HYPERLISTS

Geir Isene
g@isene.com
http://isene.com

inspired by
Jean-Dominique Warnier, Kenneth Orr
Egil Möller, Axel Liljencrantz, Marilyn Abrahamian

editing
Marilyn Abrahamian

Version: 2.4 (2019-10-05)

Everything. Concise and precise.

HyperList page: http://isene.org/hyperlist/

Contents

Abstract

HYPERLIST^S represents a way to describe anything – any state or any transition. It represents data in tree-structure lists with a very rich set of features[1]. It can be used for any structuring of data, such as:

- Todo lists
- Project plans
- Data structures
- Business processes
- Logic breakdowns
- Food recipes
- Outlines of ideas
- . . . and much, much more

The article below starts off with several examples where lists are useful as a way of describing something and where HYPERLIST^S is especially powerful. It then goes into every part of HYPERLIST^S and how it can be used to describe anything (yes – anything). Toward the end, HYPERLIST^S is used in describing itself, something very few frameworks are able to accomplish.

> *"The list is the origin of culture. It's part of the history of art and literature. What does culture want? To make infinity comprehensible. It also wants to create order – not always, but often. And how, as a human being, does one face infinity? How does one attempt to grasp the incomprehensible? Through lists [. . .]" (Umberto Eco)*

[1] The "HyperList to the power of S" signifies the iterative or fractal nature of such lists.
The name represents both singular and plural of the word – both the system and a list or lists conforming to this system.

1 Introduction

On the way in to the shopping mall, you take a quick glance at the list given to you by your mother/father/brother/wife:

5 liters of milk
2 packages of butter
2 liters of orange juice
bananas (5-8)

A simple list. But the next time you go to the shop, the list has grown to a paragraph:

> *" If they have pepperoni and that special pizza sauce, buy that and also flour, yeast, cheese and ham. If not, then buy the Indian spicy chicken with 5 or more suitable vegetables. If the chicken is sold out, be creative and decide what we should have for dinner. Also buy apple juice, eggs, washing powder and paper towels."*

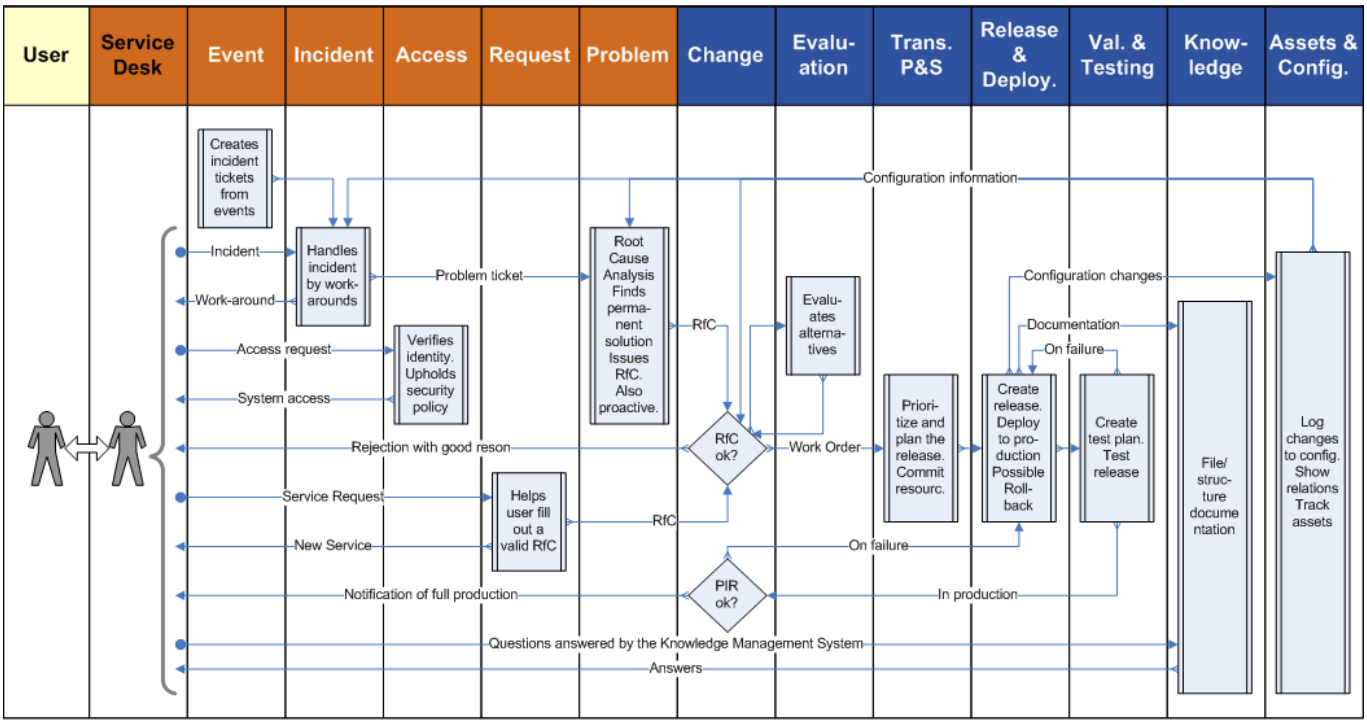
You wonder if that paragraph can be written a bit more concise.

At work you are confronted with the task of writing a few efficient processes for the IT department. You're supposed to create the overall process for IT Service Operations[2] and IT Service Transition [3] together with key personnel from that area. IT Service Operations covers that part of the IT department dealing with the daily operations while IT Service Transition coordinates and implements all changes within IT. Both areas contain several processes, supporting delivery of efficient IT services.

The task of mapping the processes would be easy if you could simply chuck them all into a room with a whiteboard and string it out. But since they are located in three different cities, you will have to do some nifty collaboration.

With the expectation from management of a colorful flowchart[4] depicting the overall work flow, you are in for a challenge. Collaborate drawing tools are not that easy.

You decide to send a draft out and let each participant amend the draft and send it back to you. After incorporating all the changes you deem good, you send out another draft, etc. After weeks of battling some people's disappointment over the ditching of some of their ideas, you finally arrive at a process map that you turn into this:

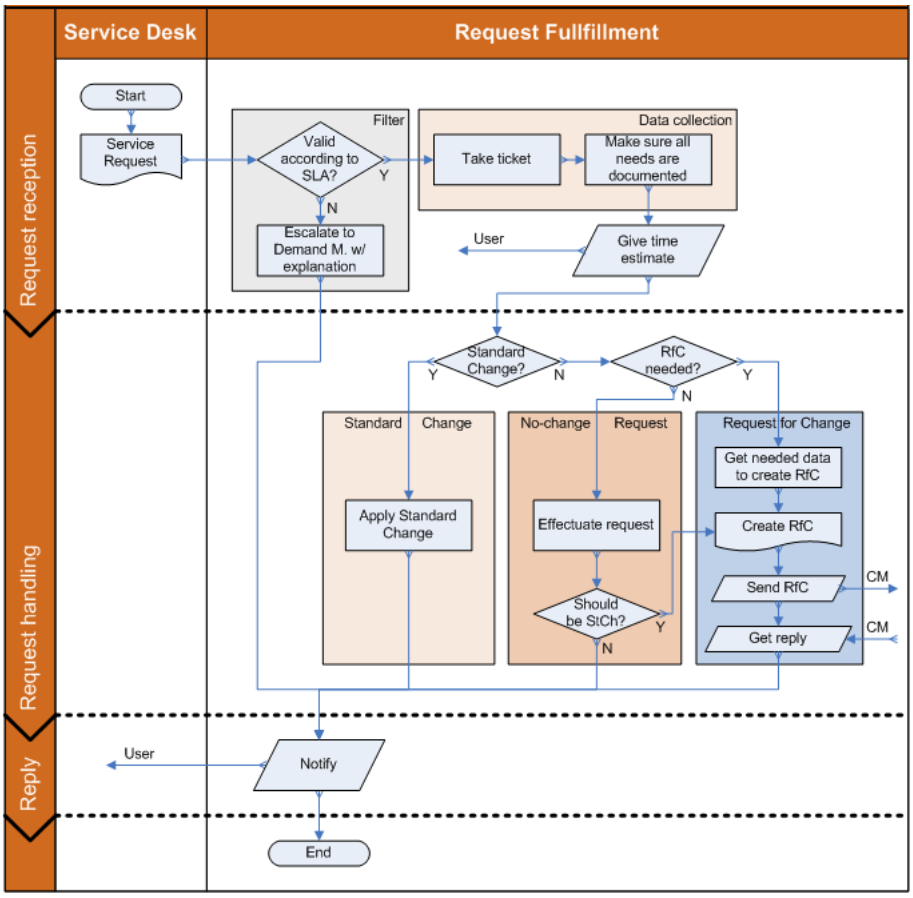


[2] IT Service Operations is part of the ITIL best practice framework,
see http://www.itil-officialsite.com/AboutITIL/WhatisITIL.aspx
and http://www.itframeworks.org/wiki/Category:Service_Operation
[3] IT Service Transition: http://www.itframeworks.org/wiki/Category:Service_Transition
[4] Flowchart: http://en.wikipedia.org/wiki/Flowchart

While working hard on creating process maps[5], you become frustrated at all the things the flowchart methodology cannot do or that are hard to represent, such as:

- Telling the user not to do something
- Repeating a set of tasks a certain number of times
- Parallel flows merging at certain times or under certain conditions
- Multiple decisions, such as picking one of 256 colors (creating one hell of a diamond tree)

Still, you settle for the flowchart methodology for now, creating several that look like this[6]:



At your desk one morning, you labor at the task of managing an ever-growing task list[7]. With half a dozen responsibilities under your command, the list is starting to get a bit messy. You end up grouping the tasks and adding subtasks in a todo-list tree to give it a more workable structure.

[5]Process maps: http://en.wikipedia.org/wiki/Process_Maps
[6]Request Fulfillment, an ITIL process: http://www.itframeworks.org/wiki/Category:Request_Fulfilment_Management
[7]Task list or Todo list: http://en.wikipedia.org/wiki/Todo_list#Task_list

In your favorite text editor, after a few minutes, and many minutes before it's completed, it looks like this:

- Marketing
  - Create a campaign for selling the new mobile service
    - Contact our advertisement agency – Done
    - Set up an initial brainstorming meeting – Done (booked for 2012-02-03)
    - Hold the meeting
      - Work out a project plan
      - Commit resources
- Public Relations
  - Contact Peter at Mobile Press
    - Present our new revolutionary mobile service
    - Give him and possibly other journalists there a free trial of the service
- Business development
  - Refine the Service Portfolio strategy
  - Set up a Project Task Team (PTT) for bailing out failing projects
    - Include Denise, Tracy and John P in the PTT
- Process design
  - Finalize the overall process design for IT
    - First target is Service Operations and Service Transition
      - Get the final proposal approved by the IT Manager – Done
        - Get approval for key process maps
          - Incident Management
          - Problem Management
          - Change Management
          - Request Fulfillment (should Access Management be included here?)
          - Release & Deployment Management (include Validation & Testing in same map?)
      - Get from the IT Manager the next focus area

A todo list or task list like this resembles a shopping list. It is also similar to describing a business process...

It strikes you that there should be a methodology that would be capable of describing anything. Any shopping list, business process, todo list, database model, food recipe, jotting down of creative ideas, building logical arguments, or anything else that it would be useful to describe in some way. It should not matter whether it is a thing, a state, a flow, a process or wild ideas that are to be described. They should be describable using the same format, the same methodology. It should be text based for easy collaboration, like we see with Wikipedia, and it should overcome the limitations of, for instance, the flowchart methodology.

You start searching. And you come across the website isene.com[8] with a document titled "HYPERLISTS". Intrigued by the quote on the front page, you start reading... this document.

You study up on the HYPERLISTS methodology and decide to take the previous shopping lists, business processes and todo lists and turn them into real HYPERLISTS ...

5 liters of milk
2 packages of butter
2 liters of orange juice
bananas (5-8)

... becomes:

```
Shopping 2012-01-15
   [5 liters] milk
   [2 packages] butter
   [2 liters] orange juice
   [5..8] bananas
```

[8] The website of the author: http://isene.com

The content in the square brackets denotes the amount or number of items on that line.

You realize with this simple convention, you can describe things that would be hard to represent in flowcharts. You try to think of ways to draw the following in a flowchart:

```
Write ad
[3] Proofread
   Correct errors
```

First write the ad, then proofread it three times sequentially while correcting errors in the text during each proofreading. In HYPERLISTS it is that easy to say that something should be done a certain number of times.

The "paragraph shopping list":

> *" If they have pepperoni and that special pizza sauce, buy that and also flour, yeast, cheese and ham. If not, then buy the Indian spicy chicken with 5 or more suitable vegetables. If the chicken is sold out, be creative and decide what we should have for dinner. Also buy apple juice, eggs, washing powder and paper towels."*

. . . becomes:

```
Shopping 2012-01-19
   OR(PRIORITY):
      Pepperoni and special pizza sauce
         Flour
         Yeast
         Cheese
         Ham
      Indian spicy chicken
         [5+] suitable vegetables
      Be creative and decide what we should have for dinner
   Apple juice
   Eggs
   Washing powder
   Paper towels
```

With the use of the "OR", the list shows that only one of the three sub-Items ("Pepperoni and special pizza sauce", "Indian spicy chicken" and "Be creative and decide. . . ") is to be chosen. The "(PRIORITY)" is a Comment specifying that the sub-Items are to be chosen in the order of priority as listed.

The overall process map for IT Service Operations and Service Transition becomes (with added valuable information):

```
IT Service Operations and Service Transition processes
   Service Desk (Service Operations function)
      Dispatch communication between User and process areas
   Service Operations processes
      Event Management
         Create Incident tickets from events, generated by monitoring systems
      Incident Management
         Create Workaround for Incident ticket
         [?] Create Problem ticket and forward to <Problem Management>
      Access Management
         Verify identity of requester for access to a given service
         Uphold Security Policy
         Grant access to service
      Request Fulfillment
         [Request is Standard Change] Handle as authorized by Change Management
         [Request is not a Standard Change] Help User fill out a Request for Change (RfC)
            Send RfC to <Change Management>
            Keep User informed of progress of the requested service
```

```
   Problem Management
      Do Root Cause Analysis on the problem reported in Problem ticket
      Create a permanent solution to the problem
      [Solution requires a Standard Change] Execute Standard Change
      [Solution requires a non-standard Change] Create RfC
         Submit RfC to <Change Management>
Service Transition processes
   Change Management
      Receive RfC
      [Evaluation needed] Forward to <Evaluation Management>
      Decide on RfC
         [Approved] Forward a Work Order to <Transition Planning & Support>
         [Not approved] Send rejection with proper reason to submitter
      Receive final Release Report from <Validation & Testing Management>
         [PIR not approved] Send to <Release & Deployment Management> for correction
         [PIR approved] Notify appropriate parties of new service in production
   Evaluation Management
      Receive request for evaluation of RfC from Change Management
      Evaluate RfC and possible alternative solutions
      Report evaluation to <Change Management>
   Transition Planning & Support
      Receive Work Order from Change Management
      Prioritize release
      Commit resources
   [Test Platform, Pre-production Platform, Production Platform]
      Release & Deployment Management
         Create release
         Deploy release
      Validation & Testing Management
         Test release
            [Test fails] Send to <Release & Deployment Management> for correction
            Ensure correct documentation in Service Knowledge Management System (SKMS)
            [Production Platform] Send final report to <Change Management> for PIR
   Knowledge Management
      Inspire knowledge sharing
      Ensure the SKMS contains appropriate knowledge
      Efficiently service all parts of the organization with appropriate knowledge
   Service Assets & Configuration Management
      Ensure a complete Configuration Management DataBase (CMDB)
      Ensure logging of any changes to Configuration Items (Cis)
      Efficiently service all parts of the organization with configuration information
```

The “[?]” denotes an optional Item, while “[Test fails]” clarifies when the option comes into play (i.e. “do this if the test fails”).

The line “[Test Platform, Pre-production Platform, Production Platform]” denotes that for each of the environments, “Test Platform”, “Pre-production Platform” and “Production Platform”, the Items below (the children) are to be done.

The specific process for Request Fulfillment also becomes a HYPERLIST^S . But as you create a HYPERLIST^S out of the process, it dawns on you how easy it is to include additional information about the process.

```
Request Fulfillment
   Description
      + Request Fulfillment (RF) handles all
       orders from users and executes Standard Changes
      Users may contact RF to discuss possible orders
   Input
      Orders from users
   Result
      Efficient delivery of orders
   Metrics
      Sum of time lapsed on open requests
      Response time
   Process
      Receive Service Request from User via the Service Desk
      [Not valid according to SLA]
         Escalate to <Demand Management>
      Take ticket
      Ensure all needs are documented
      Give time estimate to User
      [Standard Change (SC)] Apply Standard Change
      [RfC not needed] Effectuate request
         [Request is not an SC] <Notification>
      Create RfC
      Send RfC to <Change Management>
      Receive communication about the RfC
         Forward information to User
      Notification
         Give final notification of result to User
```

The "+" at the start of the third line in the list denotes that a list Item breaks over two or more lines.
The todo list at work is rather easily turned into a valid HYPERLIST^S . In fact, it looks almost the same as before:

```
Todo list
   [_] Marketing
      [_] Create a campaign for selling the new mobile service
      [x] Contact our advertisement agency
      [x] Set up an initial brainstorming meeting
      [_] 2012-02-03: Hold the meeting
         [_] Work out a project plan
         [_] Commit resources
   [_] Public Relations
      [_] Contact Peter at Mobile Press
      [_] Present our new revolutionary mobile service
      [_] Give him a free trial of the service
         [?] Give other journalists there a free trial of the service
   [_] Business development
      [_] Refine the Service Portfolio strategy
      [_] Set up a Project Task Team (PTT) for bailing out failing projects
      [_] Include Denise, Tracy and John P in the PTT
```

```
[_] Process design
   [_] Finalize the overall process design for IT
   [_] First target is Service Operations and Service Transition
   [x] Get the final proposal approved by the IT Manager
   [_] Get approval for key process maps
      [_] Request Fulfillment
         [?] Include Access Management
      [_] Incident Management
      [_] Problem Management
      [_] Change Management
      [_] Release & Deployment Management
         [?] Include Validation & Testing in same map
   [_] Get from the IT Manager the next focus area
```

You then start using HYPERLISTS for any and all things you want to describe in a concise and precise form. Even a logical breakdown of whether the universe could be deterministic – after all, the philosophical concepts of Free Will[9] and Quantum Mechanics are two of your favorite hobbies[10]:

```
A proof against determinism
   A fully deterministic system; AND:
      Governing rules must be consistent
      Governing rules must be complete
   Gödel's Incompleteness Theorems: No system of rules can be both complete and consistent
   No system can be fully deterministic
   The universe is non-deterministic
```

The "AND:" denotes that the sub-Items must both happen or must both be true.

You start using HYPERLISTS regularly at work for your todo lists in your favorite text editor. You use HYPERLISTS with pen and paper when jotting down ideas or when writing a shopping list. Or outlining an article you are about to write. A business strategy. Your New Year's resolutions. Your project for refurbishing your car. Your favorite food recipe:

```
Chocolate Chip Cheesecake
   Ingredients
      Bottom crust
         [2 cups] graham cracker crumbs
         [6 tablespoons] melted butter
         [1/4 cup] sugar
      Filling
         [2-1/4 lb.] cream cheese
         [5] eggs
         [2/3 cup] sugar
         [1 tablespoon] vanilla extract
         [1 cup] Baileys Irish Cream
         [1 cup] semisweet chocolate chips
      Topping
         [1 cup] whipping cream
         [1 teaspoon] instant coffee powder
         [2 tablespoons] sugar
      Garnish
         Chocolate curls
```

[9] My article, "On Will": http://isene.com/onwill.pdf

[10] For Gödel's Incompleteness Theorems, see http://en.wikipedia.org/wiki/G%C3%B6del%27s_incompleteness_theorems, and for the world's shortest proof, see http://blog.plover.com/math/Gdl-Smullyan.html

```
Directions
   Preheat oven to 325 degrees F
   Coat a 9-inch springform pan with nonstick vegetable spray
   Combine crumbs and 1/4 cup of sugar in the pan, stir in the melted butter
   Press mixture into bottom and 1 inch up the sides of the pan. Bake for 7 minutes
   In a food processor, beat cream cheese, add 1-2/3 cup sugar and eggs and mix
   Blend in the vanilla and Baileys
   Sprinkle half the chocolate chips over the crust
   Spoon in the cheese mixture filling
   Sprinkle with the remaining chocolate chips
   Bake for 1 hour and 20 minutes
   Cool cake completely
   Beat whipping cream, coffee powder, and 2 tablespoons sugar until peaks form
   Spread mixture over cooled cake and garnish with chocolate curls
```

You use HYPERLISTS on a whiteboard at work for easy collaboration with colleagues. No more focus on what tool to use to create pretty graphics or how to make it possible for many people to work on the same drawing at once. Sure, those tools do exist, but you see that a tool can easily distract from creating real content. After a HYPERLISTS is created, it is easy to churn out some pretty and colorful slides with flowcharts or other graphics to show off to your bosses. The HYPERLISTS remains as neat "source code" underneath the flashy diagrams as they are more easily maintained, being pure textual lists.

Many HYPERLISTS later, you are getting the hang of it. Time to dive in at the deep end :)

2 Background and definition

Having worked extensively with flowcharts, relations diagrams[11], Venn diagrams[12], Warnier-Orr diagrams[13] and other ways of representing processes, states, instructions, programs and data models, I knew there was something missing. It would have been great to have a way of representing anything – any state or action – in a way that would be more conducive to collaboration.

Most people know about flowcharts. But there are other ways of representing data, states, actions or processes: UML[14], Sankey diagrams[15], Decision trees[16], Petri Net[17], Organizational charts[18], Mind maps[19], Process Flow diagrams[20], Feynman diagrams[21], Data Flow diagrams[22], Concept maps[23], OBASHI[24], Task lists (or Todo lists), Warnier/Orr diagrams and various other diagrams. More than we can list here. They have various uses, various strengths and shortcomings.

Most methodologies for representing states or flows were born out of specific needs and were not meant to be used as generic methods for representing literally anything as simple as possible. What if there were a way to represent any state, set of things, actions, flows or transitions? What if the method were simple? What if it were also Turing complete[25]?

Enter HyperLists – a system for representing data. Any data. Static or dynamic. It can be used to describe any state: a thing or set of things, an area, a concept or collection of concepts, etc. It can also be used to describe any action or plan, transformation or transition. In fact, it can describe anything – with one complete markup set. HyperLists can be be used as an outliner on steroids or a todo-list managing system with unlimited possibilities.

After searching for a complete markup methodology for both states or things and actions or transitions, I came across the Warnier/Orr diagrams. They seemed to be the best foundation for what I needed. The methodology was expanded to be capable of describing anything as easily as possible; I removed the graphical parts of Warnier/Orr and expanded the system substantially. It turned into WOIM (Warnier/Orr/Isene/Möeller) and later got the more descriptive name of HyperLists . Egil Möeller[26] helped in the early refinements.

Besides being the name of the system, "HyperLists " is also used when referring to a list or lists conforming to this system.

The strengths of HyperLists are many:

- can represent any state or action with any number of levels
- Turing complete
- text-based (it is wiki-able – i.e. it is easy to collaborate when creating HyperLists)
- not graphical (although it can easily be made graphical for ease of consumption or eye candy)
- an easy syntax, humanly very readable
- very compact
- can represent negatives ("NOT:" = Don't do the following actions)
- can represent any number of choices in a decision (hard to do in a flowchart)
- easy to do loop counts
- easy to show attributes such as time or timing, location, person responsible, etc.
- potentially easy to map to other representation methods (graphical or otherwise)

In its simplest form, a HyperLists is just a list of Items – like your regular shopping list – but it can be so much more if you need it to be. A couple of examples will give you the basic idea.

[11] Relations diagrams: http://asq.org/learn-about-quality/new-management-planning-tools/overview/relations-diagram.html
[12] Venn diagrams: http://en.wikipedia.org/wiki/Venn_diagram
[13] Warnier-Orr diagrams: http://en.wikipedia.org/wiki/Warnier/Orr_diagramandhttp://varatek.com/warnierorr.html
[14] Unified Modeling Language (UML): http://en.wikipedia.org/wiki/Unified_Modeling_Language
[15] Sankey diagrams: http://en.wikipedia.org/wiki/Sankey_diagram
[16] Decision trees: http://en.wikipedia.org/wiki/Decision_tree
[17] Petri Net: http://en.wikipedia.org/wiki/Petri_netandhttp://www.petrinets.info/
[18] Organizational charts: http://en.wikipedia.org/wiki/Organizational_chart
[19] Mind maps: http://en.wikipedia.org/wiki/Mind_map
[20] Process Flow diagrams: http://en.wikipedia.org/wiki/Process_flow_diagram
[21] Feynman diagrams: http://en.wikipedia.org/wiki/Feynman_diagram
[22] Data Flow diagrams: http://en.wikipedia.org/wiki/Data_flow_diagram
[23] Concept maps: http://en.wikipedia.org/wiki/Concept_map
[24] OBASHI: http://en.wikipedia.org/wiki/OBASHI
[25] Turing completeness: http://esolangs.org/wiki/Turing-complete
[26] Egil Möeller's personal website: http://redhog.org/

An example of describing a state:

```
Car (example obviously not complete)
   Exterior
      Paint
      Chrome decor
      Windows
         Rubber linings
      [4] Wheels
   Interior
      Seats
         [2] Front
         [3] Back
   Mechanics
      Motor
         [6] Cylinders
         [24] Valves
      Brakes
```

A transition example (task list):

```
Walk the dog
   Check the weather
      [Rain] AND/OR:
         Get rain coat
         Get umbrella
      Dress for the temperature
   Get chain
   Call the dog
   OR:
      Go through the woods
      Walk the usual track
   AND: (concurrency)
      Ensure the dog has done its "tasks"
      Ensure the dog is exercised
        [5+] throw the favorite stick
   Walk home
```

And this can all be done in collaboration on a wiki. States are described, processes are mapped, plans and todo lists are forged. It's rather easy, and HYPERLIST[S] can accommodate any level of complexity.

Let's go through the various parts of HYPERLIST[S] and all its possibilities.

3 HyperList^S

A HyperList^S consists of one or more HyperList^S Items.

4 HyperList^S Items

A HyperList^S Item is a line in a HyperList^S . It can have "children". Children are HyperList^S Items indented to the right below the Item.

A HyperList^S Item consists of an optional "Starter", an optional "Type", "Content" and a "Separator" in that sequence. All the various parts of a HyperList^S Item are described below and in the sequence they appear in an Item, as outlined here:

```
4.1 Starter
4.2 Type
4.3 Content
   4.3.1 Element
      4.3.1.1 Operator
      4.3.1.2 Qualifier
      4.3.1.3 Property
      4.3.1.4 Description
   4.3.2 Additive
      4.3.2.1 Reference
      4.3.2.2 Tag
      4.3.2.3 Comment
      4.3.2.4 Quote
      4.3.2.5 Change Markup
4.4 Separator
```

Look familiar? Yes, it is part of the table of contents at the beginning of this article – even that is a valid HyperList^S . If you are reading this document as a PDF file, you may use the table of contents to jump directly to any part of the document, as it consists of clickable links.

4.1 Starter

A HyperList^S Item may begin with a "Starter". A "Starter" can be either an "Identifier" or a "Multi-line Indicator".

An Identifier is a unique indicator that can be used in referring to that Item. A numbering scheme such as X.Y.Z can be used, e.g. the first Item in a HyperList^S would be 1. The second Item would be 2, etc. A child to the second Item would be 2.1 and the second child of 2 would be identified as 2.2, whereas the child of 2.2 would be 2.2.1.

A shorter form consisting of mixing numbers and letters can also be used, such as 1A1A for the first fourth-level Item. The next fourth-level Item would be 1A1B. When using this scheme, there is no need for any periods. The Identifier "21T2AD" would be equivalent to "21.20.2.30", saving 4 characters.

An Item that spans more than one line must have a Starter. It does not have to be an Identifier. You may use a "Multi-line Indicator" instead; just prefix the Item with a plus sign ("+"), to show that it spans more than one line. The second line of an Item will be indented to the same level/indent as the first with an added space in front. If you use a Starter on one Item, then all the Items in that same group of Items on the same level/indent must also have a Starter.

In the example below, the first child begins with an Identifier and the second a Multi-line Indicator. A Multi-line Indicator can also be used for single-line Items when other Items on the same level span more than one line and thus require a Multi-line Indicator.

```
Multi-line Indicator = "+"
   1 Following lines are of the same indent with a "space" before
    the text
   + If one Item on a certain level/indent is multi-line, all Items
    on the same level/indent must start with a plus sign ("+") or <Identifier>
```

The angle brackets near the end will be discussed later in this article.

4.2 Type

If it is not obvious or for clarity or strictness, prefix an Item with "S:" if the Item is a static or a state Item (i.e. something which does not denote action). Use "T:" for a transition Item (an Item indicating action). Alternately, you may use "|" instead of "S:", and "/" instead of "T:". The Type indicator comes after the optional Starter.

Children of a certain Type (either state or transition) inherit their parent's Type unless otherwise specified.

4.3 Content

A HYPERLIST[S] Item must have some sort of Content. The Content can be an "Element" and/or an "Additive".

4.3.1 Element

An Element is either an "Operator", a "Qualifier", a "Property" or a "Description". Let's treat each of these concepts.

4.3.1.1 OPERATOR

An Operator is anything that operates on an Item or a set of Items. It can be any of the usual logical operators[27]. It can also be other Operators, such as "EXAMPLE: ", "EXAMPLES: ", "CHOOSE: ", "ONE OF THESE: ", "IMPLIES: ", "CONTINUOUS: ", etc. The Operator "CONTINUOUS: " makes an Item or set of Items run continuously. An Operator is written in capital letters and ends in a colon and a space.

The Operator "ENCRYPTION: " indicates that the sub-Item(s) are to be encrypted or that the following block is encrypted. The encrypted block can be properly indented in the HYPERLIST[S] , or if indentation is part of the encrypted block, it will be left justified. This is the only Item that is allowed to break the indentation rules.

If you need to include one or more lines inside a HYPERLIST[S] that include HYPERLIST[S] expressions you don't want to be interpreted as HYPERLIST[S] expressions, then use a "literal block". This is equivalent to "pre-formatted" text in HTML and some word processors. To include such a block of lines, simply mark the start and end of the literal block with a single backslash ("\") on a line.

Example:

```
\
This is a block of literal text...
Where nothing counts as HyperList markup
Thus - neither this: [?] nor THIS: <Test> - are seen as markup
...until we end this block with...
\
```

Literal blocks are useful when you want to include, for instance, a block of programming code in the middle of a HYPERLIST[S] .

4.3.1.2 QUALIFIER

A Qualifier does as its name suggests; it qualifies an Item. A Qualifier limits the use of an Item. It tells you how many times, at what times and/or under what conditions an Item is to be executed, exists or is valid. When an Item is to be included only if several conditions are met, you put all the conditions into square brackets and separate them by commas. If an Item is to be executed first for one condition, then for another and then for a third, etc, you separate them by periods. The usage is best described by a few examples:

- Do Item if "the mail has arrived" = "[The mail has arrived]"
- Do Item 3 times = "[3]"
- Do Item 1 or more times = "[1+]"
- Do Item 2 to 4 times = "[2..4]" (the user may choose 2, 3 or 4 times)
- Do Item 2 times while "foo=true" = "[2, foo=true]"
- Do Item from 3 to 5 times while "bar=false" = "[3..5, bar=false]"
- Do Item less than 4 times only while "zoo=0" = "[<4, zoo=0]"
- Do Item 1 or more times while "Bob is polite" = "[1+, Bob is polite]"

[27] Logical connectives: http://en.wikipedia.org/wiki/Logical_connective#Natural_language

- Do Item a minimum 2 and a maximum 7 of times while it rains and temperature is less than 5 degrees Celsius = "[2..7, Raining, Temperature <5°C]"
- Do Item in the context of "apples", then "oranges", then "grapes" = "[Apples. Oranges. Grapes]". With this you can reuse a procedure in many contexts.

You can add a question mark before the qualifier statement to make it clearer that it is an "if" statement. These are equivalent:

```
[? Raining] Bring unbrella
[Raining] Bring unbrella
```

To indicate that an Item is optional, use "[?]". Example:

```
Receive calls at reception
   Pick up the phone and greet the caller
   [Caller does not ask for any specific person]
      Ask who the caller wants to speak to
   [?] Ask the reason for wanting to talk to that person
   [Person available] Transfer call; END
   [Person not available] Ask if you can leave a message
      Take the message
      Send the message to the person
```

The special word "END" and the semicolon in the above example will be explained later, under "Reference" and "Separator", respectively.

Use a "timestamp" to indicate that an Item is to be done at a certain time, in a certain time span, before or after a certain time, etc.

A timestamp has the format "YYYY-MM-DD hhmmss", conforming to the standard ISO-8601[28]. The time/date format can be shortened to the appropriate time granularity, such as "YYYY-MM-DD hhmm", "YYYY-MM-DD hh" or "YYYY". One can add a timestamp Qualifier such as "[Time = 2012-12-24 17]" or simply "[2012-12-24 17]".

Timestamps that represent an amount of time may be relative, such as:

- Length of time to wait before doing the Item = "[+YYYY-MM-DD]"
- Less than a certain length of time after previous Item = "[<+YYYY-MM-DD]"
- More than a certain length of time after previous Item = "[>+YYYY-MM-DD]"
- Length of time to wait before doing next Item = "[-YYYY-MM-DD]"
- Less than a certain length of time before next Item = "[<-YYYY-MM-DD]"
- More than a certain length of time before next Item = "[>-YYYY-MM-DD]"
- Length of time to wait after doing referenced Item = "[+YYYY-MM-DD<OtherItem>]"

The last example introduces a new concept, the "Reference". References will be discussed below.
Other obvious timestamps may be used, such as:

- "[+1 week]"
- "[-2 Martian years]"

Some practical examples:

- Wait one month before doing the Item = "[+YYYY-01-DD]"
- Do Item less than 4 days before next Item = "[<-YYYY-MM-04]"
- Wait one year and two days after Item X = "[+0001-00-02<X>]"

It is also possible to have recurring Items in HyperLists. The strict format is "YYYY-MM-DD+X DAY hh.mm+Y - YYYY-MM-DD hh.mm". The first date marks the starting date and the last date marks the end of the repetition interval. The "+X" is the repetition (i.e. the number of days between each repetition) in relation to the date, while the "+Y" is the repetition of the time (the amount of time between each repetition). "DAY" represents the name of the day(s) in the week where recurring Item occurs. You use what you need, i.e. if there is no repetition within a day, obviously the "+Y" would be skipped. Some examples:

[28] The international standard for dates and times, ISO 8601: http://en.wikipedia.org/wiki/ISO_8601

- "[2012-05-01+7 13.00]" = 2011-05-01 1pm, repeated every 7 days
- "[2012-05-01+2,3,2]" = Every 2, then 3, then 2 days, in repetition
- "[2012-05-01+2 - 2012-05-01]" = Every second day for one year
- "[2012-05-01 13.00+1]" = 2011-05-01 1pm, repeated every hour
- "[2012-05-01 Fri,Sat - 2011-10-01]" = Every Fri & Sat in the repetition interval

You can also use all possible intuitive variations by leaving certain parts of the timestamp unspecified.

- "[YYYY-MM-03]" = Every third of every month
- "[YYYY-12-DD]" = Every day in every December
- "[2011-MM-05]" = The fifth of every month of 2011
- "[Tue,Fri 12.00]" = Noon every Tuesday and Friday

Here is a complex Qualifier example:

```
[+YYYY-MM-DD 02.30, Button color = Red, 4, ?] Push button
```

The above statement reads "*2 hours and 30 minutes after previous Item, if the color of the button is red, then push the button 4 times, if you want to*".

If you use HYPERLISTS as a todo-list manager or project management tool, there is a nifty way of showing Items to be done and Items done; simply add "[_]" at the beginning of the line for an "unchecked" Item and "[x]" for a "checked" Item. An unchecked Item is to be done and is indicated by this "placeholder" Qualifier, while a checked item indicates the Item is not to be done anymore.

You may add a timestamp for the completion after a checked Item ("[x] YYYY-MM-DD hh.mm:"). In this way, you combine the Qualifier with a timestamp Property. The timestamp as a Property does not limit when the Item is to be done. It supplies information about when it was done.

4.3.1.3 PROPERTY

A Property is any attribute describing the Content. It ends in a colon and a space.

Examples of Properties could be: "Location = Someplace:", "Color = Green:", "Strength = Medium:" and "In Norway:". Anything that gives additional information or description to the Content.

4.3.1.4 DESCRIPTION

The Description is the main body, the "meat" of the Item. Although an Item may not have a Description, such as a line containing only the Operator "OR:", most Items do have a Description. Many Items have only a Description, such as those in a simple todo list.

4.3.2 Additive

Additives can be used alone or in combination with Elements. An Additive can either be a "Reference", a "Comment", a "Quote" or a "Change Markup".

4.3.2.1 REFERENCE

A reference is enclosed in angle brackets ("<>"). A reference can be the name of an Item, another HYPERLISTS or anything else. An example would be a Reference to a website, such as <http://www.isene.com/>, a file, <file:/path/to/filename>, or another item <The referenced item's Description>. It can even be a relative reference consisting only of a positive or negative number (the Reference <+4>would mean 4 items below).

There are two types of References:

1. A redirection or hard Reference
2. A soft Reference

An Item consisting only of a Reference is a redirection. For a transition Item this means one would jump to the referenced Item and continue execution from there. If the redirect is to jump back after executing the referenced Item (and its children), then add another set og angle brackets, such as <<Reference>>. This makes it easy to create more compact HYPERLISTS by adding a set of frequently used subroutines at the end of the list. For a state Item, a Reference means one would include the referenced Item (and its children) at the Reference point.

There are two special redirections for transition Items:

- An Item consisting only of the key word “SKIP” ends the current HYPERLISTS level
- An Item consisting only of the key word “END” ends the whole HYPERLISTS

If the Reference is only part of an Item, it is a “soft Reference”. It indicates that one would look for more information at the referenced Item. An even softer Reference would be to put the Reference in parentheses, such as (<Reference>), indicating that the referenced Item is only something apropos. Parentheses are used for Comments, which will be discussed later.

A Reference can be to any place in the list, up or down, or to another list, or to a file, a web address or any other place that can be referred to with a unique name used as the Reference. Although the Reference is valid as long as it is unique and therefore unambiguous, you should note the following best practice for a Reference:

If you reference an Item higher up in the HYPERLISTS , a simple reference to the Item is all that is needed. One would refer to the Identifier or to the appropriate Content, usually the Description. Examples:

```
The first Item
   A child Item
      A grandchild
         <A child Item>
```

It would make sense to use an Identifier for an Item if the Description is long and you want to refer to that Item:

```
The first Item
   1 A child Item
      A grandchild
         <1>
```

If you refer to an Item further down the hierarchy, you would use a forward slash (“/”) to separate each level (like the “path” used in a URL):

```
The first Item
   A child Item (<A grandchild/A baby>)
      A grandchild
         A baby
```

If you want to refer to an Item where you first need to “climb the tree” and then go down another “branch”, you start the path with the highest common level and reference the path from there:

```
The first Item
   A child Item
      A grandchild
Another branch
   A leaf
      <The first Item/A child Item/A grandchild>
```

Or, if the referenced Item has a unique Identifier, simply use that:

```
The first Item
   A child Item
      1 A grandchild
Another branch
   A leaf
      <1>
```

You may use a unique concatenation of a path to shorten it, such as <Somewhere higher up/One lev.../Ref...>. The three periods indicate concatenation.

4.3.2.2 TAG

A Tag is a “marker” for an item. It tags an item and is used just like hashtags in Twitter. Examples: #TODO #RememberThis #First #SandraLyng. No spaces are allowed in a tag.

4.3.2.3 COMMENT

Anything within parentheses is a Comment. Comments are are not executed as HYPERLISTS commands – i.e. in a list of transition Items, they are not actions to be executed.

4.3.2.4 QUOTE

Anything in quotation marks is a Quote. Like a Comment, a Quote is not executed as a HYPERLISTS command.

4.3.2.5 CHANGE MARKUP

When working with HYPERLISTS , especially on paper, there may come a need to mark deletion of Items or to show where an Item should be moved to. To accommodate this need, Change Markup is introduced. Change markup is a special type of Tag.

If "##<" is added at the end of an Item, it is slated for deletion. To show that an Item should be moved, add "##>" at the end followed by a Reference showing to which Item it should be moved below.

To indent an Item to the left, add "##<-". Indenting an Item to the right is marked by "##->". It is possible to combine moving and indenting an Item, i.e. moving an Item and making it a child of another: "##><Ref>##->".

To signify that an Item has been changed from a previous version of a HYPERLISTS , prefix the Item with "##Text##". Inside the hash signs you may include information about the change done, e.g. "##John 2019-03-21##" to show that the Item was changed by a certain person at a given time.

4.4 Separator

A Separator separates one Item from another. A line in a HYPERLISTS is usually one Item, and Items are then usually separated by a "newline" (hitting the "Enter" on the keyboard). But it is possible to string several Items together on one line by separating them with semicolons.

By separating an Item by a newline and then indenting it to the right, you create a child Item. A child adds information to its parent.

A Separator between two Items with the same or less indent is normally read as "then". If a parent Item contains a description, the newline Separator and indent to the right (a child) reads "with" or "consists of". If a parent Item does not contain a Description, the Separator and indent to the right (a child) is read as "applies to", and a Separator between the children is read as "and". A few examples should suffice:

```
A kitchen
   Stove
   Table
      Plates
      Knives
      Forks
```

This would read: "A kitchen with stove and table with plates, knives and forks".

```
Time = 2010:
   Olympic Games
   Soccer world championship
```

This would read: "Time = 2010: applies to: Olympic Games and Soccer world championship".

```
Walk the dog
   Check the weather
      [Rain] AND/OR:
         Get rain coat
         Get umbrella
      Dress for the temperature
   Get chain
```

And this would read: "Walk the dog consists of Check the weather consists of: If rain, AND/OR: applies to children: Get rain coat, Get umbrella; then Dress for the temperature, then Get chain". Or more humanly: "Walk the dog consists of check the weather, which consists of either/or get the rain coat and get umbrella. Then dress for the temperature and then get the chain."

Now consider these four examples:

```
Production
   Station 1
      Assemble parts A-G
   Station 2
      Assemble parts H-R
   Station 3
      Assemble parts S-Z


Production
   Station 1; Assemble parts A-G
   Station 2; Assemble parts H-R
   Station 3; Assemble parts S-Z


Production
   [Station 1]
      Assemble parts A-G
   [Station 2]
      Assemble parts H-R
   [Station 3]
      Assemble parts S-Z


Production
   [Station 1] Assemble parts A-G
   [Station 2] Assemble parts H-R
   [Station 3] Assemble parts S-Z
```

The first and second examples say exactly the same thing, but the second example is more efficient as it uses the semicolon as a Separator instead of using a line break and indent. The same goes for the third and the fourth examples – they are equivalent. But how about the difference between the first two examples and the last two?

When you use a Qualifier as in examples three and four, you accommodate for the tasks getting done in any chosen sequence. In those two examples, the production *could* go, for example, “Station 1”, then “Station 3”, then “Station 2”.

The first and second examples read: “The production consists of: Station 1 assembles parts A-G, then Station 2 assembles parts H-R and finally Station 3 assembles parts S-Z”

The third and fourth examples read: “The production consists of: if or when at Station 1, assemble parts A-G, if or when at Station 2, assemble parts H-R, and if or when at Station 3, assemble parts S-Z”

As you can see, there is a subtle but distinct difference.

5 A self-defining system

Now that we have covered all the possibilities in a HYPERLISTS , it should be obvious that HYPERLISTS could extend into a vast array of descriptions. It is even possible to write a parser or compiler for HYPERLISTS and use it as a programming language.

Is it possible to compact the descriptions above into a HYPERLISTS ? Yes, indeed. The HYPERLISTS system is self-describing.

HYPERLISTS can be done in many different ways – from informal to strict and from high-level to very detailed. The power of the tool resides with the user.

To illustrate the flexibility of HYPERLISTS , we will show the definition of this methodology in three steps – from a very informal and high-level description, to a more strict but still high-level – to the fully-detailed definition of HYPERLISTS .

First, here is a simple list showing what HYPERLISTS is all about:

```
HyperList Item parts (in sequence):
   Starter (optional)
      Identifier or Multi-line Indicator
   Type (optional)
      State or transition
   Content (can be an Element and/or an Additive)
      Element
         Either an Operator, Qualifier, Property, or Description
      Additive
         Either a Reference, Tag, Comment, Quote, or Change Markup
   Separator
      Newline or semicolon
```

Now, that's a very simple HYPERLISTS . Let's make the same simple list more strict by using the system concisely:

```
HyperList Item parts
   [?] Starter; OR:
      Identifier
      Multi-line Indicator
   [?] Type; OR:
      State
      Transition
   Content; AND/OR:
      Element; AND/OR:
         Operator
         Qualifier
         Property
         Description
      Additive; AND/OR:
         Reference
         Tag
         Comment
         Quote
         Change Markup
   Separator; OR:
      Newline
      Semicolon
```

That wasn't so intricate either, and that list is more easily parsable by a computer program if one wanted to automatically create graphical representation.

How about the full and complete definition, including all imaginable details of how any HYPERLISTS could conceivably be structured. Fasten your seat belt.

The following list shows the legal structure and syntax of a HYPERLISTS . It covers all you have read above. Note: green denotes Qualifiers or semicolon Separators, blue for Operators, purple for Starters and References, and turquoise for Comments and Quotes. (For HYPERLISTS that contain Properties, red would be the color denoting these, while yellow or orange is used for Tags.)

```
HyperList
   [1+] HyperList Item
      [?] Starter; OR:
         Identifier (Numbers: Format = "1.1.1.1", Mixed: Format = "1A1A")
            [? Multi-line Item] The Identifier acts like the plus sign ("+")
         Multi-line Indicator = "+"
            + If one Item on a certain indent is multi-line, all Items on the same indent
             (including single-line Items) must start with a plus sign ("+") or <Identifier>
             and all lines on the same indent after the first line must start with a space
      [?] Type
         OR:
            State = "S:" or "|"
            Transition = "T:" or "/"
         Children inherit Type from parent unless marked with different Type
         Can be skipped when the Item is obviously a state or transition
      Content; AND/OR:
         Element; AND/OR:
            Operator
               Anything operating on an Item or a set of Items
                  [? Set of Items] Items are indented below the Operator
               Can be any of the usual logical operators
               Is written in capitals ending in a colon and a space
                  EXAMPLES: "AND: ", "OR: ", "AND/OR: ", "NOT: ", "IMPLIES: "
               Can contain a Comment to specify the Operator
                  EXAMPLE: "OR(PRIORITY): "
                     Sub-Items are to be chosen in the order of priority as listed
               To make the Item run continuously, use "CONTINUOUS: "
                  Item is done concurrent with remaining Items
                  The Operator can be combined with a timestamp Property
                     EXAMPLE: "CONTINUOUS: YYYY-MM-07:" = Do the Item weekly
               To show that an Item is encrypted or is to be encrypted, use "ENCRYPTION: "
                  OR:
                      The encrypted block can be correctly indented
                      The encrypted block contains indentation and is left justified
                           This would seem to break the list, but is allowed
               A block can be included of "literal text" negating any HyperList markup
                  Use a HyperList Item containing only the Operator "\" to mark start/end
                     EXAMPLE:
                        \
                        This is a block of literal text...
                        Where nothing counts as HyperList markup
                        Thus - neither this: [?] nor THIS: <Test> - are seen as markup
                        ...until we end this block with...
                        \
            Qualifier
               Any statement in square brackets that qualifies an Item
               Specifies under what conditions an Item is to be executed, exists or is valid
               Several Qualifiers can be strung together, separated by commas
                  All Qualifiers need to be fulfilled for the Item to be valid
                  EXAMPLE: "[+YYYY-MM-DD 02.30, Button color = Red, 4, ?] Push button"
               Successive Qualifiers can be strung together, separated by periods; EXAMPLE:
                  "[Apples. Oranges. Grapes]"
                     Do Item in the context of "apples", then "oranges", then "grapes"
               EXAMPLES:
                  Do Item 3 times = "[3]"
```

```
            Do Item if "the mail has arrived" = "[The mail has arrived]"
            Do Item 2 times while "foo=true" = "[2, foo=true]"
            Do Item from 3 to 5 times while "bar=false" = "[3..5, bar=false]"
            Do Item 1 or more times = "[1+]"
            Do Item 1 or more times while "Bob is polite" = "[1+, Bob is polite]"
            Do Item up to 4 times only while "zoo=0" = "[<4, zoo=0]"
         Optional Item = "[?]"
         Timestamp Qualifier = "[YYYY-MM-DD hh.mm.ss]"
            Shorten the format to the appropriate granularity
         Time relations
            Length of time to wait before doing the Item = "[+YYYY-MM-DD]"
            Less than a certain length of time after previous Item = "[<+YYYY-MM-DD]"
            More than a certain length of time after previous Item = "[>+YYYY-MM-DD]"
            Length of time to wait  before doing next Item = "[-YYYY-MM-DD]"
            Less than a certain length of time before next Item = "[<-YYYY-MM-DD]"
            More than a certain length of time before next Item = "[>-YYYY-MM-DD]"
            Length of time to wait after doing referenced Item = "[+YYYY-MM-DD<Item>]"
            Other obvious time indicators may be used; EXAMPLES:
               "[+1 week]"
               "[-2 Martian years]"
            EXAMPLES:
               Wait one month before doing the Item = "[+YYYY-01-DD]"
               Do Item less than 4 days before next Item = "[<-YYYY-MM-04]"
               Wait one year and two days after Item X = "[+0001-00-02<X>]"
         Time repetition
            Obvious/intuitive repetition
               EXAMPLES:
                  "[YYYY-MM-03]" = The third of every month
                  "[YYYY-12-DD]" = Every day in every December
                  "[2011-MM-05]" = The fifth of every month of 2011
                  "[Tue,Fri 12.00]" = Noon every Tuesday and Friday
            Strict convention
               Format = YYYY-MM-DD+X Day hh.mm+Y - YYYY-MM-DD hh.mm; EXAMPLES:
                  "[2011-05-01+7 13.00]" = 2011-05-01 1pm, repeated every 7 days
                  "[2011-05-01+2,3,2]" = Every 2, then 3, then 2 days, in repetition
                  "[2011-05-01+2 - 2012-05-01]" = Every second day for one year
                  "[2011-05-01 13.00+1]" = 2011-05-01 1pm, repeated every hour
                  "[2011-05-01 Fri,Sat - 2011-10-01]" = Every Fri & Sat in time interval
         Checking off Items
            Unchecked Item = "[_]"
            Checked Item = "[x]"
               [?] Timestamp Property after ("[x] YYYY-MM-DD hh.mm:ss:")
      Property
         Any attribute to the <Content>, ending in a colon and a space
            Gives additional information or description to the Item
         EXAMPLES:
            "Location = Someplace:", "Color = Green:", "Strength = Medium:" and "In Norway:"
      Description
         The main body of the HyperList Item, the "meat" of the line
   Additive; AND/OR:
      Reference
      + An Item name or Identifier, list name or anything else
       enclosed in angle brackets ("<>"); EXAMPLES:
            Reference to a website = "<http://www.isene.com/>"
            Reference to a file = "<file:/path/to/filename>"
            A relative Reference to the seventh item below the current = "<+7>"
```

```
   There are two types of References; OR:
      Redirection (hard Reference)
         An Item consisting only of a Reference is a redirection
            For a transition Item = Jump to referenced Item and execute
               + If the redirect is to jump back after executing
                 the referenced Item (and its children), then add another
                 set of angle brackets (<<Referenced Item>>)
                   + EXAMPLE: Use this when creating subroutines at
                     the end of the list
            For a state Item = Include the referenced Item
         An Item consisting only of the key word "SKIP"
            Ends the current HyperList level
         An Item consisting only of the key word "END"
            Ends the whole HyperList
      Soft Reference
         Reference is part of an Item
            Look at referenced Item for info only
         Even softer Reference = Have the Reference in parentheses
            An Item that is only something apropos
   + For an Item upward in the HyperList, a simple Reference to the
    Item's <Content> is all that is needed
   + A Reference containing several levels down a HyperList needs a "/"
    to separate each level, like a "path" (as with a URL) to the Item
   + To make a Reference to a different branch in a HyperList,
    start the Reference from the highest common level on the list
    and include all Items down to the referenced Item
      EXAMPLE: Reference from here to <Hyperlist Item/Starter/Identifier>
   + For long Items in a Reference, concatenation can be used
      The concatenation must be unique
         EXAMPLE: Reference from here to <Additive/Comment/Anything...>
Tag
   A hash sign followed by any letters or numbers, used as a marker (#Tagged)
   Is not executed as a HyperList command
Comment
   Anything within parentheses is a Comment
   Is not executed as a HyperList command
Quote
   Anything in quotation marks is a Quote
   Is not executed as a HyperList command
Change Markup; OR:
   Deletion
      Remove the Item by adding "##<" at the end of the Item
   Motion; OPTIONS:
      Move the Item by adding "##><Reference>"
         This moves the Item just below the referenced Item
      Move the Item one level in by adding "##<-" at the end of the Item
      Move the Item one level out by adding "##->" at the end of the Item
         EXAMPLE: Move an Item as a child to referenced Item = "##><Reference>##->"
   Changed Item
      Prefix an Item with "##Text##" to signify that a change has been made
         Add information inside the hash signs as appropriate
            EXAMPLE: To show who changed it and when = "##John 2012-03-21##"
```

```
Separator
   OR:
      Semicolon
         A semicolon is used to separate two HyperList Items on the same line
      Newline
         Used to add another Item on the same level
         Indent
            A Tab or an asterisk ("*")
            Used to add a child
               A child adds information to its parent
               A child is another regular <HyperList Item>
   Definition
      A Separator and the same or less indent normally reads "then:"
      [? Parent contains <Description>]
         The Separator and right indent reads "with:" or "consists of:"
      [? NOT: Parent contains <Description>]
         The Separator and right indent reads "applies to:"
         A Separator between the children reads "and:"
```

Read and re-read the HYPERLISTS above and you will be a HYPERLISTS master after a while. It's all there.
The colors are for clarification and are the same colors used in a VIM script to help you create and manage HYPERLISTS .

6 Notes on style

HYPERLISTS only dictates structure. The system is not concerned with style, such as colors or font types. You may use any coloring, font type, bold face or italics to emphasize the content of a list. The colors used in the full definition list above are merely an example. You may also write an Item description in capital letters for emphasis.

HYPERLISTS are most readable when you use fixed-width fonts (fonts where all letters have the same width), such as in all the examples in this document. But you may instead use a proportional font (like the one you are reading now, where the "m" is much wider than the "i").

You may use an indent of any width, as long as you make sure the HYPERLISTS is readable. For fixed-width fonts, an indent (tab) of three spaces is usually the best, but for HYPERLISTS consisting of many levels, you may prefer an indent of two spaces so as not to make the list too wide.

7 Tools

If you use VIM[29] as an editor, you can use my VIM plugin[30] for creating and viewing HYPERLISTS . It highlights HYPERLISTS with the colors shown above. Tabs (or asterisks, for compatibility with EMACS[31] outline mode and the Mediawiki[32] TreeAndMenu[33] extension) are used for indenting. The plugin offers simple collapsing of lists or parts of lists, reference jumping, auto encryption/decryption, "presentation mode" and much more.

There is also HyperGraph - a tool to automatically graph HYPERLISTS . This is a command line script written in Ruby that will graph your HYPERLISTS as a state (like a mindmap) or transition (like a flow chart) in several graphics formats (PNG, JPG, GIF, SVG, PS or FIG). For a list of the many options it provides, run "hypergraph -h". You will find HyperGraph on the official HYPERLISTS page: http://isene.me/hyperlist/.

[29] The VIM text editor: http://www.vim.org
[30] The HYPERLISTS plugin for VIM: http://vim.sourceforge.net/scripts/script.php?script_id=4006
[31] The GNU EMACS text editor: http://www.gnu.org/software/emacs/
[32] MediaWiki, the software used for Wikipedia: http://www.mediawiki.org/wiki/MediaWiki
[33] TreeAndMenu, a MediaWiki plugin: http://www.mediawiki.org/wiki/Extension:TreeAndMenu

Notes

[1]The "HyperList to the power of S" signifies the iterative or fractal nature of such lists.

The name represents both singular and plural of the word – both the system and a list or lists conforming to this system.

[2]IT Service Operations is part of the ITIL best practice framework,

see http://www.itil-officialsite.com/AboutITIL/WhatisITIL.aspx

and http://www.itframeworks.org/wiki/Category:Service_Operation

[3]IT Service Transition: http://www.itframeworks.org/wiki/Category:Service_Transition

[4]Flowchart: http://en.wikipedia.org/wiki/Flowchart

[5]Process maps: http://en.wikipedia.org/wiki/Process_Maps

[6]Request Fulfillment, an ITIL process: http://www.itframeworks.org/wiki/Category:Request_Fulfilment_Management

[7]Task list or Todo list: http://en.wikipedia.org/wiki/Todo_list#Task_list

[8]The website of the author: http://isene.com

[9]My article, "On Will": http://isene.com/onwill.pdf

[10]For Gödel's Incompleteness Theorems, see http://en.wikipedia.org/wiki/G%C3%B6del%27s_incompleteness_theorems,

and for the world's shortest proof, see http://blog.plover.com/math/Gdl-Smullyan.html

[11]Relations diagrams:

http://asq.org/learn-about-quality/new-management-planning-tools/overview/relations-diagram.html

[12]Venn diagrams: http://en.wikipedia.org/wiki/Venn_diagram

[13]Warnier-Orr diagrams: http://en.wikipedia.org/wiki/Warnier/Orr_diagramandhttp://varatek.com/warnierorr.html

[14]Unified Modeling Language (UML): http://en.wikipedia.org/wiki/Unified_Modeling_Language

[15]Sankey diagrams: http://en.wikipedia.org/wiki/Sankey_diagram

[16]Decision trees: http://en.wikipedia.org/wiki/Decision_tree

[17]Petri Net: http://en.wikipedia.org/wiki/Petri_netandhttp://www.petrinets.info/

[18]Organizational charts: http://en.wikipedia.org/wiki/Organizational_chart

[19]Mind maps: http://en.wikipedia.org/wiki/Mind_map

[20]Process Flow diagrams: http://en.wikipedia.org/wiki/Process_flow_diagram

[21]Feynman diagrams: http://en.wikipedia.org/wiki/Feynman_diagram

[22]Data Flow diagrams: http://en.wikipedia.org/wiki/Data_flow_diagram

[23]Concept maps: http://en.wikipedia.org/wiki/Concept_map

[24]OBASHI: http://en.wikipedia.org/wiki/OBASHI

[25]Turing completeness: http://esolangs.org/wiki/Turing-complete

[26]Egil Möeller's personal website: http://redhog.org/

[27]Logical connectives: http://en.wikipedia.org/wiki/Logical_connective#Natural_language

[28]The international standard for dates and times, ISO 8601: http://en.wikipedia.org/wiki/ISO_8601

[29]The VIM text editor: http://www.vim.org

[30]The HyperLists plugin for VIM: http://vim.sourceforge.net/scripts/script.php?script_id=4006

[31]The GNU EMACS text editor: http://www.gnu.org/software/emacs/

[32]MediaWiki, the software used for Wikipedia: http://www.mediawiki.org/wiki/MediaWiki

[33]TreeAndMenu, a MediaWiki plugin: http://www.mediawiki.org/wiki/Extension:TreeAndMenu