Decentralized Task Execution Patterns

Cesare Pautasso

10th Asian Conference on Pattern Languages of Programs,
People, and Practices (AsianPLoP 2024)
Feb. 28 - Mar. 3, 2024
Fujisawa City, Kanagawa Prefecture, Japan

Abstract

Decentralized task execution requires to assign planning, execution and control roles and responsibilities to at least two distinct parties. These include making decisions about "who does what when" so that one party can track the progress and the outcome of tasks executed by another party. In this paper we outline the design space for decentralized task execution by presenting 18 different patterns, going from imperative task execution all the way to autonomous task execution. We will distinguish how decisions about who performs a task, what needs to be done and when to do it can be made by different parties and how these alternative decision-making locations impact the interactions between the parties responsible to plan, execute and control the task execution. The patterns are applicable within human organizations with many actors delegating or performing multiple tasks but also to fully automated systems, e.g., distributed operating systems, intelligent autonomous vehicles, or high throughput job schedulers.

1 Introduction

There are two complementary reasons for executing tasks: achieving goals [5] and reacting to triggering events [12]. In both cases, executing a task requires humans or machines to perform some actions (what) taking advantage of the skills of the available resource (who) assigned to execute the task. Given limited resources and the non-instantaneous duration of task execution, tasks need to be efficiently scheduled (when).

Task execution is often combined with planning and control [3]: before execution occurs, decisions about *who does what when* need to be made and recorded as part of a plan; after execution occurs, its outcome needs to be recorded so that, for example, it can be determined whether the task was

successfully executed or not and whether the execution was carried out according to plan.

In a distributed workflow environment [4, 7] there are multiple parties involved with task planning, execution and control [11]. In this paper we assume there are at least two parties, the minimum required to break free of centralization [1, 9, 2] and already a sufficient number to generate the design space of 18 different patterns we are going to explore.

In the simplest case, one party makes all the planning decisions (determines who *will do* what and when), while the other party will actually execute the task, reporting its outcome back to the first party, which also plays the control role, checking that the task has been executed according to plan. We call this *imperative task execution* (Fig. 1). For example, a manager breaks down a complex request coming from above and assigns each of the resulting tasks to each available team member. High priority requests may pre-empt ongoing activities of the team so that tasks for a very important customer get completed as soon as possible.

Another example would shift the planning responsibility to the party in charge of the execution while keeping the control role separate. In this case, all decisions about what to do and when to do it would be taken by who actually does the task execution. The task execution outcome would be reported back to a separate party, in charge of tracking who *did* what when. We call this *autonomous task execution* (Fig. 16). For example, entering the destination address is sufficient for a fully autonomous car to start driving. After safely reaching the destination, such car will drop off the passengers and even find a parking spot nearby, all by itself.

The rest of this paper is structured as follows. In Section 2 we define the three dimensions of the design space for decentralized task execution and position within it the 18 patterns described more in detail in the following Section 3. In the interest of conciseness and readability, sections of the context and problem that are similar across multiple patterns have been uniformly addressed in Section 2.4 and 2.5. We draw some conclusions and point out future work directions in Section 4.

2 Design Space

We introduce the following three dimensions (Table 1) for defining the core design space of decentralized task execution: who, what and when. For each dimension we distinguish the location where each decision is taken: separately from the execution party (Plan), by the same party performing the task execution (Execute), or shared among both parties.

2.1 Who

The choice of who performs a task is constrained by the availability of skilled resources suitable to execute the given task. This decision can be made as part of the planning by associating each task with the corresponding resource [8], assuming that a *passive* resource will be ready to execute the task at the time the task should be executed.

Alternatively, *active* resources can spontaneously volunteer to execute given tasks [6]. The choice will be delayed until the latest possible moment: the resource ready and willing to execute the chosen task will do proceed to do so. In this case, it is critical to record the choice of who performs a task so that, for example, credit can be given for successful outcomes or resources can be blamed for failed executions.

Passive resources establish a push relationship with the source of tasks to be executed, as they wait for external requests to initiate the execution. Passive resources require to keep track of their availability and work load. Conversely, active resources pull the tasks to be executed as they internally determine they are ready to begin with the task execution.

2.2 What

Determining what to do defines the core of a task. Tasks can be seen as transformative actions, which bring the initial state (the task pre-condition) into the outcome of the task execution (the task post-condition). While the goal of a task can be described in terms of such transformation, only skilled resources may be entrusted with performing such task without further instructions on what exactly should be done to obtain the desired outcome from the given starting point.

More concretely, it may even be insulting to ask an Italian chef to prepare a pasta dish by handing over a recipe precisely describing in detail each step on how the ingredients should be cooked. For a less experienced cook, or a kitchen robot, however, having access to clear, correct and complete instructions may be the difference between a successful lunch or a failure. This outcome will nevertheless be determined by whoever gets to taste the food, and every customer has their own acceptance criteria.

Exactly what to do can be decided entirely as part of the planning, so that whoever is available to do it will directly execute the given task. The decision

	Where		
	Plan	Execute	Both
Who	Passive Resource (Push)	Active Resource (Pull)	
What	Action	Goal	Negotiation
When	Strict Schedule	Eventual Completion	Flexible Schedule

Table 1: Design Space Dimensions Summary

about what to do can be taken separately only if a sufficiently detailed and precise task description can be successfully communicated and understood by the party responsible for carrying out the task.

The decision about what to do (and how to do it) can be delayed and delegated entirely to the execution resource, who can be trusted to know what to do and only needs to agree on the goal to be achieved but does not need directions describing how to achieve it. Tasks defined in terms of goals also open up the opportunity for optimization and improvement of how they are executed.

The "what to do" decision can also be shared among both parties, with the planning party offering a set of tasks that can be done and the execution party picking which task they are willing to do. Likewise, the execution party can propose a set of tasks they are capable of doing and the planning party can narrow the choice of tasks to the ones which should actually be done. In this case, tasks could be rejected without even attempting them, due to a lack of available and sufficiently skilled resources, due to incomplete, incorrect or unclear instructions, or simply because the task initial conditions are not yet satisfied.

2.3 When

Given the finite and non-instantaneous duration of task execution, we also find decisions about when tasks should be started and by when tasks should be completed to be relevant [10]. As with the previous dimensions, also the time dimension can be handled separately or within the task execution party.

These scheduling decisions can be made as part of the planning so that the execution resource will be expected to immediately start running tasks and complete them as soon as possible. In other words, time-rigid resources should *strictly* follow the exact schedule received from the planner.

Alternatively, resources which know how to manage their own time can execute tasks at their earliest convenience and *eventually complete* them, fully owning the scheduling decisions.

Again, the 'when to do it' decision can be shared among both parties, with the planning party setting due dates for tasks within a *flexible schedule* and the time-adaptable resource independently determining when to start and finish the execution within the boundaries set by the planning party. In this context, the control party would be responsible for sending reminders and negotiating deadline extensions with overloaded resources.

2.4 Context

We use the active vs. passive resource (who) dimension to identify the context of applicability of the pattern as the type of resource can be considered as a design constraint, limiting the choice of which pattern should be introduced. Likewise, the strict vs flexible scheduling (when) dimension can

also be used to determine the pattern context, as it uncovers the assumption about whether and to which extent resources can be expected to manage their own time. Finally, the ability of a resource to make decisions about which tasks can or should be executed (what) is also an important assumption that constraints the applicability of each pattern. The combination of the three dimensions – who, when and what (using the values listed in Table 1) – reveals a different situation from which the problem arises.

While each pattern is suitable in the corresponding context, the context may change during long-running execution of complex tasks. For example, autonomous driving may work on highways as long as the car is kept below the current speed limit. When switching to city driving, a different pattern should be applied, reducing the autonomy of the task execution resource.

2.5 Problems

In the above contexts, the following problems arise:

- *How to ensure clarity in the task execution instructions?* If all decisions concerning who does what, what needs to be done, and when to do it are already taken, the task execution instructions leave no doubt about these aspects. If on the other hand some kind of negotiation or delayed decisions concern any of the previous aspects, then some ambiguity and uncertainty about which task will be executed should be expected.
- *How to assign work to a dynamic set of execution resources?* A planner with complete and up-to-date knowledge about which resources are available can pick the most suitable one and assume it will be ready to run the given task. When many resources come and go, it becomes increasingly difficult to keep track of them in a centralized location.
- *How to ensure tasks will be completed within a given time? How to schedule tasks with hard to predict durations?* The execution time of a task is affected by the nature of the task itself, by the performance of the execution resource it has been assigned to, by the input data it should process. While it is not always possible to precisely predict the task completion time, it is possible to express some expectation about the task duration within the original plan. This expectation can be communicated as a due date (or deadline) to the execution resource.
- *How to efficiently use every available resource?* While resource may not share the same skills, capabilities and performance level, leaving some resource idle may affect the overall task execution throughput.
- *How to assign scheduled tasks to resources with unknown skills? How to assign tasks to resources with unknown skills and performance?* Keeping track of resource skills adds complexity to the task execution dispatcher, especially when facing a dynamic set of resources. Resources need to advertise their skills and this information should be kept up to date. Likewise, assessing the performance of resources may require to keep detailed logs of previous task execution durations. If the resources have not yet been used, they may

be asked to run some benchmark tasks to obtain a preliminary assessment of their performance.

• *How to share access to limited capacity among execution resources?* Task execution often requires exclusive access to resources needed to perform each task. When such resources have limited capacity and are shared among multiple tasks, bottlenecks may occur due to delays in acquiring locks.

• *How to prioritize different tasks or goals?* While resources can independently pick which task to execute, they can still do so under some scheduling constraint. Having a clear deadline to achieve a goal or complete a task can help execution resources to determine the urgency of their actions. Likewise, to protect a resource, tasks with high penalties for late delivery or missed quality targets will be performed first, while the rest of the tasks with 'nice to have' requirements can be delayed.

• *How to delegate responsibility but keep accountability?* When there is no plan to follow, execution resources are fully responsible for deciding which tasks to perform and when to do so. After all, it is easier to ask for forgiveness rather than asking for permission. Still, even in this case, the outcome of such decisions should be tracked to keep the execution resources accountable.

• *How to balance task execution responsibilities?* Resources may spontaneously volunteer for work or get recruited to execute tasks. In both cases they should agree that the given task is appropriate and the corresponding schedule is feasible.

2.6 Forces

Table 2 summarizes the consequences of selecting each pattern on the following six forces:

Clarity: is the task that needs to be done known in advance? is there a clear deadline to complete it? whenever the chosen task and its deadline result from a dynamic negotiation, which could fail, the clarity of the task execution is lower (+/-) compared to when the task details can be determined well in advance of its execution time (+). Unclear task execution requests leave many doubts about what needs to be done and when to do it, decreasing the efficiency of the task execution resource (-).

Response time: task execution takes time; its duration depends on the task itself, but delays can result from tasks being queued waiting for resources to become available (-). While usually the response time should be minimized, it is also possible to introduce patterns that aim to make the response time predictable or reduce its variance (+).

Autonomy: task execution involves decisions on whether to run a task or skip it, about which task to execute and when to do so. These decisions can be taken in a centralized manner, reducing the autonomy of the task execution resources (-). Still, in most patterns task execution resources may also participate in making some (+/-) or all (+) of these decisions, raising their level of autonomy and independence.

Scheduling Cost: planning for when tasks should be carried out requires to keep track of which resources are available to run them. Collecting and keeping this information about resource availability and load level centrally can be expensive and hard to scale with a high number of resources (-). Autonomous resources which volunteer to perform tasks will do so when they are available, reducing the scheduling overhead (+). When resources can reject tasks that have been scheduled to them, or may need to coordinate with other resources, scheduling becomes more challenging (+/-).

Skill Matchmaking: assigning the task to a suitable resource capable of successfully and correctly completing it requires to match the resource skills with the ones required by the task. Again, collecting and keeping track of resource skills centrally (-) can be expensive and hard to scale with a very large number of resources (even if such information does not often change dynamically). Autonomous resources which volunteer to perform tasks should only do so if they have the required skills (+).

Protocol Simplicity: the interaction between the task planner, the task execution resource and the controller requires to exchange messages to transmit information describing tasks, deadlines, outcomes. Push-based patterns introduce a single task execution request message followed by a message to report the task outcome (+). One extra message is required for pulling tasks (+/-). In other cases, further interactions are needed to negotiate which task can be done, or send reminders about expired deadlines (-). In the simplest case only one message is needed to log the task execution outcome (++).

2.7 Scenario

Workflow management systems and business process execution engines will run complex processes by decomposing them into tasks, which will themselves be executed by a set of automated or human resources [12].

It is possible that fully automated processes make exclusive use of automated task execution resources which are entirely dedicated to the completion of the process [4]. If these resources are a known entity in terms of their skills and performance, plans can be made under the assumption such resources will be fully available when needed. Pre-allocating such resources to rarely run processes may be too expensive, and – in general – to increase efficiency, some tasks may be outsourced to external service providers, which may have different availability schedules and priorities and also be shared among multiple processes.

Human resources raise the level of autonomy even further, together with a larger variability in terms of skills and – for certain tasks – the possibility of dynamically agreeing both on the task content and its delivery schedule. This is also becoming more and more applicable to smart execution resources based on artificial intelligence. While classical software functions are called, software services are invoked, software agents – instead – should be politely asked to perform a task.

	Forces					
Task Execution Pattern	Clarity	Response Time	Autonomy	Scheduling Cost	Skill Matchmaking	Protocol Simplicity
Imperative	+	+	-	-	-	+
Voluntary	+	+	+/-	+	+	+/-
Self-Paced	+/-	-	+/-	+/-	-	+
Deadline-Driven	+	+	+/-	+	-	-
Self Imposed	+	+	+/-	+	+	-
Spontaneous	+/-	-	+/-	+	+	+/-
Synchronized	-	-	+/-	-	+	+
Independent	-	-	+	+	+	+
Optional	+	+	+	+/-	+	-
Elective	+	+	+/-	+	+	-
Facultative	-	-	+	+	+	-
Arbitrary	-	-	+	+	+	-
Lock-Stepped	-	+	+/-	+	+	+
Prioritized	+	+	+/-	+	-	+
Punctual	-	+	+/-	+	+	+/-
Autonomous	-	-	+	+/-	+	++
Passive	+/-	+	+	+	+	-
Active	+/-	+	+	+	+	-

Table 2: Forces and Consequences Overview

While we do not intentionally pre-select which patterns are suitable for human vs. automated resources, one could roughly split them according to whether the resources are passively accepting tasks or actively seeking new tasks to perform, or as mentioned before, whether the resources are capable to negotiate their task assignment.

Unless some incentive or reward is associated with the task execution request, it is not clear why the task execution resource should volunteer to perform a new task (if active), or accept a request to complete a task (if passive).

Having set the type of resource involved, the choice of the pattern can be further guided by the consequences summarized in Table 2. How to make a decision when faced with patterns that share similar quality assessments? For example, the last two patterns “Passive” and “Active” task execution have the same consequences, but are not interchangeable since the first assumes a passive resource, while the second is only applicable in the context of active

resources. The same holds for “Facultative” vs. “Arbitrary”: they have the same consequences (enhanced autonomy, minimal scheduling costs and ease of performing the skill matchmaking) while only the first is applicable in the context of active resources.

2.8 Design Space Overview

There are 2 (who) x 3 (what) x 3 (when) = 18 possible combinations in the design space (Table 3). Each pattern is positioned in one of these combinations. The order of the patterns listed in this paper attempts to traverse the space by keeping similar patterns close.

Decision	Who		What		When	
Role	Plan	Execution	Plan	Execution	Plan	Execution
Imperative	✓		✓		✓	
Voluntary		✓	✓		✓	
Self-Paced	✓		✓			✓
Deadline-Driven	✓		✓		✓	✓
Self Imposed		✓	✓		✓	✓
Spontaneous		✓	✓			✓
Synchronized	✓			✓	✓	
Independent	✓			✓		✓
Optional	✓		✓	✓	✓	
Elective		✓	✓	✓	✓	
Facultative	✓		✓	✓		✓
Arbitrary		✓	✓	✓		✓
Lock-Stepped		✓		✓	✓	
Prioritized	✓			✓	✓	✓
Punctual		✓		✓	✓	✓
Autonomous		✓		✓		✓
Passive	✓		✓	✓	✓	✓
Active		✓	✓	✓	✓	✓

Role	Plan			Execution		
Decision	Who	What	When	Who	What	When
Imperative	✓	✓	✓			
Voluntary		✓	✓	✓		
Self-Paced	✓	✓				✓
Deadline-Driven	✓	✓	✓			✓
Self Imposed		✓	✓	✓		✓
Spontaneous		✓		✓		✓
Synchronized	✓		✓		✓	
Independent	✓				✓	✓
Optional	✓	✓	✓		✓	
Elective		✓	✓	✓	✓	
Facultative		✓	✓	✓		✓
Arbitrary	✓	✓	✓			✓
Lock-Stepped			✓	✓	✓	
Prioritized	✓		✓		✓	✓
Punctual			✓	✓	✓	✓
Autonomous				✓	✓	✓
Passive	✓	✓	✓		✓	✓
Active		✓	✓	✓	✓	✓

Table 3: Decentralized Task Execution Patterns Overview

3 Patterns

3.1 Imperative Task Execution

Context: Passive resources do not know what to do but can do anything following a strict schedule.

Problem: How to ensure clarity in the task execution instructions?

Solution: Completely separate all planning decisions from the task execution.

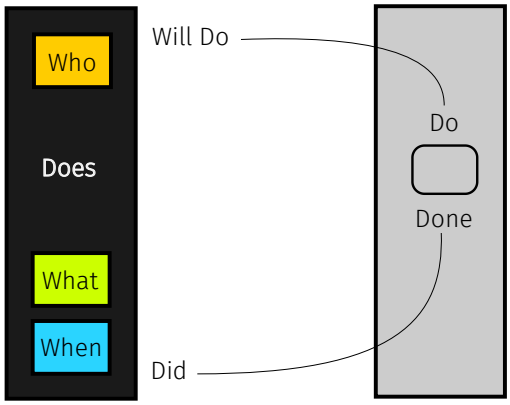

Figure 1: Imperative Task Execution

The chosen task execution resource receives complete instructions on what to do and when to do it, will perform the task following the given schedule and report the task outcome once execution has completed (Fig. 1).

Consequences:

+ **Clarity**: the task execution request defines exactly what to do and when to do it.

+ **Response time**: the task execution is expected to be carried out immediately, as soon as possible, or at least within the given schedule.

- **Autonomy**: the task execution resource cannot freely choose what to do and when to do it, not even whether to perform the task or not.

- **Scheduling Cost**: Information about resource availability needs to be collected and kept up to date to avoid sending task execution requests to unavailable or overloaded resources.

- **Skill Matchmaking**: Information about resource skills needs to be collected and kept up to date to avoid sending task execution requests to unsuitable resources.

+ **Protocol Simplicity**: The task execution request is followed by the outcome report; this is the minimum number of messages required to inform the task execution request about what to do, when to do it and control whether the task has been executed according to the plan.

3.2 Voluntary Task Execution

Context: Active resources can do anything following a strict schedule.

Problem: How to assign work to a dynamic set of execution resources?

Solution: Resources pull the tasks whenever they are available to execute them.

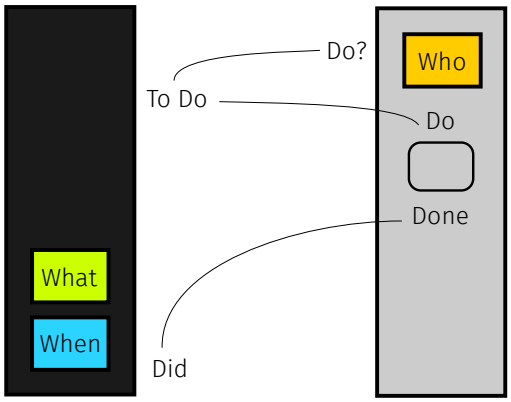

Figure 2: Voluntary Task Execution

As resources become available they ask for something to do, as soon as a task becomes ready, they receive the corresponding task execution instructions defining what needs to be done and when to do it, perform the task following the schedule and report the task outcome once execution has completed (Fig. 2)

Consequences:

+ **Clarity**: the task execution request defines exactly what to do and when to do it.

+ **Response time**: the task execution is expected to be carried out within the given schedule.

+/- **Autonomy**: the task execution resource can freely choose whether to ask for a task or not, but once the task has been assigned there is no choice about what to do and when to do it.

+ **Scheduling Cost**: by definition, resources actively seeking tasks are available to complete them.

+ **Skill Matchmaking**: As part of the request the resource can advertise the provided skills so that a suitable task can be assigned.

+/- **Protocol Simplicity**: The request for a task to be executed is followed by a response with the task description and the corresponding scheduling information. After the task completes, its outcome is reported with the third message.

3.3 Self-Paced Task Execution

Context: Passive resources can do anything at their own pace.

Problem: How to schedule tasks with hard to predict durations?

Solution: Let the execution resources drive the scheduling of tasks.

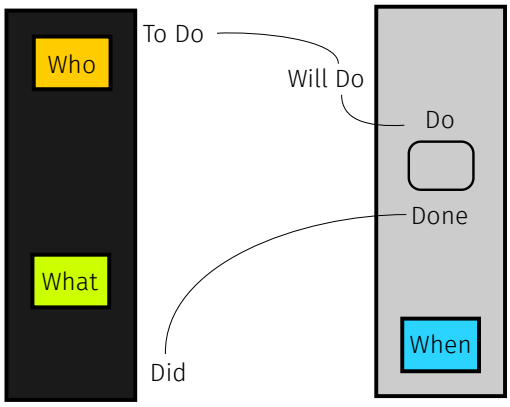

Figure 3: Self-Paced Task Execution

The chosen resource will receive the task instructions but will independently decide when to execute the task. Once the execution is completed it will report back the task outcome (Fig. 3).

Consequences:

+/- **Clarity**: the task execution request defines exactly what to do but not when to do it.

- **Response time**: the task execution will take place as soon as the resource becomes available, making it more difficult to predict when tasks will be completed.

+/- **Autonomy**: the task execution resource can freely choose when to work on the task, but once the task has been assigned there is no choice about what to do and whether to do it or not.

+/- **Scheduling Cost**: resources can queue task requests and service them whenever they can. Since resources will accept jobs even if they are busy, it becomes difficult to keep their load balanced.

- **Skill Matchmaking**: Information about resource skills needs to be collected and kept up to date to avoid sending task execution requests to unsuitable resources.

+ **Protocol Simplicity**: The task execution request only includes the task description, while the response should include the outcome as well as information logging when the task was started and completed.

3.4 Deadline-Driven Task Execution

Context: Passive resources can do anything within a flexible schedule.

Problem: How to ensure tasks will be completed within a given time?

Solution: Give a deadline but let the execution resource decide when to start executing the task.

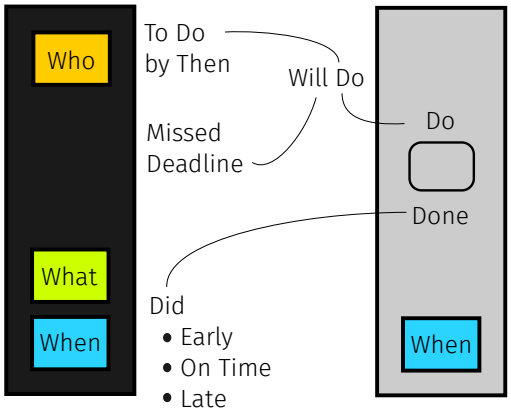

Figure 4: Deadline-Driven Task Execution

The chosen resource will receive the task instructions with an explicit deadline but without a specific schedule defining when to execute the task. The decision on when to start the task execution will be taken locally by the execution resource. If the deadline is missed, a reminder will be sent until the task execution completes. The outcome of the task execution will include information describing whether the task was completed early, on time or late (Fig. 4).

Consequences:

\+ **Clarity**: the task execution request defines exactly what to do and by when it should be done. Deadlines can be hard or soft, defined using some reward/penalty as a function of time.

\+ **Response time**: the task execution is expected to be completed by the given deadline, subject to the availability of the task execution resource.

+/- **Autonomy**: the task execution resource can freely choose when to work on the task, as long as it is completed within the deadline.

\+ **Scheduling Cost**: resources can queue task requests and prioritize them according to their deadline.

\- **Skill Matchmaking**: information about resource skills needs to be collected and kept up to date to avoid sending task execution requests to unsuitable resources.

\- **Protocol Simplicity**: reminders as deadlines are approaching or missed need to be sent.

3.5 Self Imposed Task Execution

Context: Active resources can do anything within a flexible schedule.

Problem: How to ensure tasks will be completed within a given time?

Solution: Give a deadline but let the execution resource decide when to start executing the task.

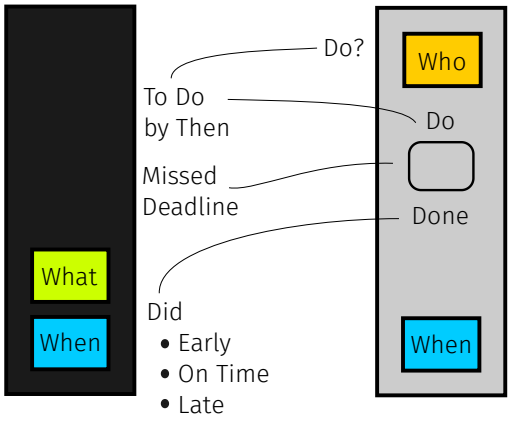

Figure 5: Self Imposed Task Execution

When available, the resource will pull the task instructions that do not include specific schedule defining when to execute the task but only a deadline for when the results are due. The decision on when to start the task execution will be taken locally by the execution resource. If the deadline is missed, a reminder will be sent until the task execution completes. The outcome of the task execution will include information describing whether the task was completed early, on time or late (Fig. 5).

Consequences:

\+ **Clarity**: the task execution request defines exactly what to do and by when it should be done.

\+ **Response time**: the task execution is expected to be completed by the given deadline, subject to the performance of the task execution resource.

+/- **Autonomy**: the task execution resource can freely choose when to work on the task, running the risk of missing the deadline.

\+ **Scheduling Cost**: resources can fetch new tasks depending on their availability.

\+ **Skill Matchmaking**: information about resource skills can be advertised by the resources so that tasks are sent to resources capable of processing them.

\- **Protocol Simplicity**: in addition to the basic task execution request and response, also reminders need to be sent as deadlines are approaching or missed.

3.6 Spontaneous Task Execution

Context: Active resources can do anything at their own pace.

Problem: How to schedule tasks with hard to predict durations?

Solution: Let the execution resources drive the scheduling of tasks.

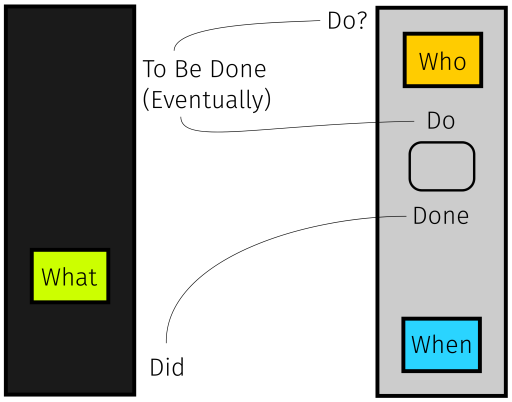

Figure 6: Spontaneous Task Execution

When available, the resource will pull the task instructions that neither include any scheduling information, nor a deadline. The decision on when to start the task execution will be taken locally by the execution resource. The outcome is reported once the execution eventually completes (Fig. 6).

Consequences:

+/- **Clarity**: the task execution request defines exactly what to do but not when it should be done.

- **Response time**: the task execution is expected to be eventually completed.

+/- **Autonomy**: the task execution resource can freely choose when to work on exactly the given task.

+ **Scheduling Cost**: resources can fetch new tasks depending on their availability and can plan when to execute them based on their local schedule.

+ **Skill Matchmaking**: Information about resource skills can be advertised by the resources so that tasks are sent to resources capable of processing them.

+/- **Protocol Simplicity**: The response to the request for tasks to be executed only carries the task description, while the third message reporting the outcome should also include the start and completion timestamp of the task.

3.7 Synchronized Task Execution

Context: Passive resources do what they need to do following a strict schedule.

Problem: How to share access to limited capacity among execution resources?

Solution: Determine the schedule but not the tasks that should be performed.

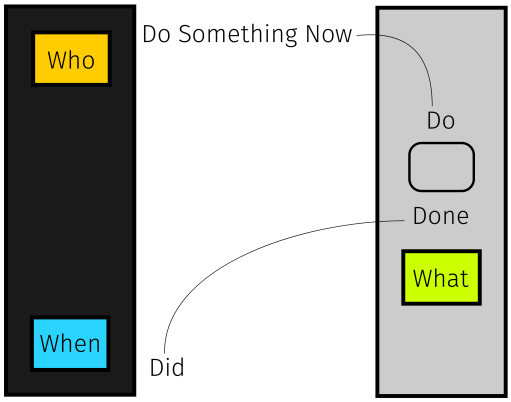

Figure 7: Synchronized Task Execution

The chosen resource is scheduled for execution but the decision of what needs to be done is left with the resource itself so that it can take full advantage of the given execution time slot (Fig. 7).

Consequences:

- **Clarity**: the task execution only triggers the resource to start running some task, without specifying which task.
- **Response time**: the task execution is expected to be completed as soon as possible.
+/- **Autonomy**: each resource can freely choose which task to perform, but it cannot refuse to work during the given time slot.
- **Scheduling Cost**: resources are activated following a centrally managed schedule. Tasks are expected to fit within the given time slots.
+ **Skill Matchmaking**: Resources can pick tasks they are capable of successfully completing.
+ **Protocol Simplicity**: the task execution request does not carry the task description but only information about the schedule. Optionally the outcome of the task execution can be returned.

3.8 Independent Task Execution

Context: Passive resources do what they need to do at their own pace.

Problem: How to ensure all available resources are used?

Solution: Let each resource do something sometime.

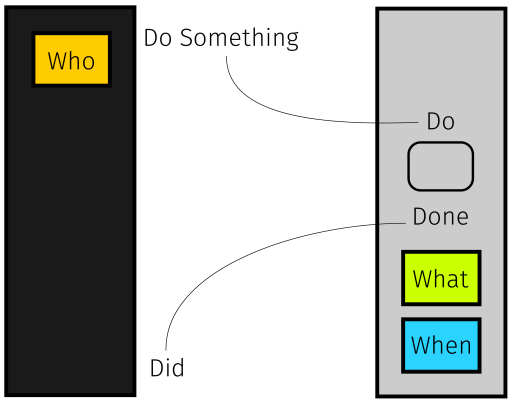

Figure 8: Independent Task Execution

The chosen resource is invoked to perform at their earliest convenience some task chosen independently. When the task is eventually completed, the outcome is reported (Fig. 8).

Consequences:

- **Clarity**: the task execution only triggers the resource to start running some task, without specifying which task or when to complete it.
- **Response time**: the task execution is expected to be completed eventually.
+ **Autonomy**: each resource can freely choose which task to perform and when to run them, but cannot refuse to work.
+ **Scheduling Cost**: Tasks can be queued by the resource which can independently plan what to do and when to do it.
+ **Skill Matchmaking**: Resources can themselves pick tasks they are capable of successfully completing.
+ **Protocol Simplicity**: the task execution request simply signals the resource to start doing something.

3.9 Optional Task Execution

Context: Passive resources do only what they can following a strict schedule.

Problem: How to assign scheduled tasks to resources with unknown skills?

Solution: Let resources accept or reject tasks assigned to them.

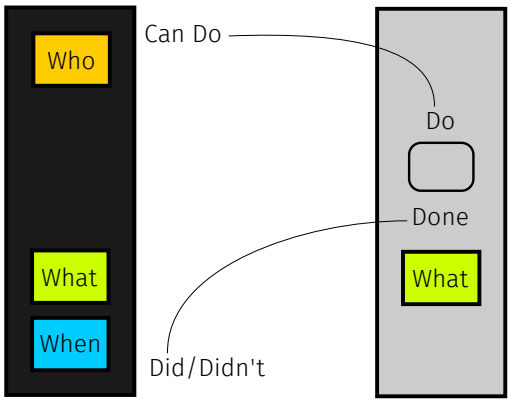

Figure 9: Optional Task Execution

The chosen resource is offered the choice to execute the given task within the given schedule. The resource should decide on whether or not to accept the given task. The decision can be made based on whether the task matches the resource skills, performance or availability timeframe. The outcome of the task execution includes whether the resource accepted or rejected the task (Fig. 9).

Consequences:

\+ **Clarity**: the task execution request defines what to do and when to do it. It is not possible to predict whether resources will accept such requests.

\+ **Response time**: if accepted, the task execution is expected to be completed within the given timeframe.

\+ **Autonomy**: each resource can freely choose whether to accept the task execution request or not.

+/- **Scheduling Cost**: Decisions on when to perform tasks are performed centrally, therefore sufficient information about resource availability needs to be kept. Since resources can refuse to run tasks, scheduling mistakes can be corrected.

\+ **Skill Matchmaking**: Resources can accept only tasks they are capable of successfully completing.

\- **Protocol Simplicity**: the response needs to report whether the task was accepted and completed or rejected. An additional message could be introduced to signal that a task has been accepted.

3.10 Elective Task Execution

Context: Active resources do only what they can following a strict schedule.

Problem: How to assign scheduled tasks to resources with unknown skills?

Solution: Let resources propose tasks that can be assigned to them.

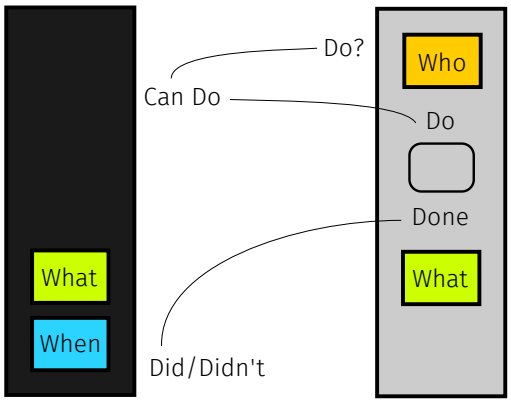

Figure 10: Elective Task Execution

When available, the resource proposes possible tasks that could be executed. A task is agreed upon and sent back to the resource to complete within a certain schedule. The resource may reject the task due to scheduling conflicts (Fig. 10).

Consequences:

\+ **Clarity**: the tasks offered by the resource need to be identified without ambiguity so that they can be chosen by the planner.

\+ **Response time**: if accepted, the task execution is expected to be completed within the given timeframe.

+/- **Autonomy**: each resource can freely choose when to ask for more work and whether to accept new tasks or not. However, they need to complete the task within the given time slot.

\+ **Scheduling Cost**: When available, resources can peek into the central task queue for work and determine whether they will be able to meet the schedule associated with each task.

\+ **Skill Matchmaking**: Resources can accept only tasks they are capable of successfully completing.

\- **Protocol Simplicity**: the response needs to report whether the task was accepted and completed or rejected. An additional message could be introduced to signal that a task has been accepted.

3.11 Facultative Task Execution

Context: Active resources do only what they can at their own pace.

Problem: How to assign tasks to resources with unknown skills and performance?

Solution: Let resources schedule tasks that can be assigned to them.

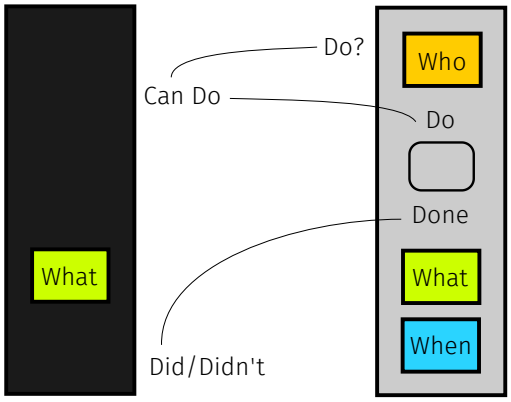

Figure 11: Facultative Task Execution

When available, the resource proposes possible tasks that could be executed. After a task is agreed upon, the resource will eventually complete it. The negotiation on which task to do may fail (Fig. 11).

Consequences:

- **Clarity**: the actual task is dynamically negotiated starting from a set of possible tasks.
- **Response time**: if the negotiation is successful, the task is eventually completed. Carrying out the negotiation delays the start of the task execution.
+ **Autonomy**: each resource can freely choose when to offer to work and when to perform the agreed upon task. The choice of what to do is constrained.
+ **Scheduling Cost**: When available, resources can propose to perform some tasks at their earliest convenience.
+ **Skill Matchmaking**: Resources propose to perform tasks they are actually capable of successfully completing.
- **Protocol Simplicity**: messages exchanged during the negotiation phase need to carry information about a set of possible tasks. The outcome of the negotiation may fail.

3.12 Arbitrary Task Execution

Context: Passive resources do only what they can at their own pace.

Problem: How to assign tasks to resources with unknown skills and performance?

Solution: Let resources schedule tasks that can be assigned to them.

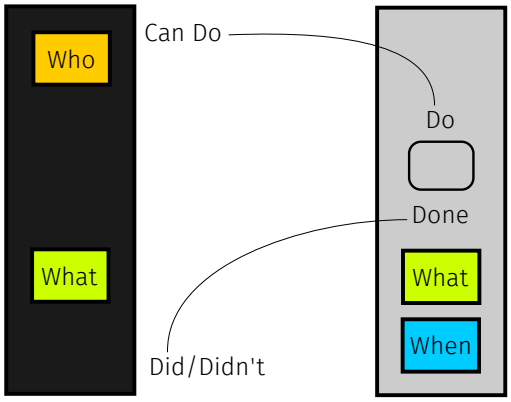

Figure 12: Arbitrary Task Execution

The chosen resource is offered one or more possible tasks to execute. If a suitable task is selected, it will eventually be completed. The negotiation on which task to do may fail (Fig. 12).

Consequences:

- **Clarity**: the actual task is dynamically negotiated starting from a set of possible tasks.
- **Response time**: if the negotiation is successful, the task is eventually completed. Carrying out the negotiation delays the start of the task execution.
+ **Autonomy**: after receiving the invitation to pick some task, each resource can freely choose which task to perform among the possible ones and when to perform the agreed upon task.
+ **Scheduling Cost**: Resources can plan when to perform the task they have been requested to execute at their earliest convenience.
+ **Skill Matchmaking**: Resources select tasks they are actually capable of successfully completing.
- **Protocol Simplicity**: messages exchanged during the negotiation phase need to carry information about a set of possible tasks. The outcome of the negotiation may fail.

3.13 Lock-Stepped Task Execution

Context: Active resources do what they need to do but can follow a strict schedule.

Problem: How to share access to limited capacity among execution resources?

Solution: Determine the schedule but not the tasks that should be performed.

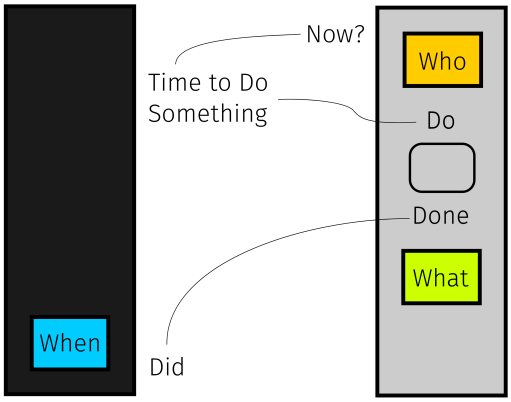

Figure 13: Lock-Stepped Task Execution

When available, the resource asks for the next available time window. It performs the independently chosen task following the given schedule and reports the task outcome (Fig. 13)

Consequences:

- **Clarity**: while the window of opportunity is known, the actual task will be selected dynamically by the active resource.

+ **Response time**: the task must complete within the given time window.

+/- **Autonomy**: while the execution time window is centrally managed, the decision of executing a task and the choice of which task to execute is taken independently by the resource.

+ **Scheduling Cost**: All coordination between task execution resources concerning when they can perform the tasks is managed centrally, making it easier to control access to a shared resource with limited capacity.

+ **Skill Matchmaking**: Resources independently select tasks they are actually capable of successfully completing.

+ **Protocol Simplicity**: messages exchanged only need to carry information about the time slot assigned to the resource asking when it can perform its work.

3.14 Prioritized Task Execution

Context: Passive resources do what they need to do within a flexible schedule.

Problem: How to prioritize different goals?

Solution: Set a deadline but do not constrain what needs to be done.

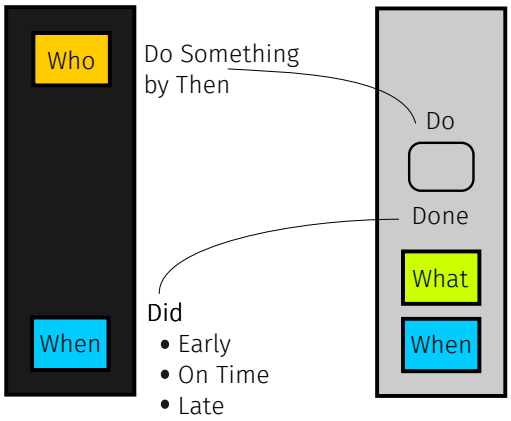

Figure 14: Prioritized Task Execution

The chosen resource is given a deadline to achieve a certain goal. The decision on what to do and when to do it is performed locally and the task execution outcome is reported and evaluated against the deadline (Fig. 14).

Consequences:

\+ **Clarity**: the goal and the time to achieve it should be specified in the task execution request, leaving the specialized decision on what task to execute to achieve the goal to the execution resource.

\+ **Response time**: if possible, the goal must be achieved within the given time window.

+/- **Autonomy**: the resource can pick the correct task needed to achieve the goal and plan the schedule to complete it before the given deadline.

\+ **Scheduling Cost**: While the deadline set needs to be achievable, the execution resource can delay the start of the execution to give priority to other more urgent or more important tasks.

\- **Skill Matchmaking**: The chosen resources should know how to achieve the given goal.

\+ **Protocol Simplicity**: the task execution request only needs to carry information about the time slot assigned to the resource, while the response should identify the task that was executed.

3.15 Punctual Task Execution

Context: Active resources do what they need to do within a flexible schedule.

Problem: How to prioritize different tasks?

Solution: Set a deadline but do not constrain what needs to be done.

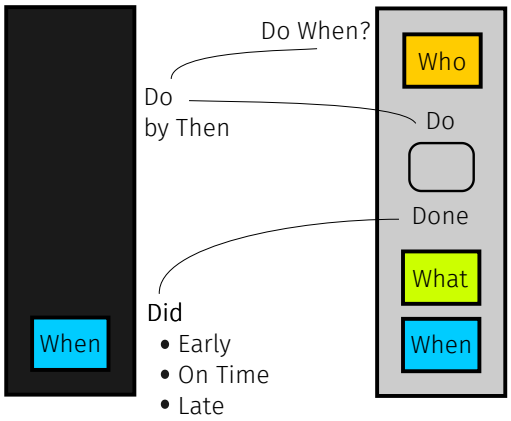

Figure 15: Punctual Task Execution

When available, the resource is informed about the deadline to complete a proposed task. The decision on what to do and when to do it is performed locally and the task execution outcome is reported and evaluated against the deadline (Fig. 15).

Consequences:

- **Clarity**: it should be possible to dynamically estimate the time by when any possible task should be completed.

+ **Response time**: if possible, the goal must be achieved within the given time window.

+/- **Autonomy**: after proposing the task, the resource is expected to plan the schedule to complete it before the given deadline.

+ **Scheduling Cost**: While the deadline set needs to be achievable, the execution resource can delay the start of the execution to give priority to other more urgent or more important tasks.

+ **Skill Matchmaking**: Resources propose tasks they are actually capable of successfully completing.

+/- **Protocol Simplicity**: the task execution reply only needs to carry information about the time slot assigned to the resource, while the final outcome report should identify the task that was executed.

3.16 Autonomous Task Execution

Context: Active resources do what they need to do at their own pace.

Problem: How to delegate responsibility but keep accountability?

Solution: Report and log the task execution outcome.

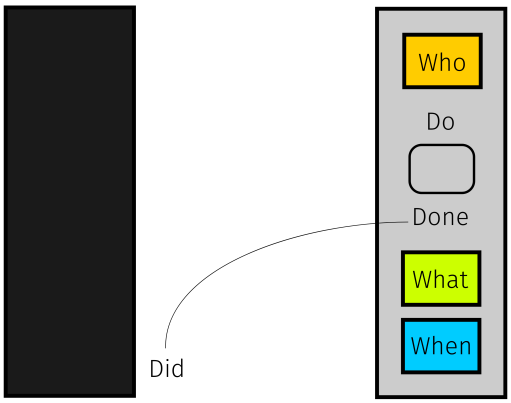

Figure 16: Autonomous Task Execution

When available, the resource makes all task execution and scheduling decisions autonomously. The task execution outcome is reported and logged so that a trace is kept about who did what when (Fig. 16).

Consequences:

- **Clarity**: what was done by whom and when it happened becomes known only after the fact.
- **Response time**: there are no constraints set on the completion time of any task.
+ **Autonomy**: the execution resource is fully in charge regarding what to do and when do it.
+/- **Scheduling Cost**: While the execution resource can independently schedule its tasks, coordination among multiple resources becomes challenging.
+ **Skill Matchmaking**: Resources select tasks they are actually capable of successfully completing.
++ **Protocol Simplicity**: Only a log of completed tasks and their outcome needs to be kept.

3.17 Passive Task Execution

Context: Passive resources only do what they can within a flexible schedule.

Problem: How to balance task execution responsibilities?

Solution: Negotiate both what to do and when to do it.

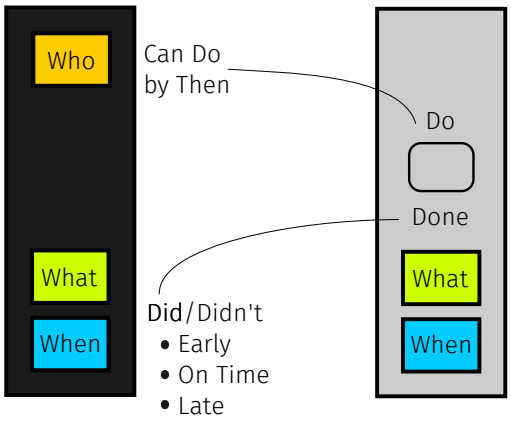

Figure 17: Passive Task Execution

The chosen resource receives an offer to perform a task by the given deadline. The resource may accept the task and autonomously schedule it for execution or reject the task as it does not match its skills or does not fit in the existing schedule. The outcome is reported and evaluated against the deadline (Fig. 17).

Consequences:
+/- **Clarity**: while the task and its time window are known in advance, they can be accepted or rejected by the execution resource at run time.
+ **Response time**: if possible, the task must be completed within the given time window.
+ **Autonomy**: after receiving the task execution request, the execution resource can reject it (e.g., the deadline is not realistic).
+ **Scheduling Cost**: If accepted, the task can be independently scheduled within the given time window.
+ **Skill Matchmaking**: Resources accept only tasks they are actually capable of successfully completing.
- **Protocol Simplicity**: The messages need to carry task definitions together with their deadlines and the protocol should support both acceptance and rejection of tasks.

3.18 Active Task Execution

Context: Active resources only do what they can within a flexible schedule.

Problem: How to balance task execution responsibilities?

Solution: Negotiate both what to do and when to do it.

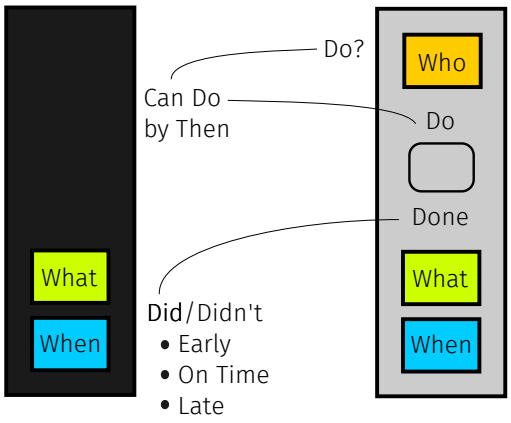

Figure 18: Active Task Execution

When available, the resource proposes to perform a task. The resource is given a deadline to complete the task and autonomously schedules it for execution. The outcome is reported and evaluated against the deadline. The negotiation on which task to do may fail (Fig. 18).

Consequences:

+/- **Clarity**: while the task and its time window are known in advance, they can be accepted or rejected by the execution resource after fetching a new task.

\+ **Response time**: if possible, the task must be completed within the given time window.

\+ **Autonomy**: after retrieving the next task, the execution resource can reject it (e.g., its deadline is not realistic).

\+ **Scheduling Cost**: If accepted, the task can be independently scheduled within the given time window.

\+ **Skill Matchmaking**: Resources accept only tasks they are actually capable of successfully completing.

\- **Protocol Simplicity**: The messages need to carry task definitions together with their deadlines and the protocol should support both acceptance and rejection of tasks.

4 Conclusion

Decentralization of task execution opens up the opportunity to explore where decisions on who does what when should be taken. As opposed to centrally planning how goals should be achieved and unilaterally determining how to schedule, perform and track the corresponding task execution, we have shown that there is an entire spectrum of 18 patterns between *imperative* and *autonomous task execution*, i.e., representing many valid alternatives to choose from between fully centralized and decentralized task execution.

To keep the complexity of the design space under control we did not discuss how data is exchanged so that it can flow between tasks, another dimension in which decentralization can have a major impact. Likewise a hybrid option between active and passive resources may be considered as well. The decision of how to decompose tasks into their constituent sub-tasks can also offer additional opportunities for further (recursively) decentralizing their execution.

Acknowledgements

The author would like to thank the AsianPLoP 2024 writers' workshop participants for their invaluable feedback: Anja Bertels, Y C Cheng, Martin Gutsche, Kiro Harada, Ali Shaukat, Joseph Yoder.

References

[1] Baran, P.: On Distributed Communications: I. Introduction to Distributed Communications Networks. RAND Corporation, Santa Monica, CA (1964). https://doi.org/10.7249/RM3420

[2] Evermann, J., Kim, H.: Workflow management on proof-of-work blockchains: Implications and recommendations. SN Computer Science **2**, 1–22 (2021)

[3] Hofstede, G.: The poverty of management control philosophy. The Academy of Management Review **3**(3), 450–461 (1978), *http://www.jstor.org/stable/257536*

[4] Jablonski, S., Schamburger, R., Hahn, C., Horn, S., Lay, R., Neeb, J., Schlundt, M.: A comprehensive investigation of distribution in the context of workflow management. In: Proc. 8th International Conference on Parallel and Distributed Systems, ICPADS 2001. pp. 187–192 (2001)

[5] Laudon, K.C., Laudon, J.P.: Management information systems: Managing the digital firm. Pearson (2004)

[6] Mengistu, T.M., Che, D.: Survey and taxonomy of volunteer computing. ACM Comput. Surv. **52**(3) (July 2019), *https://doi.org/10.1145/3320073*

[7] Muth, P., Wodtke, D., Weissenfels, J., Dittrich, A.K., Weikum, G.: From centralized workflow specification to distributed workflow execution. Journal of Intelligent Information Systems **10**, 159–184 (1998)

[8] Russell, N., Van Der Aalst, W.M., Ter Hofstede, A.H., Edmond, D.: Workflow resource patterns: Identification, representation and tool support. In: Proc. 17th International Conference on Advanced Information Systems Engineering, CAiSE 2005. pp. 216–232. Springer (2005)

[9] Schneider, N.: Decentralization: an incomplete ambition. Journal of cultural economy **12**(4), 265–285 (2019)

[10] Shirazi, B.A., Kavi, K.M., Hurson, A.R.: Scheduling and load balancing in parallel and distributed systems. Wiley - IEEE computer society press (May 1995)

[11] Siggelkow, N., Levinthal, D.A.: Temporarily divide to conquer: Centralized, decentralized, and reintegrated organizational approaches to exploration and adaptation. Organization science **14**(6), 650–669 (2003)

[12] Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, 3rd edn. (2019)