jInfer BasicIGG Module Description

Michal Klempa, Mário Mikula, Robert Smetana, Michal Švirec, Matej Vitásek
Advisors: RNDr. Irena Mlýnková, Ph.D., Martin Nečaský, Ph.D.

Praha, 2011

Target audience: developers willing to extend jInfer, specifically modify the way *BasicIGG* creates initial grammar from input files, for example by adding support for a new schema language.

Responsible developer:	Matej Vitásek
Required tokens:	none
Provided tokens:	cz.cuni.mff.ksi.jinfer.base.interfaces.inference.IGGenerator
Module dependencies:	Base
Public packages:	cz.cuni.mff.ksi.jinfer.basicigg.properties

1 Introduction

This is an extensible implementation of the *IGGenerator* inference interface. It is the only IG generator officialy shipped with jInfer.
Make sure you understand the difference between the two types of initial grammar as described in **??**.

2 Structure

The main class implementing `IGGenerator` inference interface and simultaneously registered as its service provider is `IGGeneratorImpl`. In its `start` method it enumerates all files in the input parameter (of type `Input`). For each file, based on its extension the correct *processor* is selected, executed and returned rules are aggregated. After each file has been processed, the resulting grammar is returned by invoking the `finished` method of the callback argument.

3 Processors

A *processor* is a class capable of extracting IG from an arbitrary `InputStream` (usually encapsulating a file). Various processors may handle generic XML, schemas like DTD, XSD or Schematron, query languages such as XPath, and so on.
A processor has to be registered as a service provider of the `Processor` interface from *Base* module. Due to the nature of NBP lookups, each processor is internally kept as a singleton, and should not use its inner state (refer to the chapter Lookups in **??**). Note that the factory pattern is not used here.
Each processor declares the class of inputs (documents, schemas, queries) and file extensions it is able to handle by implementing methods `getFolder()`, `getExtension()` and `processUndefined()`. Refer to JavaDoc of these methods for further details.

BasicIGG comes bundled with 3 processors: for generic XML documents, DTD schemas and XPath queries. Support for XSD queries is implemented in *XSDImporter* to demonstrate *BasicIGG*'s extension capabilities.

3.1 XML processor

XML processor registers itself into *document* input folder, *xml* file extension and declares that it can process other arbitrary file extensions.
SAX traversal is used to collect the rules; the relevant `ContentHandler` is the class `TrivialHandler`. The way it works is following: every time a start of an element is encountered, a new `Element` is created along with its attributes. This

new element representation is then placed on the top of a stack. Every other element or simple data found until its ending tag is created and attached to its subnodes. When an end of an element is encountered, the element currently on the top of the stack is closed and declared to be a rule of the IG.
After reaching the end of the document, all IG rules are returned. Note that this approach creates "simple" initial grammar.

3.2 DTD processor

DTD processor registers itself into *schema* input folder, *dtd* file extension and declares it cannot handle other file extensions.
To parse DTD files a 3rd party library is used: `dtdparser`. Translation from the object model in this library to our own is handled mostly by `DTD2RETranslator` class.
Note that the initial grammar generated in this way is complex. Therefore, this processor checks for the `CAN_HANDLE_COMPLEX_RE` capability of the simplifier following this IG generator: in positive case it simply returns the grammar, if the following module cannot handle complex regular expressions, DTD processor first invokes an *Expander* (see 4).

3.3 XPath processor

This is a rather naïve implementation of an XPath processor. It registers itself into *query* input folder, *xpath* file extension (text file containing one XPath query per line) and declares it cannot handle other file extensions.
To parse XPath queries, standard support present in JDK is used. The relevant `XPathHandler` is the `XPathHandler` class.
TODO vektor Explain what it does...

4 Expansion

Expansion is used to convert complex regular expressions in initial grammar to simple ones (concatenations of tokens) for simplifiers that can handle only the simple form. Relevant interface encapsulating any implementation of such an expander is `Expander` in `Base`. Reference implementation is `ExpanderImpl` in this module. Note that even though this implementation is retrieved using lookups, jInfer bundles only one such implementation and does not support choice among more of them. Anyone wishing to implement his own expander will thus have to either remove jInfer's implementation, or implement the usual module selection.

4.1 ExpanderImpl internals

TODO vektor

5 Data flow

Flow of data in this module is following.

1. IGGeneratorImpl walks over input files in a loop.
2. Each file gets processed by a processor based on its folder and extension.
3. Rules from all files are gathered in a single list (IG) and returned via a callback.

6 Extensibility

`BasicIGG` can be easily extended to support a new input type: just create a class implementing `Processor`, annotate it as a service provider and implement any logic needed. `XSDImporter` is an example of this.

It is possible to replace the default `Expander` implementation: but as mentioned in 4, either the old implementation must be removed or module selection must be introduced.