Sept. 15,16 2007 - Journal thoughts - Putting everything into my own perspective

A **goal** is not a task but the two are closely related. A **goal** is a desired state of the world (where 'world' is restricted to the domain of discourse or the 'business domain'). A **task** is a program intended to put the world into a given state and is represented in some language, potentially a natural written language, a computer language, machine language, etc., but also includes things like the instructions on how to build a Lego toy (all pictures) or the instructions that come office furniture (pictures, words, etc). Of the latter two, the furniture example is more precisely a task in our business world because it seems to include more information for the reader where as the Lego instructions start right at the first step without writing anything and assumes the user knows what they are doing (or that they will figure it out somewhat automatically). But another way to look at that is to imagine dumping all the Lego parts in front of a child and the furniture parts in front of a literate adult and giving them the respective instructions, neither having done the process before, and see which one finishes first. From my own experience I would bet the child finishes first, , baring that the child is too young and making sure the adult is not too old (where in either case the individuals may be useless). In other words we say that a goal provides the information about 'what' is needed and the task provides the information on 'how' to do it.

A goal G is **actualized** at time t if it describes the state of the world W at time t. In turn **actualization** means that $G(t) = W(t)$. At time t we normally stop the task execution process because "we are done" unless we have reason to believe there is some $c > t$ such that $G(c) \mathrel{!=} W(c)$ in which case we actually need to revise the goal itself. A well defined task should incorporate everything necessary for the task to end on its own, but the task is really finished when the goal is actualized, regardless if that is before the end of the task or does not even occur by the end of the task. Incongruence such as these should lead to revising of either goals or tasks. A goal is **accomplished** if it is actualized by the task that was intended to actualize it and otherwise not accomplished (but may be actualized by some other phenomena). If a goal is actualized but not accomplished we could revise the goal or task and try again or we could just accept the usefulness of an actualized goal and move on.

A **team** is a set of task executors coupled with a management system. That definition generalizes the term to apply to a group of people working together on a common goal as well as a computer system employing multi-tasking and concurrency in hardware and software (and may be applied to natural systems if we consider a theory of ethics arising from nature's implied goals).

An **executor** (for lack of a better word) is an agent or mechanism that physically triggers an effect or; in particular an executor will carry out, in order (if there is an ordering) the entire sequence of atomic steps in the task specification. For instance, to stretch the terminology, an alarm clock in a sense has one simple task of waking you up every day, although it would be somewhat illogical to say it has a goal of waking you up; you have a goal of being woken up and you decide one task that would accomplish the goal would be to have a loud sound occur at the exact time you should be awake. You could ask someone to keep track of time and at the correct time come into your room and yell and similarly you can set the alarm clock to have the same effect. Both the clock and person would be executors in the above sense, but the tasks end up being different and in the case of the clock, its not even conscious but still treated as an entity which did a job. When/while the alarm goes off, the task is in progress or being executed. When its no longer in progress if you are awake (and weren't before) then the goal is actualized and accomplished. In either case the executors could fail to execute the task though. The question is always 'how likely is a task execution failure?'.

A computer with multiple CPUs technically has multiple task executors by the above and is therefore a team. Even though a single CPU machine may seem to do multiple things at the same time, it is actually never executing more than one thing at a time and is not a team in that sense; but since multitasking still seems to be happening it could still be thought of as a team (the true distinction comes from the actual complexity and perplexity that occurs in programming truly concurrent applications). In general teams should provide expanded task execution potential but almost necessarily must also yield increased root task complexity. The specification of the goal should not depend much on the team organization, but the specification of the task is almost totally dependent on it. As described above, a team requires a management system and a task execution system. But those things are essentially required one way or another, at least implicitly; a single agent still must be a team by that definition because the management and execution of tasks must still occur.

Even when teams are not used in the traditional sense, roles must be defined and adhered too. The **task management system** (or just task manager, not attempting to relate back to Microsoft windows task manager, which should more accurately be called the 'task management console') must handle scheduling and priority of tasks, as well ordering if the ordering is not encapsulated in the task(s) and finally must provide the verification mechanism for task completion (E.g. cashier in a store is allowed to leave after his drawer has been certified by the manager to be swapped out, counted and secured). The verification part is ultimately what decides when the goal is accomplished as opposed to actualized.

In non-computer systems, it must be decided how to deal with in-progress task scheduling; humans are not as good at dropping a complicated tasks and coming back to it later. When that happens we see a huge context switch overhead and a policy for scheduling must reflect this. Even in a computer system it is possible to have quite unequal task executors as opposed to having multiple identical CPUs. For instance in a large super computer, a main computer system may send tasks off to specialized sub-systems. Maybe one such system is a cluster of inexpensive machines and the other is a cluster of machines with extremely good floating point performance. Therefore only the most floating point intensive tasks are given to the latter system and equal distribution would not be the most efficient strategy.

The following defines a general goal recursively from the top down: every non-atomic goal *G* is defined by some function of a set *S* of other goals, where *S* may contain atomic and non-atomic goals. If we draw out the resulting structure, we should have a nice dependency graph. E.g. a goal: send email to `person@place.com`. Resulting subgoals: have a *computer system*, have a network connection, have an email account (and know how to use these things, or have a person who you can communicate with and who can operate the machine!). I would assume the latter two are atomic, for instance, I got both a dial-up account and email address from UMass/OIT (although that didn't give me a phone line to dial out on, but I just ignore that here). The subgoal 'having a computer system' is not an atomic goal; it breaks down into the goal of having all the necessary computer hardware and software, and that it's configured correctly, and having electricity for the durration of time that it takes to send the email, etc. An **atomic goal** is just one that doesn't have any contingencies such as those. For these purposes a good example of an atomic goal is to solve an algebraic equation; a task can immediately be employed to actualize that goal without having a subgoal, such as, gathering extra information. If the equation to be solved involved unknown constants which depended on some physical phenomena not yet observed and the final answer needed to be a number, then a subgoal may be to do the observations (or find someone who already has done them) and derive the number. Notice here that its not saying we need to do those observations *before* we can solve the equation, we just need to do them in order for the root goal to be actualized.

A **well defined goal** or **fully specified goal** defines all the Boolean logic necessary to control a task intended to actualize that goal. After actualization of the goal, if it can be shown scientifically that the task caused (within probability threshold) the goal to be actualized then we can say that the task **accomplishes** the goal and the process used to make that determination is the **verification** process handled by the task manager.

Showing scientifically means having sufficient statistical evidence to mathematically find the conditional probability that $G(t)=W(t)$ given task $T(x)$ has been executed for $x = k$ to $x = t$ (where k is some amount of time in the past and $[k,t]$ is therefore an interval). The point is finding actual correlations. This exactly generalizes science itself. But my probability notation may be a bit rusty; its just trying to say, find the probability that, what we did, made the desired/good thing happen. If that can not be done, either for lack of a scientific model for the given situation or because the goal became actualized by some totally unrelated/unknown phenomena, we can not derive any methods for future use, which is represented by the fact that we don't say the goal is 'accomplished'. Every time a goal is accomplished, the goal and the task become library items, or in sports terms, form a "play book" (but I know nothing about sports, it just seems to fit).

When an atomic goal is accomplished, it implies that we have a method to accomplish that goal in the future (unless the nature of the goal is intrinsic to a particular time and has absolutely no bearing on the future). Similarly when a non-atomic goal is accomplished, it means that the aforementioned statement is also true of all the subgoals. If the above conditional probability is found accurately but the number is below the defined threshold then we may not say the goal is accomplished. If we know that the goal could not become actualized by default ("on its own") then we could assume that we did somehow cause the actualization somehow but we can not say that it was our task that did it, all we know is that it was some combination of the task and the state of the world (along with how the tasks were actually carried out if not carried out by a computer or in an very precise way).

The worst case is that a goal is not actualized because a goal must be actualized by definition to be accomplished. If all goals are actualized but never accomplished, the company will never be able to expand or make higher profit margins. It is helpful to note as much as possible what information might be available even when goals are not accomplished because patterns may arise over time and eventually lead to better task specifications. This is actually a natural and implicit method for human operation. Most people ignore the simple complexities in life such as physical navigation to destinations. But its hard for robots to navigate in the same way. Similarly its hard or impossible for new born humans to navigate in that way. In humans, navigational abilities evolve at the same time as abilities to formulate destinations (here allowing for the notion of 'travel' itself as a destination given that children have satisfaction of curiosity as a goal). In general humans develop subconscious systems for identifying goals and verifying actualization, and further develop systems for cataloging what works and what doesn't work. From this we see even the most primitive humans don't often make same basic mistakes twice (that is 'mistakes' as defined by their conceptual system, not ours). A mistake might be getting stung by a bee, or stepping on a sharp rock or maybe wasting all day on an inefficient method of acquiring food.

As described above, a goal may be actualized before its even thought of and a goal may cease to be actualized at some time after actualization. That is, as even if $G(t) = W(t)$ there may be some time $c > t$ such that $G(c) \text{ != } W(c)$ and even worse it may be that for every $k >= c$, $G(k) \text{ != } W(k)$. But both tasks and goals can be defined recursively, so goals and tasks can be fixed in many nice ways touched on below.

Accomplishment means scientifically proving that a well defined goal is a description of the current state of the world *where it wasn't before*, and *the change is due to the task that was undertaken for the purposes of actualizing the goal*. That is to say that, if one has a goal that is already the state of the world, I wouldn't say the goal is accomplished, I would just say its not a *valid goal*. Similarly, if a goal becomes actualized by some other means than the task that was supposed to actualize it, the goal is not accomplished. E.g. the goal of making water wet is already actualized. A more complicated example: a person's car is making an annoying noise, so the goal is to stop the noise. They decide that the task is to open the hood and then listen closely to different parts of the engine to localize the noise and upon finding the source of the noise, attempt to try to stop the noise there. If they open the hood and the noise stops immediately the goal is actualized because the noise stopped, but not accomplished because the proposed task wasn't the cause of the actualization. We may consider retroactively modifying the proposed task to allow that simply opening the hood could be the thing that actualizes the goal, e.g. goal: stop car from making noise; task: first open hood, if noise does not stop, then listen closely to the engine to find where the noise is coming from, etc. In this case we can't revise the task and repeat the experiment per se because the noise was totally random. But also, it would be inaccurate to consider that listening to every part of the engine closely (as originally proposed) somehow lead to the goal being accomplished; the accomplishment in this case is a latent function of the original task (as best we can tell) and so we can change the task and say it is accomplished, but in general we may be forced to allow the goal to remain only actualized. A good example of this in computer troubleshooting is what I call the all-powerful two: restart and reinstall. Be it an application or OS, if it doesn't work in any way, first restart it, and if it still doesn't work, then reinstall it. If it doesn't work after reinstall then either it can't work or the hardware is bad. No matter how effective the process is at actualizing goals, its infinitely ineffective at accomplishing goals. It never leads to an expanding and improving knowledge base for later use. And worse yet, it doesn't conceptually generalize at all; consider the same approach applied to fixing the car: if the car has a problem turn it off and then back on, and if the problem is still replace the engine or get a new car!

Sometimes actualization of a goal is just as good as accomplishment for the purposes of making money in the short term, but with respect to software engineering process and long term sustainable (and hopefully increasing) profit margins, accomplishment is the most important thing. Because as described above, goal accomplishment leads to a knowledge base of repeatable methods.

GAK – Goal Accomplishment Knowledge system

I think the above could be expressed by as a general theory and named GAK for goal, accomplishment, knowledge. GAK more or less describes an optimal approach to any problem area in which a process can be employed in a repeatable way to have some desired effect. The first step is recursive goal specification in its most general form which may include use-cases for engineering or general constraints. Each aspect of the goal is recursively decomposed as much as possible. Recursive task specification where actual methods are proposed which are intended to actualize each goal. Each task is executed and verified against its associated goal. For subgoals, accomplishment is always preferred but actualization must be accepted sometimes given limits on time. If some subgoal of a goal *G* is actualized but not accomplished then the *G* may not be said to be accomplished depending on the statistical significance of subgoal(s). Tasks are more or less recursively defined by the recursive definition of the goal, but a task for any atomic goal could break down in a complex way given the inherent nature of 'what to do' vs. 'how to do it'.