9. Resource Management

This section first describes some underlying principles for resource management abstractions in Joule. It then describes abstractions for resource encapsulation and ownership, the foundations for resource management. Finally, it describes market-based resource management abstractions for making resource trade-offs in complex systems.

9.1. Resource Management Fundamentals

This section describes some underlying principles in the design of resource management abstractions. The first two principles of hierarchical ownership and drawing authority are demonstrated in Chapter 6, the hierarchical accounts example.

9.1.1. Hierarchical Ownership

Hierarchical ownership makes Joule's resource management abstractions recursively applicable. It permits reuse of mechanisms within an entity without the entity losing control of its pieces. This section will explain in detail why hierarchical resource ownership is important, and gives real-world examples of its use (renters and landlords).

9.1.2. Drawing Authority

The naive way of sharing a resource among consumers is to divide it up (to allocate it). This allocation assumes prior knowledge of how the resource will be used. Allotting budgets to the consumers allows the programmer the same control over the limits of consumption, without requiring prior knowledge of resource utilization. This section will describe budgeting drawing authority in more detail, describe how it subsumes allocation, and give examples that motivate the shift to drawing authority.

9.1.3. Quantity vs. Territory

Quantity and territory are two extremes for measuring or representing access to resources. *Quantity* represents an amount of some *fungible* resource (a resource whose units are all equivalent). *Territory* represents a particular piece of some resource, analogous to real estate. Many computational resources can be represented both ways (most memory pages are fungible, for instance), and these representations are useful for different things. This section will describe the distinction and give examples.

9.2. Primitive Resources

The two fundamental computational resources are execution time and memory. Management of other resources can be built in the language, but these require support in the language *implementation*. This section will describe the primitives for reifying and encapsulating these two primitive resources, and give examples of using them.

9.2.1. Meters and Engines

Meters and Engines are two tools for encapsulating execution time. Meters support ownership of quantities of compute time; Engines support ownership of "territories" of compute time. Engines are provided to support real-time applications.

9.2.2. Space Banks

Space Banks encapsulate computer memory. This section will describe the interface to them.

9.3. Agoric Abstractions

Agoric resource management is the use of markets and prices to manage resources. This section introduces a simple system design and default strategies such that the emergent behavior of such a system is understandable; then presents mechanisms for adding programmer-defined strategies and policies for dynamically adapting to resource availability.

9.3.1. Example System Design

This section will describe a particular system for market-based resource management. It will include the definition of Workers, the virtual machine that runs on money instead of CPU cycles.

9.3.2. Default Strategies

This section will describe the default strategies from which price signals emerge. Properties of default strategies are well described in **[89]**. These policies must:

- result in the emergence of typical system behaviors, such as fairness among processes
- produce price information that reflects resource costs
- remain strategically robust against gaming by non-default strategies
- be reasonably computationally efficient

9.3.3. Using Default Strategies

This section will describe the tools for using the default strategies, including Workers (virtual machines that run on money) and Expense Accounts (budgets for Workers to draw upon). These tools make it easy to divide resources among many services, and call upon existing services that require resources.

9.3.4. Building New Strategies

This section will go under the hood of the system to describe how to build and use more sophisticated strategies for adapting resource usage to the demands of the rest of the computation. These tools are for processes that use price information, and include Agents (strategy elements) and the interfaces to standard resource Providers.

9.4. Improved Computational Model

Here we will present an abstract computational model that includes resource management and a correct state of execution even in the absence of sufficient resources.