Inversion of Control Rocks

NEED HEAD SHOT

Paul Hammant

Inversion of Control (IoC) is about software components doing what they are told, when they are told. Your OO application could well become unmaintainable without it.

IoC is a pattern that helps teams avoid the dependency hell that results when an application grows into a large pseudo-platform without taking care to adequately decouple logic; that thing that ultimately only a couple of its omnipotent architects or old-lag premium-rate contractors really understand; that system that Heath Robinson and Rube Goldberg might have made together (look them up).

The problem with small working applications that become large is that static-method entanglement does not scale. One part of a system that's otherwise fairly self-contained, statically accesses another part of the system and cannot be instantiated without invoking methods in the latter. Thus it can't easily be isolated for unit testing. What's more, it can't be developed as an independent component. Components that are developed separately can have their own development teams and may well be part of a larger design. These components will have their own source control directory and can be developed against mock implementations of their dependent components. All this will help overall development become faster, in terms of both the team's efficiency and the build time.

As an example, let's take a Personal Information Manager (PIM) application that has UI and persistence elements. The naïve implementation might have inline JDBC statements among graphical code. A componentized application would have that persistence logic separated into a persistence component with user interface logic enshrined in a view-controller component. Clearly, if the two teams developing their respective components agree on a slowly evolving interface/implementation-separated API for persistence, they can develop at their own pace and ship versions of their components whenever it's appropriate.

A third piece, which is not a component, would be the bootstrap for the application. That bootstrap may well be entirely contained in a static main method of a simple class and would merely instantiate the two components, passing one into the other's constructor before invoking setVisible(true) or similar. With the introduction of this bootstrap we can see the control aspect of the IoC pattern.

In a noncomponentized version of our example, the view-controller may well have its own main method, might instantiate a fixed version of the PIM store, or access it via an unnecessary singleton factory (public static methods are generally bad), i.e., the control is very much inside the component in question.

The word "inversion" from the pattern name is about getting control back. The containing application (often a true container or a proper framework rather than a main method) controls when a component is instantiated and which implementations of its dependent components it is passed.

IoC also dictates the configuration of components. A JDBC version of the PIM store above would clearly require some JDBC settings. Classically, developers may write a mechanism to retrieve them from a fixed properties file. IoC would insist that the configuration is passed into the component. In our hypothetical example, it would take it via the constructor and be interface/implementation separated. Thus the configuration is a component. It would be tightly coupled to the component that requires it, but subject to multiple implementations, one of which may be the "from properties file".

Inversion of Control has moved to the center stage in the last six months after a five-year gestation period. There are three types of Inversion of Control. Type 1 uses configuration data (Avalon, JContainer); Type 2 uses bean introspection (Spring, WebWork, HiveMind); Type 3 uses constructor signatures (PicoContainer).

Type 1 versus Types 2 and 3 – components for the former cannot easily be used outside of a container. As such it's appropriate to only talk of Type 1 as a true container requiring design. Type 2 and 3 components can be used outside of a container. They are as close to POJOs as possible. It's the deployer's choice to hard code and embed, or to leverage a container for management. Types 2 and 3 do not require a component to declare their components in the accompanying XML (Type 1 does).

Type 1, Apache's Avalon and JContainer's DNA, uses a series of interfaces to describe whether a component has component dependencies or requires configuration, and whether start/stop life-cycle concepts are pertinent. Many ordinary POJO coders find Type 1 a turnoff.

Type 2 (SpringFramework, HiveMind, WebWork2) uses setters to hand component dependencies and configuration into a component. Type 2 components are said to be more beanlike than Type 1. Applications can be brittle if an important setter is not called.

Type 3 (PicoContainer), like Type 2, also strives to be transparent. It uses constructors instead of settors and this is deliberately brittle, not withstanding the fact that you can have multiple constructors.

For a componentized system that uses singletons or similar for component resolution, the dependency is obscure. In IoC it is declarative. The application would have loosely coupled components and be more scalable, more maintainable, and more testable. It's a very small investment for a very large return. Inversion of Control rocks.

NEEDS TO BE TRIMMED

Paul Hammant is a still-coding architect for ThoughtWorks in London and has been programming professionally since 1989. He was a former committer on Apache's Avalon project, but left and cofounded the open-source PicoContainer project and its sister NanoContainer as well as remaining involved the post-Avalon JContainer project.

E-mail: phammant@thoughtworks.com

SYS-CON MEDIA

President and CEO:
Fuat Kircaali *fuat@sys-con.com*
Vice President, Business Development:
Grisha Davida *grisha@sys-con.com*
Group Publisher:
Jeremy Geelan *jeremy@sys-con.com*

Advertising
Senior Vice President, Sales and Marketing:
Carmen Gonzalez *carmen@sys-con.com*
Vice President, Sales and Marketing:
Miles Silverman *miles@sys-con.com*
Advertising Sales Director:
Robyn Forma *roybn@sys-con.com*
Director, Sales and Marketing:
Megan Ring *megan@sys-con.com*
Advertising Sales Manager:
Alisa Catalano *alisa@sys-con.com*
Associate Sales Managers:
Carrie Gebert *carrieg@sys-con.com*
Kristin Kuhnle *kristen@sys-con.com*

Editorial
Executive Editor:
Nancy Valentine *nancy@sys-con.com*
Associate Editors:
Jamie Matusow *jamie@sys-con.com*
Gail Schultz *gail@sys-con.com*
Jean Cassidy *jean@sys-con.com*
Jennifer Van Winckel *jennifer@sys-con.com*
Online Editor:
Lin Goetz *lin@sys-con.com*

Production
Production Consultant:
Jim Morgan *jim@sys-con.com*