|  |
| --- |
| Circle Language Spec |

## Overview

This documentation is an attempt to describe a programming language called Circle. It aims to be a diagram notation for visualizing computer code.

The documentation tries to explain how coding concepts might be expressed in Circle language: concepts like objects, classes, relations, interfaces, events, commands and parameters.

The Circle language for now is a specification in writing. At the time of this writing, no finished program was made yet to use it. Only experiments were done.

The documentation tries to shed light on the following topics:

### Overview

This overview, which attempts to summarize the topics.

*Introduction*

Tries to gently introduce what this language is about.

*Basic Diagram Elements*

Attempts to give a more or less complete overview over the basic elements that might be seen in the diagrams, like circles, triangles, crosses, squares and lines. This documentation describes what they might mean in the diagrams. Later in the documentation details about the diagram notation could be covered.

### Objects

This documentation aims to describe what objects might be in computer technology and basic concepts about objects. It also tries to shed light on drawing out objects using the Circle diagram notation.

### Classes

This documentation attempts to demonstrate the concept of *classes*. A class is sort of like a special object, that aims to describe the characteristics and behavior of other objects. The other objects might select that special object as their class and the idea is that their behavior is then guided by the rules described by that class. The basic ideas may be explained, but the concept of *static* is intended to be worked at a later time.

### Relations

When an object connects to another object, it is like a relation between the two is established. This documentation tries to clarify the concept of relations. Even though objects might relate to an arbitrary set of other objects, the concept might also be about how *classes* set the guidelines for the kinds of relations objects might have. Relations between classes form a sort of *model* of relations for objects.

### System Objects

One possible application of how this language might run in practice, is one where systems may be composed of *system objects*. System objects could be the *actual* objects, that a system runs on. Maybe this is a specific use case of the language, but describing it, seems to have allowed more generally usable things to come to light. One of the intentions of system objects could be, to manage *relations* between objects. They might also control *system aspects*, such as **Class**, **Interface**, **Execution** and **Data**. Those aspects could be controlled through *system commands*. The story moves on to assignment commands. The idea is that assignment command copy an aspect from one object to another. The *System Objects* documentation alsotries to show connectors, connections and different notational forms to regulate the aspects.

### Commands

Commands could be seen as executable objects. Other synonyms for commands might be *methods*, *procedures*, *routines*, *subs* and *functions*. But in the Circle Language Spec those executable objects would be called *commands*. In some programming languages, commands might not be objects, but constructs, that are not exchangeable with objects. But in Circle an attempt is made to see them as objects, that happen to be executable. Commands do seem to have specific (creation) behavior. This documentation tries to evaluate this behavior of commands.

### Parameters

Parameters are like instructions passed along with a command that could make the command behave differently. This documentation tries to view parameters as a relationshipbetween commands and objects. It entertains an idea of how *command parameters* and *class methods* might be interchangeable, as if they might be one and the same concept. It is not sure, if this idea will still be in there in the future. Also a diagram notation of commands and parameters is suggested.

### Globality

(The made-up word 'globality' is based on the concept of 'global' from Visual Basic. But later the term *Modules* might used instead.)

A module is like a little world in which objects and commands live. A module might be a site, a computer program, a library, a name space or other kind of module. A module may group together functionality or data. A module might be thought of as an object with a special property: something that might be called *global* access to its sub-objects. This is another suggested description of that property: Anything directly inside the module can be directly referenced from anywhere within that module.

### Execution Flow

The idea of execution flow statements is that they may regulate the order in which things are executed. Several execution flow statements are proposed. The key examples might be **If** and **For** statements. An attempt is made to build up a complete set of execution flow statements, to explore what could be desired from the diagram notation for it.

### Black Boxes

It could be said, that *Black Boxes* have to do with *Public and Private*. *Private* might make something only accessible inside an object. *Public* couldmake something accessible outside an object as well. That way, the inner workings of an object may be hidden away and this might be a way to work with the input and output only.

The notation might be changed compared to what is described here. The current version tends to focus around something, that could be called the *friend* notation. This might be because the current design of the notation seems to be made with a bit of a different view on objects and commands. Commands were sort of considered more independent and not necessarily part of an object. Which commands might access the private contents of which objects might be less obvious, when a command would not necessarily be part of any particular object. The effect seemed to be, that commands might not get access to the private contents of an object, unless that command may be declared a *friend* bythe object. Friend declaration might create somewhat of a tighter bond between the command and the object. Thinking of commands as object, seems to have affected the direction into which the design black boxing went.

In the future, a different black box method might be suggested: more like it is in other programming languages. The friend notation might still be preserved for particular use-cases.

### Interfaces

This chapter attempts to describe the concept of interfaces along with their diagram notation. Up till now the idea of interfaces was sometimes left out of the stories. This chapter tries to catch up with that. There are also attemps to describe: interface assignment, interface reference and relations that might form through the use of interfaces. As interfaces might be applied to *objects*, interfaces might also be applied to *commands*.

### Events

Events might be thought of as notification calls. Call receivers might subscribe to an event. The call sender might send a call to the receivers at times. Another variation of the events concept, that the world seems to use, if not mistaken is the *‘the observer pattern’*. The term observer may help here, to realize what an event is for: watching out for something to take place, to possibly take action. The sender might decide what the message looks like. The receiver might supply a command, that should have an interface, given by the sender.

The concept of events might be seen as a combination of constructs: an array of command references, a command interface, then supporting the command interface, registering the command inside the array of command references and then calling the commands in the array at times.

Even though It might be established by a combination of other constructs, it seems to be of enough use, that Circle attempts to accommodate it as a separate construct, like some other programming languages seem to do.

The documentation also tries to show, how events look in Circle's diagram notation.

### Inheritance

This part of the documentation attempts to describe the concept of inheritance inside the Circle language. This concept seems to have been part of some object oriented programming languages for a while. In certain languages one class might inherit from another class, to take over its characteristics before the derived class might get extended or altered. When the notation in Circle was worked out, different variations on that seemed to fall out, that might not necessarily be present in all object oriented languages. Next to *class inheritance* (could be thought of as the *traditional* form of inheritance), there could also be *object inheritance*. This might mean, that one object takes over the characteristics of a specific other object. The other object seems to melt together with the new object. When maybe changing base members of the new object, the base object’s data might also change. This could be called *object inheritance*. More forms of inheritance may be proposed. A collection might be inherited from, so that an existing 'table' of items might be extended with 'new columns'. This could be called *list inheritance*. There is also the idea for a construct, where a *system* or modulecan inherit from another system, taking over its characteristics, and the derived system can be altered, augmented, specialized and extended. This might be called *system inheritance* or *module inheritance*. If the original system would change, this may also change the derived systems. (Other technology might establish this with what may be called 'proxy classes' and extending auto-generated proxy-classes with additional aspects.) So the idea is that things can be inherited from to render specialized versions of a class, object, collection or module.

Such specialization might also be applied to *commands*. The inheritancedocumentation also aims to describe otherconcepts in the area of *specialization* and *extension*. Inheritance might be thought of related to the concept of *interfaces.*

That was the more rounded updocumentation. Next an attempt is made to list the documents, that might not be as well polished. The descriptions below, however, might be a rounded reflection on the topics.

### Type Control

Objects might relate to free to choose sets of other objects. But when the class of an object is set, some relations might be determined by the class. This could be considered a form of *type control*. Another form of type control might be assigning an interface to an object reference. This may change the types of objects that might be assigned to the object reference.

Binding an object like that could be called *explicit* declaration. Not binding an object or object reference like that might be called *generic* declaration. Both concepts could be seen as *type control*.

This documentation folder might have one document, with ideas that are possibly not as well polished, including an original description worked out in a previous version of the design of this diagram notation language.

### Object Resolution

Object resolution might take place when a reference to one object is more or less obscured by another object, shadowing or overriding the original object. (It also might apply to commands.) A reference might be set to an object, but instead the reference might be sort of non-consentiously pulled towards another object. There may be multiple situations in which object resolution takes place, like: overriding, overloading, shadowing, ambiguity, implicit conversions and default members. At first it was thought, that these kinds of situations might be due to ambiguity and the textual nature of code, but in the Circle diagram notation when drawing out the situation in a diagram, it seems to fall into place, that they might be more like automatically detoured references.

This documentation might be less smooth. It may contain ideas but partly described, not as refined into polished up text.

### Conditions

An object might set rules for what kind of values may be filled in into it and a command might be given conditions on whether the procedure might be started or not. This might perhaps be done by adding code to a procedure (or for an object, to the **Set** command). From that code may follow, that depending on certain conditions, the execution of the rest of the procedure is or is not run. That just may have described a low-level way to handle conditions. But conditions could be centric enough in programming, that they might become a construct in the Circle diagram language. A (value) object or command might propose a set of conditions, to evaluate whether the value may be assigned or if the command is allowed to execute. The documentation may be not so rich yet, apart from some loose ideas.

*Object Order*

By default, objects inside a collection might be put in an arbitrary order. The *Object Order* concept may allow sorting a list, or putting things in a manually chosen order, and also perhaps a diagram notation for it. This documentation is a collection of ideas, that might be polished up to be made neater.

### Pointers

The intention could be to move pointer-to-pointer related issues from being in more than one place to this chapter. This may or may not have been done yet.

*Misc Diagram Topics*

It would be nice if diagrams could be automatically drawn out: if the metrics and positions of the shapes and lines could be automatically done. Suggestions could be found this documentation about aspects of positioning, that might make diagrams look tidier. Also described might be ideas about how colors may be used to express things.

This documentation might still be loose ideas from previous versions of descriptions of this Circle programming language.

### Summaries of Other Languages

Attempts to summarize other languages, mainly C and C++, in the form of bulleted lists. They were made around the year 2004 during an analysis of the grammar of C and C++ to compare it to this Circle programming language.

*Previous Version*

The previous version of a description of this programming language.

**Ideas**

*The texts below are loose ideas, yet to be turned into good documentation.*

*Other Programming Languages*

Looking at the 'Subtext' programming language

- 'Subtext' implies that this is always possible, but ofcourse underlying procedures could be

- 'Subtext' code when it grows is really bound to become unoverviewable.

- His assumption that humans are not good at abstract thinking is just bullshit. Humans do nothing else.

- If he would give his functions better identifiers, you would already be better off.

J Code

Analyse the features of F# and find a place for it in my system.

http://pro.tweakers.net/nieuws/49986/microsoft-stopt-functionele-taal-f-in-visual-studio.html

JJ

Caché bestrijkt vele gebieden al, waar Code een oplossing voor biedt. Het relationeel en object georienteerd unifyen, dan zit daar eigenlijk al bijna compleet in.

Afgeleide containment relaties niet, diagrammen niet en applicatie feature generatie niet, en aspecten niet (ik zal naast standaard aspecten van JJ, zorgen dan je customer concepten kunt introduceren.) extended inheritance niet, kiezen tussen geheugen en schrijf niet, speciale security niet.

Dat soort concepten niet, maar het hele idee relationeel en object georienteerd gelijk trekken wel. Behalve echt 1 taal maken van OO en SQL

Relaties ingaan in SQL statements in Cache, is precies de manier waarop ik het van plan was. But I don't think it takes relations and classes as the base of the system. It's either specify tables with foreign keys, or declare objects with sub objects in them. I don't know if the two counterparts of a relation are synchronized either.

Cache integreert wel al met bestaande grote standaarden, zoals ODBC, XML, SOAP. And it provides in importing data into it from other database systems.

Wat nog steeds wel origineel is, ook ten opzichte van Cache, is de visie om OO ondergeschikt te maken aan het relationele model. In OO - Relational oplossingen wordt in object-relational mapping, OO bovenop relationeel geplaatst. Chaché zet de OO benadering ook bovenaan, en de relationele benadering als handig alternatief.

*OO Versus Relational Database*

Ik zie ook beweringen op het internet, dat een OO data benadering geavanceerder is dan relationeel. Maar er zijn voordelen aan relationeel, waar dan niet meer over gesproken wordt.

Ik denk dat het ook een zaak is twee-kampen. Ik zie in beide methodes het licht.

Anderen vinden OO bijvoorbeeld het beste, en voegen hier relationeel aan toe, en andere mensen vinden relationeel het handigste, en voegen hier OO aan toe. Don’t want to consider

Je moet dus echt aansporen dingen in de objectstructuur te embedded en daar met recursie op te lossen, in plaats van EEN moeilijke procedure erlangs schrijven

*Other People’s Software*

Windows doesn't allow shortcuts with the same name in a folder even when the short cuts have different target types.

Oracle 8.0i

Zoek op in HTML for Dummies hoe je naar een 'bookmark' springt.

Ideas,

Ik heb het vermoeden, dat als we alle gegevensverwerkingen via webservices zouden laten lopen, dat de boel niet vooruit te branden is.

JJ

*Other*

Computer Language,

Referenties naar een copy functie wil je

ook niet in de in de copy command definitie zelf bijhouden.

Maar je zou wel de mogelijkheid willen hebben om te querien

welke kopieeracties er binnen een bepaald systeem zijn.

Je kunt altijd een ruwe sequentiele zoek-query uitvoeren op

een subsysteem.

Maar je wilt het misschien ook centraal bijhouden. Dan

zou je een filter index moeten kunnen maken,

maar een filter index gezet op een elders gedefinieerde

method of class.

Ik heb er toch best moeite mee, dat je

in een stuk diagram niet ziet wat er allemaal naar

een bepaald object verwijst, maar alleen waarnaar

de objecten in de diagram verwijzen.

O, wacht, dat gebeurt voor objecten wel, omdat

de gerelateerde objecten als sub objecten worden getoond.

Heen en weer relaties tussen objecten in principe gelijkwaardig.

Maar bij methods is het anders. Die hebben altijd een richting,

en de relatie terug is echt de backwards verwijzing.

Het is zeg maar een kwestie van 'belachelijk om allemaal bij te houden'.

Alleen soms wil je voor een definitie, die zijn referrers niet bijhoudt,

toch referrers bijhouden.

Eigenlijk moet dan een systeem de referrers naar een definitie van een

ander systeem bij kunnen houden.

Je maakt bij methods eigenlijk ook relaties tussen method definitions aan.

Die zouden dan ook referrers bij kunnen houden, en een gesynchroniseerde

relatie aan kunnen gaan.

JJ

Software System, General

An application of this new model, that is my software system, would be

for instance, that someone comes up with something that makes any lengthy

process pauzable, items in it skippable, etcetera. Just a handy concept

for handling lenghty processes.

A handy user could inherit an existing system and extend it with a new

concept, that was not applicable to the system yet.

I think in order for a system to be adaptable to new concepts, there

must be a set of standards to be upheld. Otherwise a program might be just

one big method, and that can not adopt new concepts.

So you have to impose design rules in order to make a program

adaptable to new concepts.

JJ

Software System

- Crap first

- Objects second

- Methods third

Not:

- Methods first

- Object second

- Crap third

JJ

Computer Language,

Windows Workflow raakvlakken met diagrammen.

JJ

Computer Language,

I saw something call speficying the structure of a program, rather than what happens step-by-step is called declarative programming, as opposed to imperative programming.

JJ

Computer Language,

I don't have ... for user program flow, for instance when using

multiple windows seemingly arbitrarily, but not...

And also not for workflow.

I don't know yet how workflow fits into the system.

Probably as internet threads.

JJ

Computer Language,

Ik wil gewoon dat workflow en methods die elkaar aanroepen gewoon hetzelfde concept zijn. Method stappen kunnen parallel lopen en seriele punten hebben en vertragingen en gezette tijden hebben.

JJ

Stereotyping

Perhaps another typing can be assigned to a relation, instead of containment. For instance: *ownership* or *usage*. Perhaps a few standard ones, and it may be possible to define your own typing by specifying a String.

JJ

Computer Language,

Dead links.

JJ

Collection,

In Collection kan je ook vanalles van verschillende modules van verschillende sites combineren, tot 1 home-page. Je hoeft dus niet te kiezen tussen sites, waar je een home-page op kunt maken. Je kunt zelf alles dat ze aanbieden met elkaar combineren, zonder al te veel heisa. Ook een leuk argument om het Collection te hoemen.

JJ

Computer Language,

Important statement to use: objects first, procedures second.

JJ

Computer Language,

People aren't realizing that a computer program IS a domain-specific language.

JJ

Computer Language,

Met diagrams in computer language wordt de systematiek echt zichtbaar.

Op het moment programmeren we eigenlijk allemaal blind en op de tast.

JJ

Computer Language,

The idea of object oriented is: objects first, procedures second.

JJ

Computer Language,

The diagram expression also makes it possible to

use hand signs in the air to draw out relations between objects,

in a technical story you try to explain.

JJ 2008-06-03

Computer Language,

De kracht van programmeertalen nu, zit hem niet in de taal zelf,

maar in het framework.

What makes programming languages today powerful,

is not the language itself, but the framework.

JJ

Computer Language,

2008-08-15

Other programming languages and diagram expressions:

http://www.obsolete.com/dug/sorcery/oop.htm

JJ

Computer Language,

Other programming languages and diagram expressions:

Leuke zoekterm om op te googlen:

"my own programming language"

JJ

Computer Language,

2008-08-15

Other programming languages and diagram expressions:

http://jolt-lang.org/

JJ

Single paradigm for all digital objects,

See the physical disk as an object.

See the IO on it,

and what source does IO on it.

Be able to see which reference has the most activity.

Be able to navigate through the system,

so that you can trace the source of the activity.

That way you not only fly through the internet,

and the applications, but you can also navigate through

the internal workings of your computer.

I guess I do want to see magnitude of activity

in the diagrams.

JJ

Computer Language,

2008-09-02

Some people seem to like isolated memory space.

But this is just for some protection that it gives.

Memory leaks only live inside a process.

When it is stopped, then the memory leaks

are released.

When something crashes, it is just that process, that crashes.

The rest keeps running.

But in my own system, the whole internet is really just one process.

So what needs to be done, is analyse the advantages

people think they can only get from isolated processes,

(advantages, that are probably only precautions for problems)

and see what solutions for those problems will be implemented

in my own system.

JJ

Computer Language,

2008-09-02

Google using a separate process for each tab in its

Google Chrome browser, is just a practical

solution, relatively easy to implement.

That's why process isolation offers a solution

to this problem. It is not, that with more effort,

a not-yet existing solution to this problem,

that does not use the isolation of processes,

would not offer a better or equally practical

solution, would products like that be finished

today and free to use.

JJ

Computer Language,

Ik ben me er wel van bewust,

dat een nieuwe programmeertaal,

of nieuwe manier van besturen van je computer,

een bijdrage is, niet een vervanging van alle

andere software. Daar zijn software giganten

veel te innovatief voor geworden.

JJ

Computer Language,

2008-08-23

The way it is right now, the system can not change the system as much as a user can change the system.

JJ

Computer Language Functional Design,

2008-08-31

Sometimes it is just clearer to have an article,

with diagrams in it straight away, without

any article with just textual explanations.

In the future, the whole form of the documentation

might change as such and have diagram expression

be more present in the conceptual explanations.

JJ

2008-11

The roles that symbols get when connecting them with lines.

JJ