A Redesign of a Visual Programming Language Inspected with Cognitive Dimensions of Notations

URBAN LILJEDAHL

MASTER OF SCIENCE IN TECHNOLOGY
Computer Science

Department of Computer Science and Electrical Engineering
Division of Computer Science
Supervisor: David A. Carr

2003:49 • ISSN: 1402 – 1552 • ISRN: LTU – DUPP – – 03/49 – – SE
A redesign of a Visual Programming Language inspected with Cognitive Dimensions of Notations

Urban Liljedahl

November 19, 2003
Abstract

This report describes a redesign of a process modeling program, reMind. The new design, called reMind+, is presented and evaluated from a usability perspective. reMind lets the user model an industrial process by using a graphical user interface. The model is defined by a node-link diagram, and it is possible to add functions to the nodes. This model is then transferred to a tool that optimizes the process.

Remind uses a graphical editor to construct a node-link based model of a process. reMind has however usability problems. For example, only a limited set of functions are available for process components. The design of reMind+ blends user requirements together with research results in the area of Human-Computer Interaction to overcome reMind’s limitations.

The most important user requirements are possibility to add user defined functions, support for reuse of models that been built, and an improved model editor. An evaluation of reMind+ is performed with a system called Cognitive Dimensions of Notations (CD). CD focuses on usability and the design trade-off. This report concludes that the cost for the added functionality is an increased learning threshold for the application.
1 Introduction

reMind is a software tool used by engineers to optimize industrial processes. A user requirement for a less constraining tool implied a redesign of reMind named reMind+. This work describes the design of the new system and an evaluation of its usability. The evaluation showed that the design changes affected the usability in many different perspectives. Many of these perspectives are in trade-off relation with each other.

IKP (The Department of Mechanical Engineering at Linköping Institute of Technology) has a research area called Energy Systems. They invented a method called MIND (Method for analysis of INDustrial energy systems). It was developed as a part of Ph.D. thesis (Nilsson, 1993). It has been used by pulp, paper and steel industries and by graduate students at IKP. The method aims to find a combination of investments and production, which gives minimal costs. The basic principle is to describe the system by a set of equations, inequalities and a goal function and transform it to a format that optimizing software can use. Seven students rewrote the software during the years 2000-2001. The result, called reMind, used a graphical user interface that lets the user build a model of the industrial process by building a graph with the graph editor instead of defining an equation system in text. The model is defined as a directed graph with nodes and links. The incoming and outgoing material and energy are named as resources and the resources flow through the model. However, the users of reMind needed improvements and this has led to a redesign called reMind+. Four main problems are:

- a constrained graph editor
- weak support for reuse of a part of a model and including a model into another model
- monolithic model structure
- a constrained set of predefined functions

The last problem was the most severe for users at the department of Energiteknik at Luleå Tekniska Universität and was the main cause of the redesign.

The redesign is mainly supported by research in the area of Visual programming languages, end-user programming and existing end-user programming systems. Visual programming languages (VPL) are a group of programming languages that let the user define the program in a two-dimensional graphical environment instead of a textual language. The approach benefits from the fact that humans have used images for communication since the cradle of human culture and the relaxation from strict sequential solutions. Many VPLs are designed with human cognition and problem solving in focus instead of a more traditional technical focus.

End-user programming systems are often designed with human cognition in focus in order to avoid or minimize complex constructions. It can be control structures or complex boolean expressions that must be transformed into easier constructions simple enough to learn, yet still powerful enough to be effective.

A design must be examined to see if it meets the user requirement and the design goals. This design was evaluated with Cognitive Dimensions of Notations...
(CD), which is a discussion tool used to evaluate information systems, both interactive and non-interactive. The inventors of CD claim that it is broad-brush approach and focus on the whole rather on details, but still specific enough to capture different usability problems in a system. The evaluation of reMind+ stated a set of trade-off relations and effects of the redesign e.g. that the increased degree of freedom has a cost, in this case in increased abstraction, which affect the learning threshold for beginners and the cognitive burden on the skilled user.

The following section relates this project with findings in the area of end-user programming, Visual Programming Languages and visualization of structures. Then follows the sections with introduction to cognitive dimensions, description of reMind, problems and requirements, the design of reMind+, how it solves the problems, the evaluation of reMind+ with CD followed by future work and conclusions.
2 Related Work

The redesign of reMind is based on the requirements from the users, study of other systems and applications, and by applying findings from the area of end-user programming and Visual Programming Languages. This section introduces concepts such as end-user programming, systems and applications, psychology of programming, visual programming languages, representation and presentation and a brief introduction to graph layout and graph aesthetics.

2.1 End-User Programming

End-user programming can be described as programming performed by non-expert programmers, for example an economics expert performing an analysis with spreadsheet software as a tool. An end-user programming system must be easy enough to learn and powerful enough to be effective and efficient. A professional programming system has the same requirement but usually tolerates a higher learning threshold traded off against increased effectiveness and efficiency.

Pane et al. [15] studied non-programmers’ solutions to programming problems. A programmer must transform the solution of a problem into the language specific domain, and this is often a large part of the programming task. They suggest that future systems should minimize the gap between the mental plan and the way it must be carried out in the system. The gap is also known as the “gulf” between goals and actions [6]. The predecessor of reMind known as Mind was created as an optimization tool [16] with a user interface based on text. The successor reMind fulfilled the same purpose but took advantage of a graphical user interface in order to reduce the gap between the goals and the actions. The goal of reMind+ is to improve usability further among other things by reducing the ”gulf” between goals and actions one more step. Pane et al. conclude that ”the distance between the plans and the programming language” should be minimal, the language system should provide help for beginners to make a user transform a solution with imprecision into a precise solution because programming is often a task of precision. Important design suggestions derived from their study [15] are:

- Aggregate data access through set creation and manipulation.
- Use event-based style of programming.
- Try to substitute complex boolean expressions with other constructions that are easier to use.
- Use a different programming paradigm than imperative programming. This programming style is not the most natural choice.
- Exploit the fact that many novice programmers used pictures and diagrams in their solutions.

A subsequent study by Pane and Myers tested alternatives to textual boolean expressions [14] and explored a graphical approach instead of a boolean expression to select an entity within a set. Their evaluation indicated that the graphical approach performed significantly better than boolean expressions for selection tasks.
2.1.1 Systems and Applications

There are many systems today that are designed to offer programming facilities
to non-expert programmers. This section will describe LabView\(^1\), ProGraph\(^2\)
and the spreadsheet. The first two systems have similarities with the reMind and
the third is one of the most common end-user programming systems. LabView
and ProGraph use the data-flow approach, and the user defines the program
with a directed graph. Each node acts on the flow and the graph does not de-
fine the order of control as a state-machine. Pane et al. [15] claim that flow of
control, or an imperative approach, is unnatural for many non-expert program-
mers despite the fact that natural words like if, else and then are used in their
program descriptions. LabView let the user build a scenario with components
and connect them with wires, building on a metaphor of the prototyping board
for electronic experiments. The system lets the user build a model of a system,
for example a circuit with a battery, a switch and lamp. LabView has an ad-
ditional layer that hides the components except for interactive components like
lamps, switches and displays. The model is built in the first layer at one level.
LabView accepts user-defined components but the model remains flat. On the
other hand ProGraph lets user define new nodes in a sub window with a graph.
Users of ProGraph have described the approach as a box inside a box. It is like
a Russian doll that lets one graph hide inside a node, deeper and deeper. Green
et al. [7] use the term *programming games* for plain goals that demand complex
actions to reach, e.g. finding the smallest number in a vector of integers by a
loop combined with variables for storage and comparison performed in an im-
perative programming language C. Both LabView and ProGraph avoid many
of the programming games that are involved in looping and set operations in
common imperative programming languages.

Many different programs use the spreadsheet paradigm. The general idea
is an electronic spreadsheet that lets the user type data into a matrix with
addressable cells, automated recalculation of all cells and type-checking of input.
The key to success was the possibility to include a formula that related to other
cells and recalculates when the sheet is changed. The concept was easy enough
to learn and powerful enough to be both effective and efficient.

The spreadsheet paradigm has supported end-user programming since the
introduction of VisiCalc in late 1970s. The program lets non-programming
experts use the computer like a programmer and the success of the paradigm is
well known.

Lewis [13] uses the spreadsheet solution to build interactive software with
additional features like vector graphics connected to spreadsheet cells. He adds
as little as possible to the paradigm so the benefits with the original system
will persist in the new system. His system is called NoPumpG referring to the
fact that the user should not think about reading input, process it and send it
to the display, called pumping in his paper. The user can instead work with
input, data and displaying graphics in the same area, a sort of spreadsheet.
The design of reMind+ uses the same approach as NoPumpG. The spreadsheet

\(^1\)LabView (Laboratory Virtual Instrument Engineering Workbench) is a programming en-
vironment that builds on the data-flow paradigm. It is used for data acquisition, analysis,
display and control applications.

\(^2\)ProGraph is a visual programming language following the data-flow paradigm with object-
oriented facilities.
idea is reused, and the addition of new functionality is done with the paradigm in mind in order to restrain the foundation as much as possible. The model is structured as a directed graph and the nodes control the flow of resources. The node is defined by a set of functions and the design of reMind+ lets the user define the node in a spreadsheet-like environment that gives the user much higher level of freedom than the predecessor system, what was limited to a static set of predefined functions. In addition, reMind+ lets the user define new functions in a spreadsheet-like environment.

2.1.2 Psychology of Programming

The research in the area of Psychology of Programming has evolved during the last three decades, but language design seems partly to ignore these findings. The design of many modern programming languages seems to focus on scalability, efficiency, provable correctness and reusable constructions rather than HCI-related issues when programming languages are designed. Pane et al. [14] recommend that a part of the language design criteria should regard the perspective of HCI and psychology of programming.

van der Veer et al. [17] describe an overview in the area of mental models and usability of systems. The mismatch between the mental model of the system and the actual system is a source of errors and dissatisfaction. These can occur at different levels, for example on task-level when the user carries out a task in the wrong way because of an inadequate idea of the system or on keystroke-level where the key-binding is other than the user expects. The redesign of reMind+ tries to support the mental model of the user by adding new features for the modeling. The intention is to support a modeling process that decreases the gap between the real system and the model built by the user. This is accomplished by the recursive model definition that lets the user organize the model as a number of sub models. The main blocks of the real system can match sub models in the model.

2.2 Visual Programming Languages

Visual Programming Languages (VPLs) transform the task of programming from a textual environment to a graphical environment. Whitley et al. [5] mention that the idea of visual programming is as old as the computer, but communication with images is a part of the human nature and as old as human culture. VPLs try to benefit from the fact that image communication is a natural behavior of human beings. They claim that textual programming has a strong legacy from Indo-European languages with symbols that become meaningful in connection to others, like a variable that is a meaningless symbol that represents an unspecified object outside its context. They study the ProGraph system as a programming language and conclude that ProGraph is relaxed from the Indo-European rooted paradigm.

ProGraph lets the user build the program without the constraints of a flow-of-control. This is achieved by a graphical representation of a data flow and class hierarchy. The two-dimensional display offers a pseudo-3D solution with windows that can be opened to view details about a pictorial element as the third dimension. reMind lacks this possibility, and the resulting program is built as one monolithic structure. reMind+ has adopted the solution of ProGraph.
in order to let the user build a top-level model that can be viewed and defined element by element, each in new windows.

A survey by Alan F. Blackwell et al. [18] focuses on how academia and industry think about VPLs and concludes that researchers expected improved understanding through visualization of the program while programmers are more concerned about improved productivity.

Boshernitsan et al. [2] state a formal definition of a VPL: "A spatial arrangement of icons that constitutes a visual sentence that is a two-dimensional counterpart of a one-dimensional arrangement of tokens in conventional (textual) programming languages". They present a survey of some VPLs and continue with a classification of VPLs. The classes may overlap and a language can be a mix of the following classes:

- Purely visual languages
- Hybrid text and visual systems
- Programming-by-example systems
- Constraint-oriented systems
- Form-based systems

They mention that the hybrid class has two approaches where one use graphical elements to build a program translated into textual language and the other combines text and visual programming. The redesign of reMind changes the classification from a combination of purely visual language and form-based system to a hybrid text and visual system.

2.3 Visualization of a Structure

Each programming system must visualize the structure that the user builds. Even a textual system can contain more than just the text. Additions include colour cues that highlight keywords or different syntactic constructions to semantic constructions that hide details and shows available functions. A visual programming system relies even more on the presentation. The semantics can be described as an image, for example, a graph. Graph structure includes two elements, representation and presentation together with layout and aesthetics.

The following section is Representation and Presentation. It deals with the connections between an underlying structure and the presentation. Then follows a section on graph layout and graph aesthetics that covers some specific issues regarding presentation of graph structures.

2.3.1 Representation and Presentation

Representation and presentation can be thought of as a pair. Representation is the underlying structure and presentation is the view of that structure. Carpendale et al. [3] use this approach when they present a framework for presentation of a substantial amount of information on a limited display. The display is a limiting factor, e.g. a map must be viewed at low magnification, only a part of it or distorted in some way. Presentation deals with the problem of showing details and still letting the user keep track of the whole. Carpendale presents
a framework called Elastic Presentation Framework, e.g. a map viewed by a
virtual fisheye lens. Many different strategies have been explored. Hornbaek
et al. [12] present a comparison between two approaches when searching and
navigate geographical maps. The first is called Overview+Detail User Interface
and uses a small frame for an overview of the map and a bigger frame for the
details of interest. The second is called Zoomable User Interface (ZUI). The
ZUI approach lets the user navigate in a map and zooming in to explore details
in a selected area. They conclude that subjects are faster with the zoomable
user interface when it is combined with a multilevel map. A multilevel map is a
map were details are viewed as the user zooms in and details are hidden as the
user zooms out. Subjects preferred the Overview-Detail interface in their study,
but the authors found certain benefits with the interface without the overview.
The overview must be displayed on the screen so less screen area is available for
the details.

Pad++ was an early framework [1] that supported the Zoomable User Inter-
facing paradigm and has been used in several projects, for example zooming web
browsers and document visualization. The Jazz framework is a similar frame-
work but based on Java instead of more platform specific techniques. It has
been used in different applications such as PhotoMesa and TimeSearcher. This
framework is now replaced with Piccolo framework [9], a less complex structure
with almost the same functionality.

The design of reMind+ uses two different approaches for presentation. One
of these is a Zoomable User Interface, where the user uses low magnification
for navigation and overview together with high magnification to view details
in the model. The idea of a multilevel map implementing a semantic zooming
interface is used, and details are viewed depending on the current scale when
the user zooms in and out the model. This concept is also referred as Constant
density zooming. Representation is the underlying model and organizes the
information in a way that supports the presentation. The representation model
in reMind+ is redesigned depending on two requirements. First, it must support
the hierarchical module-based presentation where the graph can be grouped
into modules, and secondly, the representation must support the possibility to
reuse a subset of a model in order to both extract and include these modules
or sub models into another model. The concept of object orientation includes
the possibility of recursive class definitions. For example a model class can be
defined by a set of attributes, e.g. a node class, a link class and model class.
This concept is used in the design of reMind+ to meet the two requirements
above.

2.3.2 Graph Layout and Graph Aesthetics

The graph layout and the graph aesthetic perspective affect the understanding
and the construction of the mental model in a system based on graphs. Both
flat graphs such as the graphs in LabView and hierarchical constructions as in
ProGraph rely on the visualization of a problem. But a bad graph layout can
lower the effectiveness [8].

A certain structure can be presented in many different ways in a graph. Some
graphs are easier to read and understand than other graphs even when
the underlying structure is the same. Some recent findings from the area of
graph aesthetics is presented by Purchase et al. [10]. They carried out experi-
ments to reveal potential usability of automated graph layout algorithms based on aesthetics. They conclude that minimizing the number of crossings is the most significant factor of graph aesthetics and usability. Orthogonality in terms of only using horizontal or vertical lines is important in some contexts, for example together with Unified Modeling Language (UML) class diagrams [11]. The conclusion is that graph layout algorithms that don’t bother about the context of the graph will not necessary produce useful layouts.

They studied the aesthetics from two different perspectives: syntactic performance and semantic preference. The participants answered questions about the graph, for example what is the shortest path between node A and node B, during the syntactic phase. The semantic preference used graphs with different design in a semantic context. The result relied on what graphs the participants preferred.

Their experiment with syntactic approach on abstract graph structures indicated that reduced number of crossings and bends and increased symmetry increased performance. Orthogonality did not influence the result on these abstract graphs. The semantic approach explored diagram that modeled a system. In this case, UML class diagrams and how the semantic context relates to graph layout. They presented a priority order of important graph aesthetics as a result of the experiment. The first four factors were minimize edge crossings, orthogonality, information flow and minimize edge bends. The redesign of reMind led to a suggestion where the number of links in the same view is visible and an environment that encourages modular design. Both factors may reduce the number of crossed links, which [10] indicated as the most important factor of syntactic and semantic understanding of a graph. reMind+ renders the link as the shortest line between two nodes without any bends, which may lead to more link crossings compared to a system that allows links to take a different path to avoid crossings. According to [10] straight links leads to fewer errors, but increased crossing correlates with increased number of errors.
3 Introduction to Cognitive Dimensions of Notation

Cognitive Dimensions of Notations (CD) is a usability inspection method defined by T.R.G. Green et al. [8]. The method provides a set of “dimensions” used for the analysis of informational artifacts including both interactive and non-interactive artefacts. They are called dimensions because they are conceptual idealizations and a change of one dimension affects other dimensions, for example three “dimensions” temperature, pressure and volume in physics. T.R.G. Green claims that the tool is broad brush, useful for analysis and serves as a discussion tool.

3.1 Cognitive Dimensions Related to reMind and reMind+

T.R.G. Green et al. [7] perform an analysis of the two VPLs ProGraph and LabView using CD. They claim that VPLs are easier to use than many textual languages because of easier syntactic planning, operators at a higher level that reduces the amount of programming with primitives and finally a less restricted order of the programming activity. They discovered on the other hand that the visual languages needed more entities compared with its textual counterpart for a computation of speed depending on acceleration and gravitation. One of the studied VPLs was highly inefficient when the user needed to reorganize the program structure. They continue by mention some important design issues for future VPLs:

- Stronger support for comments or informal notation.
- Avoid complex constructions for flow-of-control. They claim that the representation of control flow remains a problem in the studied VPLs.
- Strive for a system with low resistance to change. The studied systems were surprisingly time-consuming to maintain; e.g., adding a new element could cause serious re-organization, which takes time and may introduce errors.

The redesign of reMind was influenced by this advice. By allowing textual notation inside a node definition, the possibility to add notes inside the graph as a metaphor for Post-it notes on a desktop, possibility to select background colors for a sub model. reMind+ follows the data flow paradigm and the node definition is done in a spreadsheet environment that is more of a declarative approach than a control-flow approach. The design of reMind+ graph editor must include features that help users to organize and reorganize the graph using direct manipulation with drag and drop on groups of selected objects in order to lower the resistance of changes.

3.2 The Full Set of the Dimensions

Here follows the full set of the dimensions combined with a short explanation. The dimensions are presented briefly and a sub set of them is described more carefully with an example 3.5.
Abstraction

*Types and availability of abstraction mechanisms.* A system, for example a programming language can belong to one of the classes abstraction-hating, abstraction-tolerant and abstraction-hungry where:

- Abstraction-hating systems do not allow the user to introduce new abstractions.
- Abstraction-tolerant systems allow new abstractions but do not demand it.
- Abstraction-hungry systems force the user to define at least one abstraction before the artefact can be used.

Hidden Dependencies

*Important links between entities are not visible.* A hidden dependency is a link between two objects but the relation is only visible in one object. The dependency can be one-way and/or local. One-way dependency is a pointer from one object to the other and local dependency means a dependency chain where only the first link is visible.

Premature Commitment

*Constraints on the order of doing things.* When a system has constraints on the order of actions, and the user does not have enough information for a decision when the action must be committed.

Secondary Notation

*Extra information in addition to formal syntax.* Let the user make informal notations like comments and spatial layout that don’t affect the functionality of the system.

Viscosity

*Resistance to change.* The viscosity can be of repetition or knock-on nature. The first appears when a change of a high level abstraction implies changes in the underlying structure. The latter appears when a small change induces many other changes.

Visibility

*Ability to view components easily.* The visibility is a measure of how much of the whole system is visible in one view.

Closeness of Mapping

*Closeness of representation to domain.* The closeness of mapping measures the closeness of a representation of a system is to the system. High closeness lowers the cognitive burden of translation between a system and its representation.
Consistency

*Similar semantics are expressed in similar syntactic forms.* A consistent system lowers the learning threshold and lowers risk for syntactic errors.

Diffuseness

*Verbosity of language.* The diffuseness describes how the entities are expressed. Long labels give more information and are less diffuse than short. An expression with many labels is on the other hand easier to parse if the labels are short.

Error-Proneness

*Notation invites mistakes.* The artefact can be more or less directed and increased degree of freedom usually increase the error-proneness.

Hard Mental Operations

*High demand on cognitive resources.* Typical signs of a hard mental operation are when the user starts counting on fingers, supporting memory with small notes or pointing at the screen to remember a track.

Progressive Evaluation

*Work-to-date can be checked at any time.* For example, a programming task involves many steps, and a system that lets the user validate parts before the whole system is ready supports progressive evaluation.

Provisionality

*Degree of commitment to actions or marks.* Provisionality is the degree of how persistent the results of actions are. High level of provisionality makes the actions less persistent and is usually preferred when a user performs exploratory design.

Role-Expressiveness

*The purpose of a component is readily inferred.* The role-expressiveness of a component is strong if the role or purpose of that component is easily perceived. Components with a strong role-expressiveness supports the comprehension of the system and the role of that component in the system.

The Trade-Off Between Different Dimensions

The overview approach helps users of CD to remain focused of the whole rather than the details. The analysis results in a profile rather than a list of bugs or an "overall difficulty measure". It is useful in finding factors in a trade-off relation. The dimensions are connected, more or less and a change of one dimension often affects other dimensions. T.R.G. Green [8] mentions that little research is reported in the area of trade-off relations between dimensions.
3.3 Match an Activity with a Profile

A CD analysis produces a profile. There is no absolute wrong or right profile. Different activities best supported by different profiles and the four main activities are [8]:

- *incrementation* adding a formula to a spreadsheet
- *transcription* converting a formula into an index card
- *modification* modify the spreadsheet to compute a different problem
- *exploratory* design programming on the fly (hacking)

These activities are performed in the context of an informational artefact, e.g., the spreadsheet program or a time-table. An artefact has a notation, environment and a media and these may change during the interaction process described as different layers.

3.4 The Informational Artefacts

The informational artefacts are described with models that are constructed in some way and can be manipulated. The model uses three concepts notation, environment and medium to capture an artefact. The following example uses the point of view of a programming language. Notation is the programming language with rules, syntax and conventions. The environment is the editor, compiling and executing tools. The screen is the media of the edited text. These three concepts are used to classify an interactive language as a notation, split an artefact in different layers and identify sub-devices of an informational artefact.

3.4.1 Interaction as a Notation

The interaction of an informational artefact is regarded as a notation. It must follow some implicit or explicit protocol. An interactive artefact like the telephone needs a set of actions or commands to be used and these commands are interactive language or the notation. A non-interactive informational artefact like a timetable is directly expressed in the notation.

3.4.2 Different Layers

The notation, environment and media may change over time or work in parallel within the same artefact. This is known as the concept of different layers. A programming system has one layer with the programming language as notation, the editor as the environment and the memory as the media. It has a second layer with the keystrokes and mouse actions as the notation, the keypad is the environment and the medium is of short duration when the notation is interpreted to editor actions. It is important to be aware of the layers during the analysis and discussion to avoid a discussion on the wrong level of interaction.

3.4.3 Sub-Devices

A sub-device is a part of an informational artefact but has its own notation, environment and media. A paper and pencil on the desk can be a sub-device of a programming system, used for sketching and notes during the process.
An important sub-device is the abstraction manager, e.g. a macro scripting language within a word processor or a system with a sub-device that lets user extend the functionality by adding new functions to the informational artefact.

3.5 CD Described by an Example

Here follows an example that includes the three dimensions Closeness of Mapping, Viscosity and Abstraction Gradient. These are described by a brief analysis of a generic word processor. Our word processor uses the What You See Is What You Get (WYSIWYG) paradigm.

3.6 Closeness of Mapping

The first dimension is Closeness of Mapping, which can be described as the gap between the model and the real world, in our example the word processor display and the result when the document is printed out. Green [7] claims that Closeness of Mapping is the answer to the question: what programming games need to be learned? Closeness of Mapping of the visible part of the document is tight if the display picture of the document looks like the document printed on paper. The Mapping is Closer to the metaphor of writing with a pencil if the word processor lets the user select font, font size and color by direct manipulation rather than typesetting the document by codes in the text. A system that maps to a real world metaphor reuses some of the users previous knowledge. It can be a drawback when the metaphor prevents a more efficient solution. For example, most users would prefer a word processor with dynamic word wrap instead of a new-line character at the end of each line even if the new-line is closer to a traditional typewriter. If an additional half line is added in the beginning of the paragraph the whole paragraph must be reformatted. This is not the case with dynamic word wrap that automatically reconsider the line breaks in a paragraph after each typed character.

3.7 Viscosity

Viscosity of a system is a dimension described as resistance to change. Assume that our document in our generic word processor is a report with page numbers and a table of contents and the page numbers are added manually on all pages and the table of contents is made “by hand”. If a page is added in the beginning of our report the page numbers on the following pages must be changed and all page numbers in the table of contents as well. The Viscosity is high and the remedy is the introduction of some kind of automated system for page numbers and generation of the table of content.

3.8 Abstraction gradient

An automated feature normally increases the Abstraction Gradient, which is the third CD in this brief example. The Abstraction Gradient is high if the system is abstraction hungry and demands user created abstractions from start. A system is Abstraction tolerant if it let users start without abstractions but let users create new and use predefined abstractions. Our generic word processor allows users to create a new document and start typing. The user can add
a formatting script later if it’s needed. The word processor can be described as Abstraction tolerant because abstractions, like the formatting script, can be added but are not required.

3.9 The Cognitive Dimensions are Related to Each Other

The cognitive dimensions are related to each other. Some dimensions like abstraction gradient are strongly connected to other dimensions while other dimensions are loosely coupled to the other dimensions in the set. Consider a system that lets the user automatically number the pages and generate the table of contents. This system will probably increase the learning threshold but lower the resistance of changes known as Viscosity. A word processor with automated formatting of page-numbers, page headers and page footers increases the Abstraction, lowers Viscosity and decreases the Closeness of Mapping. The Cognitive Dimensions are helpful when trade-off relations are analyzed and discussed during a design process in this way.
4 reMind: The Old System

reMind is an application software used for optimization of industry processes. The software provides a graphical user interface and lets the user build a model of the process. The model is represented with a directed graph where the nodes represent processing of resources and the links represent flow of resources. (See Figure 1). The user defines the nodes by adding functions and connecting nodes with links. The model is finally imported into an optimization tool that solves the equation system.

![Figure 1: A complex model with many nodes and links](image)

4.1 The Workflow of reMind

The workflow starts when the user builds a model or opens an earlier defined model. The user defines different boundaries for the production quantity, relations between different flows of resources and changes the cost for the incoming resources, called sources. The model is then exported by translating it to an equation system consisting of equations, inequalities and a goal function. The goal function is built of all sources functions, i.e. the incoming resources. An optimizing tool that strives to minimize the goal function solves the equation system. Analysts manually study the result and iterate the process for further investigations.

4.2 An Example of the Workflow

The usage of reMind described by a simple example: the process of heating water with oil and electricity. The system has a furnace that heats water with oil combustion or electricity heating. The resultant flow is heated water. (See Figure 2).

![Figure 2](image)

Each node has a function, and the functions look like this in the model:
Figure 2: The main window of reMind with a four-node, three-links model

Oil source  
Electricity  
Furnace  
Furnace  
Heat  
Heat

It is possible to drag and drop new functions to the nodes. The principle is that a resource, e.g. oil, needs a source and a destination function. Each function has a specified dialog window that guides the user, e.g. the Boundary function dialog. (See Figure 3). Some kind of boundary function is needed to make an optimization meaningful and the other functions are used to relate flows to each other. The user translates the model into an equation system when the graph is complete. The model passes the validating phase before translation, including a check of all flows. Each resource must have a source and a destination but all processing functions in between are optional.

The equation system is processed and the goal function is minimized, in this case a combination of the source functions and the cost of the resource as coefficients. The goal function: \(7.60 \times \text{oil} + 0.50 \times \text{electricity}\). The optimization suggests 100 percent use of electricity in order to minimize the cost for the heat production. The workflow starts over again with changed settings for the functions, export of data and data processing with the optimizing software.

A real world system demands a more complex model to be optimized, often consisting of hundreds of nodes. (See Figure 1). This model contains over 300 nodes and models steel production. The high number of nodes makes the model richer but harder to overview, especially in this flat presentation. The user has organized the model spatially to keep track of different parts of the model.
Figure 3: A function dialog window in reMind
5 Problems and Requirements

The introduction of reMind improved usability. The user could define the model by building a graph. The graph mapped closer to the real system than the text-based predecessor Mind. New requirements and desires came when reMind was used with real world problems. The design goals for reMind were mainly improved usability in terms of effectiveness, efficiency and more intuitive tool, but the users needed more flexibility than reMind offered. An improved graph editor, possibility to reuse a model, improved representation and presentation and flexible node definition are important issues for the design of reMind+. These are discussed below.

5.1 A Very Constrained Graph Editor

The graph editor let the user build a model, but lacked functionality such as copying, cutting and pasting, mark a region and drag a group of nodes. The nodes must be moved one by one if the graph has to be reorganized. This causes severe problems if a model contains hundreds of nodes and they have to be moved one by one to make enough space in the middle of the graph. When half of the nodes are moved the model looks different and important information about the model may have disappeared, including information that the user needs to complete the current task.

5.2 No Possibility to Reuse a Model or a Part of a Model

The reMind software lets the user build a model, save and open it. The model must be treated as one big chunk, which corresponds to the output of the software. Still the users lacked the possibility to reuse a collection of nodes or part of a model. The problem is also connected to the graph editor that did not let the user group nodes in any other way than by placing them closer or in a special pattern. The requirement for reusing models or part of models connects both to the representation of the model and how it is presented and edited in the graph. The user must be able to build a library of standard units that can be included into bigger systems in order to make the process of model building efficient.

5.3 Monolithic Model Structure

The desire to improve the graph editor and the possibility to reuse parts of a model brings an important aspect into the light. The representation of the model is strictly flat and without any hierarchical structure. The model is a map of a real world system that is organized in units and units together into bigger parts. The lack of hierarchy and grouping creates an unnecessary gap between the real world and the simulated system and makes the model harder to understand. The lack of grouping and hierarchy also makes it harder to reuse a part of a model. What is a part in that case? A group of nodes within a certain distance or a group of nodes that are less than three links away from node x? The presentation of the model is flat and presented in a two-dimensional area. The user has two choices when the presentation is bigger than the view: zooming out or panning. Selecting a zoom level in the menu carries out the
zooming and the scrollbars support the pan. When the user has zoomed out he must rely on the graph layout to keep track of the model. The text becomes unreadable when zooming out but on the other hand, a detailed view needs a lot of panning. When panning the user is likely to lose the orientation in a model with hundreds of nodes. The navigation can be extensive when two distant nodes must be compared. The node dialog window can only be viewed one at a time and blocks the rest of the application until it is closed, so-called modal dialog windows.

5.4 A Constrained Set of Predefined Functions

Engineers of different disciplines use reMind, and the predefined set of functions was not enough to cover the needs of all. This lead to industry specific components. The desire for special components can be expressed as a requirement for a flexible node definition that lets the user define new functions. The reMind software let the user build a model consisting of flow of resources and processing of resources. The model is represented by a graph where nodes represent processes and links represent resource flows. The node is defined by the functions that are applied on the node and the settings for each function. The only way to add a new function is to add code for both a new dialog window and the underlying structure. The framework needs an update in at least three modules to include a new function. The source code is available under the conditions of Gnu General Public License, which makes it possible but not trivial. Every complex update is likely to introduce bugs in the source code, so testing and maintenance requires resources too. Finally, the process is slow compared to a system where the user have freedom to define new functions.

The reMind system improved the usability by migrating from a textual environment to a graphical user interface. However it needs design improvements before it is useful in terms of effectiveness and efficiency.
6 reMind+: The New System

The design of reMind+ aims to build an end-user-programming environment instead of completely predefined application software. The programming follows the data-flow paradigm as its predecessor does. This is in contrast to the control flow paradigm of many programming languages. It lets the user define a directed graph of nodes and links that control the flow of resources. The system uses concepts from Direct Manipulation as well as the power of textual language. The model has an input of resources. These are processed and possibly converted into new type of resources as a product of the process. The design mixes concepts from Visual Programming Language and textual language programming. reMind+ introduces new abstractions in order to be more efficient to use.

6.1 The Graphical User Interface Design

The GUI inherits many concepts from the legacy system such as a menu bar with a given set of commands, a frame for graph presentation and a frame with predefined nodes. It differs in the way the model is represented and presented. ReMind+ adds an abstraction level to the graph, which lets the user group a set of nodes into a sub model. These sub models are treated as a single node, possible to save and import in a new scenario. The user interface is also affected by the new design of the node definition. While reMind offered a specific dialog window for each applied function, reMind+ lets the user define the node in one generic window designed like a spreadsheet.

6.2 The Model Representation and Presentation

The representation is the foundation. reMind is a tool that helps the user to define a model of an industry process with a graph consisting of nodes and links. reMind+ is also a tool with the same purpose and a similar model. It contains nodes and links as before. In addition, it can contain models. The inner model is called as sub model of the outer model. This makes the definition recursive, and the reason is a need for increased abstraction of the model. The recursive model makes it possible to handle a part of the model as a unit and hide the details about it. The support for saving and including sub models in a model is also a benefit.

The graph editor interprets the structure of the model and presents the picture of the representation. The graph representation and presentation is connected to the recursive model definition of reMind+. The presentation design supports the new model representation. The hierarchy of the sub model is presented with its nodes inside the border of a square. The user can label and select a background color for each sub model. The user can add comment in the graph in the shape of a rectangle for ASCII text, adding comments like a comment in a textual language to support secondary notation. The comment follows the Post-it on desktop metaphor and let the user move around the note, add or delete text and remove it. This feature moves beyond the Post-it metaphor and lets the user iconify the comment, moved on top or behind the other graph elements including notes.
The new graph presentation comes in two flavors, each with different methods. The first approaches the problem of showing both overview and details by semantic zoom. The second is a hierarchical presentation where the user can view details in sub windows.

Semantic zoom presentations let the user zoom and pan the graph. The semantic zoom hides details when the user zoomed out to a specified level as in Figure 4. The details show up when the user moves closer as in Figure 5 and Figure 6. The panning action does not affect the hiding/showing details. The design tries to relieve the cognitive burden by hiding details at the high level. The details are presented as the user zooms in.

![Figure 4: The semantic zooming approach hides the details at the high level.](image)

![Figure 5: Details shows up when the user zooms in.](image)

The hierarchical presentation lets the user handle the model from a high level and any part of interest can be viewed in a new window by a doubleclick.
with the mouse. When the user has reached the bottom of the hierarchy when the new window only contains nodes and links as in Figure 7. It is possible to dig deeper in the model as long as the new window contains one or more sub models to expand. Figure 8 shows a subwindow of Part A in Figure 7.

Figure 7: The hierarchical browsing approach uses few powerful nodes at the high abstraction level.

6.3 Node Definition

reMind had a predefined set of functions that the user could apply to the node in order to define its behavior. reMind+ extends this concept and lets the user either use predefined functions or mix them with user-defined expressions. A set of arithmetic operators let the user build expressions and mixes them with
functions as in Figure 9. The user can also extend the set by defining new functions at runtime. It is possible to add new functions and save them for later use. The definition is inspired by the spreadsheet concept. It is one of the most successful end-user programming environments today. The design uses the same semantics as the spreadsheet, for example a function starts with ‘=’, and the worksheet consists of addressable cells just like the spreadsheet. The main difference between them is the presentation. reMind+ shows the definition, but a spreadsheet normally shows the numerical result. The reason that reMind+ only show the definition is because reMind+ lets the user make a definition, not a calculation. The spreadsheet concept lets the user write notes and comments in the same sheet as the mathematical definition and the user can arrange the comments freely. A comment starting with a + sign is treated as public comment and used in the graph. The comment shows up like a ”tool tip” when the user positions the mouse pointer over a node in the graph.
6.4 Type Validation

The system needs a stronger type validation for two reasons. First, the increased abstraction level makes a more flexible system, but also a system that is harder to grasp. Second, the system helps the user to define a model, not to calculate, optimize or any other operation that gives feedback about the definition. Validation increases the abstraction by introducing a type tree. (See Figure 10). The user can define a type tree and a set of conversion rules that the system applies on the model before it is finished. The tree construction lets the user handle different resources as one by creating a tree node with leaves. All leaves inherit the conversion rule from their parents.

![Type tree diagram]

Figure 10: The type tree editing window in reMind+.

For example we could have a type tree suitable for the simple model presented in Section 4. The resources are oil, electricity and heated water. The type tree has Resource as root with Energy and Material as children. Energy has the children Heat and Electricity and Heat has the three leaves Steam, Heated water and Combustion emission. Material has the child Furnace Material and Furnace Material has the three children Coke, Oil and Coal. The set of rules
consists of three simple rules: Electricity to Heat, Furnace material to Heat and Steam to Electricity. The abstraction level of the tree let the user define an extensive set of Resources with a set of rules that are still possible to overview. The example above lets the system validate all resource conversions before the optimization is executed.
7 How the Design of reMind+ Solves the Problems

The design of reMind+ aims to solve a set of problems, which are more or less interconnected. The main problems with reMind are a static and tight system for node definitions, limited opportunities to reuse defined models and a graph editor with limited functionality. The graph editor is directly mapped to the model definition, so a richer model definition affects the graph editor. The model must be developed to support a more advanced graph editor. The model is thus the representation, and the graph is the presentation. They must be treated in parallel. The design of reMind+ tries to meet the problems by introducing a richer model definition, an improved graph editor with support for a hierarchical structure and a node definition with higher degree of freedom. All these changes add complexity to the system so a stronger validation of the model is also suggested to prevent errors in the model.

7.1 Representation

The new representation aims to meet the demand of model reuse and model building with modules in contrast with reMind where the model is one monolithic structure. One or more nodes, connecting links and zero or more models define the model. This recursive structure is the foundation for both reuse and hierarchical presentation. reMind+ lets the user treat a model within a model as a node: drag and drop it into the graph, connect and disconnect with links to other nodes and also save it for later reuse. It is possible to build a library of more or less general modules that can be reused. A module can be added into the model in one or more places. For example, the user defines a group of nodes and links as a power station, saves it in the library and imports it in three different locations in an industry model. Another effect is stronger support for abstractions. A complex model can be treated as a set of modules that let users leave details and focus on the overview. For example, a complex industry process can be treated as a set of modules consisting of Coking Plant, Blast Furnace and Steel Plant. This improved model is easier to handle for the engineer, lowering the learning threshold when a person needs to work with a ready made model. It guides the user to think of the model as a set of modules instead of complex cloud of nodes and links.

7.2 Presentation

The presentation strategy follows two paths. One path is named Semantic zooming and the other path Hierarchical browsing. Both strategies try to support the intention of the modularized representation. The new graph editor lets the user drag and drop individual nodes or marks a region of nodes and moves them together. The hierarchy is exploited in two different ways: the Semantic zoom lets the user zoom out to hide details, and see the overview, while zooming in reveals details. (See Figure 5 and Figure 6). Panning vertically and horizontally are possible when the presentation is bigger than the view port. It is performed with traditional scrollbars. Hierarchical browsing presents a top-level model where only the modules are possible to see. For example, Part A to Part E in Figure 7. It is possible to open a sub window with a double click in the area of
Part A module to view details about it. (See Figure 8.) The browsing is finished if the Part A is organized as a flat model, but if it is build with modules. These can be browsed in the same way. Both of these presentation concepts support the user in the process of building a mental model of the system.

7.3 Node Definition

Combining two strategies solves the problem with a tight and static set of functions for node definition. The node is defined in a dialog window that uses the concept of spreadsheet, one of the most successful end-user programming environments. The spreadsheet, a set of functions, logical and arithmetical operators let the user define the node. The user can also define new functions. The functions within a spreadsheet relax the need for a specially designed dialog window for each function but give less guidance for usage. The absence of the dialog is compensated by help instruction that is available for all predefined functions. User-defined functions depend on the instructions added by the user. Node dialog windows in parallel improve the reuse of a single node, and the user can copy and paste a solution from one node into another.
8 Inspection with Cognitive Dimensions of Notations

CD is a used by experts to evaluate a system with focus on usability. Cockton et al. [4] defines usability in terms of a system that is effective, efficient and enjoyable to use. CD builds upon a set of dimensions and tries to find trade-off relations and remedies for usability problems. A CD-based inspection of the design of reMind+ compared with its predecessor reMind follows.

8.1 The Design of the Graphical User Interface

The usability is affected by the new representation in many ways. The combination of a hierarchical representation, presentation and improved graph editor lowers the cost for many operations; in other words lowers the resistance to change known as Viscosity. It is possible to handle part of a model as a unit, move it around and reorganize the graph much easier than before.

Hidden Dependencies are typically low in data-flow languages, but the new representation increases the gradient of this dimension. The model validation uses a set of rules for conversion of resources, and these rules are not visible in the graph. The problem is increased even more by the possibility of creating nested models. The user must traverse the whole model to see how the resources are used to know if a rule is used or when a rule can be removed.

The user has to start the layout at some point of the graph editing area. The decision about that position must be taken before the user knows how to dispose the area and before the model is built. This problem connects to the dimension Premature Commitment. The new system has not improved the possibility to avoid Premature Commitment, but the decreased cost for rearrangement of the graph is an effective remedy for this.

Modularity is another positive effect of the new representation that can be used to lower Premature Commitment. The user can build a part of a model and validate it before including it into a larger context. Modularity can be used to avoid a spaghetti graph. The top-level can have a limited number of visible nodes. The details appear when each node is examined.

Viscosity is decreased by the introduction of a new abstraction level, i.e. the recursive model. The increased abstraction is an increase of the dimension Abstraction gradient. reMind was Abstraction-Hating because the low number of abstractions used and no possibility to create new abstractions. reMind+ is classified by CD as Abstraction-Tolerant because it allows the user to create and use new abstractions, but can be used without them. The model can be defined totally flat without nested models inside the model, as it is done in reMind.

The Closeness of Mapping is greatly increased by the new abstraction. A real world plant can now be modeled as a set of models combined into a complete model. Each model can be divided further into less complex parts. reMind+ let the user define modules in parallel and use test them one by one, adding, removing or collect them into libraries of useful modules.

The Secondary Notation is improved by a possibility to attach a comment to a node like a Post-it note connected with the object with a dashed line. This note gives the model builder a possibility to explain the design and the role of different entities. The effort of writing a note makes the graph building process
more viscous and the note may also hide a part of the graph behind. The latter effect lowers the Visibility. The note can be minimized to increase the Visibility but makes viewing more viscous when the user must expand all notes before they appear as visible text.

The Hierarchical browsing presentation gives a better overview but less Visibility of the details. It is possible to view different sub models in different windows. This increases the possibility to juxtapose sub models. The Semantic zooming presentation is the other way around. It is possible to view details when zooming in but impossible to juxtapose details that are spatially far away from each other in the graph.

Closeness of Mapping is increased by the conversion rules. The type tree and the conversion rules are a framework that can be used to make the model more realistic. The user can model rules that for example prevent a conversion of lead to gold.

Consistency is an important issue for a usable programming system. The new presentation with Hierarchical browsing has a graph with ordinary nodes and sub models. Sub models are defined by a graph and nodes are defined in the spreadsheet environment. A sub model is distinguished from a node by a slightly thicker frame around it. One can argue that this creates an inconsistency with two node types that look similar but are defined in a very different way. The possibility to hide part of a model in a sub model node can be used to present a more abstract overview. This can be used on the top level to emphasize the overview of the system. This comes with a trade-off relation with Diffuseness that is increased. A careful choice of comments and structure can minimize the Diffuseness and help the understanding on the other hand.

The graph editor has many new features such as selection of a group of nodes in different ways. This group can then be treated as a unit, for example dragged around when the graph must be restructured. Consider the difference between moving 100 nodes one by one or in one single drag and drop action.

The media is a fundamental characteristic for a graph. It was originally designed for paper or blackboards. The computer display lacks the possibility to draw a path with a pencil and make small notations. This media-related problem can make operations in the graph into Hard Mental Operations. Typical signs are when the user points on the screen to keep track of a location or usage of pencil and paper in parallel with the computer usage. The possibility to break down the structure in sub models will probably lower the cognitive workload. A possibility to explain the model with comments may also lower the risk for Hard Mental Operations.

Another positive effect of the recursive model definition is improved Progressive Evaluation. The user can build a part of a large structure. This part can be validated or even tested and used before it is included in a larger context. This possibility makes it possible to achieve Progressive Evaluation. The modularity has also a positive effect on the Role Expressiveness. A careful choice of top-level abstractions will help the user to understand the meaning. Each part communicates its role in the model. Secondary Notation in the form of comments and cues increase the Role Expressiveness in the graph, for example to distinguish a node from a sub model.
8.2 The Node Definition

The urge for a node definition that let the user define new functions is literally an urge for increased Abstraction gradient. The new design is strictly more complex than its predecessor and increases the learning threshold for beginners. The user can use different paths using a function as a side-effect of the increased level of freedom, compared with reMind where each function offered one solving path.

The new node definition takes advantage of the freedom of the spreadsheet. The spreadsheet uses one-way, directly or indirectly by addressing cells. The user must check each cell in order to check if a certain cell is in use by another cell. This is one form of Hidden Dependency. The other form of Hidden Dependency is also represented in the spreadsheet: a chain of dependencies where cell A is dependent on cell B which is turn dependent on cell C.

All functions are presented in one area in the node. This is not the case in reMind where every function uses its own dialog window. The possibility for increased overview is known as the dimension of Visibility. It is how much of the code that is simultaneously visible.

It is possible to add a function directly in the node definition dialog window. This avoids the Premature Commitment of its predecessor where a function must be added directly in the graph before the node definition is viewable.

The spreadsheet gives a rich opportunity to write comments in connection with an expression. The spreadsheet concept helps the user to make changes to the definition of different functions without closing and opening the dialog window. It is easy to grasp the node definition due to the increased Visibility. The time step resolution make separate node definition for each time step. The user must toggle between these different time steps in the same dialog window. This makes them impossible to Juxtapose. reMind+ can though view different node definition in parallel, which increase the Juxtaposibility. This can be helpful for comparison or transcription activities.

The Closeness of Mapping is low when a node definition is compared with the real system counterpart. The process in the real system can be a furnace process and the definition is basically a spreadsheet. Engineers to model a real system use reMind+, and they describes the process by some mathematical rules. The mapping between this mathematical description and the node definition is closer in the new design. This makes the mapping closer than its predecessor where a mathematical description must be transformed into a limited set of functions. The definition makes the functions more consistent by a common interface with an identifier and a number of arguments. The user-defined functions can be more or less Diffuse depending on the choice of identifier, comments and arguments. The pre-defined functions are provided with descriptions and instructions in order to minimize the Diffuseness. This model is probably more Error-prone than its predecessor. Help instruction and model validation are the remedies used to compensate for increased Error-proneness.

Hidden dependencies may occur if the user refers to other cells in the sheet. Backtracking a definition can become a Hard Mental Operation if the dependency chain is long. The user might use paper and pencil or point with the fingers on the display in order to keep track of a dependency path. The definition is validated when the dialog window is closed. The windows are kept open if the process discovers invalid definitions. The syntax error is highlighted and
8.3 Type Checking and Model Validation

The increased Abstraction gradient for the node definition lets the user fulfill a task in different ways and increases the possibility for errors. An increased level of freedom is tightly coupled to the Abstraction dimension, and one remedy is an automated validation of the model. The type checking system lets the users define a tree structure for all resources in the model and which resources can be converted to other resources. A correct type tree and set of conversion rules is a counterbalance for the increased Abstraction of the node definition. However, the user must define the type system and as a new abstract component it must be introduced and carefully set to avoid problems. The type tree itself increases the Abstraction gradient of the system.

The validation system creates Hidden Dependencies. A node definition must regard the type conversion rules. A function that converts a resource into another resource must follow the stated limits. The whole model must be traversed if the user wants to know if a rule is used by a function within the current model.

A workflow must start with a definition of some type of resource. This forces the user into Premature Commitment. The type tree and the conversion rules are built in advance of the model as much as possible. The rest of the tree will probably be built in a more ad hoc fashion. A new type of resource and a corresponding conversion rule is created when a new resource is needed.

A new resource can easily be included in the type tree. A conversion rule demands more of attention before it is included. It is costly to remove a resource from the tree. All conversion rules and all node definitions must be examined to see if the resource is in use. A conversion rule can be used by the resource or by some resources in the sub tree of that resource. It is a demanding task to check if a conversion rule is in use because of that and the fact that all functions must be checked. The Viscosity for restructuring the type tree and the conversion rules can be extremely high for a complex model with many node definitions, types of resources and conversion rules.

The Visibility of the type tree is good as long as it is completely visible on screen. Sub trees can be shrunk into one node as in common file handlers. That construction increases the overview of big trees but hides details. A sub tree can be expanded with a single mouse click. It is possible to Juxtapose two sub trees if the rest of the tree is compressed.

The fundamental idea with a type tree and conversion rules is model validation. The user can decide how close the model must map to its real counterpart. The Closeness of Mapping is related to the type system created by the user. The type tree is integrated with the resource-managing tool in order to achieve Consistency. The level of Diffuseness of the type system relies on the user. A good structure and self-documenting naming of the resource identifiers lowers the Diffuseness. The reMind+ type system cannot ensure a well-structured type tree but some properties support the user, for example a type tree manager that is easy to handle. Both Error-proneness and Diffuseness will probably decrease if it is easy to create and modify the type system. The type system makes the model creation less Error-prone. The type system itself can be hard to maintain properly and can be a source of errors. The user must traverse the whole model to check if a certain conversion rule is in use. This will become a Hard Mental
Operation for a big model. The user needs to remember which nodes that are checked and which are not.

Progressive Evaluation is an important property of a programming system. The type system can provide a useful source for model validation but also an obstacle for Progressive Evaluation. The user is recommended to postpone the creation of the conversion rules during an exploratory phase. This may improve the foundation for the exploratory phase.
9 Conclusions

The most important conclusion is that all design changes have both advantages and disadvantages. The decisions are more complex than just choosing between bad or good solutions. Cognitive Dimension of Notations has been useful to evaluate the design suggestions. CD led the evaluation further behind of just bad or good solution. It focuses on the relation between different usability aspects and their trade-offs. It is clear that CD can be useful for that process. But, are the results relevant in practice?

The fulfillment of the requirements increased the complexity and the abstraction gradient of the tool and probably made it more powerful. Usability testing is needed to confirm that the tool is simple enough to learn, as an end-user programming system should be, and powerful enough to be effective and efficient. Usability testing can also be useful in the evaluation of the CD usage.
10 Future work

The redesign process and the evaluation are inspired by theoretical studies and software engineering principles. The evaluation is a theoretical discussion tool. How relevant are the conclusions and the results? Here follows a list of suggestions for future work of this project:

- Usability testing with zooming and browsing interface and usability testing of the function definition. Are the design suggestions on the right track? How should the suggestions be tuned to improve the usability? Are there matching results between usability testing and the theoretical evaluation, in other words if the CD evaluation gave relevant results?

- The user can save a model or a part of a model and reuse it later. How should the library be built?

- Evaluate what size and sort of sub models are easiest to reuse.
11 Acknowledgement

I would like to thank David Carr for supervising this project including this technical report. Many of the design decisions rely heavily on his suggestions.

Mikael Larsson and Jan Dahl at Energiteknik/LTU contributed with knowledge about the usage of reMind and requirements for reMind+.

I thank my family for encouragement and support.
References


