Towards a Fundamental Structure for Tasks

Andrew Frederick Cowie
Operational Dynamics

Abstract

No one can agree what the magnitude of a single step of a procedure is since different actors in a system will have differing understanding of the internal complexity of their tasks. Thus a rigid hierarchy is inappropriate when using procedures to describe highly detailed activities or larger composite processes. There appears to be, however, a fundamental unit to procedural activity which we identify as the "step", and which can be used to describe structured activities of any level of complexity.

Background

In the past few years we've advocated a specific, fixed, taxonomy for naming the actions to be taken when planning and executing a procedure. Each step had one or more named individuals working concurrently on a series of individual tasks. Steps were conducted sequentially, with progress on the next step only beginning once all the tasks within the current step were completed. Finally, steps were grouped into sections, with each section boundary describing a stable point that could used as a safe holding point or, in extremis, rolled back to.

This schema was very successful when planning and executing massive change and upgrades to mission critical IT systems. [1] We were able to extend this to the activities of small operations teams (and even in the largest companies the teams doing routine updates and responding to incidents are invariably small).

The rigid hierarchy had some problems, however. Sometimes we wanted to talk about the detailed actions that someone had to take within a task. That was awkward, because one person might have a single task (send an email, say), whereas another might have dozens (specific commands to be run on a server to perform a maintenance action). In the latter case we want to record the exact instructions, otherwise what's the point of having a medium that can be refined and improved? [2] We tried to call this fifth level of hierarchy "detail", but the syntax for recording this in documents was reaching its limit. Bullet points nested underneath tasks?

While we were grappling with this we also had experiences where despite our best efforts people started using step, task, item, and action pretty much interchangeably. The hierarchical relationship (and importance of not proceeding with the next step until all the tasks in the precious step were complete) was soon lost in conversation … and then in execution. This highlighted that different participants would see their units of work, their tasks, at different sizes. People at different levels of an organization will necessarily have different views of their sphere of responsibility, and so consider different magnitudes of action to be a "step".

And then we began to realize that this was a feature of any work considered as a procedure.

Keywords: Procedures, Composition, Schema Hierarchy, Distributed Tasks, Agent Based Systems

A. F. Cowie, "Towards a Fundamental Structure for Tasks" at *CodeCon 2011*.
Mount Wilson, NSW: Peter Miller and Company, Oct. 2011.

Procedures

Consider a procedure to accomplish a task. The task could be an infrastructure related such as provisioning a virtual machine, or an operations procedure like updating the company website. Or it could be a step in a larger process, preparing the department's annual accounts for example.

Fundamental Unit

We start with the description of the task to be initiated. Then something has to start it. That could be a user running a program, a manager instructing a subordinate, or an item being added to a work queue. It doesn't seem to matter whether the agent performing the task is a human or a computer program.

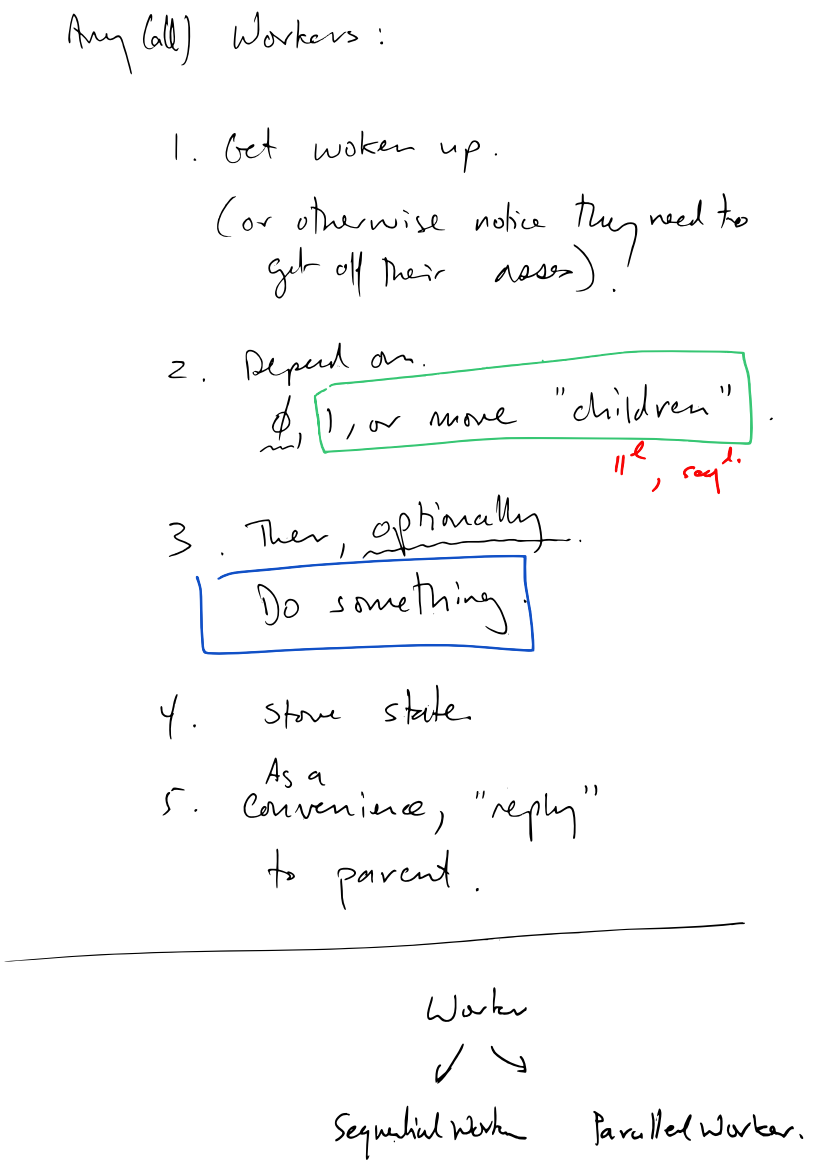

Figure 1: The original statement of the fundamental unit of procedure

Accomplishing that task might require other tasks to be initiated, or for the agent performing the task to wait for those other tasks to be finished. It might in turn be a part of a larger system, so finally the entity performing the work needs to let whomever initiated the task know the work is done. The size

of the task is not relevant to this definition, and the structure appears to apply regardless of scale or nature of the task. Thus, as shown above (Figure 1), we have:

> An agent performing work in a procedure system: gets woken up (or otherwise notices it has work to do). It depends on 0, 1, or more "children", and waits for all to complete. Then, optionally, it performs a task. The worker then durably stores the state of it being completed along with a result value from its task if any. Finally, as a convenience, it signals its parent indicating it is complete.

We'll call this the *fundamental unit of procedure*.

Composition

Steps can be composed in parallel or in sequence (Figure 2). A procedure that depends on multiple steps has those dependencies run in parallel. A procedure that depends on a single step is collectively a set of steps that run in series. And a step with no dependencies is a leaf node in the procedure.

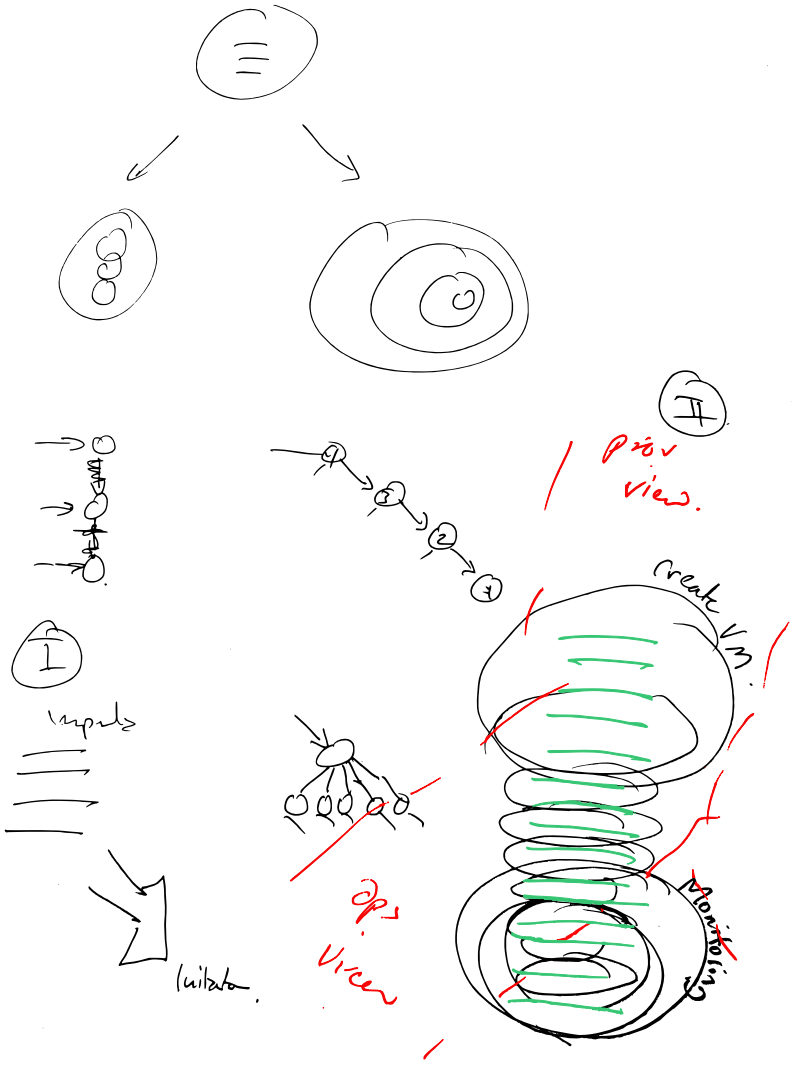

Figure 2: Description of the different dependency variations. There are steps that can run in parallel whereas a step that depends on a single step which in turn depends on a single step represents a set of operations that run in series.

Procedures can be grouped by their dependencies. A system of automation can be developed as one step depends on another (Figure 3).

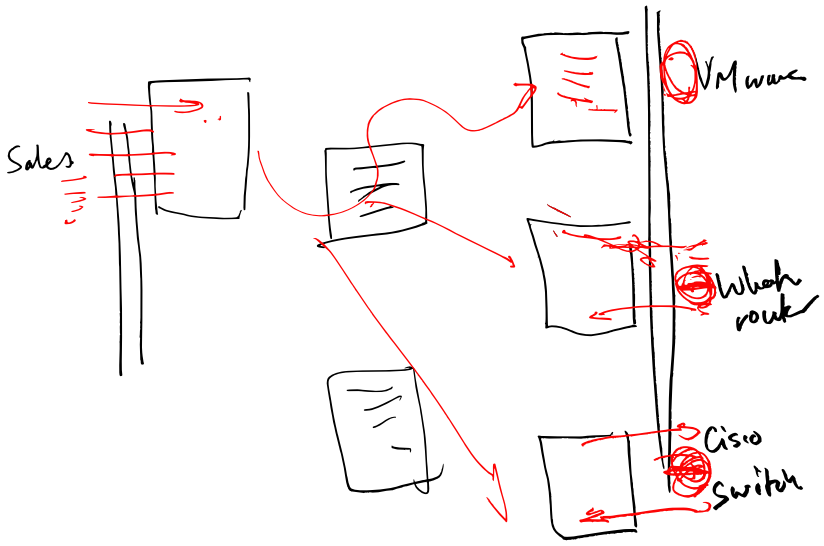

Figure 3: A network of steps showing internal structure, dependencies, and boundaries between automation and external players

This example was written recalling the process a 2000s-era managed hosting company needed to follow to provision a virtual machine. Starting from data collected by Sales initiating an order, internal processes were followed leading to an instance being defined in their VMware management interface, the switches being configured to route traffic, and subsequently updates being made in their billing system.

We gain the interesting property that different participants can have a differing view of the system. While a nested set of tasks is a tree structure, we don't have to be limited to just one such construction. A project manager would care about certain tasks and milestones, whereas an operations team would be focused on a different set of requirements. These do not need to be mutually exclusive. If the results of one procedure could be included in as an input dependency in several others the multiple "views" are possible as shown above. This could offer a novel way to address the information asymmetry that arises from the organization separation described by Conway's Law. [3]

Boundaries

When an external boundary is encountered, actions can be taken by workers *outside* the system, with results then recorded *inside* it (Figure 4).

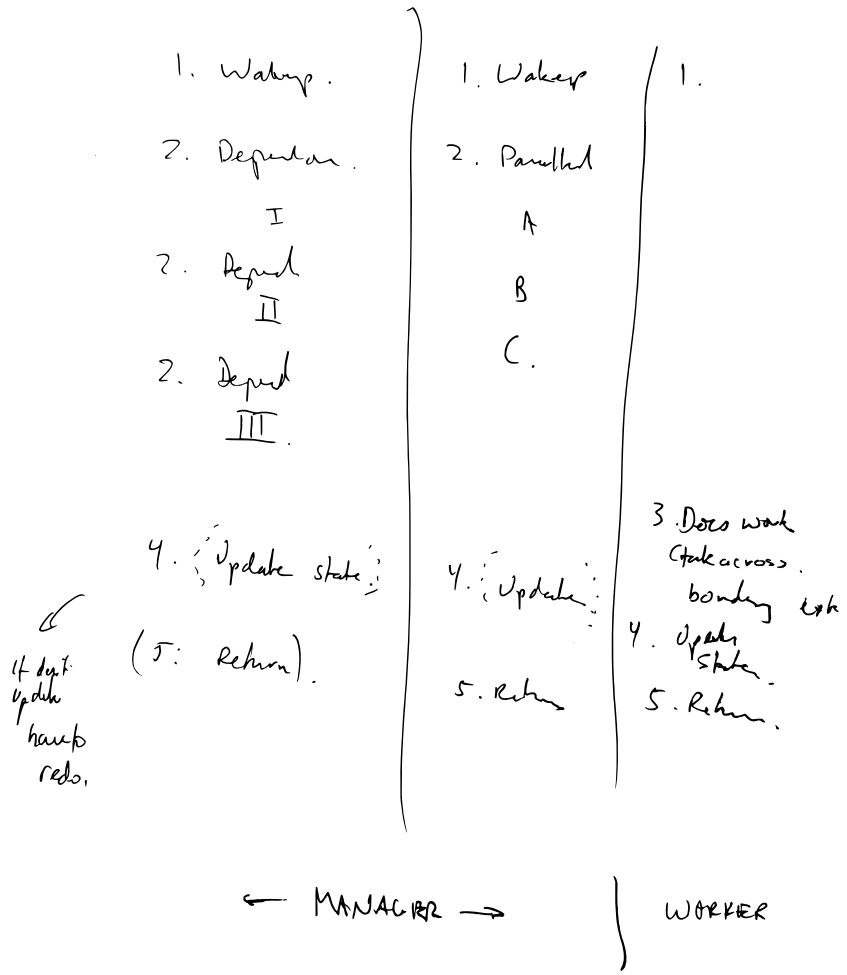

Figure 4: Exploring the boundary between higher-order steps (the "manager" layers in this example) and leaf nodes at the edge of a system of procedures (labelled "worker" here).

This opens a fascinating direction for exploration. So often when building "automation", systems administrators working on IT infrastructure write scripts which encode repetitive tasks, but of course that automation only expresses things which can be done by the scripts. [4] It is understood that human operators will run the scripts (or press the button that will cause a scripts to be run, or…), but what happens when a step in the procedure needs to be done by a human?

Description and Instantiation

There is a distinction to be drawn between the static description of a procedure to be followed versus the live instantiated version of a procedure being executed (Figure 5).

The schema or description of a procedure is a static form; when run it becomes a live network of nodes. As nodes execute their tasks they will emit telemetry—a familiar notion to systems designers but in

this case the telemetry will be transmitting the state that the node is in and the information that it is completed when it has performed its work.

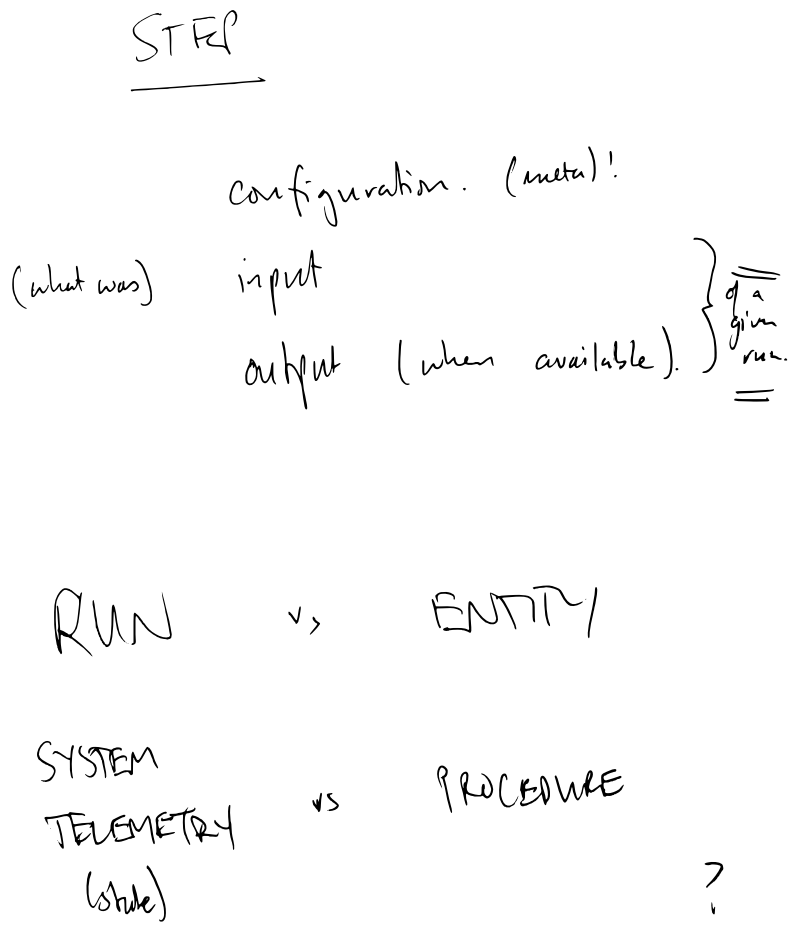

Figure 5: First use of the term «step» to describe a unit of procedure, and consideration of the relationship between an instantiated procedure at runtime versus the entity describing the procedure to be executed.

Realizing a system

Finally we consider what it would take to realize such a system in a machine or network of machines. Servers would host an agent program, which in turn could host the instantiation of one or more actors running procedure steps (Figure 6 below).

Our previous work on [5] envisioned multiple participants collaborating on a procedure. Here this notion is extended and generalized to the procedure itself being not just a participant but the point of view of the event.

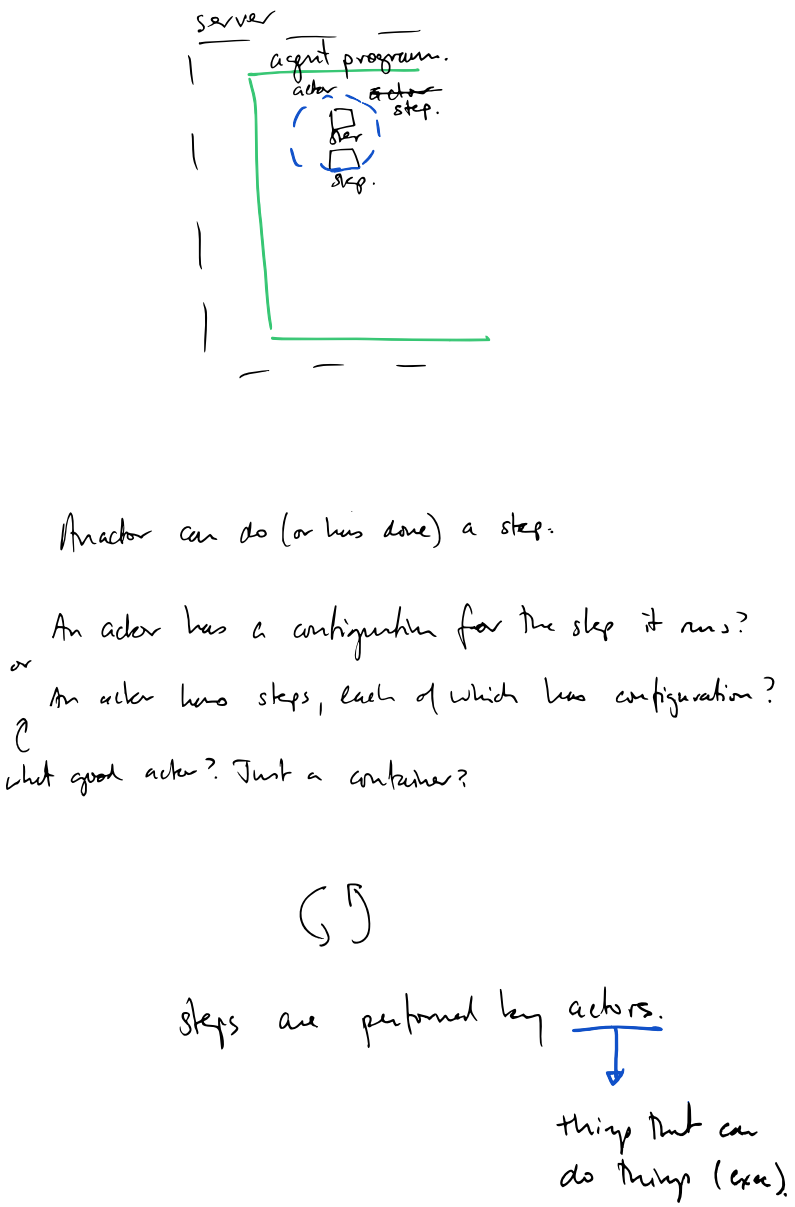

Figure 6: Consideration of the structure of a runtime that would support a network executing these procedures, showing the relationship between servers, agents, and actors

This then leads to questions of identity. Where is the configuration for a given actor to reside? Is there a unique configuration for every instantiation of a step, or does an actor get instructed and then can carry out a given task repeatedly on request according to that already set configuration? This is the direction our for future work.

Conclusion

A rigid, fixed, pre-defined structure for a procedure is insufficient when we attempted to articulate the detail within a task, or needed to aggregate procedures at larger scales than the hierarchy supported. Examining these two extremes uncovered the fact that a procedure is scale independent, leading to the definition of a fundamental unit of procedure presented here.

Acknowledgements

The outline of these ideas were first presented in [5] and were subsequently developed in discussions with Peter Miller, Lindsay Holmwood, Silvia Pfeiffer, Erik de Castro Lopo, and Robert Collins whose insights and encouragement were greatly appreciated.

The illustrations in this paper are the hand-written notes originally developed by the author when exploring this topic, ultimately building towards this presentation.

References

[1] A. F. Cowie, "Mastering Massive Changes and Upgrades to Mission-Critical Systems," in *Proceedings of the 19th Conference on Systems Administration (LISA '05)*, D. N. Blank-Edelman, Ed., USENIX, Dec. 2005.

[2] A. F. Cowie, "Surviving Change," in *Systems Administrator Guild of Australia Annual Conference Proceedings*, SAGE-AU, Aug. 2004. doi: 10.31224/5596.

[3] M. Conway, "How do Committees Invent?," *Datamation*, vol. 14, pp. 28–31, 1968.

[4] P. Miller, "Recursive Make Considered Harmful," *AUUGN Journal of AUUG*, vol. 19, no. 1, pp. 14–25, 1998.

[5] A. F. Cowie, "Realizing a Graphical User Interface for Procedures," in *CodeCon 2007*, Peter Miller and Company, Sep. 2007. doi: 10.31224/5768.