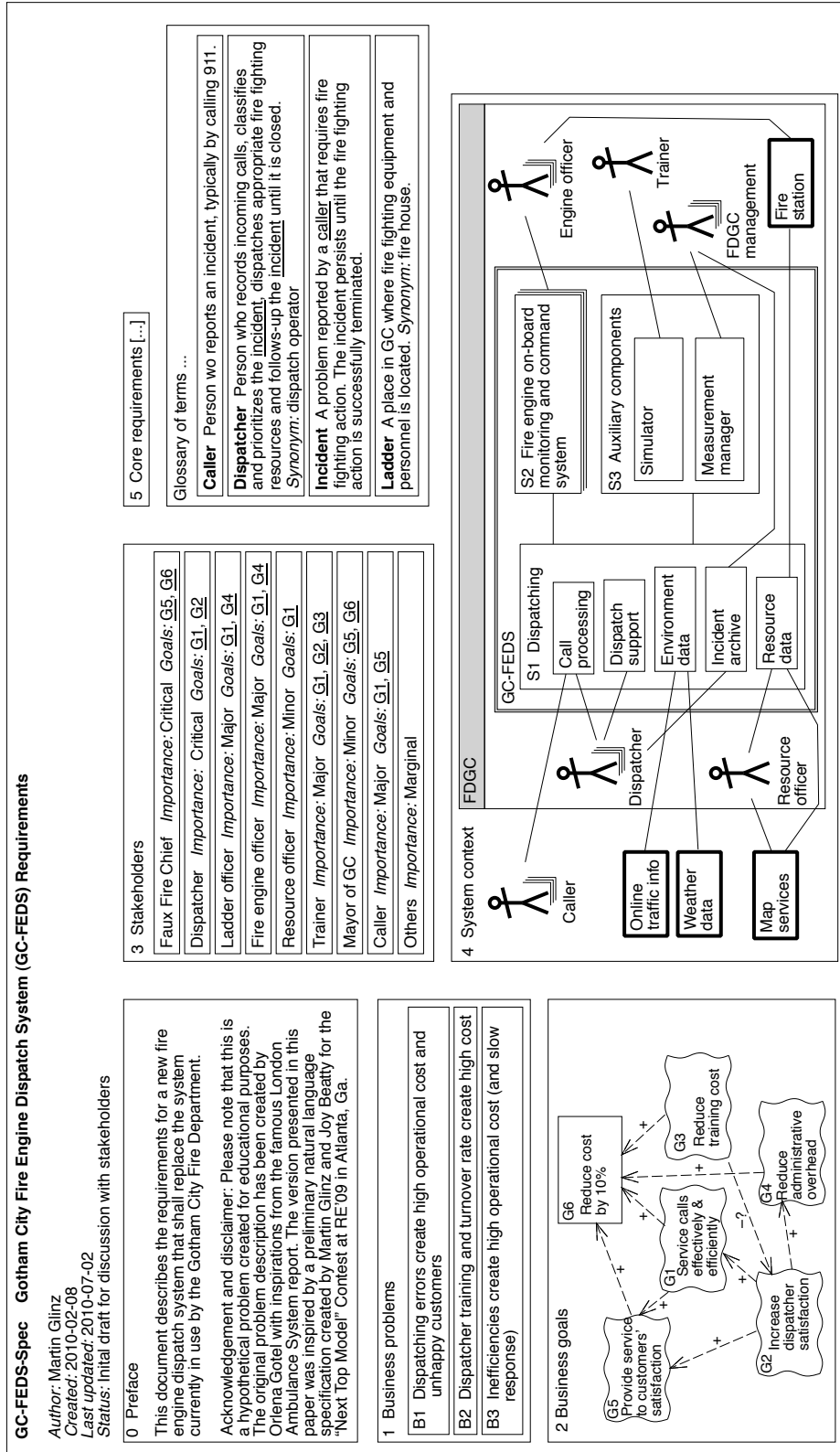
We hope that this paper will stir the discussion on ultralightweight requirements modeling languages for early-stage requirements specification and motivate other researchers to contribute critique and ideas. In our own research we plan to further investigate both the language and tool design issues. We are convinced that the lightweight model structure will make ULM specifications significantly better than plain natural language ones while retaining the flavor of naturalness and ease of use.

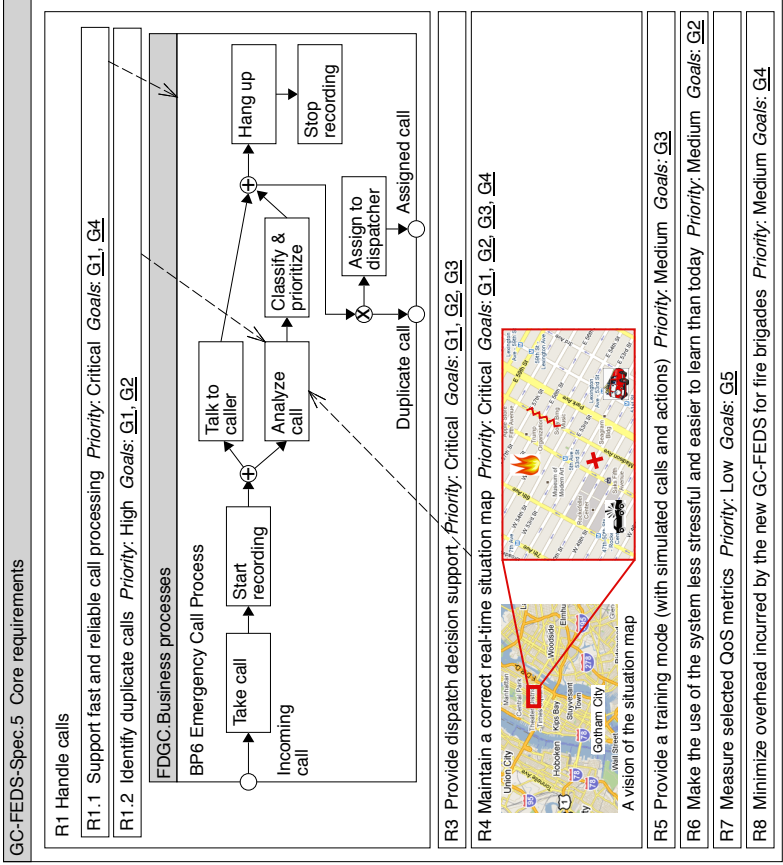
REFERENCES

- [1] Ambriola, V., V. Gervasi (2006). On the Systematic Analysis of Natural Language Requirements with CIRCE. *Automated Software Engineering* **13**, 1 (Jan 2006). 107-167.
- [2] Berner, S., S. Joos, M. Glinz, M. Arnold (1998). A Visualization Concept for Hierarchical Object Models. *Proc. 13th IEEE International Conference on Automated Software Engineering (ASE'98)*. 225-228.
- [3] Chen, Q., J. Grundy, J. Hosking (2008). SUMLOW: Early Design-Stage Sketching of UML Diagrams on an E-Whiteboard. *Software Practice and Experience* **38**, 9 (Jul. 2008). 961-994.
- [4] Cleland-Huang, J., B. Berenbach, S. Clark, R. Settimi, E. Romanova (2007). Best Practices for Automated Traceability. *IEEE Computer* **40**, 6 (Jun. 2007). 27-35.
- [5] Furnas, G. W. (1986). Generalized fisheye views. *Proc. ACM CHI86 Conference on Human Factors in Computing Systems*, Boston, Mass. 16-23.
- [6] Glinz, M. (2000). Problems and Deficiencies of UML as a Requirements Specification Language. *Proc. Tenth International Workshop on Software Specification and Design*. San Diego. 11-22.
- [7] Glinz, M., S. Berner, S. Joos (2002). Object-Oriented Modeling with ADORA. *Information Systems* **27**, 6. 425-444.
- [8] Glinz, M. (2007). On Non-Functional Requirements. *Proc. 15th IEEE International Requirements Engineering Conference (RE'07)*, Delhi, India. 21-26.
- [9] Glinz, M., C. Seybold, S. Meier (2007). Simulation-Driven Creation, Validation and Evolution of Behavioral Requirements Models. Dagstuhl-Workshop Modellbasierte Entwicklung eingebetteter Systeme (MBEES 2007). Informatik-Bericht 2007-01, TU Braunschweig, Germany. 103-112.
- [10] Glinz, M. (2010). Very Lightweight Requirements Modeling. To appear in: *Proc. 18th IEEE International Requirements Engineering Conference (RE'10)*, Sydney, Australia.
- [11] Gotel, O. (2009). *GC-FEDS Replacement System: Request for Requirements Modelers*. Case description distributed at the Next Top Model Contest at RE'09 [12]. <http://www.gotel.net/ntm/brief.htm>.
- [12] Gotel, O., J. Cleland-Huang (2009). Next Top Model: A Requirements Engineering Reality Panel. *Proc. 17th IEEE International Requirements Engineering Conference (RE'09)*, Atlanta, GA. 357-357.
- [13] IEEE (1998). *IEEE Recommended Practice for Software Requirements Specifications*. IEEE Std. 830-1998.
- [14] Jackson, D. (2002). Alloy: A Lightweight Object Modelling Notation. *ACM Transactions on Software Engineering and Methodology* **11**, 2 (April 2002). 256 - 290.
- [15] MacDonell, S.G., K. Min, A.M. Connor (2005). Autonomous Requirements Specification Processing using Natural Language Processing. *Proc. ISCA 14th International Conference on Intelligent and Adaptive Systems and Software Engineering (IASSE 2005)*, Toronto. 266-270.
- [16] Moody, D.L. (2009). The “Physics” of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Transactions on Software Engineering* **35**, 6. 756-779.
- [17] Object Management Group (2009). *Unified Modeling Language: Superstructure*, version 2.2. OMG document formal/2009-02-02. <http://www.omg.org/spec/UML/2.2/Superstructure/PDF/>
- [18] Ouyang, T. Y. and Davis, R. (2009). A Visual Approach to Sketched Symbol Recognition. *Proc. 21st International Joint Conference on Artificial Intelligence*, Pasadena, Ca. 1463-1468.
- [19] Plimmer, B., M. Apperley (2002). Computer-Aided Sketching to Capture Preliminary Design. *Proc. 3rd Australasian Conference on User interfaces*, Melbourne. 9-12.
- [20] Reinhard, T., S. Meier, R. Stoiber, C. Cramer, M. Glinz (2008) Tool Support for the Navigation in Graphical Models. *Proc. 30th International Conference on Software Engineering (ICSE'08)*, Leipzig, Germany. 823-826.
- [21] Robertson, S., Robertson, J. (2006). *Mastering the Requirements Process*. 2nd edition, Addison-Wesley.
- [22] Seybold, C. (2006). Simulation teilformaler Anforderungsmodelle [Simulation of semi-formal requirements models (in German)]. Doctoral Thesis, University of Zurich. Aachen: Shaker Verlag.
- [23] Sezgin, T.M., T. Stahovich, R. Davis (2001). Sketch Based Interfaces: Early Processing for Sketch Understanding. *Proc. 2001 Workshop on Perceptive User Interfaces (PUI'01)*, Orlando, Florida.
- [24] Teichroew, D., Hershey III, E.A. (1977). PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Information Processing Systems. *IEEE Transactions on Software Engineering*, **SE-3**, 1 (Jan. 1977). 41-48.

APPENDIX: THE GC-FEDS PROBLEM MODELED IN AN ULTRALIGHTWEIGHT MODELING LANGUAGE

This specification is based on a hypothetical letter in which the Fire Chief of the Gotham City Fire Department asks a requirements engineering consultancy company for help. He wants a replacement for his current fire engine dispatch system and informally describes his problems and needs. The specification below is intended to capture this initial information for a meeting with the stakeholders. The original problem description has been created by Gotel [10] with inspirations from the famous London Ambulance System report.





EXPLANATIONS

GC FEDS-Spec is an object representing a document. Its top-level nested objects serve as an organizational structure.

In the Business goals object, nested objects are arranged in a diagram, in this case a goal graph. The 'fuzzy' modifier is used to denote soft goals. Relation annotations are used to indicate positive and negative influence.

The details of the Core requirements object are hidden from the overview diagram. The '...' marker indicates that more details are available.

The Glossary of terms object is incomplete (name followed by '...'). The currently available information is displayed in full (no '...' marker).

Italics indicate *attributes* that are followed by values. Underlining a word (e.g., G2) indicates a reference to an item with that name.

In the System context object, modifiers mark the context boundary of the GC-FEDS system and indicate which elements are external to it.

The object BP6 belongs to two hierarchies: (i) to Core requirements by embedding, (ii) to FDGC.Business processes by an explicit context path.

The Call processing object is shown in isolation. It is contextualized by the context paths of the two hierarchies that this object is embedded in.