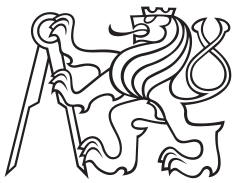
Master's Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Graphical Editor for Job Structure Configuration in Java EE 7 Batch API

Bc. Tomáš Milata
Study programme: Open Informatics
Specialisation: Software Engineering

May 2015
Supervisor: Ing. Jiří Pechanec

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Graphics and Interaction

DIPLOMA THESIS ASSIGNMENT

Student: **Tomáš Milata**

Study programme: Open Informatics
Specialisation: Software Engineering

Title of Diploma Thesis: **Graphical Editor for Job Structure Configuration in Java EE 7 Batch API**

Guidelines:

Batch API is a part of the new Java EE 7 specification. It allows to define batch processes which are described as potentially complex workflows in simple XML language.
Study the API.
Design and implement an Eclipse plug-in that will:
*Allow user to graphically draw the structure and links of the elements of the workflow,
*Allow text editing with bi-directional propagation of changes,
*Integrate with Eclipse to provide dynamic names for referred beans.
Analyse existing solutions on other platforms. Follow Eclipse UI conventions.
Create documentation, offer sources to communities and prepare a distribution channel (update site).
Provide an example job that uses all the basic elements: step, chunk, checkpoint, batchlet, flow, split, decision, transition. Implement it using the developed tool to test and demonstrate it can produce valid configurations. Also provide test cases of the invalid ones that the tool should detect. Perform testing with user to evaluate the final software.

Bibliography/Sources:

JSR-000352 Batch Applications for the Java Platform (Final Release)
[https://jcp.org/aboutJava/communityprocess/final/jsr352/index.html]

Diploma Thesis Supervisor: Ing. Jiří Pechanec

Valid until the end of the summer semester of academic year 2015/2016

prof. Ing. Jiří Žára, CSc.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, November 7, 2014

Acknowledgement / Declaration

I would like to express my gratitude to following people. Ing. Jiří Pechanec and Ing. Tomáš Černý, MSc. for bringing education and professional experience closer together, which gave me a great topic for my thesis. JBoss Tools open source community for being very welcoming, yet providing valuable feedback. All of the participants taking part in usability testing for their willingness. My family for their support, my sister also for helping me to correct my English, and my girlfriend Pája for her understaning for me not being very supportive last weeks.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Abstrakt / Abstract

Java Batch API, část specifikace Java EE 7, představuje jazyk k popisu dávkových úloh založený na XML, označovaný JSL. Úlohy definované JSL utváří struktury, jež si lze představit jako vývojové diagramy. Nedostupnost žádného vizuálního editoru na platformě Eclipse pro takovéto úlohy byla inspirací k vytvoření nového nástroje. Tato práce analyzuje Batch API a již existující nástroje. Na základě těchto analýz se ukazuje jako nejvýhodnější rozšířit existující projekt JBoss Tools. Klíčovými funkcionalitami tohoto nového nástroje jsou možnost graficky upravovat elementy úloh a spojení mezi nimi nebo obousměrná synchronizace změn mezi grafickým a textovým editorem. V této práci je popsán proces návrhu, implementace a vyhodnocení nástroje pomocí testů s uživateli. Softwarový příspěvek se podařilo začlenit do následující připravované verze JBoss Tools, jejíž vydání je naplánováno na říjen 2015.

Klíčová slova: Java EE 7, Batch API, JSR-352, dávkové úlohy, grafický editor, diagram

Java Batch API, part of the Java EE 7 specification, introduces an XML-based Job Specification Language. Jobs defined by this JSL form structures which can be visualized as workflows charts. Absence of visual graphical editors for Java Batch jobs on Eclipse platform inspired us to create a new tool. This thesis analyzes the Batch API and existing tools. Based on these analyses, extension of existing JBoss Tools project shows up to be the best option. The key features of the new editor are ability to graphically edit job elements and their transitions or bi-directional synchronization of changes between the graphical and a text editor. The thesis describes the process of design, implementation and evaluation of the tool by usability tests. Finally, the software contribution is integrated into the next prepared version of JBoss Tools to be released in October 2015.

Keywords: Java EE 7, Batch API, JSR-352, batch jobs, graphical editor, diagram

Contents /

Figures

Chapter 1
Introduction

"To look at the history of batch processing, you really need to look at the history of computing itself", says Michael Minella, member of the JEE Batch specification [1] expert group in [2]. One of the most important use cases of first computers was to run **long-running** and **non-interactive** tasks. But is this programming paradigm still applicable to today's applications? As Minella [2] claims, *"the business world runs on batch"* and he adds valid **reasons**:

- Sometimes all data is not available immediately. A printed bank account statement has to be generated at the end of the month after all transactions are settled, not after every single transaction.
- Sometimes it is just worth waiting from the business point of view. It makes sense to wait a few hours to check whether user cancels his order and process it with other orders in a batch than to put what user has ordered immediately after he click Buy button.
- Also efective planning and scheduling can save resources. E.g. idle computing time at night can be utilized better when scheduled in a predictable way.

The new Batch API, part of the Java EE 7, provides a standardized technology for developing batch applications. The main benefits of using Java and an open source tool for batch processing are listed in [2]: *maintainability, flexibility, scalability, development resources, support and cost.*

The Batch API introduces an XML-based *Job Specification Language* for description of job workflows. Although the whole Java EE 7 [3] was released in 2013, it is surprising that some parts such as Batch have not been **not reflected much in support of IDEs**.

Typical user of the JSL is a Java programmer who can orient quickly in Java code. Orientation in code written in any *domain-specific language* such as JSL code may be harder as there is a higher chance that developer does not use it as often as the primary language (Java).

The other drawback of the JSL is the potential comlexity of the whole structure of large jobs.

Therefore it would be more convenient to represent batch jobs visualized in a structure similar to **workflow charts**. A tool able to support such visualization could save development time to a potentially **high number of Java developers**. The most important assignment of this thesis is to develop such tool.

Eclipse was chosen as a reasonable target platform for our new tools due to its high extensibility and a fact that many similar visual editors exist and therefore we have a real proof of concept that the idea is feasible.

Except for developing a visual editor for batch jobs, these are **aims** of this thesis:

- An analysis of the Batch API needs to be perfomed.
- Existing solutions have to be analyzed.
- Source code should be available with open source.
- The editor should be demonstrated on examples and user documentation written.

- The final software should be tested with users to evaluate its usability.

The editor should also fulfill following **requirements**:

- It should be able to create and manipulate elements of the job worklow in graphical way, as well as their connections.
- Text editing should be also supported and changes should be propagated between the source code and the visual represention in both directions.
- The application should provide content assist for existing Java artifacts.
- Eclipse conventions should be followed in the sense of user interface and distribution of application.

The requirement to offer source code to open source communities implies two possible scenarios. First is to develop own new tool and try to persuade others to contribute. The other one is to contribute to an existing project.

If either one of these strategies is successful, it will guarantee (apart from teamwork skills) a certain **level of quality** of the final software and therefore should be one of the main **criteria of success**.

Chapter 2
Overview

This chapter focuses on Java EE Batch both as a technical specification and as a tool from programmer's point of view; describes its domain model, language and API. We will also shortly mention its relation to Java standardization process and to other projects similar to Java EE Batch.

2.1 JEE Batch

By **JEE Batch** or **Batching API** or other similar names we mean the *Java Specification Request* (JSR) no. 352 [1] from 2013 which is a specification document issued as one of many standards by the *Java Community Process*[1]). This JSR belongs to the *Java EE 7* standard [3].

It is worth mentioning that, in general, by Java EE we do not represent any particular library, SDK or application server. It may be understood as a set of standards that need to be fulfilled by an application server in order to be *Java EE compliant*. E.g. for the Java EE 7 standard we have Oracle's *Glassfish 4* reference implementation, *Wildfly 8* by RedHat and others.

Some Java EE standards may be based on already existing and successful implementations, such as **Spring Batch**, which was an inspiration of JSR-352. Although the standardization process may seem like re-inventing the wheel from one point of view, it is important because it gives us a certain guarantee level for the future, e.g. support and bugfixing or possibility to easily switch to other implementation which complies with the same standard.

The JSR 352[1] defines itself as a description of "*the job specification language, Java programming model, and runtime environment for batch applications for the Java platform.*" It speaks about *batch processing* as typically bulk-oriented, non-interactive execution in the background. The specification [1] addresses some common requirements of batch applications: checkpointing, logging, parallelism, control and interaction with batch instances like initiation, stopping or restarting.

2.1.1 Building blocks

The main idea of JEE batch is to compose batch **jobs** (an abstraction of a task to be accomplished) of **steps**. There may be either item-oriented steps (called **chunks** consisting of **reader**, **processor** and **writer** units or a free-form steps called **batchlets**. The domain model includes many other entities, such as **flows** that compose steps into encapsulated units, various listeners, checkpoints, parallelization on both item and step levels, flow control elements. The step-level parallelization can be achieved by using **split** elements which are comprised of nested flows. These flows are executed concurrently.

[1]) `https://jcp.org/`

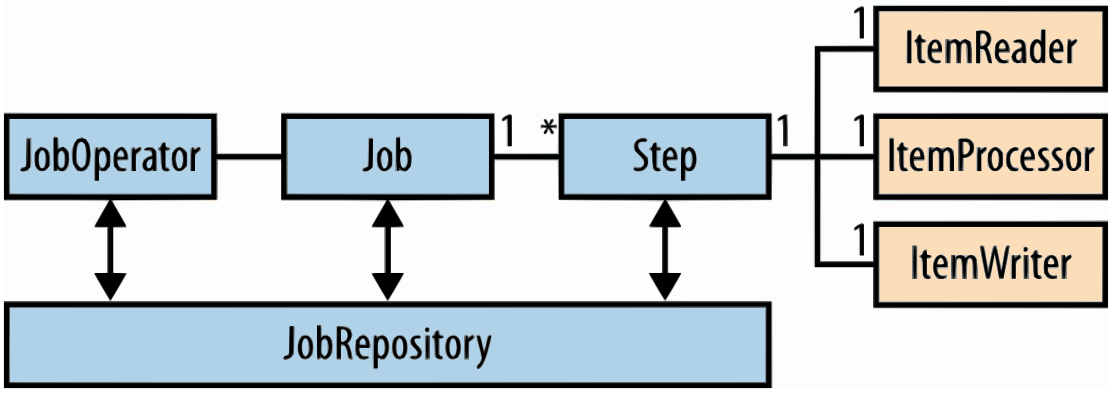


Figure 2.1. Diagram of JEE Batch domain model taken from JSR 352 [4]

Jobs can be launched via the **JobOperator** and information about their execution is stored in a **JobRepository**.

There is a difference between jobs, **jobs instances** and **job executions**, as shown in figure 2.2. While a job is a container of steps, a job instance is a logical instance of this job. There may be more instances, we may e.g. create one instance every day. And each of these instances may have multiple executions, because e.g. the first try to execute one instance has failed and we want to run it again. Therefore, a job execution represents a single attempt to execute a job instance.

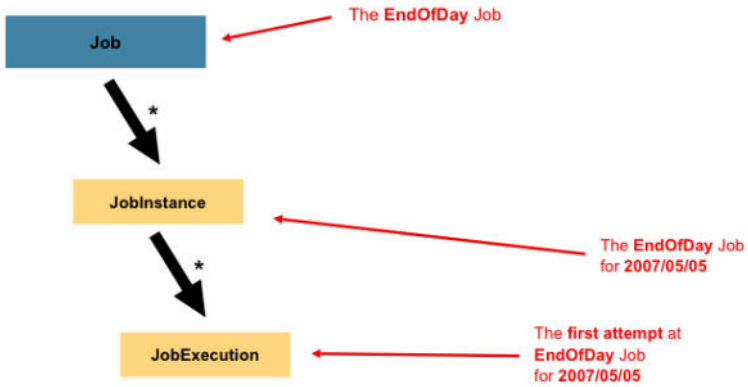


Figure 2.2. Job instances and executions taken from JSR 352 [1]

2.1.2 Java API and Job Speficiation Language

From batch programmer's perspective, writing JEE Batch application consists of writing batch job files in *Job Specification Language* (JSL) and providing Java classes implementing Batch API interfaces which are referred to from the JSL files.

The JSL is a XML-based language the structure of which corresponds to the batch model. The `*.xml` files with batch jobs must be present in the `META-INF/batch-jobs` directory and the file names serve as identifiers of the jobs. Here is an example of a simple job consisting of one chunk step:

```
<?xml version="1.1" encoding="UTF-8"?>
<job id="job" xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="1.0">
    <step id="myStep">
        <chunk item-count="3">
            <reader ref="myItemReader"/>
```

```
            <processor ref="myItemProcessor"/>
            <writer ref="myItemWriter"/>
        </chunk>
    </step>
</job>
```

The Java API consists of a set of interfaces and abstract classes. The abstract classes serve mainly for developer's convenience as he might be satisfied with some default behaviour. This is the `ItemReader` interface as an example:

```
public interface ItemReader {

    public void open(Serializable checkpoint) throws Exception;

    public void close() throws Exception;

    public Object readItem() throws Exception;

    public Serializable checkpointInfo() throws Exception;
}
```

2.1.3 Chunking

The specification document [1] describes an algorithm of processing items as follows (see figure 2.3): An items is read by reader and passed for processing to processor. This repeats until the number of the read items reaches the commit interval. Then the processed items are passed to the writer, written all at once and the transaction is committed.

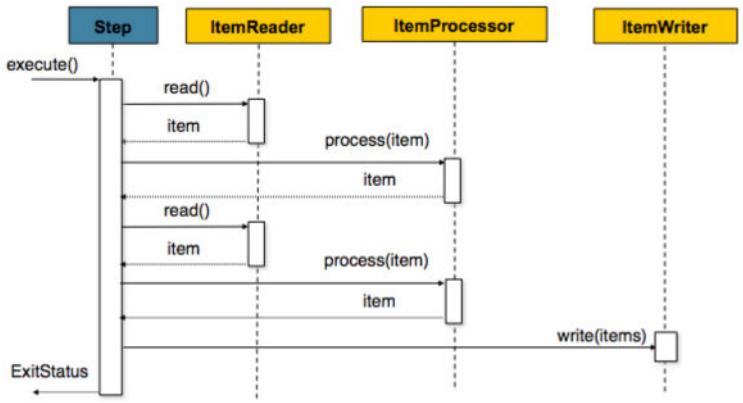


Figure 2.3. Algorithm of processing items taken from JSR 352 [1]

2.1.4 Transitions

JSR-352 specification [1] contains 3 types of transions:

- **Unconditional next transitions** — specified by `next` attribute of steps, flows and splits. (Note that only one such transition may be specified for one step, flow or split).

```
<step id="step1" next="step2"/>
<step id="step2"/>
```

- **Conditional next transitions** — represented by nested `<next>` elements within a step, flow or decision with a required `on` attribute which defines an exit status to be matched. If the exit status of a step execution is equal or matches (`*` wilcards are allowed as well) to the `on` attribute, the transition is used.

```
<step id="step1">
    <next on="MY_STATUS" to="step2"/>
</step>
<step id="step2"/>
```

- **Restart transitions** — defined by `<stop>` elements that work the same as conditional next transition from the status matching point of view. When a stop element is matched by status, job is restarted. Instead of a `next` target they have a `restart` target element which specifies the element to continue with after restart.

```
<step id="step1">
    <stop on="MY_STATUS" restart="step2"/>
</step>
<step id="step2"/>
```

The JSL contains also `<end>` and `<fail>` *outcome elements*, which are not considered to be transitions, though, as they do not specify a target element and they always terminate the job. The end element indicates a successful execution and the fail element an error status.

```
<step id="step1">
    <stop on="STOP_STATUS" restart="step2"/>
    <fail on="FAIL_STAUTS"/>
    <end on="END_STATUS/>
    <next on="NEXT_STATUS" to="step2"/>
</step>
<step id="step2"/>
```

If we take a look at the batch elements and slightly generalize our view, we may infer that there are 4 types of **flow elements** and these can transition from one another. By flow elements we mean **steps**, **flows**, **splits** and **decisions**. This may suffice for general overview and basic understanding. The further facts are, however, essential for our purposes to develop an editor:

- A job may not begin with a decision. When a job is run, it begins with a first flow element in the XML file. But there is no use running a decision first as its purpose is to decide which status to return based on executions of previous steps. That is the reason for signature of `decide:` method in `Decider` interface:

```
    String decide(StepExecution[] executions) throws Exception;
```

- There can be no condional transitions from a split. Since flows inside a split are executed in parallel, there is no single unambiguous exit status that could be used for matching. Therefore, only unconditional transitions are allowed from a split.
- There can be no unconditional transitions from a decision — as a decision serves for decision purposes, it does not make sense to always transition to a single element. Such decision could be omited and replaced with a direct transition.

2.1.5 Transitions and Nested Flows

One of possible puporoses for having flows in the JSL is a sort of encapsulation known from object-oriented programming languages. As many other sources do, also Arun Gupta [4] refers to a flow as an execution element which *"defines a sequence of execution elements that execute together as a unit."*.

The term **sequence** not very descriptive, though. On the one hand, the final execution is a sequence as JSR-352 does not allow cycles. On the other hand, contents of a flow can be a much more complex **tree-structure** (with other nested flows etc.)

An important restriction that provides the encapsulation is that `next` transitions may point only to target element contained directly in the same parent as the source element. It means that only direct children of a job or a flow can be interconnected. Figure 2.4 illustrates valid and invalid transitions.

The only exception is the stop (restart) transition which may target only a child element of the root job element. From the point of view of structured programming, it looks like a `goto` command.

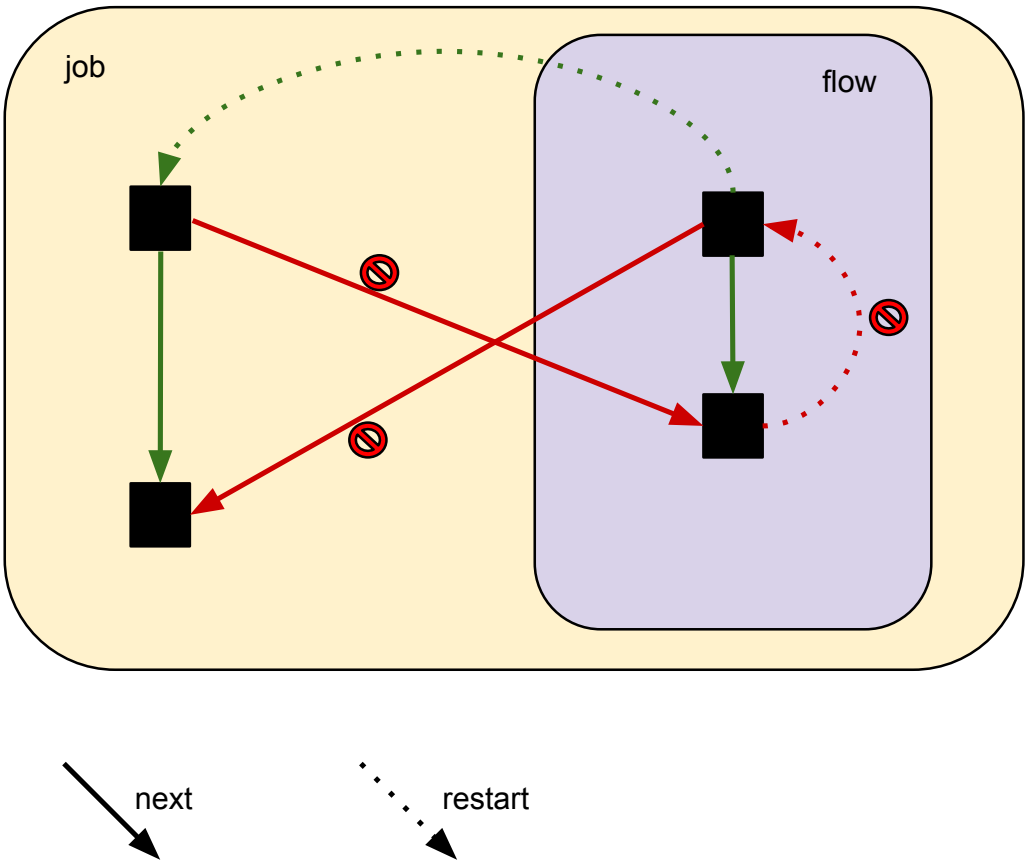


Figure 2.4. Difference between next and restart transitions

2.2 Spring Batch vs JEE Batch

It is no secret that design of JSR 352 was very much inspired by Spring Batch [1]) and that JSR 352 brought similar funcionality that Spring Batch had already supported. Although there were some incompatibilities for a certain time after the JSR 352 release, the Spring documentation [5] now claims that since 3.0 version, Spring Batch is in compliance with the standard.

As a result, there are now two possible ways how to use Spring Batch, either the old Spring-specific API with its own `org.springframework.batch.*` classes and slightly different XML language, or stick to the standard JSL and `javax.batch` API.

Neverthless, Spring Batch provides much more functionality above the standard, e.g. various readers/writers for specific usages (certain databases, file system, etc.) or support for passing of type-safe properties instead of standard `String` properties from `java.util.Properties`.

[1]) `http://projects.spring.io/spring-batch/`

Chapter 3
Analysis

In this chapter, we will review the state of the art in this field and summarize existing solutions. We will also analyse requirements of potential users that will result in a specification of features that our tool should provide.

3.1 Existing Tools and Related Works

This section captures a list of known tools (on any platform) that provide support for Java EE Batch or Spring Batch and are available at the time this analysis is being performed (November 2014).

Following points were considered in the analysis:

- Similarity to the assignment of this thesis
- Obvious features and concepts in the UI from the usability perspective
- Code reusability for our purposes

3.1.1 jBatchSuite

At first sight, this project [6] extending NetBeans IDE appears to be a close match to our needs as it provides a GUI and drag and drop approach to configuration of JSL files. It is also able to generate implementations of `javax.batch.api` interfaces according to the drawn diagrams.

The downside of this one-man project is that it is not very active (only 8 commits so far in the Subversion trunk) and therefore potentially to immature and early code base to be built upon.

What is missing from the usabilty point of view is that the diagrams created by user are persisted in different artifacts that the `.xml` job file itself. That means also that JSL files created by hand are not recognized for batch support. Propagation of changes is only one-way (from diagram to XML), triggered by user clicking on a button, and user also has to jump between the diagram file and the generated XML by hand; there is no natural connection of those 2 artifacts.

Since this tool is based on JSR-352 API, some source code might be reusable, e.g. classes that model Batch entities.

3.1.2 Intellij IDEA

IntelliJ IDEA has become quite popular last years among Java developers, partly due to its advanced and intelligent code assistance. However, their approach seems to prefer intelligent and smooth text editors to visual tools. And so is IDEA's support for JEE Batch.

Except for standard support for XML editing according to XSD (which is a general feature of IDEA for all XMLs) such as code completion and validation of entered elements and properties, there are some more advanced JEE Batch-related functions.

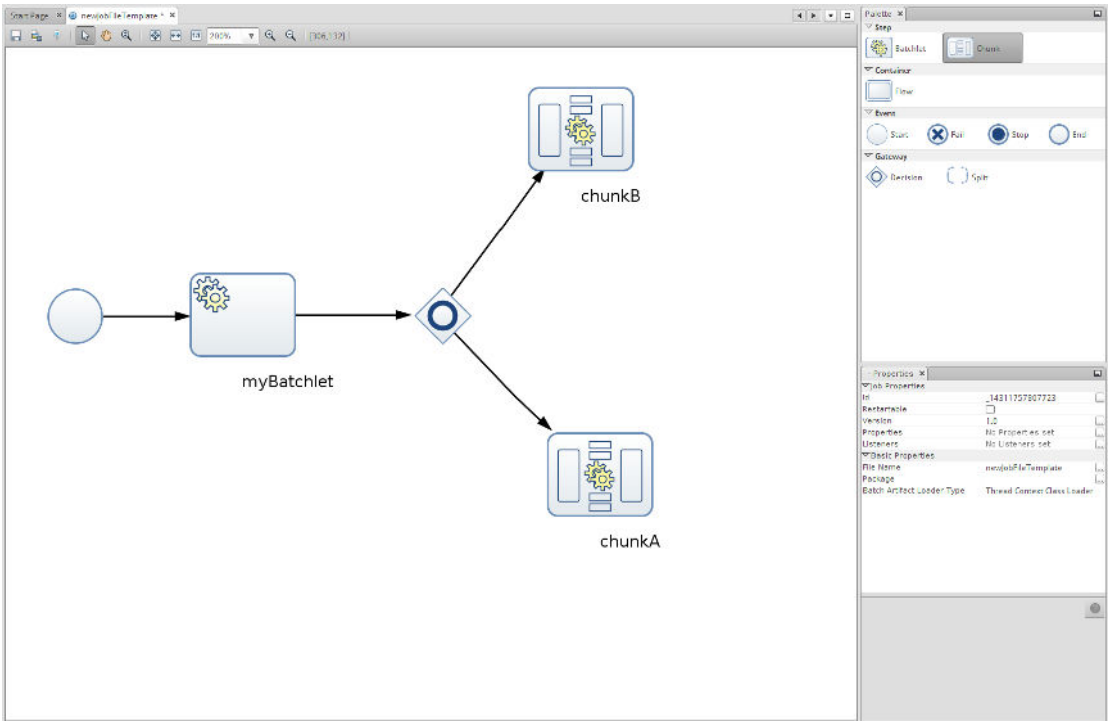


Figure 3.1. jBatch Suite graphical editor

Namely, these include search for usages and refatoring support, code completion for names of entities (defined both in java and xml) and navigation to definition or warning if entity does not exist, as is presented in [7]. Batch files containing a valid XSD location are discovered automatically, there is no need to set up the Batch nature of the files.

IntelliJ in Ultimate version, where Java EE support is available, is a commercial tool with mostly closed source code (at least the Java EE part) which prevents us to reuse any code from it.

3.1.3 Spring Batch Support in Intellij IDEA

Support of Spring Batch in IntelliJ is much the same as for JEE Batch. The difference is that it takes into account the Spring dependency injection model. Thanks to Spring's advanced options of passing properties to Batch entities from XML, it provides some extra features like Expression Language resolution.

3.1.4 WebSphere Developer Tools

This Eclipse-based IDE provides simple support for editing JEE Batch files in a form of XML tree editor and some standard Eclipse-style wizards to create Batch related JSL artifacts and Java classes. The XML uses the usual pattern that allows for switching between tabs with text source and a tree view.

Unfortunately, the lack of ability to display jobs in a graphical diagram makes this tool only a context-aware XML and tree-form editor.

The source code is, unfortunately, not publicly available.

3.1.5 JBoss Forge Addon

This tool does not really fit among other tools from this selection since it targets different group of users due to its command-line nature.

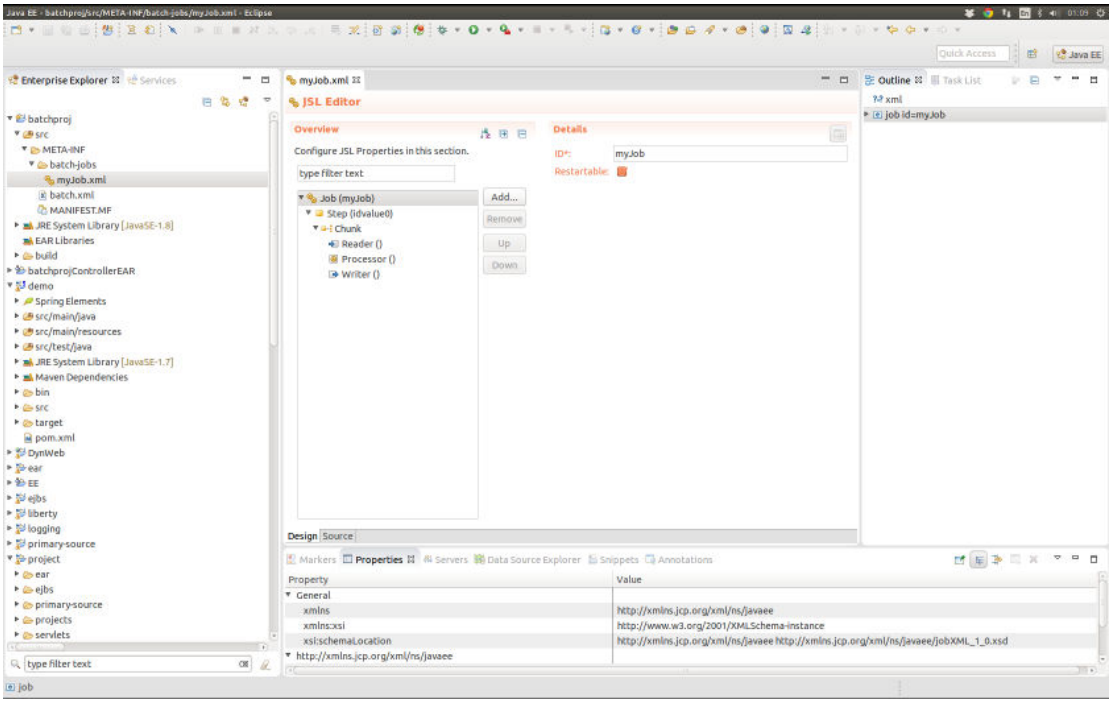


Figure 3.2. Tree-form XML editor in WebSphere Developer Tools

Neither its source code [8] seems reusable, because it does not contain any domain model layer. It operates directly with XML ojbects and generates artifact from file templates.

3.1.6 Spring Tools Suite

Spring Tools suite appears to be very advanced tool which covers our criteria very well, except the fact that it is build upon a different model. Unfortunately, it supports only the non-JSR version of Spring Batch.

Nevertheless, its features are multitab editor of Batch XML artifacts (classic text XML editor with code completion, tree-form XML editor and a visual drag & drop editor), 2-way propagation of changes (XML to GUI and GUI to XML) or e.g. integration with Spring beans model.

The codebase is split into two modules. Spring IDE is just a set od Eclipse plugins, Spring Tools Suite contains plugins from Spring IDE bundled together with other features and bundled as a full distribution. Source code of the Eclipse plugin has been released under Eclipse Public License v. 1 which means that some code might be reused for our purposes even tough its underlying model is different. Code of Spring Tools Suitei is, however, not public. Therefore a more thorough analysis of code dependencies between these two parts would be required if this project was chosen as a base for extension.

3.1.7 Spring Netbeans Module

NetBeans IDE has also certain support for Spring Framework, including Spring Batch. But the Spring NetBeans Module [9] does offer nothing more than just an XML editor with code completion.

Project is neither used much at the moment, nor actively developed anymore.

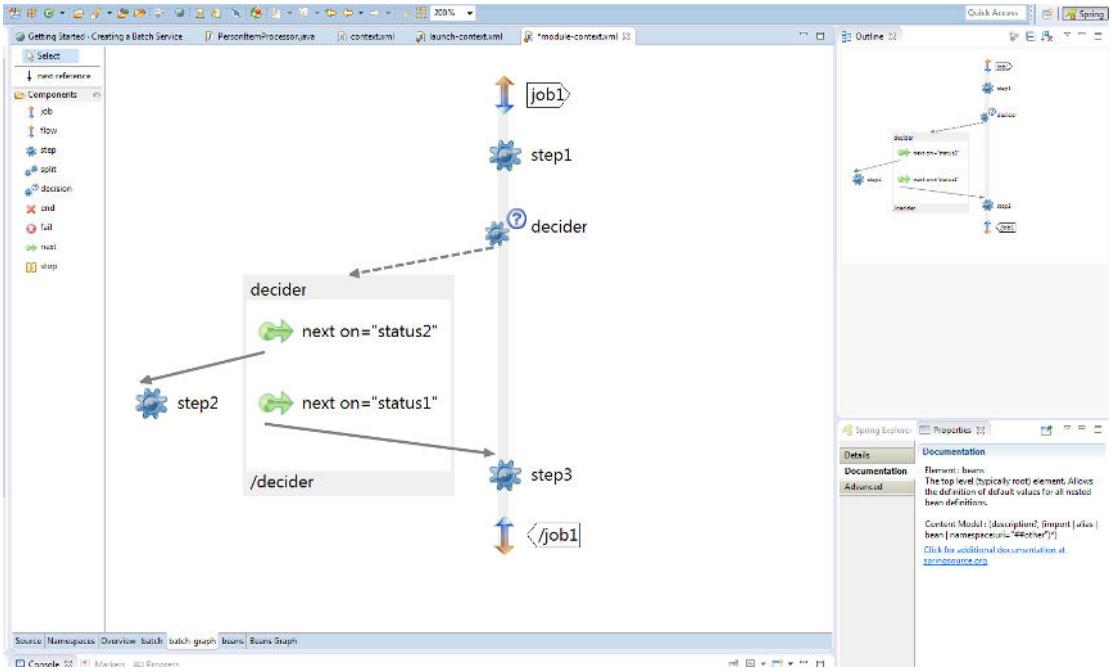


Figure 3.3. Graphical batch editor in Spring Tools Suite

3.1.8 JBoss Tools

This very new project started development quite late compared to others (even the JIRA feature request [10] is from 22nd of September 2014). Alghough involving this project into the research at the last minute, after some decisions had been already made, meant re-evaluting these decisions, it could not be simply overlooked. In may 2015, the project still has not been released in a final version, which is the reason it could not have been found during the first research for existing editors in autumn 2014.

Among other JEE related features, JBoss Tools offers Batch support. It takes form of the standard Eclipse 2-tab editor. The former is a XML editor with some validation accoring to XSD and content assist reflecting the Batch nature. The latter, a tree-form editor, does not differ much from the WebSpher Developer Tools editor. Unlike the XML editor, no content proposal is present.

A clear advantage is that the tree-form editor is implemented using Eclipse Sapphire framework which also supports creating visual (diagram) editors. Diagram and tree-form editors are able to work with a common Saphire model which means that the existing model form tree-form editor could be reused.

Source code is open-source and it is licensed under EPL v 1.0.

3.2 Typical User and User Needs

Target users of the visual editor are developers. More precisely, Java developers who are familiar with basic Elipse concepts. Therefore, this thesis does not focus on teaching new developers how to control their IDE. Let us assume that user understands the concept of Eclipse perspectives (different layouts of the whole scren, consisting of several views, that differ based on the task context they are used for). He or she should also be able to open a desired file editor, and be familiar with editor tabs and be aware of the fact that some common views as *Properties* may have dynamic content based on what

action is being performed in other views. Although these concepts might cause some minor troubles for Eclipse newcomers, they are very routine for regular Eclipse users.

One of the main reasons users might want to use our tool is to get a nicely organized and uncluttered view of the job strucure. As the title of this thesis suggests, the main design goal should be to visualize the **structure**. That means the tool should emphasize important structural features that are not **well visible** in XML and table-based (or tree-form) editors. Rather than being a tool for quick editation of any property value anywhere in the job tree, the tool should help with orientation in the whole job. It should make easier to analyze existing large and complex jobs and help to create basic **architecture** of new jobs.

One may raise a valid question — whether a certain form of user research should be conducted. Ideally, qualitative research should be taken first to formulate hypotheses about user needs and potential problems and then a quantitative study to test these hypotheses. After a discussion with a supervisor, we decided not to include user research in this thesis. One reason is the limited time. Secondly, several companies or people have already invested some time into designig and implementing visual diagram tools. Spring Batch visual editor or jBatch Suite are based on either slightly different programming model or different platform than JSR-352 Eclipse editor, the basic paradigm from usability point of view may be reused though, or at least serve as an inspiration. And last but not least, since the target user is a developer, the problem of *designing for yourself* does not apply to this design so much.

3.3 Features and Requirements

Let us summarize the facts from previous observations and from the assignment of this thesis concerning features and requirements of our tool.

According to the assignment, user should be able to create the structure of jobs and links between elements with drawing. This implicitly assumes that the entities will be displayed in a kind of a diagram or 2D structure. This is quite natural requirement because the job structure might become very complicated quickly (e.g. when flows are nested) which is not very transparent when displayed as a XML tree. The already working proof of concept is Spring Batch support, part of Spring Tools Suite.

Except for the graphical editor, a classic text editor is a must because developer must be able to edit the file in any possible way. A XML tree view does not seem to be necessary because it provides view to the data in the same structure (hierarchy) and the only difference is in methods of input.

If these two different views of the same file are provided, there should be also a quick way to switch from one to the other, such as editor with tabs.

Changes should be propagated in both ways and directly without any explicit trigger or user input. There are tools that require user to e.g. click a button but unless there is a reason to prevent automatic update, it should be done automatically.

The more advanced tools we have examined show that there is no need to persist other artifacts than the XML job files and Java classes themselves.

There is also no defensible reason for allowing user not to create his job XML files by hand. And if so, it still should be possible to open them in the editor.

A standard way of distributing Eclipse plugins is an Eclipse update site. Our tool should have one as well.

The final list of basic features that the tool should support is:

- Tabbed editor with text view and graphical view

- Two-way immediate automatic propagation of changes
- Source code as a first-class entity (no other persistent artifacts)
- Installation via standard plugin channel (marketplace).

Chapter 4
Solution Design

This chapter discusses the main desing decisions that have been made. Reasons for choosing Sapphire framework and its comparison to others are presented. It also contains a captured UI design process and an intial design of the whole software architecture.

4.1 Initial Decisions

From the interaction style point of view, Spring Batch support in Spring Tools Suite and jBatch Suite tools appear to be the closest match to the concept of our tool. Both of them offer a visual, diagram-style, editor. An advantage of the first one lies in its wide feature set which covers well our assignment, the latter in its domain model which is the same as ours and also quite close set of features.

However, they both do not seem to be a wise choice to build on. The main disadvantage of Spring Tools Suite is its different domain model and a codebase dependent on the spring environment. The jBatch Suite's weaknesses follow from immature codebase, different platform and lack of certain features which are necessary for us, e.g. bi-directional editing.

JBoss Tools seems to be the most interesting choice for extension. Support for diagram editors by Eclipse Sapphire framework and ability to reuse its existing model makes a promise of possibility to create a new diagram editor without writing a lot of new code. Its open-source nature also creates chances for merging the finished editor into the existing project. It has several advantages over creating a project that is not intented for contribution. In case it is merged into existing JBoss Tools, the potential users can find and start using this project much easier. A self-distributed project would require a sort of marketing to find its way to new users. A succesful contribution to JBoss Tools would also imply that a separate Eclipse update site for installation of the editor and its updates does not have to be created because the update site by JBoss Tools fulfills the requirements for an update site from the assignment of this thesis.

After considering all preceding arguments, JBoss Tools was chosen as a a base project that will be built on.

4.1.1 GUI in Context of Frameworks

Even though there is not much to decide about framework choice when this decision has already been made by JBoss Tools team, it still necessary to know main advantages and disadvantages of common Eclipse frameworks so that they can be utilized as much as possible.

All Eclipse UI framoworks are based on SWT in the low level. SWT is an alternative to Swing with the main difference that it is implemented differently using native API on each platform. It is not reasonable to use SWT directly as more high-level frameworks exist. It provides basic building blocks like button, dialog or tab.

JFace is one abstraction level above SWT and allows for features like data binding or field assist. Its bulding block are mostly aggregated SWT widgets into more complex structures like Wizards.

GMF (Graphical Modeling Framework), GEF (Graphical Editing Framework) and EMF (Eclipse Modeling Framework) are all separate Eclipse projects, yet supposed to be used together. EMF can be used to describe domain using a description language (to get a metamodel) and then generate model classes. GEF is a technology for creating graphical editors and GMF utilizes EMF and GEF to create graphical editors based on a given model. GMF might serve for our purposes to create a diagram editor quite well but its disadvantage is quite complicated structure (different modeling language than Java etc.) and many steps needed to get a simple thing done.

Sapphire framework internally used GEF but its model is written by developers directly in Java using interfaces and annotations.

The *core premise* as Sapphire documentation [11] says, *"is that the basic building block of UI should not be a widget (text box, label, button, etc.), but rather a property editor"*. Due to a model-first approach, result code written with Sapphire is very declarative. A big advantage is that the same model is reusable for both tree-form and diagram editors. The model can also have a resource (e.g. XML file) attached which can be reflected in a XML text editor. Then all changes are automatically synchronized among all of the 3 editors. The declarative approach means that in case of form editors, developer has to choose properties to edit and groups them together. In case of diagrams, developer has to specify which model entities correspond to diagram nodes, which entities represent edges and how shapes of nodes should look like.

It appears to be a quite reasonable choice for use cases like Batch API where the model is really unlikely to change. Therefore analysis can be made in advance to be sure that the framework is expressive enough to grasp a particular model.

4.2 GUI

4.2.1 Eclipse Conventions

Standard Eclipse guide for developers on how to create good UI is very detailed and covers a wide range of possible cases. The guidelines document [12] emphasizes main priciples: *user in control, directness, consistency, forgiveness, feedback, aesthetics, and simplicity.*

A great advantage of using a high-level framework as Sapphire is that it handles most of the cases where developer might make a wrong decision. A Sapphire editor developer does not need to handle platform matters like lifecycle of views, saving or preferences. If property editors are chosen and placed wisely and any interaction with Eclipse platform API is avoided, UI guidelines should be conformed.

A much simplified version of guidelines applicable when using Sapphire would be:

- Use only Sapphire UI API.
- Group items reasonably (follow the structure of the model).

4.2.2 Discussion of Diagram Structure

Spring Tools Suite and jBatch Suite and were taken as an inspiration. But some changes had to be made in order to both be able to conform to the rules of Sapphire framework and to make the final product as usable as possible.

- **Nested Flows**

  The two existing editors have chosen a different approach to the problem of flow recursion. Spring Tools Suite displays flows nested in an (potentially) unlimited depth while jBatch Suite requires flow to be opened to see what is inside.

  Capabilities of the Sapphire limit us to avoid nesting. However, if we look at the problem from the perspective of what flows should actually represent, it seems like it does not really matter. The encapsulation of flows exists for a good reason — no one from outside should care about its internals. And if someone needs to create flow internals, he or she does not need to know (almost) the outer context.

  If an appropriate navigation UI is provided, the flow traversal should not be a problem at all and it should work like e.g. browsing through directories in a file explorer.

  Another benefit is that this approach scales well with size of the whole job. Editors displaying all nested nodes globally might reach their limits at quite a low count of total nodes.

- **Start and Terminating Nodes**

  The jBatch Suite tool has shown a questionable way of presenting start and termination of the job. There are *start* and *end* entities that can be placed directly in the graph. An advantage of this way is that from the "flowchart" perspective, it is obvious where the job starts end ends. On the other hand, it does not represent well what the job XML file really contains. The correspondence between JSR-352 entities and graph entities is weaker, e.g. there is no such thing in the JSL as a transition from a start node to a step. And terminating elements belong directly to an element (e.g. step), it is not a separate entity.

  Since the described representation has its disadvantages and we are also restricted by Sapphire to use only the first-class entites (like step, split etc.) to represent nodes, there will be no start and terminating nodes and no transitions from/to them. An indicator of a start element and terminating elements should be visible directly in the diagram, though.

Note: Restart transitions are not intended to be displayed at all. Unlike `next` transitions they do not obey rules regarding nesting and always point to a top-level element. From within a nested flow there is no node displayed that would serve as a target endpoint. In the rot job there is a target element visible but source may not be visible as it may be located in depth inside a flow. There could be even one flow with more than one descendant flow with a restart transition. On the top-level view, it would be even more confusing because there would be more transitions from one flow.

4.2.3 Sketches & Prototypes

Some sketches (see figure 4.1) were drawn during discussions about the problem of displaying flow internals vs. navigation inside. They served to discover some potential design flaws like e.g. limited space for elements inside nested flows.

After the structure of digrams was clear, a first small application prototype was created. It was the so-called *evolutionary prototype*, i.e. its source code eventually evolved into the final software. The reason for this choice was the fact that a proof-of-concept application had to be developed as soon as possible to make sure that Sapphire framework and existing model is really applicable to this use case. Prototyping was also quite "cheap" regarding time costs as writing in the declarative way is really fast, especially when the Sapphire model is ready.

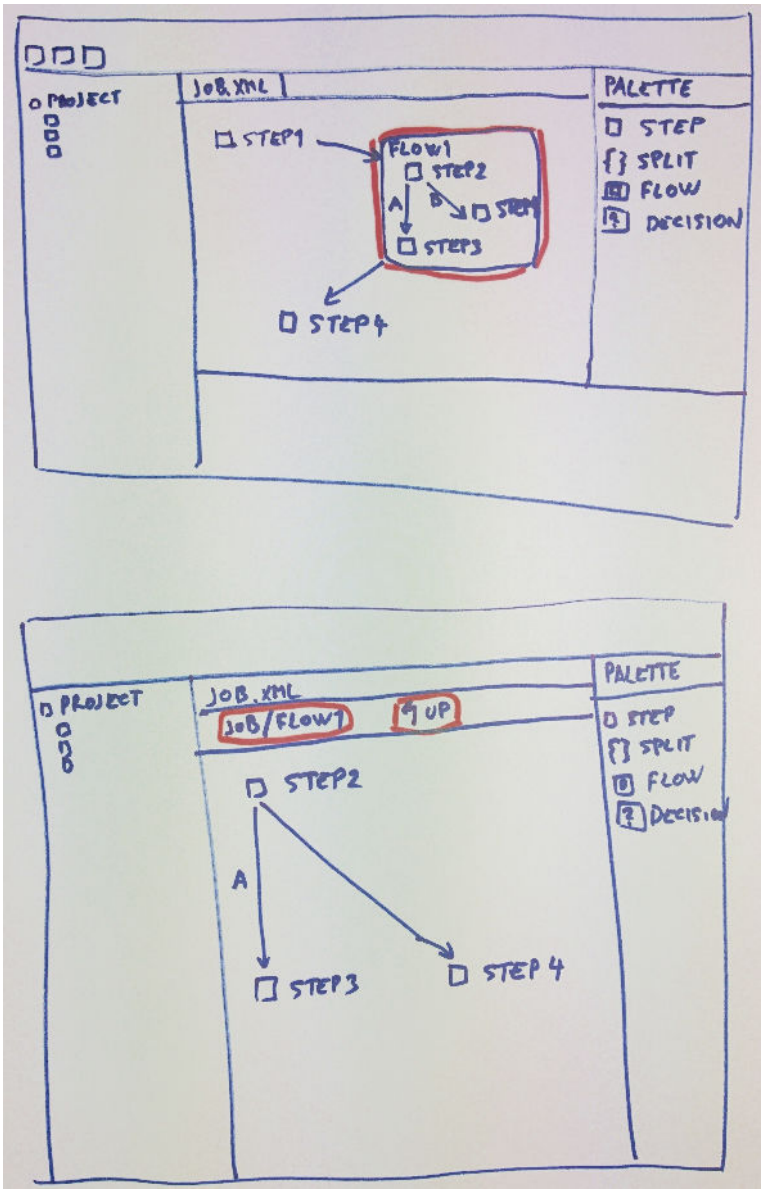


Figure 4.1. Sketches — ways of displaying nested flows

4.3 Software Architecture

In open source world it is not always very easy to manage to get new changes merged into an existing project. Especially if developer:

- Is new to the project or does not know it in detail.
- Has not contributed to it yet.
- Is not well known in open source communities.
- His changes are too big to process.

And since one of the key criteria of this project is to be merged into the existing codebase, a key strategy for the software design is to change as little existing code as possible.

4.3.1 Eclipse

Everything necessary for development of batch editor is located in the `batch` subdirectory of the repository.

```
batch/
|-- features
|   |-- org.jboss.tools.batch.feature
|   |-- org.jboss.tools.batch.test.feature
|-- itests
|   |-- org.jboss.tools.batch.core.itest
|   |-- org.jboss.tools.batch.ui.itest
|-- plugins
|   |-- org.jboss.tools.batch.core
|   |-- org.jboss.tools.batch.ui
|-- target
|-- tests
    |-- org.jboss.tools.batch.ui.test
```

The `plugins` subfolder contains all executive code. Ideally, only code in `org.jboss.tools.batch.ui` should be changed. The `org.jboss.tools.batch.core` contains support for Batch nature of Eclipse project and utility classses which allow e.g. to query for certain batch artifacts.

The `itests` directory is important as it contains integration tests which should not be broken. Also new test cases for the diagram editor should be added there.

The `features` directory contains only definitions for the Batch feature so that it can be installed via Eclipse update site.

Eclipse dependencies and extensions to Eclipse platform are specified in `plugin.xml` file. Since all Sapphire dependencies are already present and the Batch editor is already registered, there is no need to modify anything.

4.3.2 Sapphire

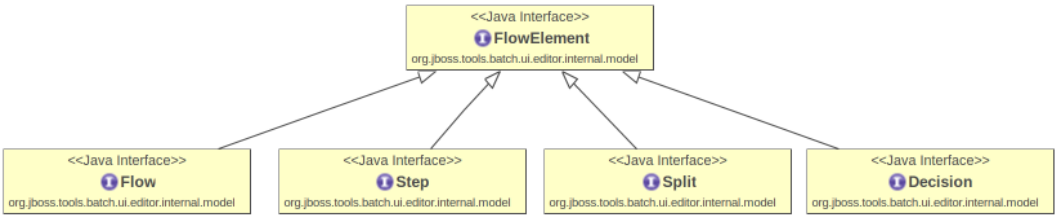


Figure 4.2. Part of initial Sapphire model — flow elements

The existing Sapphire model shown on figures 4.2, 4.3 and 4.4 by JBoss Tools team is very straightforward and copies the structure of the JSR-352 model. Similar elements classes are aggregated under a common ancestor.

The whole UI description resides in a single `.sdef` file, for both form and diagram editor. Form definitions for the tree-form editor may be reused for so-called *Properties View Contribution Page*, which is a dynamic content displayed in the standard Eclipse *Properties* view.

When a Sapphire editor is initialized, an object representing XML element is passed to it as a root of editor model. Traversal through nested flows may be achived by passing different roots to the editor.

The design of the implementation is as follows:

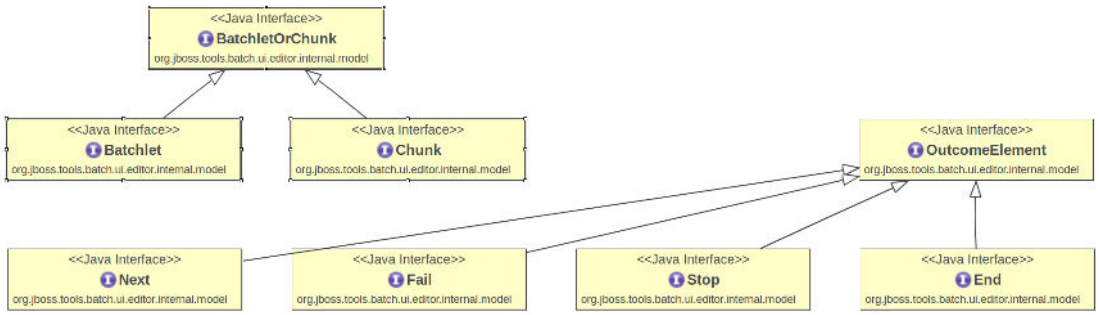


Figure 4.3. Part of initial Sapphire model — batchlet, chunk and outcomes

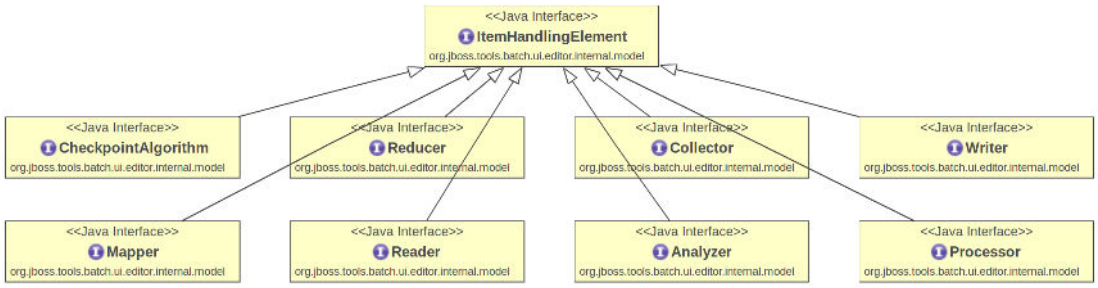


Figure 4.4. Part of initial Sapphire model — item handling elements

- Reuse the existing model and modify it only when necessary.
- Add a new tab for diagram editor using the Sapphire API.
- Map diagram nodes to all flow elements in the job.
- Achieve navigation through nested flows by passing different model elements.
- Find out a way to map `next` attributes and elements to diagram connections.
- Properties not editable directly in diagram should be accesible using the *Properties* view.

Chapter 5
Implementation

This chapter focuses on the process of implementation and contribution to the existing project. Readers should get a basic idea how the editor was implemented and why. An overview of used technologies is presented and some developer insights from the working process are shown.

5.1 Contributing to JBoss Tools project

JBoss Tools is a large project providing IDE support for various technologies somehow related to Java EE and JBoss (or Wildfly) application server. Users may use tools offered by this project either as a pre-packaged IDE (called JBoss Developer Studio) with all tools pre-installed or as a manually installed Eclipse plugin (called JBoss Tools). Sources of JBoss Tools are available online, mostly under Eclipse Public License v1.0. They are separated into many git repositories [13]. The Batch module is a part of the *jbosstools-javaee* [14] repository which also contains e.g. IDE tools for CDI, JSF or Seam.

5.1.1 GitHub Flow

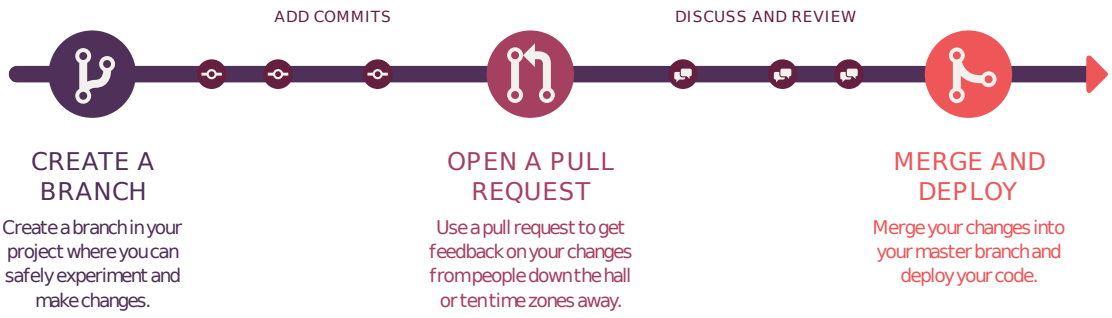


Figure 5.1. GitHub Flow, taken from [15]

JBoss Tools team uses so-called *GitHub flow* (shown on figure 5.1) for managing their source code and code from external contributors. GitHub describes the process on their website [16] as follows; its consists of these steps:

1. **Create branch**
   Meaning that all our changes are done in a new branch. Important recommendations are that the new "topic" branch should be created off of the master branch and anything in the master branch should be always deployable. Also, names of the topic branches should be self-decriptive.
2. **Add commits**
   Commits of the changes in the code should have meaningful descriptions explaining reasons of changes as well. These basic separate unit of changes have their meaning

in rolling-back changes or they can serve as a base for creating other changes in a new branch.

3. **Open a Pull Request**
   This step means that changes are ready for reviewing and a discussion about them should start. Developers may also accompany their code with a short documentation written in markdown.
4. **Discuss and review your code**
   Reviewers may add feedback to the requested changes and developer has a chance to work on their additional requirements. Additional changes are reflected and are displayed in a single timeline together with comments.
5. **Merge and deploy**
   It means merging changes to the master branch. The pull request is closed but remains searchable for future.

Note: The workflow may differ a little based on the chosen approach to repostory access. In the **"fork & pull"** model, a separate fork of the original repository is created and the final changes are transfered from the topic branch of the fork to the master branch of the original repository. In the **shared repository** model, changes in the topic branch of the repository are transfered to the master branch of the same repository.

5.1.2 Workflow of JBoss Tools

JBoss Tools use GitHub "fork & pull" workflow model. One of the biggest advantages of such workflow is that it allows completely strange developers to work independently on their own fork and also instead of managing access rights to the main repository for every user who would like to push their changes, users just request someone who has access rights to merge his changes.

A README repository documentation file contains a guide [17] on how to manage source changes which recommends following:

- Changes should be made in a topic branch named after an issue number. All issues of JBoss Tools, including feature request are tracked by JIRA issue tracking system.
- The master branch of the forked repository should be kept up-to-date with the main repository master branch. Changes from the topic branch should be applied to master using the `rebase` command, not `merge`.

Although the final repository file contents after the `rebase` and `merge` commands should not differ, the history of commits makes the difference (see figure 5.2). While `merge` creates a new commit with two or more predecessors, `rebase` reapplies changes from one branch on top of head of the other branch. An article at Attlassian web [18] mentions pros and cons of both approaches. Regarding `rebase` it states that *"The major benefit of rebasing is that you get a much cleaner project history. First, it eliminates the unnecessary merge commits required by git merge. Second, as you can see in the above diagram, rebasing also results in a perfectly linear project history".*

Note: What has proven successful during the development whas the interactive (`rebase -i`) command which allows to "squash" several commits into one. It leads to much cleaner log of commits becase we may eliminate similar consecutive commit messages like e.g. "Added files", "Added forgotten file". Interactive rebase is, however, not the only way to squash commits, as Chacon and Straub describe in [19].

5.2 Software Structure

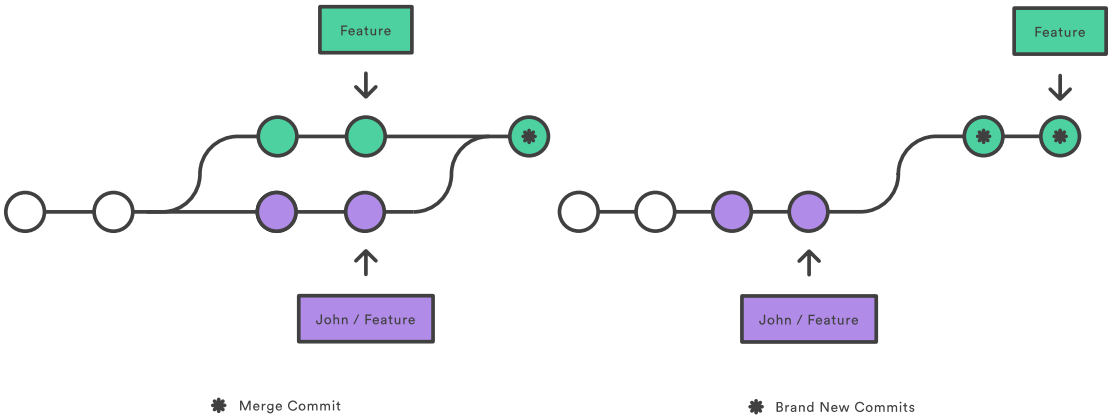


Figure 5.2. Merging vs Rebasing, taken from [18]

5.2.1 Model

Classes of `org.jboss.tools.batch.ui.editor.internal.model` package represent the JSR-352 model written using Sapphire framework. The model consists only on interfaces with annotated fields and getters / setters which are used by the framework to generate imlementing classes.

The original model by JBoss Tools team had to be modified to conform some requirements of the diagram editor.

`Flow` and `Job` classes were given a common ancestor `FlowElementsContainer` which can be used as a root model element for the diagram editor.

`OutcomeElement` fields of `Steps`, `Flows` and `Decisions` had to be split into lists of `TerminatingElements` and `NextElements` as they need to be handled differently in the diagram UI.

The old model contained fields corresponding to the `next` attributes only as a `String` value. Even though there was an implementation of `PossibleValueService` attached that provided data for content proposal in the UI, it was replaced by `ReferenceService` which also provides content proposal and also an object-oriented access. It implies that using this service we may get a type-safe reference to a target element of the next attribute. It simplifies handling value changes in implementation of a `ConnectionService`(see below).

Also some optimization was done to reduce repetition of fields of the same type. E.g. all fields corresponding to `next` attributes were extracted to a common superinterface. These improvements were done only in situations where it could save a lot of writing new code. In situations without this obvious savings it was not performed because of the strategy to change as little code as possible.

5.2.2 Actions

Actions are one of the core features of Sapphire. Using actions a functionality can be split into specification of how an action be fired and into a executive code of this action. Using action handler filters it can be decided dynamically whether an action is applicable to a given context. Custom actions implemented for the editor can be found in `org.jboss.tools.batch.ui.editor.internal.action` package.

5.2.3 Content Proposal

The `org.jboss.tools.batch.ui.editor.internal.services.contentproposal` package contains implementation of `ContentProposalService` which serves for pro-

viding content assist in both form and diagram editors. The service can determine its context (in which model element it is used) and based on that information it provides content assist for very specific types of batch artifacts.

At first, several separate classes were used instead of common `RefProposalService`. It had advantages in a clean code without reflection. But after Mr. Kabanovich's suggestion ins discussion about pull request [20] that the mapping between these classes and batch articats could be externalized for possible reuse, the implementation was changed to his suggested way.

5.2.4 Connections

Sapphire unfortunately does not support connections from an element that would be represented using its attribute. That is the case of a transition using `next` attribute. Although these connections cannot be declared in the `.sdef` file, it is possible to implement a custom `ConnectionService` to provide the necessary behavior. Since two connection types exist in the diagram editor and on of them can be handled by the `StandardConnectionService`, the custom implementation (`BatchDiagramConnectionService`) determines the connection type and manages method calls itself or delegates to the standard service accordingly. For a more detailed description see package `org.jboss.tools.batch.ui.editor.internal.services.diagram.connection` and its javadoc which is attached.

5.2.5 Layout

Due to some incompatibilities between the standard connection service and customizations of the diagram editor (described further in section 6.5, the diagram layouts are not persisted and automatic layout is used when editor is reopened. The `BatchDiagramLayoutPersistenceService` class serves just to override default behaviour not to save diagram layouts.

5.2.6 Extensions

Extensions to Sapphire are added via `sapphire-extension.xml` file. In our case a `BatchPathFunction` extension class was added to display a path from root element to a flow node which is currently set as a content of the editor. The class adds a function into the Sapphie Expression Language which means that it can be called directly from the `.sdef` file.

5.2.7 Utilities

The `org.jboss.tools.batch.ui.editor.internal.util` package contains so far just one utility class that provides mapping of model element types to batch artifacts.

5.3 Used Technologies

All the technical details regarding Eclipse Platform and Sapphire can be found in section 5.2 as it is related to it closely. A short descrition of all other interesting technologies used follows below.

5.3.1 Maven

Maven is an industry standard for building Java projects. Except from running build tasks it serves as a dependency manager. Tests are configured to be run automatically during each build.

An interesting speciality for this project is that running `mvn install` is not recommended (according to the documentation [14]) on development machines as installing target platforms from maven may interfere with local development Eclipse instance. Therefore `mvn clean verify` should be used to check whether build is correct and tests pass.

5.3.2 Eclipse Tycho

This technology is useful mostly for running automatized integration tests. Tycho is a maven plugin which allows to specify and configure a complete target Eclipse instance and run it. Although for development it is not very useful as running is slower than on a regular (installed) Eclipse instance, it is very convenient for running tests. The headless runner allows to run a completely functional Eclipse from command line even when no Eclipse is installed, it just downloads all required dependencies from maven repositories.

5.3.3 JUnit

JUnit is the main testing framework used on the project. In a testing terminology, most of the test that were already existing, were integration tests. No layers are mocked during such tests, which means that a real platform is run and operations are executed in the same way as user would execute them. Some tests, however, use Platform API to control the tests, which means it is not a real black-box testing as is used e.g. with Selenium tests for web applications. The usage of Platform API means that the same method calls that are used internally are also used for test control, e.g. mouse is not emulated or forced to perform an action to open an editor but a Platform method `openEditor()` is called instead.

5.3.4 Travis CI

Travis CI is a continuous integration service. It is free for open source and has very easy integration with GitHub. This service provided an invaluable help to check whether new commit has broken build or tests.

After a few-click setup of repository and adding `.travis.yml` configuration file, service was ready to react to every new commit in the GitHub repository. It ran a Maven command to build the whole batch project and run all test suites. Since the whole development changes consited of approx. 80 commits, it would have been a lot of manual work and waiting time on a local PC to verify each of these commits. It was used for most of the time of development until the pull request. The configuration file had to be removed before the merge to JBoss repository.

The service uses virtual linux machines to run the tests so the configuration file can contain almost any script. In this case a customized setup had to be done in order to run GUI Eclipse test, nevertheless the time spent on configuration was really worth it.

The history of all builds can be found at [21].

5.4 Work Process of the Project

5.4.1 Preparation

Before start of development and a final decision whether to choose JBoss Tools, it was important if they are interested in such contribution. Since their reply in an internal communacation was positive, development could finally begin.

It was unclear for a while whether JBoss Tools team will continue with development using Sapphire at all due to a bug [22]. Luckily, the bug in underlying SWT library was fixed in March and Sapphire framework was not dropped for Jboss Tools Batch project.

5.4.2 Implementation Challenges

The definitely worst experience from the implementation part was trying to figure out how to achieve specific what was needed in Sapphire. Framework documentation is very poor and the only source of examples online are the very few official examples from Sapphire repository.

The most difficult part to implement was the custom connection service. On the one hand, it is easy to find what needs to be implemented. On the other hand, ”details” like necessity to broadcast specific so that new connections are displayed cannot be found anywhere and it took long hours and days reading framework code and debugging to resolve cases like this.

Fortunately, Sapphire developers were always helpful and also managed to fix quickly a bug [23] that was found during development which would block the whole work if it was not fixed quickly.

5.4.3 Merging changes

The pull request [20] with a contribution of diagram editor changed 64 files in total. 3509 lines of code were added and 456 deleted. It was merged after a few days of discussion and additional improvements. Comments of JBoss Tools team members, description of changes and a promotional video with features of batch diagram can be found in [20].

JBoss Tools team required to have the whole change set in a single or a few commits, Therefore the whole history of approx. 80 commits was squashed into one. A nice view of the work history also with build results can be found at [21].

Chapter 6
Evaluation

This chapter evaluates the diagram editor from various aspects. A list of implemented features is presented, then these features are demonstrated on examples. Next follows a usability testing section describing the whole usability testing process from preparation to resolutions of the found issues. Finally, a list of know issues and possible future extensions is provided.

Due to a fact that reviewing changes of large contributions may take a very long time to process, it was necessary to send a pull request as soon as possible. And since usability testing is also a very time-consuming work, a decision was made to try to merge existing chages as soon as possible in a first pull request, then perform the usability testing and after that, send potential patches in additional pull requests.

6.1 Features Implemented

The following list contains features that were not present in the JBoss Tools Batch editor before and were added. Please note that it does not necessarily mean they were all implemented from scratch. In some cases existing features were reused to create new ones.

- `<step>`, `<flow>`, `<split>` and `<decision>` elements can be created in the diagram by selecting from the *Palette* and placing onto the background.
- Transition using `next` attribute and conditional transitions using `<next>` element can be drawn between elements.
- Tri-directional propagation of changes (all ways between XML, diagram and form editor)
- Invalid transitions cannot be created:
  - No conditional transitions from splits.
  - No unconditional transitions from decisions.
  - Transitions only on the same level (a transition cannot go between a nested flow and outer element).
- Navigation through nested flows. (Both regular flows and flows inside a split).
- Content proposal:
  - For Java beans by name according to `@Named` annotation
  - By classname
  - Filtered by context (job listeners, step listeners, mappers, reducers, processors etc.)
  - For Batch XML artifacts used in references
- Property editors with dynamic content based on currently selected element in the diagram for editing non-visual properties.
- Ids of elements editable directly on the node.
- An action to set an element as a start of the job (move to the first place).

- Terminating outcomes shown directly in the diagram next to nodes (end, fail, and stop on status).

6.2 Demonstration of the Editor

6.2.1 Example Weather Application

An example weather forecast application was created to demonstrate capabilities of the editor. A slightly modified version of the same application was used for usability testing with users. The application does not perform any real calculation. More than a full application it is rather a skeleton to show the Batch part. It is deployable and runnable, all the batch parts work as any production appplication would, only the Java code inside the referred beans only prints informative text to standard output instead reading real files, writing to DB and calculating forecast. The purpose of the demonstration is to show that the XML result produced by the editor can be runnable as a regular JSR-352 job.

Motivation: We need to calculate weather forecast based on data from weather stations that are sent to us daily. Input data from one day is stored in `N` CSV files on file system. The data needs to be read and saved into are database. Then long-running forecast computations can be run. Since rain and temperature forecast results do not depend on each other, they may be run in parallel. After all forecast calculations are done, it is time to check whether some notifications have to be sent. If the forecast looks fine, the batch job finishes. If rain is expected, a rain notification is sent. If snow is approaching, a snow notification is sent.

Description:

- The first `csvStep` is a chunk step with an item reader and writer referring to corresponding Java beans. Also a custom checkpoint algorithm is assigned (its implementation is overriden to make no checkpoints) just for demostration. The step always transitions to a split.
- The forecastSplit contains two flows. Each of them contains single batchlet step for forecast calculation. As soon as both calculations are finished, application continues to a decision along the `next` transition.
- The decider bean reference form the decision returns a status which is used to determine direction of next transition.
- Based on the status of decider, either rain or snow notification step is called. These are, again, simple batchlets.
- Java Batch classes are annotated by `@Named` annotation so that the can be reffered to by name.

There is a short **video** enclosed to this thesis which captures the process of creating this application. All Java code had been prepared before start, the job file is started from scratch. The source code is attached as well.

Running:

Prerequisities: You must have Java and Maven insalled and a running instance of Wildfly 8 or 9 to be able to deploy to it by Maven.

- In command prompt, `cd` to the root of the application (the directory where the `pom.xml` file is located).
- `mvn clean install`

- `mvn wildfly:deploy`

Assuming your Widlfy instance is running and you did not change the default configuration such as ports, you should be able to visit url `http://localhost:8080/com.example.batch-1.0-SNAPSHOT` which starts the batch job. In the output of Wildfly you should see the log of the batch that was run, something like:

```
23:18:46,927 INFO  [stdout] (Batch Thread - 1) Reading file 1
23:18:46,928 INFO  [stdout] (Batch Thread - 1) Reading file 2
23:18:46,928 INFO  [stdout] (Batch Thread - 1) Reading file 3
23:18:46,929 INFO  [stdout] (Batch Thread - 1) Saving data to DB.
23:18:46,930 INFO  [stdout] (Batch Thread - 1) Saving data to DB.
23:18:46,930 INFO  [stdout] (Batch Thread - 1) Saving data to DB.
23:18:46,939 INFO  [stdout] (Batch Thread - 3) Calculating
temperature forecast.
23:18:46,941 INFO  [stdout] (Batch Thread - 2) Calculating
 rain forecast.
23:18:47,443 INFO  [stdout] (Batch Thread - 2) Rain forecast finished.
23:18:47,940 INFO  [stdout] (Batch Thread - 3) Rain temperature
 finished.
23:18:47,942 INFO  [stdout] (Batch Thread - 1) Decision status: SNOW
23:18:47,944 INFO  [stdout] (Batch Thread - 1) It is going to snow so
 drive carefully!
```

6.2.2 Invalid Configuration Handling

To demostrate that the editor also handles negative cases, following examples of invalid configurations were prepared together with description how the editor handles them.

- **Case: Next attribute on a decision**

```
<decision id="decision" next="step"></decision>
<step id="step"></step>
```

Behavior: The transition cannot be created using the diagram at all. If such configuration is created using XML editor, a validation marker is shown in the XML editor with a warning that `next` attribute is not allowed according to [XSD]. If switched to the diagram, no connection is shown here. *Note:* The validation against XSD is a feature from the original editor by JBoss team.

- **Case: Next element on a split**

```
<split id="split">
     <next on="aaa" to="step"/>
</split>
<step id="step"></step>
```

Behavior: Same as the previous case: Diagram does not allow to create such transition. If this code is written in text editor, XSD validation error is shown and diagram does not display this transition.

- **Case: Missing element id**

```
<step>
</step>
```

Behavior: A validation marker shows up in the diagram node, in the form editor next to the *Id* field and on the corresponding line in the XML editor. The validation message says *Id must be specified.*

- **Case: More that one batchlet inside a step**

```
<step id="step">
   <batchlet ref="myBatchlet"></batchlet>
   <batchlet ref="myBatchlet"></batchlet>
</step>
```

Behavior: The *Add* action offers batchlet as an option only when no batchlet or chunk is present yet. Therefore the diagram and form editors will not create such configuration. If someone does so in the text editor, a validation marker is displayed in XML, form and diagram editors with a label *Must have at most 1 items*.

6.3 Usability Testing

There exist certain rules on how to set an appropriate number of participants for usability testing. E.g. a common ”rule of thumb” about 5 usability tests discovering ”most” of the usability problems. Such approach may be, of course, criticised, as e.g. in article [24] by Spool and Schroeder.

On projects with limited resources (time, money, etc.) the more important question is which number of usability test is the **most profitable**. Even on this project a decision had to be made whether to spend more time on usability tests or to develop new features, fix existing bugs, etc. Even though there exist studies, such as one [25] by Nielsen and Landauer claiming that for a medium-large software project *the highest ratio of benefits to costs is achieved for 3.2 test users*, as shown on figure 6.1, such value cannot be quantified for our project since it is hardly measurable by money. Therefore the final value of 4 participants was intuitively set as an estimate of the optimum value.

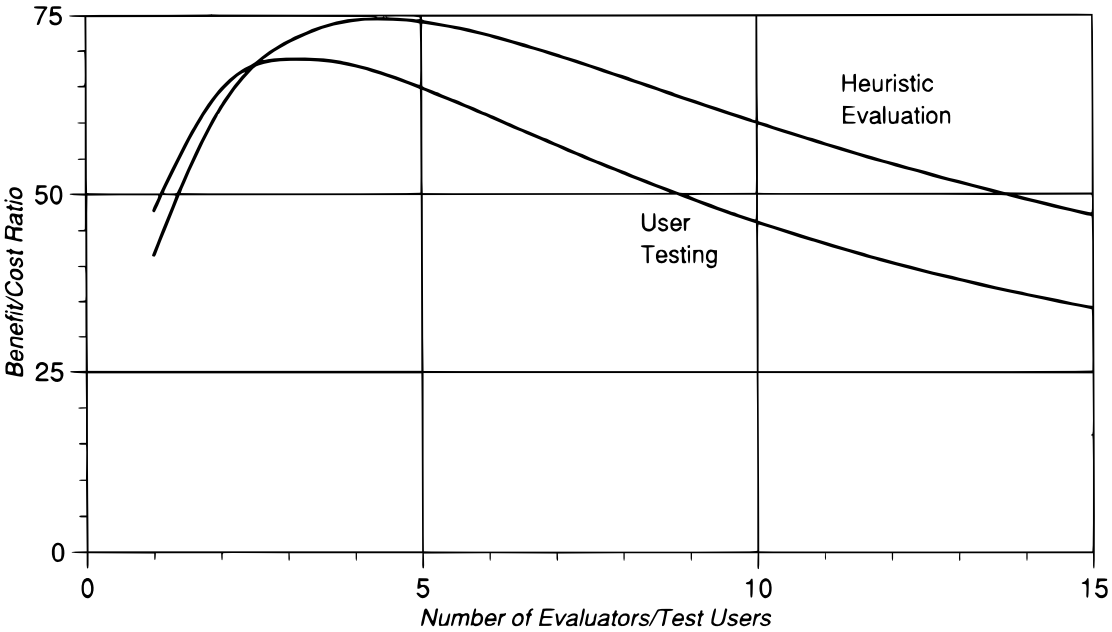


Figure 6.1. Ratio between benefits and costs for using various numbers of heuristic evaluators and test users to find usability problems in a medium-large software project.

6.3.1 Participants

4 volunteers participated in the usability session. All of them work for a single SW company with offices in Prague. Each of the 4 participants was familiar with the Batch

API but participants with different levels of knowledge were chosen on purpose. Also both regular Eclipse users and Intellij IDEA users were included to get some basic insights how user interaction on these two platforms might differ.

Participant 1:

- Java developer
- Familiar with JSR-352
- Has actively used Spring Batch in his projects
- IntelliJ IDEA user but also has used Eclipse for a longer time
- Male
- 27 years old

Participant 2:

- Analyst and Java developer
- Has used Spring Batch in his projects
- Uses Eclipse daily
- Male
- 38 years old

Participant 3:

- Java and Grails developer
- Has read about JEE Batch but has never used it on a project.
- Intellij IDEA user but has know Eclipse from the past
- Male
- 35 years old

Participant 4:

- Java developer
- Uses Spring Batch in one of his projects
- Uses both IntelliJ IDEA and Eclipse, it depends on current project.
- Male
- 28 years old

6.3.2 Scenario of the Session

Schedule of test session was planned as follows:

- Ice breaking, briefing (What is participant expected to do, whan he can expect), agreement on recording
- Tasks 0 - 5 (30 minutes)
- Follow-up interview, debriefing

No pre-test interview was performed as data about participants were collected when scheduling the date of the session.

Participants were given 6 tasks, each of them consisted of a printed assignment and a XML file they should create in the diagram editor as a result of their task, such as this exmaple of Task 4:

```
<step id="csvStep" next="forecastSplit">
    <chunk>
        <reader ref="csvReader"></reader>
        <writer ref="dbWriter"></writer>
```

```
        </chunk>
    </step>
    <split id="forecastSplit" next="forecastDecision">
        <flow id="rainFlow">
            <step id="rainStep">
                <batchlet ref="rainForecastBatchlet"></batchlet>
            </step>
        </flow>
        <flow id="temperatureFlow">
            <step id="temperatureStep">
                <batchlet ref="temperatureForecastBatchlet"></batchlet>
            </step>
        </flow>
    </split>
    <decision id="forecastDecision" ref="test.ForecastDecision">
        <end on="OK"/>
        <next to="snowNoticifactionStep" on="SNOW"/>
        <next to="rainNotificationStep" on="RAIN"/>
    </decision>
    <step id="rainNotificationStep"></step>
    <step id="snowNoticifactionStep"></step>
```

They were guided by a moderator so that they can focus rather on their work with the tool than trying to understand how the whole application should work. Participants were explained details about the application before the first task. They were informed they are expected to create only the job XML file and existing Java classes can be just used without knowing their internals.

Task assignments were in Czech but their translated versions are provided below. They are accompanied with motivation (what can be learned during the tasks). Please note that following questions are not hypotheses for quantitative testing. They are just topic with respect to which the scenarios were designed.

- **Task 0:** Open the existing `job.xml` file in the project. Explore the IDE and editors any way you like.

  Motivation: To see how user reacts to the new editor and to let him get familiar himself. Does he understand meaning of the three editor tabs?
- **Task 1:** Create a step named `csvReader` that serves for saving the CSV data to database. The step should contain a chunk. The chunk should contain a reader that references the `csvReader` Java bean and a writer with a reference to `dbWriter`.

  Motivation: To see if user understands that nodes represent XML elements, how can he add them to the step and how to do basic property editing. Does he know where to edit inner properties? To see how he uses content assist. How does he react to changes in *Properties* view based on current selection. Is the concept of transitions clear? How does he create transitions?
- **Task 2:** For parallel calculation of two different weather forecast, create a split containing two flows, a `rainFlow` and a `temperatureFlow`. Each flow should contain a step with a reference to appropriate batchlet.

  Motivation: How does user understand elements nesting? Is he able to open a nested flow? Does he realize he is in it now and how to navigate back?
- **Task 3:** Now, two different types of notification must be sent according to the result of the forecast.

Create a `forecastDecision` that will:

- end on `OK` status
- transition to a `rainNotificationStep` on `RAIN` status
- transition to a `snowNotificationStep` on `SNOW` status

Motivation: Do conditional transitions cause any troubles?

- **Task 4:** Replace the two notifications with one common `badWeatherNotification`. Each transitions should point to the new node.

  Motivation: What methods does participant use to accomplish the tasks? Does he move the lines end endpoints and how?
- **Task 5:** Add an additional step named `excelStep` (for reading Excel files) at the beginning of the job so that it continues with `csvStep`.

  Motivation: Is the *Set as Start* action clear?

6.3.3 Test Setup

Usability testing session were performed in 2 days with 2 participants per each day. Both session took place directly in developers' offices during a regular working day. The only difference compared to their own working environment was that they were performing the tasks on a different computer, yet with the same operating system they use (Windows 7).

Such conditions are more accurate in the sense of simulating realistic results compared to distraction-free laboratory tests.

The disadvantage of this choice was that participants were willing to spend only approx. 30 minutes with this experiment.

Figure 6.2. Usability Test Environment

6.3.4 Test Execution

In general, the test execution completed successfully and provided valuable feedback. After the first participant it showed that the original estimate of 30 minutes for all tasks had been a little over-estimated. Therefore other participants were told a shorter estimate (slightly over 20 minutes). The other 3 participants completed all tasks in about 25 minutes. Screen and sound records of all participants are available on the attached CD.

Some participants were being disturbed a little during their sessions, e.g. because of an important phone call, collegue's laugh or other colleagues entering the room and saying hello. However, all session were managable and the real vs. laboratory test balance was very adequate.

Next follow the session logs for all participants. Only relevant events are included, i.e. only those referring to questions that were posed and those which bring important new ideas. Execution times for each individual tasks are not logged as it makes no sense to analyze them statistically on such small data set.

Participant 1

[3:55] P1 understands the core concept of diagram, i.e. the nodes from the palette have to be insterted into the diagram area.

[4:01] After inserted a step, P1 recognises that a text input for step id was activated and that it is a requried field.

[4:24] Although P1 was not supposed to focus on setting step as a start, he proactively wants to do so. He notices the *Set as Start* button and correctly identifies the start indicator with the same icon.

[4:30] P1 thinks he should add an item reader and writer directly to the csvStep. That is just a matter of Batch API knowledge.

[5:05] P1 tries to add item reader using the context menu (right click) on the chunk area. Such action is however, not available. Although he sees the *Show in Properties View* action, he thinks it will not help him accomplish the task.

He is guided by moderator how to open the Properties View so that he can continue with the task.

[5:55] P1 succesfully finds the Reader tab in property views and identifies the correct row for entering a reference to the `csvReader` bean.

He expects content assist but since he performs no action to trigger it, he gets no assist. That might be caused by the fact that his main IDE used at work is Intellij IDEA which triggers content proposal automatically (as he himself explains at the end of the session).

[7:30] P1 correctly recognizes that transitions using the `next` XML attribute can be modeled using the *Next* transition from the pallete and can be applied to exisiting objects in the diagram.

He is not aware about the protocol for creating diagram connections, i.e.: click transition, click source node, click target node. He assumes that at first the source node should be selected, then transition chosen and then target node selected.

At the end of the session he explains he was used to that procedure from a different visual editor (maybe Enterprise Architect but he is not sure).

[8:20] When trying to add a flow inside a split, P1 at first makes a mistage and drops a new flow next to existing split instead of creating one inside. He also tries to create a reference to existing outer flow from inside of the split using the property view editor.

After moderator's hint to see generated source code he realizes his mistakes by himself and is able to recover to a correct state.

The reason of this mistake might be the lack of API knowlegde. He probably wanted to use same approach with reference to an external flow which is valid for Spring Batch but not for JSR-352.

[10:30] P1 successfuly finds a way to open a flow and uses it.

[11:10] After moderator's hint that content proposal is available, he successfully finds out the correct keyborad shortcut (CTRL + Space) for activation.

[11:35] P1 obviously understand the concept of navigation in inested flows and finds a correct button to use to navigate up to the parent.

[12:15] With knowledge gained when creating the first batchlet, P1 is able to apply it to a similar task again without any problems.

[13:30] P1 is able to add a terminating element with status code reference.

[14:30] P1 recognizes that connections using `next` element with `on` status reference correspond to the *Next On* transition in the palette view. He is also aware he can start typing the status code immediately after he creates the transition.

[15:55] P1 reveals the possibility of dragging an endpoint of a connection from one target node to another and use it to accomplish his task.

[17:00] P1 Finds a correct way to set a step as a start element.

He also states that after few minutes of using the tool it finally starts to feel intuitive.

Follow-up discussion: P1 suggests to open Properties View by default when editor is opened. He says he "quite likes" the editor.

Participant 2

[3:00] P2 Recognizes where to find nodes representing Batch entities and how to add them to the job.

[3:30] He proactively uses *Show in Propoerty View* and *Show in Source* actions and understands their meanings.

[4:00] P2 opens the content assist dialog for the `next` attribute. He says "There is none" and selects *OK* for the default (empty) option. But he does not notice that the first option was selected as default and the field did not remain empty.

[4:30] After a little moment of searching, P2 find the right way to insert a chunk into step.

[4:35] After inserting the chunk, P2 immediately recognizes where forms for specifying readers and writers are. he obviously understands the concept of Eclipse property views.

[5:45] P2 asks moderator whether editor provides content assist for Java beans. When moderator avoids to answer, he tries it himself and succesfully determines the correct shortcut for it.

[7:10] Instead of creating a new transition, P2 reuses one he accidentaly created before. His action indicates he undrestands that arrows represent transitions and how to reconnect their endpoints.

[8:00] P2 expresses his expectation of a working drag & drop gesture for inserting flows into a split. When he accidentally drops a flow into the step, he realizes that he is in an inappropriate location.

[9:40] P2 tries again to insert elements into other elements by dropping them. Soon after that he discovers the *Open Flow* button and uses it correctly.

[10:40] P2 has problems finding a wat to insert batchlet into a step. Trying to find any way to accomplish that he finds out that content of the Property View

depends on the current focus in the diagram. After a short break when he tries to show moderator something in the parent step he finds the right menu. But he points out that he would expect a menu for adding batchlets in the Property View because in the context of a split, there is possibility to add flows.

[13:20] P2 states that meaning of the *Parent* navigation label is not clear to him. He would expect a possibility to navigate in the structure of the job using nodes under the node of the XML job file in the *Project Explorer* view on the left.

[15:55] P2 says he would expect a buton for transition in the context menu of a node. Then he thinks the best way to specify transition is using the text field. He is not aware of the option to draw transition from node to node using the icon in the *Palette* view.

[17:40] Participant uses wrong procedure of adding transition, he tries to drag it from one node to another. Soon he realizes how to perform this task right.

[20:30] P2 succesfully finishes task that focuses on changing transition endpoints by dragging.

[21:15] Also the task to set step as start is executed without problems.

[21:45] P2 asks about existence of *automatic layuot* feature which indicates he would like to have such function.

Follow-up discussion: Participant's answer to question whether he has mistaken validation markers for delete buttons is positive. He would also like to have error marked by a validation marker in the editor shown in the common Eclipse *Markers* or *Problems* view and also on appropriate location in the XML editor. He sees the biggest problem in absence of transition button in the node context menu. Other things were intutitive for him, he claims. Then he likens the diagram editor to JBoss Drools editor, the knowledge of which is applicable for him to the Batch editor.

Participant 3

[3:01] P3 succeeds in adding a step to the job from the palette. He recognizes that the default text input which is activated after step was inserted refers to step's id.

[3:30] P3 also figures out how to add chunks into steps.

[4:26] Participant has problems finding a way to add an item reader. The *Show in Properties View* action does not seem important to him so he tries to drop another step onto the chunk. After a minute of wondering he is guided by moderator.

[4:55] P3 makes an observation concerning validation markers. When he clicks on it and sees the *Id is required.* label, he infers that the marker is a delete button which cannot be used at the moment because the id is missing.

[5:55] When entering a reader reference, P3 just types the text by hand instead of activating content assist. A possible reason for that might be he is a Intellij IDEA user.

[7:35] P3 has no problem with creating a `next` transition on the first try.

[7:55] Participant tries to insert a flow into split by dropping it onto it which is not supported. However, he finds a working solution using Properties View soon.

[8:44] The *Open Flow* action is found easily. P3 undrestand he can start inserting elements inside the flow when it is open.

[9:40] P3 has some problems finding an action to add a batchlet, even though he has already added a chunk, which is done in the same way. Neverthless, he succeeds eventually.

[11:15] Afer some unsuccessful retries of returning back to parent step, P3 finally completes this subtask. He expects a possibility to navigate back using tree nodes in *Package Explorer*, the same as P2.

[11:55] After moderator's hint to expect a content proposal, P3 tries the correct key shortcut to activate it.

[14:15] P3 has no problem creating `next` element transitions with `on` condition.

[16:00] Participant successfully reconnects exisiting transition to a new node.

[17:28] The *Set as Start* button is discovered and used properly.

[24:15] P3 tries to change a source node of a transition by dragging it to another element. He fails because it is not supported.

Follow-up discussion: Participant suggests having title bar clickable. It should be able to navigate directly to the clicked path segment. He also states he would expect the *Outline* view to display the whole job even when an inner flow is open. And it should be clickable. Another option for navigation up could be a button in the context menu of an element. He also believes a different icon for validation markers would be less confusing. And he would be satisfied if the *Show in Properties View* action was hooked by default on double click. He appreciates flexibility of the transition lines.

Participant 4

[4:45] P4 manages to add a new step to the job.

When he tries to input and id for the step, he does not realise that the `<id>` label serves only as a placeholder. He changes the value to `<csvStep>`. Perhaps he thinks he is writing XML code directly (because of the `<` and `>` characters).

The same error happens also during execution of some other tasks.

[5:25] Participant tries to click on validation markers. He might guess that the serve as delete buttons.

[6:40] Trying to add a reader into a chunk, P4 tries to drag the `CsvReader.java` file from the *Project Explorer* and drop it onto the chunk in the diagram. After an approx. half minut of searching he finally tries to use the *Show in Property View* action and finds the *Reader* tab there.

[7:50] After P4 is given an advice to try to use content assist, he knows he should use `Ctrl + Space` shortcut.

[9:00] No matter the task assignment, participant wants to understand how the content proposal works internally. It might he appreciates the content assist being smart.

[10:15] Transition using the `next` XML attribute is created without any problems.

[10:40] Same as some of previous participants, also P4 tries to add a flow into a split using a drop gesture.

[11:55] P4 deals with opening the flow without noticable troubles.

[12:55] Participant returns to the parent job and asks why his layout has changed. His guess is it is because the nodes are sorted automatically (which is true).

[14:00] P4 notices some strange characters (`\<`) in the `id` attribute of an entity. This is the XML encoded `<` character which he entered in one of previous tasks. Without moderator's explanation he would probably be very confused.

[15:50] Participant feels a little inconsistency in the fact that step node labels contain step id but batchlet or chunk contain only *Batchlet* or *Chunk* label. according to his suggestion, the label should contain name of the referenced batchlet bean.

[17:55] End outcome element created without any troubles.

[18:45] P4 makes a mistake by double clicking a step node while his mouse cursor is in the transition mode. It creates an unwanted transition loop.

User tries to use a *Revert* action (`Ctrl + Z`) but he is dissapointed that it is not possible.

During his consequent effort to fix his mistakes he has serious troubles to select the new transition. He tries it in the right way, yet he does not point his mouse cursor precisely so he fails.

[21:00] The tasks focused on reconnecting endpoints is unfortunately finished in a different, less efficient, way by deleting steps first and recreating transitions to the new step. Being advised explicitely to try to reconnect the transitions, participant accomplishes the task without much effort.

[22:30] P4 plays with connection bending points and likes to possibility to change shape of the lines.

[23:50] The way how to set element as a start is understood and task executed successfully.

Follow-up discussion: P4 claims that after a few minutes of using the SW he starts to get used to the features he has had problems with on the first attempt, such as selecting items. The hidden *Properties* view has been unintuitive to him. He would also expect the view to be on a different location but, as he says, it might be due to different layouts of different WYSIWYG editors he is used to. The *Parent* navigation button seems alright to him, although he would expect *Back* label instead. He would also expect the diagram layout to persist, not to lay out nodes autmatically but on demand instead.

6.3.5 Test Evaluation

All of the previous findings were analyzed and grouped together if they were repeated. They can be divided into 2 groups:

1. Actions that cause some troubles during the first try but users finally figures out the correct way himself and has no problems with repeating the learned procedure next time.
2. Problems which user might have problems to solve himself without any advice.

Since none of the users showed any signs of frustration when having problems from group 1, only problems from group 2 were further analyzed and necessary steps were done to solve them:

1. Users cannot find a way to edit inner elements of steps and other entities. They cannot see any corresponding form when the *Properties* view is hidden. Once the view is show, they have no problems finding particular form.

   Resolution: The issue was fixed by opening the *Properties* view by default. An issue in JIRA [26] was created together with a pull request.
2. When no content proposal in the form editor is shown automatically, users think this feature is not present. *Resolution:* Eclipse editors are not very consistent in this. Some provide on-demand assist and some trigger it as soon as user starts typing. Some forms add a small ”lightbulb” indicator with a tooltip saying *Content Proposal Available*. Such indicator is, however, not supported in Sapphire and therefore a feature request [27] was created in Eclipse BugZill
3. User tries to add a nested flow using drop gesture.

Resolution: The Sapphire framework unfortunately does not support such drop gestures. Therefore an enhancement request [28] was created in Eclipse BugZilla.

4. User does not know how to navigate back from nested flow to parent.

   Resolution: Unfortunately, Sapphire framework probably does not support a way to change location of the button to the left of the title in titlebar. Possible fixes could be:

   - An action in the context of the editor (right click)
   - To populate the job xml file in the *Package Explorer* view with additional sub-nodes representing nested flows

   Since there is no obvious solution for this and it would make no sense to send a quick patch without being completely sure, JBoss Tools community was asked [29] for help to share their experiences and insight.
5. Users expects a button for creating transitions in the node context menu.

   Resolution: This feature is present in a newer version of Sapphire framework, which will by updated before the final release of the editor, as you can read in the Jira task by Alexey Kazakov [30].
6. User expects the diagram layout to be persisted and to perform the auto-layout on demand.

   Resolution: Since this issue is not critical (see the difference on figures 6.4 and 6.3), it remains as a possible future extension, as described on page 40.
7. Validation markers (with a cross) are mistaken for delete buttons.

   Resolution: Markers were moved from right to left, which is the standard location in Eclipse. An issue in JIRA [31] was created together with a pull request.
8. User expects validation errors to be shown in the common *Errors* view.

   Resolution: When trying to reproduce this issue in a new workspace, errors started to appear in the view as they should. The bug during the test could have been caused by an inappropriate workspace handling. (The whole workspace versioned with Git so that it could be restred after each participant and the refresh probably caused an inconsistent state).
9. User thinks that the `<>` characters in the id default value have some meaning in syntax.

   Resolution: The confusing placeholder was removed. A request for enhancement [32] was sent to Jira together with a pull request.
10. User feels that the *Batchlet* label displayed instead of value of batchlet bean reference creates an inconsistency in the UI.

    Resolution: From a different point of view, if the label contained the `ref` value of the batchlet, there would be an inconsistency that all nodes have an `id` in the label but batchlet a reference. Ids are the values from XML but `ref`s are names of Java beans and therefore it is not a good idea to mix them. The other reason for not doing that is that chunks do not have ids too, but for chunks there is not a straightforward choice of field to be reflected in the label. The reason for this change would have to be validated by a quantitative test at first, so it will not be done for now.
11. User misses a *Revert* action.

    Resolution: This should be definitely handled by the framework as it makes not sense that every should implement the feature himself. A request for enhancement [33] was raised.

6.4 Known Issues

All problems found during the usability testing were either fixed and a patch sent or a request for enhancement was created if it was an issue of framework. The only exception is the problem with *Parent* navigation button where an obvious solution was not known and that is why community was asked [29] for their advice as they have also right to decide because this project became a part of theirs.

Note: The additional patches with fixes of usability issues were not merged yet into the main project repository.

6.5 Possibilities of Future Extensions

6.5.1 Persistence of Diagram Layout

Although the Sapphire framework itself is capable of persising layout of diagram nodes and connections, it is not applicable to our editor due to its certain customizations.

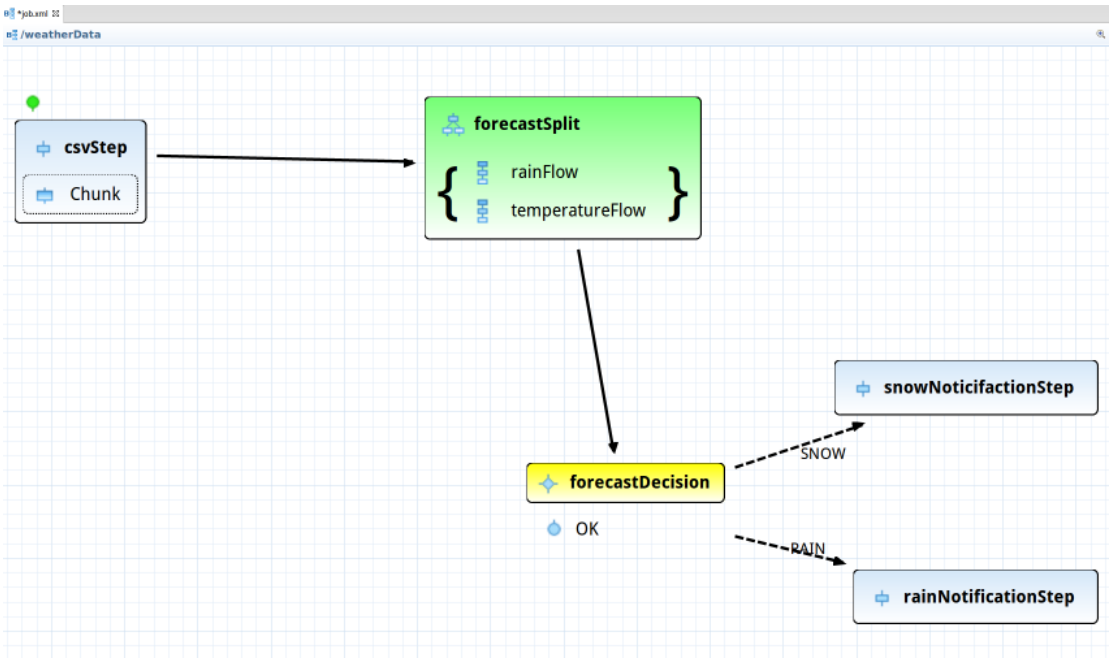


Figure 6.3. Custom Layout

The problem is that the Sapphire standard diagram layout persistence service calls a connection service which is used for the editor to return all existing connections. The connection service always returns all connections for the current editor content. This content is, however, not always the same in our case. Sometimes a nested flow needs to be displayed and in this case the connection service has to list only connections from the current flow. And when the editor is in the root (job) element, the connection service cannot include connections from nested flow into the list of all connections.

Implementing a custom persistence service would require writing of some additional hundreds lines of code and so it will be kept as an opportunity for contributions in the future.

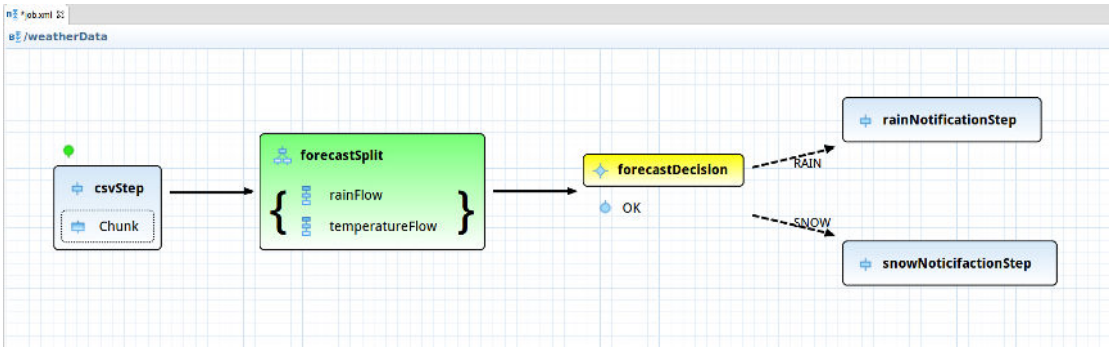


Figure 6.4. Automatic Layout

6.6 Author's Personal Insights

If I, as the author of the thesis, may add some personal opinions, I would say that working on this project was a great experience. It has shown me that open source world can be welcoming, yet factual and just. Even though I sill believe Sapphire was the right choice of framework, I know I should think twice next time before starting with a tool with such a poor documentation. And the decision for extending JBoss Tools was definitely the right one. Not only it let me learn many new things regarding teamwork but it also gives me prospects for my open source activities in future as the JBooss Tools team has already added me to their development group.

Chapter 7
Conclusion

To summarize what has been done and achieved during work on this thesis, let us discuss all the objectives that were set:

- A analysis of the Batch API was performed, resulting in observations how Batch entities should be modeled and represented in the visual editor.
- Based on the conducted research of existing solutions, main design points regarding UI and software architecture were made and JBoss Tools project was chosen to become a base for the following implementation.
- A contribution was made to the JBoss Tools project by merging the developed features back to the project code base. Consequently, the application itself is available for free online as well as its source code (which licensed under EPL).
- The editor was demonstrated on examples of both valid and invalid user input. An example application was developed using the new tool, its sources attached and the process development captured into the attached video.
- User documentation was created and attached.
- A usability test was performed with real Java developers. The tests provided valuable feedback and findings which were used to fix usability issues or provide suggestions for improvement.

Regarding the requirements for the new editor we can state that:

- The main feature of the editor — which is graphical manipulation with batch entities and their links — was implemented.
- Apart from the diagram editor, the tool comprises also of a XML editor and a tree-form editor, the content of which is automatically synchronized with others.
- The editor provides a dynamic content proposal. Names of Java beans and batch artifacts are suggested.
- The tool follows Eclipse conventions and is installable directly to an existing Eclipse instance using the standard *Update Site* mechanism.

A serious bug [23] in the Sapphire framework preventing diagram editors from working on certain circumstances was found thanks to the discoveries during work on this thesis.

One of the issues found during the usability testing, a problem with *Parent* navigation button, which might be hard to find for some users, was not resolved completely as section 6.4 describes.

Support for persistence of diagram layout might be a nice feature improving usability and remains as a possibility for future extensions 6.5.

One of the most noticable indications of a certain level of quality of the software is the fact that the contributing pull request was merged [20] and a feature request [34] for batch diagram editor in Jira issue tracker was resolved.

The JBoss Tools team provided valuable feedback and requests for improvements, yet their reactions mostly showed appreciation. Such as Max Rydahl An-

dersen, lead of JBoss Tools, who commented the pull request [20]: *This looks great! Will take some time to review but want to say Good Job.*

The editor was incorporated into the `4.3.0` version of JBoss Tools. Its final release is scheduled to 15th October 2015.

References

[1] JSR-000352. *Batch Applications for the Java Platform* [online]. Version 1.0 Final Release. Java Community Process, 2013. Available at: https://jcp.org/aboutJava/communityprocess/final/jsr352/index.html

[2] MINELLA, Michael T. *Pro Spring Batch: Batch Processing with the Spring Batch Framework*. New York: Apress, 2011, xiv, 487 p. Expert's voice in Spring. ISBN 978-1430234524.

[3] JSR-000342. *Java Platform, Enterprise Edition 7*. [online]. 2013. Available at: https://jcp.org/aboutJava/communityprocess/final/jsr342/index.html

[4] GUPTA, Arun. *Java EE 7 essentials*. 1st ed. O'Reilly Media, 2013, xvi, 343 pages. ISBN 14-493-7017-9.

[5] Spring Batch - Reference Documentation: JSR-352 Support. [online]. [cit. 2015-02-10]. Available at: http://docs.spring.io/spring-batch/trunk/reference/html/whatsNew.html#whatsNewJSR-352Support

[6] JBatch Suite: Netbeans plugin. *Java.net* [online]. [cit. 2014-11-19]. Available at: https://java.net/projects/jbatchsuite

[7] Java EE 7 and Batch Processing in IntelliJ IDEA 13. In: *YouTube* [online]. 2013 [cit. 2015-05-09]. Available at: https://www.youtube.com/watch?v=416mX1mWb-c

[8] GUPTA, Arun. Forge Addons for Java EE 7. In: *GitHub* [online]. 2014 [cit. 2015-05-09]. Available at: https://github.com/javaee-samples/forge-addons

[9] *Spring Netbeans Module* [online]. [cit. 2015-02-10]. Available at: http://sourceforge.net/projects/spring-netbeans/

[10] JSR-352: Java EE 7 Batch. In: *JBoss Issue Tracker* [online]. 2014 [cit. 2015-05-03]. Available at: https://issues.jboss.org/browse/JBDS-3160

[11] Sapphire Developer Guide: Introduction. *Sapphire* [online]. 2015 [cit. 2015-05-09]. Available at: http://eclipse.org/sapphire/releases/8.1.2/documentation/introduction/index.html

[12] WEINSTEIN, Jesse. ECLIPSE.ORG. *User Interface Guidelines* [online]. 2013 [cit. 2015-05-09]. Available at: https://wiki.eclipse.org/User_Interface_Guidelines

[13] JBoss Tools. [online]. [cit. 2015-05-02]. Available at: https://github.com/jbosstools

[14] The JavaEE Tools project. [online]. [cit. 2015-05-02]. Available at: https://github.com/jbosstools/jbosstools-javaee

[15] The GitHub Flow. [online]. [cit. 2015-05-03]. Available at: https://guides.github.com/pdfs/githubflow-online.pdf

[16] GitHub Guides: Understanding the GiHub Flow. [online]. [cit. 2015-05-02]. Available at: https://guides.github.com/introduction/flow/

[17] The JavaEE Tools project: Contribute fixes and features. [online]. [cit. 2015-05-03]. Available at: https://github.com/jbosstools/jbosstools-javaee#contribute-fixes-and-features

[18] Atlassian Git Tutorial: Merging vs. Rebasing. [online]. [cit. 2015-05-03]. Available at: https://www.atlassian.com/git/tutorials/merging-vs-rebasing/

[19] CHACON, Scott and Ben STRAUB. *Pro Git*. New York: Apress, 2014, 456 pages. 2nd edition. ISBN 978-1484200773. Page 305.

[20] MILATA, Tomáš. Visual (diagram) editor for JSR-352 batch job files: Pull request #326. In: *GitHub* [online]. 2015 [cit. 2015-05-10]. Available at: https://github.com/jbosstools/jbosstools-javaee/pull/326

[21] Builds - tomas-milata/jbosstools-javaee. *Travis CI* [online]. 2015 [cit. 2015-05-10]. Available at: https://travis-ci.org/tomas-milata/jbosstools-javaee/builds

[22] KAZAKOV, Alexey. Unhandled event loop exception when opening Batch Job Configuration editor. In: *JBoss Issue Tracker* [online]. 2015 [cit. 2015-05-10]. Available at: https://issues.jboss.org/browse/JBIDE-19270

[23] ZHOU, Shenxue. Connection issue with node templates declared on same base type. In: *Eclipse Bugzilla* [online]. 2015 [cit. 2015-05-10]. Available at: https://bugs.eclipse.org/bugs/show_bug.cgi?id=463917

[24] SPOOL, Jared a Will SCHROEDER. Testing web sites: Five Users Is Nowhere Near Enough. *CHI '01 extended abstracts on Human factors in computing systems - CHI '01*. New York, New York, USA: ACM Press, 2001, s. 285-286. DOI: 10.1145/634067.634236. Available at: http://portal.acm.org/citation.cfm?doid=634067.634236

[25] NIELSEN, Jakob a Thomas K. LANDAUER. A mathematical model of the finding of usability problems. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '93*. New York, New York, USA: ACM Press, 1993, s. 206-213. DOI: 10.1145/169059.169166. Available at: http://portal.acm.org/citation.cfm?doid=169059.169166

[26] MILATA, Tomáš. Open 'Properties' view together with the Batch diagram editor. In: *JBoss Issue Tracker* [online]. 2015 [cit. 2015-05-09]. Available at: https://issues.jboss.org/browse/JBIDE-19781

[27] MILATA, Tomáš. Content proposal indicator. In: *Eclipse BugZilla* [online]. 2015 [cit. 2015-05-09]. Available at: https://bugs.eclipse.org/bugs/show_bug.cgi?id=466884

[28] MILATA, Tomáš. Add items to element list by dropping them onto a node in diagram. In: *Eclipse BugZilla* [online]. 2015 [cit. 2015-05-09]. Available at: https://bugs.eclipse.org/bugs/show_bug.cgi?id=466883

[29] MILATA, Tomáš. Users have problems finding the ”Parent” button in Batch diagram editor. In: *JBoss Issue Tracker* [online]. 2015 [cit. 2015-05-10]. Available at:
`https://issues.jboss.org/browse/JBIDE-19783`

[30] KAZAKOV, Alexey. Update Sapphire to latest 9.0.0.x version. In: *JBoss Issue Tracker* [online]. 2015 [cit. 2015-05-08]. Available at:
`https://issues.jboss.org/browse/JBIDE-19751`

[31] MILATA, Tomáš. Validation markers in Batch diagram seem like delete buttons to users. In: *JBoss Issue Tracker* [online]. 2015 [cit. 2015-05-09]. Available at:
`https://issues.jboss.org/browse/JBIDE-19782`

[32] MILATA, Tomáš. Confusing ¡id¿ placeholder in Batch Diagram editor. In: *JBoss Issue Tracker* [online]. 2015 [cit. 2015-05-09]. Available at:
`https://issues.jboss.org/browse/JBIDE-19780`

[33] MILATA, Tomáš. Revert Action in Diagram. In: *Eclipse BugZilla* [online]. 2015 [cit. 2015-05-10]. Available at:
`https://bugs.eclipse.org/bugs/show_bug.cgi?id=466931`

[34] MILATA, Tomáš. Visual (diagram) editor for JSR-352 batch job files. In: *JBoss Issue Tracker* [online]. 2015 [cit. 2015-05-10]. Available at:
`https://issues.jboss.org/browse/JBIDE-19717`

Appendix A
User Documentation

A.1 Prerequisities

- You must have Java version 7 or higher installed on your system.
- Download an Eclipse `4.5M7` release from `http://download.eclipse.org/eclipse/downloads/drops4/S-4.5M7-201504301445/` or any `4.5` final release.

A.2 Installation

- In your running Eclipse, click *Help*, then *Install New Software*.
- Type `http://download.jboss.org/jbosstools/updates/nightly/mars/` into the *Work with:* field (see figure A.1).

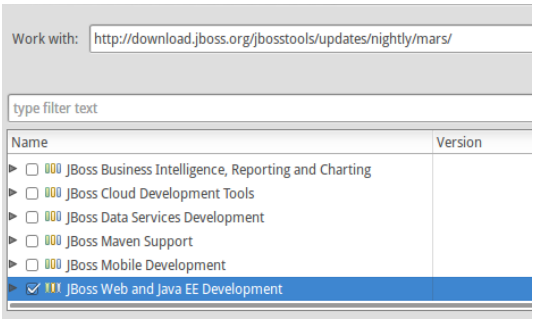


Figure A.1. Installation from Update Site

- Select `JBoss Web and Java EE Development` and click *Next*.
- Procced installation by clicking *Next* and confirming license agreements.
- After installation finish and IDE restart, the software should be successfully installed and ready to use.

A.3 Usage

Tip: See a video of almost all features at `https://www.youtube.com/watch?v=wmWFQKvTWSc`.

- Let us suppose you have created a Java project using menu *File*, *New*, *Java Project*.
- A new batch job can be created by right clicking the project and selecting *New*, *Other*, typing `batch` and selecting *Batch Job XML File*. See figure A.2. Select a name of your job file and click *Finish*.

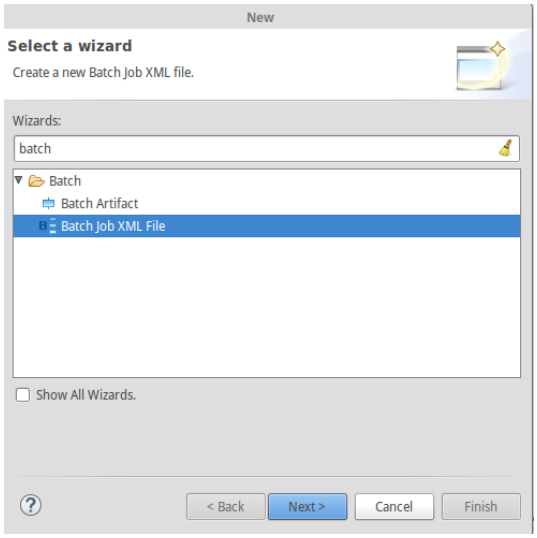


Figure A.2. Batch Job Wizard

- Your job file should open and you should see three tabs: *Design*, *Diagram* and *Source*.
- In the *Diagram* tab you should see a *Palette* view on the right, containig icons for creating batch elements and transitions.
- New elements are created by clicking on one of the *Flow Elements* on the right and dropping it onto a desired position. A text filed for assigning and id to a new element is activated right after the drop.
- You can create a new transition by clicking a *Next* transition in the palette, then clicking the source node and then the target node.
- At any time, you can view the source code by right clicking on any element and selecting *Show in Source*.
- Properties of elements can be edited using a form editor by right-clicking an element and selecting *Show in Properties View*. Context of the *Properties* view is switched according to the currently selected item. See figure A.3

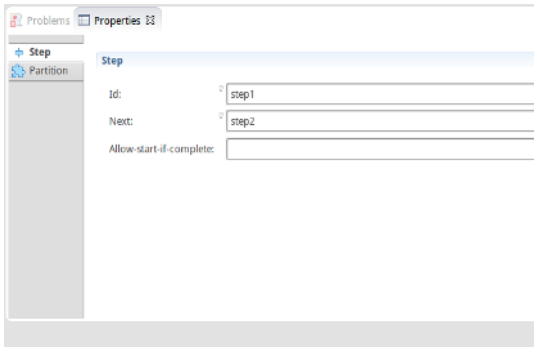


Figure A.3. Properties View

Appendix B
Screenshots

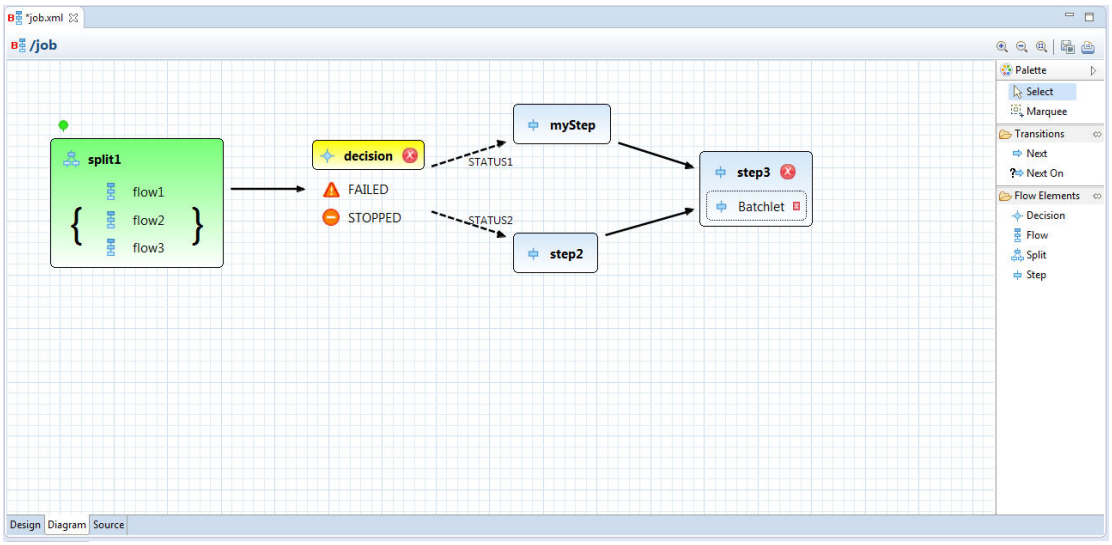


Figure B.4. Screenshot of the diagram editor

Appendix C
Contents of the Attached CD

- `javadoc.zip` — Javadoc generated from the Batch UI plugin.
- `src.zip` — Source code of the Batch UI plugin. Only this plugin was extracted from the whole repository due to a limited attachment size.
- `usability-testing-p*` — Records of usability tests.
- `example.mp4` — Record of development of the weather forecast example application.
- `example.zip` — Source code of the weather forecast example application.
- `java-ee-batch-editor.pdf` — This document.

Appendix D
Glossary

CDI	Context and Dependency Injection
CSV	Comma-Separated Values
EMF	Eclipse Modelling Framework
EPL	Eclipse Public License
GEF	Graphical Editing Framework
GMF	Graphical Modelling Framework
IDE	Integreated Development Environment
JEE	Java Enterprise Edition
JSF	Java Server Faces
JSL	Job Speficiation Language
JSR	Java Specification Request
SDK	Software Development Kit
WYSIWYG	What you see is what you get.
XML	eXtensible Markup Language
XSD	XML Schema Definition Language