CC - AN INTRODUCTION

Rieks Joosten
KPN Research

This document is an introduction to the 'Calculate with Concepts' (CC) method[1] for describing contexts. It does not detail the process (and its pitfalls) of how to come up with such descriptions, but the agile reader should be capable of making his first attempts to do so. This document is not intended to be a course in CC, but rather aims to give the reader an idea of what it is about, where it can be applied, and its promises for the future.

So What's The Problem?

The problem we try to get under control is that of (mis)communication between people, the root of which is that most people confuse syntax and semantics, i.e. language and meaning. Only few people realize that a person they talk to assigns (his own) meaning to the words he hears, and that this meaning can differ significantly from what the speaker tried to convey. For example, as I write this text, I realize that you, reader, in your attempt to understand what I try to convey, assign a meaning to these words that you think I try to convey.

As an illustration of this, and even stronger: as an illustration of people being capable of assigning inconsistent meaning to a single notion within 2 minutes, we have come up with a simple experiment. This experiment requires that the subject has a notion of what data networks and protocols are about. We asked the subjects to imagine a single network. Not multiple networks, just one. Then we asked the subjects how many protocols there are in the network. Basically, we got three answers:
1) there are no protocols in this network.
2) there are dozens (rather: more than one) protocols in the network.
3) there's only one protocol in the network.

While this already illustrates that people think differently about a relatively common technical notion such as 'networks', there is more to be learned. Let's examine the answers with a bit more detail.

1) There are no protocols in this network. This answer was given by a theoretician. Basically, he says that since he must think of one network, and no more, there is no equipment that would put data on the network, and hence no protocol appears on the network. While this answer might theoretically be sound, it is useless for all practical purposes. Still, it shows that there are people that think different things than you (or did you think of this answer too?) even when it is about something as 'normal' as networks.

2) There are multiple protocols in the network. This answer is given by most people (by far). Basically, they see a network cable, envisage what a sniffer would 'snif', and there you have it: dozens of protocols. However, if you subsequently ask them to identify where this single network begins, and where it ends, people get confused. At the IP-level, they would say that the network pretty much stretches around the world. But at the ATM or

[1] The method was originally invented by prof. S.M.M. Joosten, Ordina partner, founder of Anaxagoras Proces Architects (http://www.anaxagoras.com/), and professor at the Open University in Heerlen, The Netherlands.

Ethernet level, the network is only locally present. But hey, are we now all of a sudden talking multiple networks here (one at the IP-level, another at ATM/Ethernet level)? Here is where people experience their inconsistent/ambiguous notions of 'network'. They get confused, and need time to sort things out.

3) There is only one protocol in the network. This answer is given be the remaining (few) people. They usually say it's not the *right* answer, nor the 'truth'. It is one of the ways one can think about networks, and it has the advantage that you know where a given network begins and ends. And it comes at a cost: in order to think e.g. about an internet browser connected to a web-server, you need to have a means of stacking different networks on top of one another, of tunnelling, and so on.

The experiment thus shows that it is far from obvious that two people that talk to one another, even when they use the same words, actually talk about the same thing. This has dramatic consequences for ICT.

In ICT, it's common that processes (workflow, procedures, tasks) implement business needs, and they in turn are supported by systems (computers running software applications. The manual processes are executed by people. The nice thing is that people can think, and that means they have this great ability to recover from mistakes, even if they get complex. The automated processes and tasks (running on computers) are executed by unintelligent hardware: the hardware simply does as instructed - no more, no less. It's up to the process- and software engineers to write these instructions with such precision that when the hardware executes these instructions, everything 'goes as expected'. That we, people, can do this is wishful thinking, as process designers and software designers are only human, and hence cursed with the communication problem as demonstrated in the experiment. This not only results in flawed software, but also in continual exceeding of project deadlines, and the seemingly impossibility to fix data-models, the practical uselessness of corporate data-models, etc. Now imagine what thre is to win, even if we were to (only partly) solve this problem.

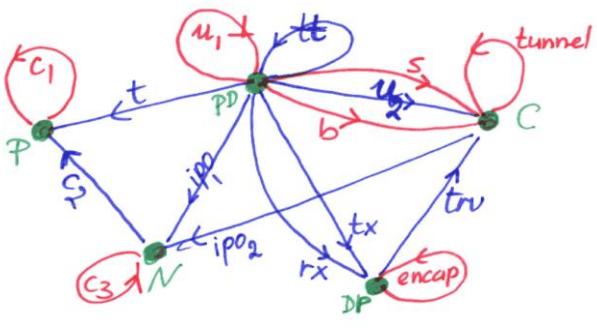
Is There a Solution?

Not that we know of. However, there's a technique that helps us to control the problem - the CC-method (where 'CC' stands for 'Calculate with Concepts'). Basically, the method helps us to find 'laws' in a context of concepts, and relations between them. This is quite similar to 'laws of physics' (like Maxwell's laws on electrodynamics/electrostatics, and Newton's laws on the movement of objects). These laws capture the essence of electrodynamics/electrostatics, or the movement of objects. And while these laws are not necessarily 'true' (Einstein, for instance, found that objects that travel at a speed that is near the speed of light, don't obey Newton's laws), they have a context in which they apply, and can be put to practical use. And that's why we agree to use them, *as if* they were true. Analogously, the CC-laws are not necessarily true, but the have a context in which they are relevant and useful - at least to the people that have agreed to these laws.

CC-Contexts

The idea of the CC-method is to get a good grasp on the communication problem between the individuals in a group of people as they talk about a given topic (context). The CC-method does so by establishing a 'context-language', describing the core-concepts and core-relations between such concepts. Core-concepts and core-relations only become part of the context-language after all people in the group have recognized them as being relevant to the context. The CC-method also aids in establishing the 'laws' that the whole group recognizes as 'truths' within the context.

We have done this for the topic (context) 'networks'. Our group of people came to agree on the relevance



of the following concepts (denoted as green dots in the adjacent figure):
- Network (N)
- Protocol (P)
- Network Adapter (PD)
- Connection (C)
- Data Packet (DP)

Note that we apparently talk about data networks. We could just as well have talked about analogous (voice) networks, electricity networks, railroad networks, etc.

The relations (blue and red arrows between the green dots) that we found relevant are best described in a table, with the colums as follows:
- Abbr. is the abbreviation of the relations name. The abbreviation appears in the figure.
- Src, Dst are the source and destination of the relation-arrow respectively
- Relation is the word that identifies the relation (i.e. the name of the relation-arrow)
- Description has the natural language sentence that relates the Src and Dst concepts.

Abbr.	Src	Dst	Relation	Description (natural language centence)
c1	P	P	carry	Protocol p1 carries protocol p2.
c2	N	P	carry	Network n carries protocol p.
c3	N	N	carry	Network n1 carries network n2.
t	PD	P	talk	Network adapter pd talks (speaks) protocol p.
ipo1	PD	N	is part of	Network adapter pd is part of network n.
ipo2	C	N	is part of	Connection c is part of network n.
rx	PD	DP	receive	Network adapter pd receives data packet dp.
tx	PD	DP	transmit	Network adapter pd transmits data packet dp.
encap	DP	DP	encapsulate	Data packet dp1 encapsulates data packet dp2.
trv	DP	C	traverse	Data packet dp traverses connection c.
s	PD	C	setup	Network adapter pd sets up connection c.
b	PD	C	breakdown	Network adapter pd breaks down connection c.
u2	PD	C	use	Network adapter pd uses connection c.
u1	PD	PD	use (call)	Network adapter pd1 uses (calls) network adapter pd2 to obtain a service that pd2 offers. Think of this as pd1 stacked on top of pd2.
tt	PD	PD	talk to	Network adapter pd1 talks to network adapter pd2. Think of this as pd1 talking to pd2 on the same network level.

These concepts and relations are what we agreed on as being relevant with respect to (data) networks. Note that if we have a common understanding of what the concept 'network' means, all we can say is that a network is anything that:
- may carry a protocol (relation c2)
- may carry a(nother) network (relation c3)
- may be carried by a(nother) network (relation c3)
- may have network adapters as parts of it (relation ipo1)
- may have connections as parts of it (relation ipo2)

So far there's nothing here that says e.g. how many protocols are allowed on the network, or whether or not a network can exist without a network adapter or a connection. Such restrictions are what gives 'meaning' to this context-language, and they are the 'laws' that we're after. Basically, there are two types of 'laws'; one that defines cardinalities (e.g. 'a network carries one and only one protocol'), and another that does not (e.g. 'all network adapters that are part of a network talk the protocol that is carried by that network'). Such laws don't exist for us to find; they are what the group decides to think of as truths. In the network case, we decided on the following laws (note that *all* laws *only* use concepts and relations as defined above):

Cardinality laws
- A network n carries one and only one protocol.
- A network adapter talks at least one protocol.
- A connection is part of one and only one network.

- A data packet is transmitted by one and only one network adapter.

General Networking laws:

- All network adapters that are part of a network talk the protocol that is carried by that network.
- If network adapter pd1 has talked to network adapter pd2, then both network adapters are part of the same network.
- If a network adapter has used a connection, then that network adapter and that connection are part of the same network.
- If a network adapter has broken down a connection, then it has also set up that connection.
- If connection is part of network, then the connection has been set up by a network adapter that is part of the same network.

Data Transport laws:

- A network adapter can only talk to a(nother) network adapter if they speak the same protocol.
- If a network adapter has received a data packet, then the network adapter has used the connection that the data packet has traversed.
- If network adapter pdtx has talked to network adapter pdrx, then pdtx has transmitted a data packet that was received by pdrx. Also, if pdtx has transmitted a data packet that was received by pdrx, then pdtx has talked to pdrx.
- If network adapter pdtx has talked to network adapter pdrx, then there is a connection that both pdtx and pdrx have used.

Network Stacking laws

- If network adapter pdh has setup a connection ch, then pdh has used a network adapter pdl that has used a connection cl which tunnels ch.
- If network adapter pdl, being part of network nl, has been used (called) by network adapter pdh which is part of network nh, we then say that network nl carries network nh.
- If network adapter pdh has used (called) network adapter pdl for transmitting a data packet dpl that encapsulates data packet dph, we then say that network adapter pdh has transmitted data packet dph.
- If network adapter pdh has used (called) network adapter pdl for receiving a data packet dpl that encapsulates data packet dph, we then say that network adapter pdh has received data packet dpl.
- If connection cl tunnels connection ch, then there are network adapters pdl and pdh such that pdl uses cl and pdh uses ch, while pdh uses (calls) pdl.
- If network adapter pdh talks protocol ph, and pdh uses (calls) network adapter pdl which talks protocol pl, we then say that protocol pl carries ph.
- If data packet dph traverses connection ch, and dph is encapsulated in data packet dpl which traverses connection cl, we then say that cl tunnels ch.
- If data packet dph is encapsulated by data packet dpl which traverses connection cl that tunnels connection ch, we then say that dph traverses ch.

There are more laws that you might want to agree on. For instance, if all people involved in the discussion only consider point-to-point communication over confidential networks (i.e. disregard things like multicasting and broadcasting), they might decide to add a law such as 'any data packet is received by one and only one protocol driver'.

As you may have noticed, the CC-context of 'networks' has a particular 'formal' taste to it, in particular the laws. While concepts and relations are easily expressed in natural language, the laws have a distinct logical structure. And with good reason, as we will see. Nevertheless, this does not necessarily prevent group members from talking about networks the way they were used to. However, if they get to disagree, they have this write-up to settle their disputes, in a similar fashion as physicists draw on the mathematics to settle disputes.

A common misconception of CC-graphs (such as in the figure above) is to assume it identical with Entity-Relation diagrams, or similar diagrams, as they appear e.g. in OO software development. The reason is that most software engineers are taught to think in objects and their attributes. There's nothing wrong with that, except when objects are confused with concepts. In CC, the decision of whether to map a concept onto an attribute, an object, or a

class is basically *derived* from the context language and the laws. That's why we explicitly distinguish between between object models (as in OO) and conceptual models (as in CC).

Mathematical Fundaments of CC-Contexts

You may skip this section if you're not interested - it just gives a bit of mathematical background to the CC-contexts as described above. The mathematics are introduced in a casual way. For a formal description see <afstudeerverslag van Remco>.

Mathematically speaking, the CC-concepts are (mathematical) sets, the name of the set being the same as the name of the concept, and the contents of the set being the instances of the concept. For example, the concept 'protocol' is associated with the set that is also called 'protocol'. The set 'protocol' may contain elements such as 'HTTP', 'IP', 'ATM', etc.

Relations between concepts are also sets. Suppose we have a relation *r* that relates (source) concept-set X with (destination) concept-set Y with statement *s*. Then *r* is the set of all tuples (*x*, *y*) with *x* in X and *y* in Y for which statement *s*(*x*, *y*) is true. For example, consider the relation *ipo1* (is part of), that relates network adapters with networks through the statement 'network adapter x is part of network y'. Then *ipo1* is the set of all network adapters *x* and networks *y* for which the statement 'network adapter *x* is part of network *y*' is true.
Mathematically: *ipo1* = { (*pd*, *n*) | *pd* in PD, *n* in N, and the sentence 'network adapter *pd* is part of network *n*' is true }

Relations can be connected to form composite relations. For example, the relations *tx* and *drv* can be combined into the composite relation *tx;drv*, where the composite relation is the set of all tuples (*x*, *y*) with *x* in PD, and *y* in C such that the statement *s*(*x*, *y*) = 'There is at least one *z* in DP such that the relations *tx*(*x*, *z*) and *drv*(*z*, *y*) are both true'. In natural language, the relation *tx*;*drv* is the set of all combinations of network adapters and connections, for which there is at least one data packet such that the network adapter has transmitted the data packet and that same data packet has traversed said connection.
Mathematically: *tx*;*drv* = { (*x*, *y*) | x in PD, y in C, and 'there is a *z* in DP such that *tx*(*x*,*z*) and *drv*(*z*, *y*)' }.

Relations, including composite relations, are subsets of the set formed by the cross-product of the concept-sets that they relate. For example, the relation *ipo1* is a subset of the set PD x N, Mathematically: *ipo1* $\subseteq$ { (*pd*, *n*) | *pd* in PD, *n* in N } = PD x N. We have the same for composite relations, e.g. *tx*;*drv* $\subseteq$ { (*pd*, *c*) | *pd* in PD, *c* in C } = PD x C.

The non-cardinality laws are all of the form $r \subseteq s$, where *r* and *s* are both (composite) relations that share the same source and destination concept sets. For example, the law ' All network adapters that are part of a network talk the protocol that is carried by that network.' can be rephrased to 'If a network adapter is part of a network, then that network adapter talks the protocol that is carried by the network'.
Mathematically: $ipo1 \subseteq t;ⓘc2$ (where the ⓘ-character denotes that *c2* is used inversely - this does not introduce much of a meaning, it is more a mathematical notational question).

Cardinality laws for a relation *r* are of the form $r;ⓘr \subseteq id$, $id \subseteq r;ⓘr$, $ⓘr;r \subseteq id$, $id \subseteq ⓘr;r$ where '*id*' is the identity relation (i.e. *id* = { (*x1*, *x2*) | *x1*, *x2* both in X, and *x1=x2* }). For example, the cardinality law for *c2* ('a network carries one and only one protocol') is mathematically denoted by $ⓘc2;c2 \subseteq id$.

In the previous section we said that we could derive the object-model (data-model) from the conceptual model (cc-model, i.e. context language + laws). Now we are in a position to summarize how this works: consider a concept, say X. Find a (composite) relation *r* to another concept, say Y. If this relation *r* is such that for every *x* in X there is at most one *y* in Y, we say that *r* is a 'functional relation', and then Y is an attribute to X. Note that if there is another relation *s* from X to Y, and either *s* or *r* is a functional relation, then you can mathematically prove that the other relation is functional as well. Also note that if *r* is such that for every *y* in Y there is at most one *x* in X, then ⓘ*r* is a functional relation and X is an attribute to Y.

Now that we have a rough idea of how the concepts, relations and laws are represented mathematically, let's return to the world of the natural language again.

Glueing CC-Contexts Together

The CC-method wants to focus on small contexts - that is to say: contexts with less than ten concepts. The reason for this is that people are not good at dealing with large numbers of concepts, and complex relations between them. As a rule of thumb 7 +/- 2 concepts is about the maximum you want in a context language.

However, the world is more complex than 7 +/- 2 concepts. One way to deal with this is to create several small enough but related contexts, and then find relations that occur in multiple contexts. The decision that two relations that come from different context languages are in fact the same is non-trivial. It implies that the source concepts are the same sets, and the destination concepts are the same sets too. It also implies that the (source and destination) set in either context must now also satisfy the relations in the other context, and must obey all laws that apply to the set in the other context. This allows for modelling notions such as 'inheritance'.

Mathematically you can also do calculations, in particular with the cardinality laws. For instance, if a relation in one context language is glued onto a relation in another context language, and either relation is functional, then the other becomes functional as well, as well as all other (composite) relations over the source and destination concepts. From this, and the way we construct data-models from CC-models, it follows that glueing contexts have a good chance of having to change the data-models. This coincides with the practical experience that data models can hardly ever be fixed during the lifetime of (large) projects.