

Site

An interface to a set of related application services.

Table of contents

1 Overview.....	2
1.1 Intent.....	2
1.2 Problem.....	2
1.3 Discussion.....	2
1.4 Structure.....	3
1.5 Example.....	4
1.6 Check List.....	5
1.7 Rules of Thumb.....	5
2 Design.....	5
3 Constraints.....	6
3.1 Allowed Child Dependencies.....	6
3.2 Allowed Parent Dependencies.....	6
4 Attributes.....	6
4.1 Exported Attributes.....	6
5 Commands.....	6
5.1 Start.....	6
5.2 Status.....	7
5.3 Stop.....	8
5.4 Update.....	9

1. Overview

Site: *An interface to a set of related application services.*

1.1. Intent

- Define an object that encapsulates a set of related Services that comprise an integrated software system.
- Provide a unified interface used to coordinate a set of Service objects to simplify the management of the software release process and runtime state control.
- Abstract the coordination of objects which may be distributed over a network hosts.
- Wrap a set of complicated procedures with a simple command.

1.2. Problem

We want to control the deployment and runtime state of distributed application services, but each service must be updated and controlled individually, yet there is an implicit procedural order. The procedural order should reflect the runtime dependency hierarchy that is often inherent to sets of integrated software services.

1.3. Discussion

To an end user, the services of an application may appear to be supported by a single component, but often times this impression is really the result of a large set of diverse components interacting with one another. These interactions are defined in terms of a software configuration that describes the communication integration points. Upon startup, the application components connect to their colleagues. The implementations of the application service components dictate how they will behave if they fail to connect to their external dependency; sometimes logging error messages and other times, entering a hung state.

Because services can be distributed differently depending on environment, the procedures used to perform an orderly deployment and restart vary widely. For example, in a development or test environment, services might be installed on two machines, while in a staging or production environment, services might be spread over tens or even hundreds of machines.

Finally, depending on the task at hand, the administrator will sometimes need to perform actions that are isolated to a subset of the services while in other cases the administrator will have to act on all of the services.

The diagram below describes two scenarios. In the first scenario, the blue figure, admin 1,

wants to stop just the web servers. This requires the admin knows which nodes the web servers reside, logging into each machine and executing the required command. In the second scenario, the orange user, admin 2, needs to update the deployments of all the servers. This requires the admin know which machines the deployments reside, and which order to stop servers after their software has been updated.

These scenarios highlight the following considerations:

- network abstraction: a means to abstract the location of the objects deployed within the network
- sequencing: a means to describe the ordering of procedures that recognize the runtime state dependency of the software components.

An approach that accommodates these considerations would be an abstraction of a set of integrated software services that both hides the differences in how services are deployed in different environments and encapsulates the sequencing logic for each of the routine high level procedures.

This offers several advantages. Administrators can approach managing a set of services as a single entity, use the same procedures in different environments without knowing beforehand how services are deployed in the network, and not have to know the runtime dependencies of the underlying services.

The diagram above illustrates the role the Site abstraction plays.

The Site object provides a standard encapsulation of routine procedures such as Stop, Start, Status and Update, leaving the details for how these procedures should be coordinated to implementations of derived classes.

Instead of each administrator invoking the correct succession of commands on each node (either locally or via a remote shell session), they request the Site object to perform those actions.

Lastly, because the Site object provides a unified higher-level interface to the set of related Services, it wraps a complicated set of actions with a simpler interface that makes the procedure easier to perform. Of course, it does not replace the procedures for the underlying Services, enabling the user to still manage objects individually when desired.

1.4. Structure

The Site type inherits the command dispatching capabilities from Mediator but defines standard operations for updating and managing the runtime state for a set of related Service objects.

1.5. Example

Runtime State Control

The Site type defines an object that provides methods to manage the runtime state and deployment of a set of related software services.

Stop

Service objects contain a property called `startup-rank` which can be utilized to order its set of services according to their `startup-rank` values. For example, one can assign smaller numeric values to services which should be started up before others that should be started afterwards.

The following pseudo code describes an algorithm that can be used by the Site's Stop command; it finds all its services, sorts them in descending order, iterating over the results calling the Stop action on each Service object.

```
FOR each service in the result of sortByStartupRank(set: services, order:
descending)
    DISPATCH-CALL Stop on service
END FOR
```

The diagram below describes a scenario where a Site object depends on two kinds of Service objects, web and app. The web objects have startup rank 2 while the app objects have startup rank 1. According to the algorithm above, this indicates that first the web services would be stopped and then the app services, since app services have a lower startup-rank value.

The pseudo code, `DISPATCH-CALL Stop on service`, implies that instead of directly calling the Stop action on the service object, to go through a dispatch process that provides a lookup determining where the object resides in the network. It is the `DISPATCH-CALL` mechanism that will then call Stop either locally or remotely depending on the location of the service object.

Update

Because the structure of an integrated software system can take many forms, each with its own policies governing a consistent runtime state, two examples of the Update command are discussed.

The default logic of the Update sequence for a Site does not assume that the Site remain functional during the update process. In this case, the Site object first instructs all the Services to stop, then install their software dependencies, configure themselves, and then start up. This approach matches the life cycle stages of the Service object itself and is a simple mapping between the high level procedure offered by the Site object and its related

Service object dependencies.

Some integrated software systems cannot be completely shutdown during the deployment process, they must remain functional, albeit in a reduced capacity. In this situation, the Site object's Update workflow must be designed to iteratively update its underlying Service objects. How a set of services can be updated while not interrupting the utility of the application will, of course, depend on the architecture of the whole integrated software system.

The diagram below describes an Update workflow that manages the deployment release process according to application tier. The Update sequence begins by invoking the Update action to the "App" tier, and then the "Web" tier, subsequently invoking the Update action to the whole set of Service objects.

In this example, it is assumed that while each instance of the "App" Service is updated (i.e., stopped, its software installed and configured and then started), the Service objects in the "Web" tier continue to function by connecting to an alternate "App" Service object.

1.6. Check List

- Establish the boundaries of what constitutes the integrated software system you wish to control as a single entity.
- Identify the types of Service objects that comprise the integrated software system.
- Determine the routine deployment and runtime state management procedures needed to maintain the set of Services as single operations.
- Understand the runtime state dependency requirements for updating a live Site. The generic method is the simple - Stop, Packages-Install, Configure, Start sequence. For sites that must remain functional during a deployment, it will be necessary to define a specialized sequence that may target subsets of Service types.

1.7. Rules of Thumb

The intent of Site is to capture the procedures that translate to calls to underlying Service object commands.

2. Design

Super Type [Mediator](#)

Role	Concrete. (Objects can be created.)
Instance Names	Unique

Notification	false
Template Directory	
Data View	Children, proximity: 1
Logger Name	Site

3. Constraints

3.1. Allowed Child Dependencies

- DeploymentSetting
- [Service](#)

3.2. Allowed Parent Dependencies

- Node
- [Updater](#)

4. Attributes

4.1. Exported Attributes

Name	Property
basedir	deployment-basedir
install-root	deployment-install-root

5. Commands

Note:

Commandline options displayed in square brackets "[]" are optional. If an option expects arguments, then angle brackets are shown after the option "<>". Any default value is shown within the brackets.

5.1. Start

starts all services

Usage

Start [-command <Start>]

5.1.1. Workflow

- dispatchCmd

5.1.2. Success Handler

Email	
Subject	[\${context.type}:\${context.name} @ \${framework.node}] \${command.name} - SUCCESS
File	\${modules.dir}/Deployment/templates/notice.html

5.1.3. Error Handler

Notify	Email Subject \${modules.dir}/Deployment/templates/notice.html File \${modules.dir}/Deployment/templates/notice.htm
Report	Workflow failed.

5.1.4. Options

Option	Description
command	<i>command to dispatch</i>

5.2. Status

calls status for each service

Usage

Status [-command <Status>] [-keepgoing]

5.2.1. Workflow

- dispatchCmd

5.2.2. Success Handler

Email	
Subject	[\${context.type}:\${context.name} @ \${framework.node}] \${command.name} -

	SUCCESS
File	<code>\${modules.dir}/Deployment/templates/notice.html</code>

5.2.3. Error Handler

Report	Workflow failed.
--------	------------------

5.2.4. Options

Option	Description
command	<i>command to dispatch</i>
keepgoing	<i>if an error occurs continue workflow</i>

5.3. Stop

stops all services

Usage

Stop [-command <Stop>]

5.3.1. Workflow

1. dispatchCmd

5.3.2. Success Handler

Email	
Subject	<code>[\${context.type}:\${context.name}] @ \${framework.node}] \${command.name} - SUCCESS</code>
File	<code>\${modules.dir}/Deployment/templates/notice.html</code>

5.3.3. Error Handler

Notify	Email Subject <code>\${modules.dir}/Deployment/templates/notice.html</code> File <code>\${modules.dir}/Deployment/templates/notice.html</code>
Report	Workflow failed.

--	--

5.3.4. Options

Option	Description
command	<i>command to dispatch</i>

5.4. Update

run the update cycle

Usage

Update [-command <Update>]

5.4.1. Workflow

1. dispatchCmd

5.4.2. Options

Option	Description
command	<i>command to dispatch</i>