



Introduction to the *featherlite* framework

Author: Michael Gatto, Hansjörg Meier, Martin Smock
Type: Documentation
Date: 2011-02-18
Version: 0.1.2

Abstract

This document explains the core features of the *featherlite* framework and the motivations behind it.

1 Background and Motivation

In the beginning of 2004 we started to build, with a small group of developers, a framework of re-usable components that would allow us to implement projects in planning, detailed scheduling and control for logistics and manufacturing. This development was mainly motivated by the fact that most of the existing software in the area of planning, scheduling and control was either standard software or individually built software to be applied to and optimized for single projects only. Since our goal was to build customer projects on top of a single framework of software components, allowing us to individualize the projects fast and easily, the existing software products did not fit our needs. Thus, we decided to build our own framework of re-usable software components and run-time components, which was the beginning of the development of what has evolved into the *featherlite* framework.

Since all of us had experience with software frameworks (some of it good, a lot of it bad) we decided to build the framework keeping in mind the design principles which we felt to be most important:

Support agile development methods Keep simple things simple, make complex things manageable.

Quick initial setup and starting We don't want a heavy weight infrastructure. We want to be able to install, start and use it quickly.

Lots of components ready to use Even in the area of planning, scheduling and control, lot of work is replicated implementing the same code over and over again like re-inventing the wheel. We want to avoid this by serving the core functions as well as commonly used functions as part of the framework.

Generic extensible object model We don't know what requirements the future will bring. Thus, we want our business objects to be generic and extensible by plugging software components together to build complex objects.

Flexibility It is our feeling that databases are no holy grails, but services to be used according to one's needs. We wanted to be independent of database products and even be able to run without a database at all.

Openness The market in the area of planning, scheduling and control was and still seems to be dominated by proprietary interfaces. We don't want to depend on proprietary software at all and thus use open interface protocols wherever possible.

Interoperable Already in 2004, there were a lot of ready to use infrastructure frameworks around and their number is still growing. Thus we did not want to re-invent the wheel but rather integrate our framework to the existing application servers, service oriented architectures, message queues, etc. Keeping that in mind, we designed the framework to easily integrate to the commonly used open standard infrastructures.

Keeping these principles in mind, we started to implement the core framework and run-time components. Early success in industry projects, where we were able to implement quite large projects with a few developers in only a few months, confirmed our approach. The *featherlite* framework and its run-time components now provide a technically mature basis to implement individualized software solutions. Nevertheless, we continue to work on the framework, both to keep it up to date and to add new functionalities to open new fields of application.

2 What is the *featherlite* framework?

From an application point of view, the *featherlite* framework provides a basis to quickly develop individualized software solutions with focus on — but not limited to — planning, scheduling and execution control in logistics, manufacturing and process driven applications. The *featherlite* framework provides:

1. a framework to build individualized software solutions, mainly in the field of planning, scheduling and execution control in production and logistics,
2. a library of re-usable software components solving common problems in planning, scheduling and execution control,
3. a large set of run-time components with pre-built algorithms,
4. reference implementations, to facilitate the development of individualized solutions.

From a technical point of view, the *featherlite* framework implements the following patterns:

inversion of control We want to provide as many components as possible as part of the framework. Thus we focused on the separation of generic, re-usable and the individualized parts of applications in planning and control context. Now, most of the generic part of the application flow of control is served by the framework components, accessing individualized components via their programming interfaces only.

dependency injection We want to favour configuration against programming, whenever possible. Thus we decided to use tool or file supported configuration up to the registration of individualized components called by the generic application components. Most of the individualized behaviour can be now configured through files or by tools at run-time.

object composition From the very beginning, we intended to use the *featherlite* framework in a broad range of application areas. To do so, we decided to keep the core business objects small and to allow to configure the individualized business objects by composition. Now, a developer can choose from a large set of core components to compose the individualized business objects with the help of configuration files or at run-time using the provided tools.

service architecture¹ In our experience, many applications lack simplicity due to a too high degree of cross linking of functionality. From the beginning, we decided to reduce the application's complexity by using a service-based architecture and a one service per use case concept. This choice has led to a large set of generic and re-usable services, queries and commands available to the developer and to service or command interfaces that one can extend to implement individualized functionality.

modularisation We favour modularisation against parameterization. Thus we designed the framework to be fully modular. For this we organized the code in OSGI plugins, which can either be deployed on a JEE WEB application container or run as standalone application in a Java Run-time Environment. But even inside *featherlite*, the dependency injection of the run-time components required for individualized solutions is used. Now you can select from a set of run-time components for special functionalities (for example Legacy Integration, Shop Floor Integration, Persistence, . . .) and inject them into your application by configuration.

In the past years, the library of components has grown. And since we ourselves prefer to test and study software technologies starting from working examples, *featherlite* comes with a set of reference applications as part of the framework. So, from the beginning, you can install and run the reference applications to test *featherlite* functionality. Now, you can execute and test nearly all of the services and queries provided by *featherlite* using the included rich clients. But you can use the reference implementations as a starting point to implement your own applications (we do so too). For example, with the help of the reference client user interfaces you can

- configure the business objects at run-time,
- configure complex manufacturing processes or material flows at runtime,
- import and export object configurations to files,
- plan and schedule complex processes,
- control manufacturing, logistic and business processes,
- simulate execution of concurrent processes,
- inspect the object model at run-time,
- analyze the processes,

Thus, the reference implementations can be easily used for rapid prototyping, for case studies and to test and verify your individualized application.

¹ This shouldn't be confused with the widely used term 'service oriented architecture (SOA)'. Service oriented architectures usually provide an infrastructure to deploy services, while with *featherlite* we provide services implementing various business functionality, which can be deployed in service oriented architectures

From the early beginning the *featherlite* framework was build with openness in mind. One the one side we exclusively used open standard technologies (Java, XML, OSGI, and others). One the other side, *featherlite* comes with an integration framework. The integration framework is accompanied with a library of components for open standard data interfaces (XML Files, CSV Files, and other) as part of the *featherlite* framework. With this you can easily integrate to the legacy (for example existing ERP systems) or the shop floor environment. The integration framework is concieved to allow manipulation of the data that *featherlite* shares with the environment.

In addition, we offer commercial components for advanced solutions and proprietary interfaces to integrate the *featherlite*-based solutions with the existing environment (IDOC/BAPI interface to exchange data with SAP R3, Beckhoff or Simatic shop floor controllers, and others). You can access these components through the *featherlite* website.

The development of the framework wouldn't have been possible without using third party products. For this we carefully evaluated the available open source software and used them whenever they made our life easier, that is, when they were reliable, simple and well documented. Beyond others, we use the following free or open source technologies, which became very popular or even standard these days:

Java the programming language, the run-time environment and the core libraries used by the framework,

OSGI the plug-in infrastructure helping *featherlite* to be modularized,

Eclipse Rich Client Platform the libraries and components used for the fetaherlite rich clients,

Java Server Pages the libraries and components used for the *featherlite* WEB clients,

Hibernate the framework used to persist *featherlite* core objects in relational databases,

Log4j the well known java logging framework from the Apache Software Foundation,

Dom4j a library used to easily serialize *featherlite* core objects to XML streams.

You may notice, that all third party products used are published under open source licenses (LGPL or cognate licenses), which neither put restrictions on the license of the software using them nor requires license fees to be payed to run.

3 Why use *featherlite*?

3.1 Advantages of using *featherlite*

It's quite hard to summarize and point out the advantages of the *featherlite* framework without falling into the common speech bubbles of marketing. Nevertheless, in what follows we give a short list of advantages of the *featherlite* platform, which we feel (and

experienced in our own projects) to be quite unique in frameworks or software platforms with focus on planning, scheduling and execution control. You can:

1. choose from a large library of software components which can be easily combined to meet your requirements,
2. easily extend the *featherlite* functionality by Java programming,
3. easily install, setup and start to test and to develop,
4. use agile development methods, starting small and grow in time,
5. successfully manage even projects with complex requirements with small teams in short times.

3.2 When should you try *featherlite*?

There are many reasons to use *featherlite*. We have used *featherlite* in various industry projects, with very different scopes. We feel that using *featherlite* is most effective if you

- want to use *featherlite* planning and scheduling functions in a standalone application or as part of your own product,
- want to use *featherlite* execution control functions in a standalone application or as part of your own product,
- need a platform providing object composition configuration to build your own individualized application or running as part of your own application,
- need a platform for rapid prototyping or case studies in planning, scheduling and execution control,
- want to use the existing integration components,
- want to use commercial components based on *featherlite* to meet your needs, thus saving you from a lot of work to address a problem we have already solved for you.

3.3 When should you not use *featherlite*?

Although *featherlite* comes with its own business process modeling language and tools, it's not a pure business process modeling tool. The focus of the *featherlite* modeling is to define processes for planning, scheduling and execution control of concurrent processes. If you are looking for a business process modelling tool and are not interested in planning, scheduling or execution control of concurrent processes, *featherlite* is perhaps not the best choice.

Although *featherlite* comes with an integration framework for synchronous and asynchronous integration, it's not a pure integration platform. The focus of the *featherlite* integration framework is to inter-operate with the environment by transforming legacy or shop floor data to *featherlite* core objects to work with. If you are looking for a platform that only transfers or manages data, without operating on the data, *featherlite* should perhaps not be your first choice.

Although *featherlite*'s execution control strategy covers functions one usually finds in the shop floor controller (PLC), *featherlite* is not intended to replace shop floor controllers in machine control. The philosophy behind the *featherlite* control strategy is to take over the high-level decision making from the PLC to the control system, but leaving the final decision to the shop floor controller. To give an example in the context of logistic, the decision of when a motion should be made may be left to the *featherlite* execution control, but the check of the sensor states to ensure that the motion can actually be performed is left to the controller. This is so, since the controller runs in hard real time (with a guaranteed response time), while *featherlite* runs in soft real time (fast enough, but has no guaranteed response time).

4 How to obtain *featherlite*?

The *featherlite* framework comes in separate plug-ins (modules) which can be downloaded from the *featherlite* homepage at www.featherlite-framework.com.

The core plug-ins are all provided free of charge. With them you can easily install one or more of the *featherlite* reference implementations to study, test and run the tutorials, which are also available for free on the *featherlite* homepage.

Commercial add-ons solving specific problems for you and save your time are also available to download on the *featherlite* homepage. Visit the homepage for descriptions, documentation and to download the plug-ins.

5 How to begin?

Beginning is simple and carries no risks. You can download one or more of the reference implementations we compiled for you from the *featherlite* homepage at www.featherlite-framework.com.

The reference implementations are shipped as independent and self contained packages for the most common operating systems. Just unzip the package to a folder of your choice and start the server and client executables. To study and test the reference implementations download the tutorials. Do the 5 minute tutorial first. The tutorials will give you a quick insight of what the *featherlite* framework provides.

If you are already using the Eclipse IDE, you can alternatively download the *featherlite* plug-ins and reference implementations, copy them your workspace and run them from inside eclipse. In this way, you can also have a look at our source files to see how

things are done.