

## Intent

Regroup resources under the supervision and control of a responsible entity. This decouples the resources being managed from the management functions operating on them.

## Motivation

This pattern applies in the following context: Imagine a large system of collaborating components that cooperate in order to provide a service, such as a telecommunication network. Such a system is often managed from a central console which controls all the components in the system and which is typically called a *manager*. The problem is to present a unified management interface for such systems, or for subsets thereof.

This problem has to deal with the following forces:

- There can be a large variety of management functions to be performed, increasing the complexity of the manager.
- There can be an extremely large number of components to manage, which might overload any single manager.
- The components can present various types of management interfaces, functionalities, and semantics, which need to be unified somehow.
- It might be desirable to have a portion of the system manage itself in an autonomous manner.

The proposed solution is to first isolate the management functionalities in one or more *manager* objects. Their task is to handle all the management aspects of the system. Then, the system is partitioned into a number of subsystems, which are controlled by individual *agents*. The agents take responsibility for a group of “related” resources (functional nature, logical relationship, manufacturer, etc.). Moreover, they represent their respective subsystems to the managers of the system. Managers and agents can use a single protocol to communicate with one another. Managers may interact with multiple agents within the system in order to handle a given management task. Similarly, an agent may report to more than one manager. This solution can be recursively applied to a large subsystem in order to simplify the agent in charge of it. The agent then becomes a manager for its subsystem.

The result of this solution is twofold. First, the management policy of the system is decoupled from the system itself. Furthermore, management responsibilities may be delegated to subsystems, resulting in a “divide and conquer” approach to system management.

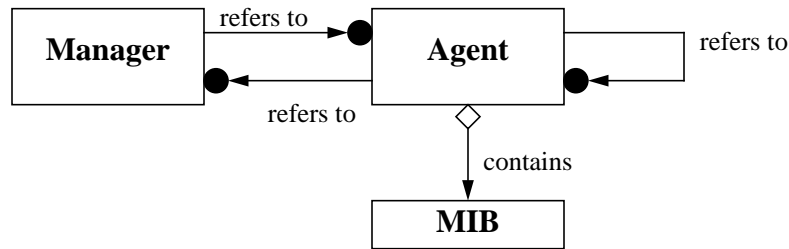
The set of resources grouped under an agent are referred to as a *Management Information Base*, or MIB.

## Applicability

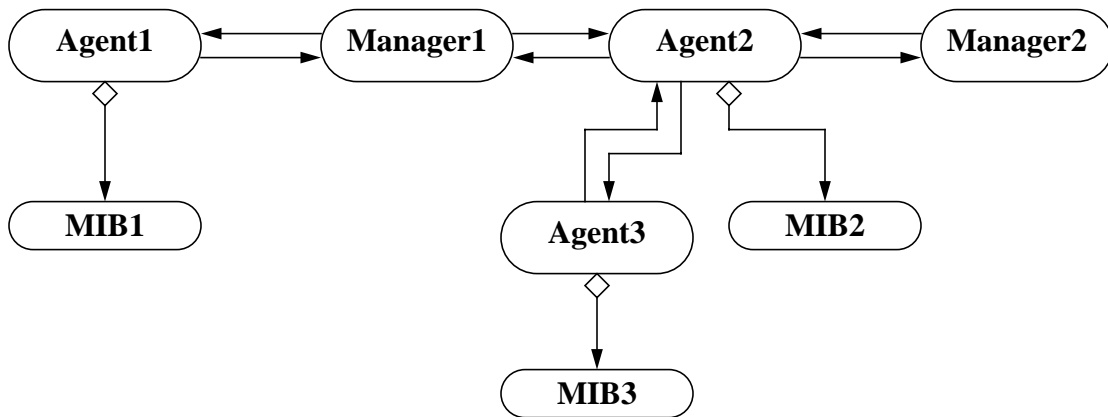
Use the Manager-Agent pattern when there is a large number of low-level resources that need to be handled similarly. The Manager-Agent works best when:

- It is impossible to modify the actual interface of the resources so as to make them conform to a single management protocol. There are many reasons why this could be so, some of them technical, political, or even financial.
- The resources are so numerous and show so much variation that the introduction of individual adapters is not a viable solution.

## Structure



A typical object structure might look like this:

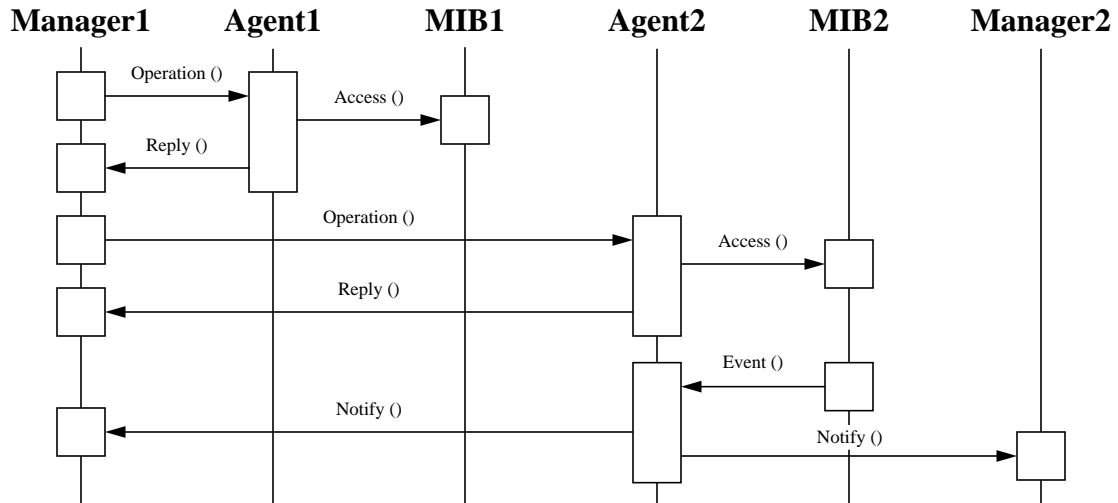


## Participants

- **Manager** (Manager1, Manager2)
  - Responsible for one or more management aspect of the whole system.
  - Sees the systems as a set of Agents.
- **Agent** (Agent1, Agent2, Agent3)
  - Responsible for managing a subset of the resources in the system.
  - The resources may include other Agents.
- **MIB** (MIB1, MIB2, MIB3)
  - The set of resources represented by an Agent.

## Collaborations

- The Manager dispatches management operations to one or more Agents in order to perform its (the Manager's) management functions.
- The Manager collects event reports coming from one or more Agents, and may take some action in response to them (depending on its management function).
- The Agent handles management operations on its resources on behalf of one or more Managers. Those resources may include other Agents.
- The Agent monitors its MIB and/or agents for reportable events and forwards them back to one or more Managers. The Agent may also take some action in response to those events. The resources may include other Agents.



In the above example, MIB1 and MIB2 are coupled to Agent1 and Agent2, respectively, and interact solely with them. Note that Manager1 performs operations on both Agent1 and Agent2. Note also that Agent2 reports to both Manager1 and Manager2. Managers and Agents are thus more loosely coupled than the MIBs with their respective Agents.

## Consequences

The Manager-Agent pattern has the following benefits and liabilities.

1. *The Manager and the Agent use a single protocol to communicate.* This encapsulates the proprietary protocols used by the resources in the Agent and simplifies the implementation of the Manager. The Manager is then able to communicate with any and all the resources in the system, regardless of their origin or nature. Note however that this single protocol, in order to incorporate all the proprietary functionalities, can become quite complex and might introduce a lot of overhead.
2. *The hierarchy of the system is expressed through the Agents.* This means that the Managers do not need to maintain their own maps of the system. Modifications in one area of the system need only be reflected in the relevant Agents, not in the Managers that manage the system.
3. *This pattern is a variation of the client-server architecture.* This architecture closely relates to the client-server architecture, where the agent plays the role of the server and the manager that of the client. But in traditional client-server interactions, all the interactions originate in the client-side, not the server-side. Here, both the agent and the manager can be the originator of an interaction at any given time.

## Implementation

Here are some useful techniques for implementing the Manager-Agent pattern.

1. *The Agent comprises a level of indirection for accessing its resources.* The Agent, in order to portray accurately the MIB, needs either to maintain a special database or to apply a set of translation rules. In the first case, the special database may stock the proprietary values in a format that is legal for the management protocol. This database needs to be kept synchronized with the actual resources. In the second case, translation rules are applied dynamically in order to fulfill requests from a Manager. In both cases, there is an added layer of processing when accessing management data, and this layer can degrade

performance when manipulating large amounts of data. The selected mechanism needs to be implemented with care.

2. *The MIB must be described by architecture-neutral means.* The specification must capture the details of the MIB in a manner that is as generic, yet as precise as possible. This is to ensure the interoperability of distributed components, regardless of their underlying origin or implementation, and to promote a unified view of the system.
3. *The management protocol must be architecture-neutral.* Again, this is to ensure the interoperability of distributed components, regardless of their underlying implementation. The protocol between managers and agents needs to be flexible enough to handle various types of data, regardless of the actual implementation of the data.
4. *The relationships between managers and agents must be maintained adequately.* One solution is for each manager and each agent to maintain a list of its collaborating opposites, but such a solution is inflexible. Instead, the Mediator [GHJV94] pattern may be used to maintain all these relationships. Moreover, the Remote Operation pattern may be used to provide location transparency. Finally, the Broker [BMR<sup>+</sup>] pattern can be used to provide both at the same time.

### Known Uses

This pattern can be found in both CMIS, a part of OSI [TS92], and SNMP, a part of Internet [Ros91]. Both are flavors of NMIs. CMIS has been successfully implemented in two independent NMI frameworks: Layla and OSIMIS, the latter being from the University College of London [PKMB95].

### Related Patterns

**Observer:** The Observer [GHJV94] pattern could be used to implement the notification messages that travel from the agents to the managers. In current implementations however, all data regarding notification is carried in the initial message, in order not to overburden the system with such data transfers during later execution.

**Remote Operation:** The set of interactions between agents and managers (the communication protocol) can be defined using instances of the Remote Operation pattern.

**Mediator:** The Mediator [GHJV94] pattern can be used to handle the communication between the managers and the agents. In this way, each side doesn't have to possess any knowledge of the other. New managers or new agents can be inserted in the system without affecting the already existing managers and agents.

**Broker:** The Broker [BMR<sup>+</sup>] pattern adds the concepts of service negotiation and location transparency, which are not explicitly addressed by the Manager-Agent pattern. A broker can be inserted between the managers and the agents. The broker can then direct manager operations and agent notifications to their proper destination. Operations can be routed to equivalent agents without involving the emitting manager in the selection process, or they can even be broadcast to multiple agents. The same goes for agent notifications. The broker can also take care of aggregating multiple responses. In essence, it incorporates characteristics of both the Remote Operation pattern and the Mediator pattern.

- Reactor**: The event loop that drives the managers and the agents can be viewed as an instance of the Reactor [Sch95] pattern.
- Composite**: Complex parameters in the operations between managers and agents can be viewed as instances of the Composite [GHJV94] pattern.

## References

- [BMR<sup>+</sup>96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture - A System of Patterns*. John Wiley and Sons, 1996.
- [GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
- [PKMB95] George Pavlou, Graham Knight, Kevin McCarthy, and Saleem Bhatti. The OSIMIS platform: Making OSI management simple. In Adarshpal Sethi, Yves Raynaud, and Fabienne Faure-Vincent, editors, *Integrated Network Management IV*, pages 480–493. Chapman and Hall, 1995.
- [Ros91] Marshall T. Rose. *The Simple Book: An Introduction to Management of TCP/IP-based Internets*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1991.
- [Sch95] Douglas C. Schmidt. Reactor: An object behavioral pattern for concurrent event demultiplexing and event handler dispatching. In James O. Coplien and Douglas C. Schmidt, editors, *Pattern Languages of Program Design*, chapter 29, pages 529–545. Addison-Wesley, 1995. (Reviewed Proceedings of the First International Conference on Pattern Languages of Programming (PLoP'95), Monticello, IL, 1994).
- [Tes96] Jean Tessier. An application framework for OSI network management interfaces. Master's thesis, Université de Montréal, Montreal, Quebec, Canada, April 1996. In French.
- [TK97] Jean Tessier and Rudolf K. Keller. Manager-Agent and Remote Operation: Two key patterns for network management interfaces. In Collected Papers from the PLoP'96 and EuroPLoP'96 Conferences, Washington University Department of Computer Science, wucs-97-07, pages 4.8.1-4.8.14, February 1997.
- [TS92] Adrian Tang and S. Scoggins. *Open Networking with OSI*. Prentice-Hall, Inc., 1992.

## Contacts

<http://www.iro.umontreal.ca/~keller/Layla>

Jean Tessier, AT&T Laboratories, Advanced Technologies Division, Holmdel, NJ.

Jean.Tessier@att.com

Rudolf K. Keller, Département d'informatique et de recherche opérationnelle, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal (Québec) H3C 3J7, Canada.

keller@iro.umontreal.ca