DCI Glossary

Trygve Reenskaug with lots of assistance from
the Object Composition Group
July 16 2014

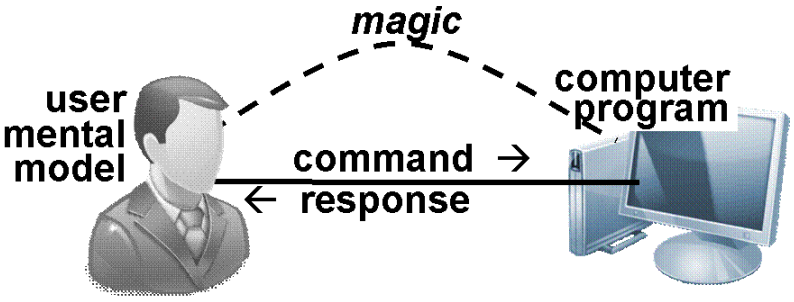
The purpose of this Glossary is to encourage people to use the same terms for the same concepts. The glossary is not a textbook; a prior understanding of DCI is assumed. There are plenty of background sources; the Wikipedia[Wikipedia] article on DCI is a good starting point

Background terminology

A ***Mental Model*** can be defined as *an explanation in someone's thought process for how something works in the real world* [Wikipedia]. A human can have many mental models in his or her mind simultaneously and is very good at switching between them as the need arises. Most of these models are intuitive, but a few are concrete and expressible in some language.

There are two aspects of mental models that are of interest here. One aspect is the user's knowledge of how to operate the computer by giving commands and interpreting the results. Such interaction is symbolized by a solid line in the figure below and is related to *what the system does*.

Another and equally important aspect is the user's understanding of the information model involved in whatever task the user is working on. This aspect of the mental model relates to *what the system is*. The corresponding data model in the computer may or may not be similar to the user's information model. The data model exists concretely in the computer hardware, but the user can only perceive it indirectly through exploration of the user interface. The ideal is that user's information model magically corresponds to the computer's data model. This ideal situation is symbolized by the dashed line in the figure.



Alan Kay introduced the term ***object orientation*** as a common tool for thinking about how things work as well as how we can make things work in the computer. The following quote is coached in terms of Smalltalk programming, but the idea is applicable to describing computer programs well as human mental models:

> *"In computer terms, Smalltalk is a recursion on the notion of computer itself. Instead of dividing "computer stuff" into things each less strong than the whole--like data structures, procedures, and functions which are the usual paraphernalia of programming languages--each Smalltalk object is a recursion on the entire possibilities of the computer. Thus its semantics are a bit like having thousands and thousands of computers all hooked together by a very fast network. Questions of concrete representation can thus be postponed almost indefinitely because we are mainly concerned that the computers behave appropriately, and are interested in particular strategies only if the results are off or come back too slowly.*
>
> *Though it has noble ancestors indeed, Smalltalk's contribution is a new design paradigm--which I called object-oriented--for attacking large problems of the professional programmer, and making small ones possible for the novice user. Object-oriented design is a successful attempt to qualitatively improve the efficiency of modeling the ever more*

complex dynamic systems and user relationships made possible by the silicon explosion." Kay-93

The 'computers' in Kay's definition are *objects*. At Xerox PARC, the Smalltalk group's definition was: "*An **Object** is an entity that has identity and that encapsulates state and behavior"*. The identity is immutable; it can be shared and communicated so that it is meaningful to reason about networks of communicating objects. Encapsulation means that there is a clear distinction between an object's external properties and its internal construction. Externally, an object is characterized by its provided interface, i.e., the set of messages that the object can receive. Its internal construction (its class) is irrelevant to the object's collaborators as long as the object behaves appropriately. The notion of object orientation is recursive since an object can encapsulate an inner network of communicating objects.

A very important goal for DCI is to give the user an illusion of working directly with his or her mental model when working with the computer. Object orientation is the key to the DCI way of achieving this goal. The data model can be implemented in the computer as a structure of objects. The user's mental information model can be a similar, often intuitive, object model. With DCI, system behavior is implemented as interacting roles within a Context. The user can similarly explain what the system does in terms of interacting roles.

With DCI, the boundary between user and computer fades away; the computer acts as an extension of the user's mind. This magic supports the "no surprises" principle and is illustrated with a dashed line in the above figure.

The DCI Glossary

The following defines terms for important concepts in DCI. The definitions are based on regular English and the terms defined in *Background terminology* above.

The terms related to the system's handling of a Use Case are of special interest:

Use Case
→ Scenario
→ Command
→ System Operation
→ Context
→ Role
→ RoleMethod

Command	A user input that triggers the execution of a System Operation. Part of a Scenario.
CRC Cards	*Candidate (role), Responsibility, Collaboration* (CRC) cards[CRC] are a brainstorming tool that can be used for determining the Roles and their collaboration structure in the implementation of a System Operation. These CRC cards are an object oriented refinement of the original, class oriented cards.

Context	A Context implements one or more System Operations. The static (class) side of a Context specifies networks of communicating objects as similar structures of interconnected Roles. Object identity is represented by the Role's position in the structure. A Context method coherently maps all the Roles onto objects, keeping the result in the RoleMap of the Context instance.
Data	The DCI Data objects are the objects of the user's mental model. Examples are domain objects, MVC[MVC] model objects, and database objects.
DCI	A paradigm defining a program architecture where a program is seen in different perspectives. Each perspective is a filter that exposes certain properties of the program and hides the rest. The essential perspectives are Data, Context, and Interaction
Habit	A Habit is like a Use Case, but one that is the purview of the programmer rather than the end users. See Coplien-Bjørnvig p 183. A Habit does not in itself achieve a business goal.
Injection	RoleMethod injection is a mechanism that maintains the invariant: "For any given Role, its RoleMethods are shared among all its RolePlayers."
Interaction	In DCI, specification of how a network of communicating objects realize a System Operation. The network nodes are RolePlayers; their behaviors are specified in their RoleMethods. (Note that in some Use Case methods, the term 'interaction' refers to the interaction between actors and the system under discussion.)
Role	A Role is an alias for an object in a network of interacting objects within a Context instance. A Role is an abstraction that highlights an object's identity and external properties while it ignores the object's internal construction. A Role has a responsibility that is part of the responsibility of the enclosing Context instance. The Role forms a bridge between the compile time and the runtime properties of a system. A Role explicitly or implicitly specifies a RoleObjectContract that must be provided by all its RolePlayers.
RoleObjectContract	An assurance in some form (depending on programming language) that only objects reifying a required set of messages can play a certain Role.
RoleMap	Every Context instance has a RoleMap that maintains the current mapping between Roles and RolePlayers.
RoleMethod	A stateless method that is a feature of a Role and that is shared among all the Role's potential RolePlayers. Polymorphism does not apply to these methods; RoleMethods have priority over methods specified in the RolePlayer instances. An executing RoleMethod can access its actual parameters and temporary variables. It can also access the current RolePlayer and the current Context.
RolePlayer	An object that fills the position of a Role in a network of communicating objects. Examples are Data objects and other objects that participate in an Interaction. The object must provide the Role's RoleObjectContract.

Scenario	A scenario is a narrative describing foreseeable interactions of types of users (characters) and the system. Scenarios include information about goals, expectations, motivations, actions and reactions. Scenarios are neither predictions nor forecasts, but rather attempts to reflect on or portray the way in which a system is used in the context of daily activity.[Wikipedia] A Scenario is non-branching; conditionals are expressed using alternative flows. A Scenario entails one or more user Commands as part of the interplay between user and computer. Coplien-Bjørnvig pp 167ff
System Operation	A System Operation is behavior implemented by a Context. A System Operation is always executed within a Context instance and is realized as an Interaction within this Context.
Use Case	A use case is a description of a potential series of interactions between program and a user, which lead the user towards a business goal. The potential series of interactions are described in a series of Scenarios. (Note that 'interaction' here has a different meaning from the DCI Interaction.)

References.

Coplien-Bjørnvig	Coplien, J. O., Bjørnvig, G; *Lean Architecture for Software Development*; ISBN 978-0-470-68420-7; Wiley, Chichester, United Kingdom, 2010.
CRC	R. Wirfs-Brock, A. MacKean: *Object Design. Roles, Responsibilities, and Collaborations*; ISBN 0-201-37943-0; Addison-Wesley, Boston, MA 2003.
Kay-93	Alan Kay: *The Early History of Smalltalk*; ACM SIGPLAN Notices archive; 28, **3** (March 1993);p.70; An HTML copy at [web page] *http://c2.com/cgi/wiki?EarlyHistoryOfSmalltalk* and http://www.smalltalk.org/smalltalk/TheEarlyHistoryOfSmalltalk_TOC.html
[MVC]	Reenskaug, T.; *The original MVC reports*; [web page] http://www.duo.uio.no/sok/work.html?WORKID=52648 Reenskaug, T.; *The Model-View-Controller (MVC). Its Past and Present*. Dept. of Informatics, University of Oslo; August 2003; [web page] http://heim.ifi.uio.no/~trygver/2003/javazone-jaoo/MVC_pattern.pdf
[Wikipedia]	*WikipediA, the free enceclopedia*. [web page] http://en.wikipedia.org/wiki/Main_Page