(Title Page)

In Memoriam,

To my grandfather Osvaldo Pinali
and all other anonymous, imperfect heroes.

CONTENTS

1 Introduction

Science is a way of trying not to fool yourself.

– Richard P. Feynmann

1.1 Motivation

This is a research motivated by problems of old technology, and by problems of new technology. Objects are today taken for granted in development of most new software, and complementary tools like Design Patterns are becoming increasingly of mainstream usage. In the opposite end, maintenance / reengineering of legacy code is a problem that only grows as more software is written.

New techniques require new support from tools. Design Patterns are good for many things. They help communication by defining a higher level vocabulary to express involved collaborations between objects. They help development by providing solution guidelines for recurrent problems, and effectively encapsulating the community's experience – what we find out to be good pieces of design – as reusable components.

The full potential of design patterns will depend on comprehensive, seamless support of development tools: repositories, forward and reverse engineering, refactoring, critics / metrics / quality, formal specification, and others.

Legacy used to be synonymous of mainframe applications written in languages such as COBOL, but right now there is quite a good deal of object-oriented legacy, written in the first OO languages to become popular like Smalltalk and C++. Unfortunately, not all of this code is easy to understand just because it is OO code. The legacy of the future is being written today, and not all of it will be well documented. Unless some revolution happens to this industry, we will always have to handle software that is obscure, badly designed, and poorly documented.

Roundtrip engineering will benefit from techniques of reverse engineering and code understanding even for new developments. New software also qualifies for the benefits of code understanding. Incomplete, in-development software often lacks proper documentation. New developers may join a project and need to catch up with its current state, and this learning is a problem [Brooks]. Learning big class libraries (even when mature and extensively documented) is another well-known time sink, because too often documentation shows endless detail about features, but fails to show how everything fits together, which are the fundamental architectural concepts.

Simpler design and programming practices, e.g. structured programming, lead to simple reverse engineering techniques: tools would be able to inspect source and produce flowcharts, call graphs, or entity-relationship models, without special difficulty. (On the other hand, such tools wouldn't help a lot in handling complex software; both code and design would miss expressive power).

As both languages and methodologies evolve, they become increasingly more abstract and more expressive. A huge part of this abstraction is lost in the long way going from top-level design representations, like Patterns or Architectural specifications (*if* they are ever written), to the floor level of production sources and compiled binaries. Recovery of that abstraction requires additional effort and new approaches. As of today, the recovery of structural information from sources of OOPLs – back to the equivalent class diagrams, for example – is a common facility, but not much additional help is available.

[Chikofsky] provides a list of terms that characterize system renovation, commented by [Brand]. We are directly interested in:

- Reverse engineering "*restricts itself to investigating a system. Adaptation of a system is beyond reverse engineering but within the scope of system renovation*".
- Design recovery, where "*domain knowledge and external information is used to make an equivalent description of a system at a higher level of abstraction. So, more information than the source code of the system is used*".

Additionally, we are interested in understanding complex software (either old, new or in production) by means of patterns, and in the general problem of tool support for design patterns.

1.2 Objectives and approach

It is our aim to investigate automatic documentation of design of object-oriented programs: in particular, detection of not-so-obvious artifacts as design patterns.

Documentation, in our context, means creating some information that is not previously available, or that is not explicit. Class diagrams are not sufficient documentation because they are obvious in the sources of OO languages. They serve as a skeleton, a starting point for the full design information, usually manually provided by a developer.

There is need of tools to do the desired extraction of high-level design information from code, such as patterns.

The detection of patterns requires investigation of a range of subjects, from the basics of parsing and reverse engineering to class structure models, to the representation of programs and patterns, methods and heuristics for detection. Even the parts that we can consider well known may demand special treatment for this particular task. Detecting patterns will require both a methodology, and tools that realize it.

Additionally, we decided to not "hardwire" our efforts to the primary objectives of this research, but to design a generic framework to enable a multitude of work and experimentation with program code, design, patterns, reverse and forward engineering, and related issues. We consider programming language independence a particularly important, additional objective, because we wish to investigate the fundamentals of this problem, while solutions tied to specific environments would more likely depend on *ad hoc* strategies and shortcut general, higher-level solutions.

1.3 Presentation

In section 2: State of the art, we present previous research in this field, and enter in detail on relevant topics for our own work.

In section 3: A Framework for pattern detection, we present the tool developed as part of this research, with some details on its implementation.

In section 4: A Method for Pattern Detection, we present the methodology developed in parallel with that tool.

In section 5: Results, we present the result of running our software on a number of test cases and discuss them.

In section 6: Conclusion, we summarize our findings, contributions, and horizons for subsequent research.

In Appendix: Models, the UML models for all software developed can be seen.

In Appendix: Output from test cases, we make available the data for all test cases, on which the section 5 is based.

2 State of the art

For every complex problem, there is a solution that is simple, neat, and wrong.

– *Henry L. Mencken*

If I have seen farther than other men, it is because I stood on the shoulders of giants.

– *Isaac Newton*

2.1 Presentation

High-level languages, methodologies, paradigms and tools aim to improve software development – to make it faster, easier, more reliable, flexible, etc. and they do it by adding layers upon layers of abstraction over the low-level operations that computers are designed to run.

One thing computers understand and run is a branch. A structured language will make this branch with a procedure or function, abstracting memory addresses with symbolic names and adding other constructs as needed (parameters, return values).

An object-oriented language will add virtual dispatch, where some call might map to different implementations of the same method accordingly to the dynamic type of the receiver; now the concrete procedure is also transparent for the user.

A sophisticated object-oriented language will have very flexible dispatch rules, and additional complexities like methods that can be defined dynamically, manipulated, passed as parameters, or even defined at runtime. It may also define exception-handling facility (as in Java or C++), which is a very complex form of branch.

A design language, such as the UML, will put yet another mask over these mechanisms, and will deal with even more abstract entities like collaborations or transitions in state models.

When highly abstract design is realized into compilable sources and finally executable code, most of this abstraction should be removed because

- The mapping from one layer to the next below it is rarely perfect (semantic gap),
- Efficiency: abstraction is slow. CPUs prefer immediate branches to complex dispatch mechanisms.

The histories of languages and design methods, and compilers and runtime technology (such as interpreters or garbage collectors), are parallel and opposed, as the latter need to be continually improved to remove the abstraction added by the former. The progress of programming languages can even be tracked to depend on advancements of implementations. New technologies that introduce additional layers of abstraction (such as interpreted code or objects) are usually not massively adopted by the industry until better compilers and runtimes are created to eliminate (or even take advantage of) the additional abstraction. When it comes to design methods, users need powerful tools that are mature, integrated with other tools, and make automatic the "bureaucratic" work that is always part of any methodology – such as drawing boxes and arrows, or maintaining repositories.

One big problem not addressed by current technology is the process of recovery. Moving from implementation sources, or even worse, binary executables back to *architectural* schema, is a hard task if the design documentation is not available, because it consists in producing information that was discarded along the forward-engineering process. And very often this information only exists in the engineer's mind. The engineer himself may ignore a lot of it consciously, because he or she works "instinctively", basing decisions on past experience, and not documenting things that seem natural, irrelevant, or obvious.

Reverse engineering is a well-established practice. CASE tools are common to produce good quality structural models out of source code. This is good enough for last generation engineering practice, though. Producing flowcharts out of structured code was very easy (a tool-generated syntax scanner does most hard work), and producing class diagrams out of object-oriented sources is easy enough to produce good results (all information is explicit in the sources, at least in static-typed languages). But we are considering basic-level design practices, i.e. diagrams that map straight to the sources. Recent years witnessed the growth of more high-level mechanisms.

Design patterns, defined "officially" in [GoF], are one such popular, successful higher-level tool. Aimed to object-oriented code, patterns abstract over classes, methods or attributes, and work with more ethereal entities like roles and collaborations. These are not usually found explicitly in object-oriented languages. Even while working on the pure model, developers will convert their patterns to more down-to-the-earth, detailed class diagrams that are then implemented by classes in the OO language. The semantics of patterns is basically lost.

2.2 Difficulties of detecting design patterns

- **Languages have no special constructs to bind roles to types.**

This is usually an informal relationship; the most common hint is use of literature-standard naming. For example, a class named *ClientFactory* is very likely to be an Abstract Factory where the *Client* class is the subject.

We cannot rely on such "hints" because they will have too much variation (we don't have any data on this, but we can expect that when natural languages are involved).

- **Languages have no special constructs to bind messages to collaborations.**

One factor that makes this harder is, collaborations may span messages from multiple object types. Encapsulation would make hard to define language syntax to capture collaborations as whole entities, because the communications between objects have additional levels of indirection.

Ironically, well-written code may be harder to handle. Code that does not break the Law of Demeter [Lieberherr] can make harder to track chains of messages, because these chains are broken in multiple methods instead of being intact, in a single place.

- **Patterns can usually be implemented in a variety of ways.**

Most developers do not follow the pattern diagram they found in a book down to the fine print; this is not the idea. The patterns are guidelines, not final code, and they are supposed to be adapted for specific needs. Some books on patterns present implementation code, but that serves for illustration only. Even the class diagrams cannot be trusted to appear *as is* in the pattern instantiations.

For example, distributed environments like CORBA will generate stub and skeleton classes for the remote objects. These classes implement some patterns (like Factory and Proxy) but they are highly "polluted" by other code – marshalling, error handling, communications, and CORBA-specific bookkeeping like its method lookup, type system, extension and management hooks. This is a lot of "noise" that should be filtered, so we can identify the patterns.

- **Several patterns can be mixed.**

The same class can participate in different patterns. Patterns that lead to common implementation artifacts may share them.

For example, a *Bank* class can be a Singleton (because the application is designed to manage only one bank, which is the root of the system) while at the same time being an Abstract Factory for more than one subject (like *Agency* or *Transaction*). This makes less likely that the developer will use naming conventions to identify patterns; the same class cannot usually have multiple names (*AgencyFactory* and *TransactionFactory*). Having aliases to classes or instances would create more confusion than they would solve. Delegating these tasks to other objects would be a good idea in some cases, but not in all cases.

- **Building blocks used by patterns are usually very common code idioms that can be used by other patterns or by any other code, so we cannot naively match these idioms.**

For example, both a Factory class and a Singleton class may use a method that returns an instance of its class (the Factory may be its own subject, using a class method). They will be otherwise differentiated. The point is that we must pay attention to all details of each pattern. In this case, we should see if the class is able to create additional instances against request from clients (Factory) or if the class stores special instances somewhere (Singleton).

It is important that we don't require too much detail either; otherwise we will miss patterns implemented in a more loose way. The balance is difficult to find, and one reasonable approach is assigning different grades of likeliness, using them to flag or order findings, to help users to filter manually.

- **Not all patterns are implemented consciously.**

Patterns are not usually created, but "found", extracted from experience. When some pattern is published in a canonical form, it has already been implemented multiple times by developers who haven't read the canonical form because it did not exist.

Even after that, lots of developers will keep reinventing the same patterns independently, because they didn't read all the recipes, or because they still lack tools to automatically instantiate patterns and big repositories full of good patterns, or even because the patterns evolve in the code gradually.

- **Not all patterns are well implemented.**

This is partially related to the first problem of having no language constructs for patterns (compilers and other tools have no chance to check anything), and partially to the usual sources of bad implementations.

To give one example, some patterns will demand data (like a Singleton's single-instance reference) and while designing a detection tool, we may consider reasonable that this piece of data is implemented as a private (or read-only) class variable. But somebody can use a public variable – or even a global variable, if the language supports that. Common wisdom points that the number of wrong (or inelegant) implementations of anything will be significant, and the prerequisite that patterns are implemented in a reasonably decent manner is a generally accepted, although implicit, tenet of the research in this field.

Jan Bosch's approach is having language support for design patterns [Bosch]. That idea would make several things much easier, but is not yet supported by languages used in the industry. Bosch also summarizes the problem of traceability: "*The traceability of a design pattern is often lost because the programming language does not support a corresponding concept. The software engineer is thus required to implements the pattern as distributed methods and message exchanges, i.e. the pattern which is a conceptual entity at the design level is scattered over different parts of an object or even multiple objects. (...)*"

2.3 Previous works

A number of papers and previous research contributed significantly to this thesis. Some of them provided different perspectives, while others hinted at the pitfalls.

We will analyze important aspects of this research's previous work, and use them to introduce some decisions.

2.3.1 Starting points

Some previous texts have summarized existing research and papers for this field, and provided the roots for our study.

2.3.1.1 A comparison of utilities for development with patterns

Isabelle Borne and Nicolas Revault [Borne] cover several tools for automating development with patterns. They present the objectives of such tools (modeling, code generation, validation and detection) and necessary features (navigation, production of libraries, instantiation and manipulation, design and consistency checking).

The tools for forward engineering are certainly more numerous and mature than those for detection. There are several approaches for generation and manipulation of tool-generated pattern instances, ranging from libraries to "wizards" or integrated environments.

Their discussion of (dis)advantages of integrated tools has a relevant point for us, where they see the possibility of detection or retro-conception of patterns "*eventually, by means of explicit marking [of code]*" but "*this possibility depends on the reflective, or at least introspective, capabilities of the language*". The use of "marker code" is a common practice of many roundtrip engineering tools, and it makes sense in environments where the patterns are instantiated by a tool in the first place. Reflection seems more relevant for situations like refactoring, as the pattern tool could track [user's] operations on the object definitions; and the analysis of objects in the runtime provides again valuable information about collaborations, that may help the detection work.

2.3.1.2 Documenting design and architecture using patterns: How to detect patterns

Bernet Frédéric, Ceberio Martine and Heulot Olivier [Frédéric] do a summary of existing research for this purpose. They provide an overview of existing research, including BACKDOOR [Shull] and PAT [Krämer].

The classification of utilities *a priori* versus utilities *a posteriori* is interesting. In the first category go environments where information about patterns is available because they are instantiated by tools and tracked by them. Our work would fit in the second category – trying to find patterns in raw sources, which are not structured, organized or complemented by any tool-friendly metadata.

2.3.2 General reverse engineering

Before trying to detect patterns, we should usually climb the step of going from programs to structural models. Even if models (e.g. class diagrams) are available, or if there are available tools that will do them for us, what is available may be insufficient for our needs.

2.3.2.1 Extracting source models from Java programs: parse, disassemble, or profile?

Ivan T. Bowman, Michael W. Godfrey and Richard C. Holt [Bowman]'s paper appeared post facto for our work, but it address some of the same problems we attacked, presenting a similar strategy.

The authors developed tools to reverse engineer Java in three different ways – parsing source code, parsing bytecode (.class files), and profiling the execution of programs (using the JVM's profiling interface). They are interested in analyzing the advantages and shortcomings of each approach, and tabulate all kinds of information each method can obtain.

We have also used three different mechanisms to reverse engineer Java into model data, but we used reflection instead of profiling as a third mechanism. For our conclusions on the differences, see 3.3.2.3: The Mechanism layer.

2.3.2.2 Dynamic reverse engineering of Java software

Tarja Systä [Systä] presents an environment for run-time reverse engineering of Java code, called SCED, to produce state diagrams of applications. The static analysis is fairly easy to do in a language like Java, where static types, control structures and many other artifacts are available, explicit, and easily identifiable in the code (even bytecode). The dynamic, runtime behavior is something harder and very interesting for a number of uses – code understanding and test coverage being obvious cases.

For the design pattern viewpoint, runtime data would clarify collaborations that are not completely understood after static analysis. One reason for that may be absence of type information, which is possible even in Java, when one uses opaque references (*Object*), i.e. in heterogeneous collections. Another reason is the effective impossibility of doing a full analysis of data and control flow statically.

This research produces information as code sequence, concurrency, coverage, and memory management. It starts from static information extracted from the bytecode, and running the application under a debugger automatically generates the event data. The SCED program will visualize the event trace as scenario diagrams, and extract state diagrams from that.

Although the research does not mention design patterns, we could make use of such kind of information. There are patterns that would benefit from state machine information – the State pattern being the obvious case, but we could find others, like Strategy, and also many concurrency patterns.

2.3.3 Methodology

Before creating tools, we need some theoretical perspective on what to do. We should also know what use we will do from the patterns eventually detected.

A number of authors attacked more specifically the methodological issues, or contributed more for our research from this perspective.

2.3.3.1 An inductive method for discovering design patterns from object-oriented software systems

Forrest Shull, Walcélio L. Melo and Victor R. Basili [Shull] contribute an inductive method to discover design patterns. This is an interesting work because they do no tool; it is rather a "pure" methodology as they present "*…a set of procedures rigorously defined in order to be repeatable and usable by practitioners who are not acquainted with the reverse architecting process. Guidelines are provided and a case study is shown that demonstrates the usefulness of the approach*". The BACKDOOR method (Backwards Architecting Concerned with Knowledge Discovery of OO Relationships) approaches the problems of capturing design experience, organizing the patterns, producing metrics on the patterns (like reusability or defect-proneness), and refinement.

This work is centered in a Pattern Knowledge Base, a common point in other related work. We will always need to deal with knowledge about code and patterns, and making this knowledge persistent not only enables forward engineering but could also be used to refine the detection tool's intelligence.

The method itself includes a list of steps going from initial review and preliminary (structural) modeling up to validation interviews, with identification of potential patterns and filtering of such candidates in a cyclic fashion. One interesting detail is the definition of a "correspondence level" for detected patterns; a 4-point scale that considers partial vs. complete matches in the "implementation" axis (as well as in the "purpose" axis. The implementation part is more of our concern (and possibilities), and they put a good question, "*…could the patterns we had discovered be used to create a knowledge base of future use to developers? Or would they need to be extensively generalized before they would make good patterns?*" One interesting path we didn't explore is the partial detection of patterns, or on-purpose detection of "almost good" patterns, e.g. aimed to refactoring work (see 6.2: Future work).

The case study presented used very small student's applications; the authors propose to create "*tools to assist in the process of reverse architecting and automate the feedback loop as much as possible*"; they also mention program slicing techniques and metrics as possible tools to make the method practical.

2.3.3.2 Using patterns for design and documentation

Georg Odenthal and Klaus Quimberly-Cirkel [Odenthal] focus in documenting by designing: unification of design and documentation phases by means of patterns. They report examples taken from one particular project, and after introducing the method to produce patterns, extend it to work on documentation. This includes discussion of organization of patterns as hypertext, and presenting patterns for pattern documentation.

"*Our pattern-oriented approach to documentation concentrates on the* reflective *use of (on-line) pattern texts, be they proprietary or standard. (…) Pattern-oriented documenting logically continues pattern-oriented designing. (…) the design is*

structured independently of the problem domain, establishing a meta-*level documentation: For those designers who are not familiar with the problem domain, but familiar with design patterns, there is a* neutral *access to understanding the system*"

Although reverse engineering / automatic detection of patterns is not a subject, these are exciting and very complimentary ideas. We can imagine an environment of round-trip engineering that expands to the reverse and forward engineering of patterns, and pattern-driven documentation. But we are going to focus a different part of the puzzle.

2.3.4 Pattern management tools, or "environment" approach

These are tools that handle patterns, usually with an emphasis on the full engineering cycle (forward and reverse). They may be able to perform the detection the "easy way" because the patterns live in a well-controlled environment – they are generated from standard templates, changes are tracked, everything is consistent to some repository, etc. Even if a tool attempts to detect patterns in arbitrary code, it helps a lot if a rich environment enables the user to visualize, filter and use the findings.

2.3.4.1 Tool support for object-oriented patterns

Gert Florijn, Marco Meijers and Pieter van Winsen [Florijn] develop a prototype tool that supports design patterns for maintenance of OO programs. They work in the Smalltalk environment to easily integrate / extend source browsers. There is an integration of several views of the program: code (standard browser), design (the OMT tool) and design patterns.

The pattern tool assists in instantiating patterns; integrating them into existing program elements, and checking if (user-modified) instances of patters are still well formed. Their objective is "*introduce patterns as first-class citizens in an integrated OO development environment*".

This tool addresses the problem with the approach "Design environment support" described by [Bosch]: No language support for patterns, but rather having an environment that automates the instantiation and tracking of the patterns in a conventional language.

Central to this work is the representation of patterns. There is a repository of pattern definitions, organized as *fragments*. There is a fragment database, a fragment browser / inspector, and a fragment model. Fragments represent design elements of some type (like class, method, pattern, or association) and they relate to each other, arranged in graphs. A single-level model contains both pattern definitions and pattern instances, which are generated by cloning in a prototype-based fashion; similarly, pattern definitions may be generated from existing code by simple copying.

Reverse engineering of patterns, from code that was not done with this tool, is not an objective; but it is an objective to validate transformations (manual edition, refactoring) to keep implementation consistent with the definition in the fragment database. The authors' initial attempts, adding constraints over instance elements and checking them with procedural code, resulted too dependent on a strict structural definition of the patterns. This issue was addressed with a declarative approach and better modeling of relations than originally available. This is relevant basis to our own work, and the important things to remember are:

- **The pattern definition is complex.** We need more than the information typically found in class diagram representations of patterns. Additionally to the structural elements, and structure relationships like “extends”, we need collaboration / behavior relationships like “creates” or “uses”.
- **A declarative approach works better.** The verifications to be done will be often possibly indirect, requiring recursive scanning of graphs. Universal and existential quantification, and “on-demand” production of needed information, will make this work substantially easier.

This work contains some limitations when put against our objectives. First, it is language-dependent. Second, the approach of implementing validations with Smalltalk code works fine to verify one or a very few elements at any given time, because the whole environment induces to incremental validation of the changed code only. That would probably not scale to reading some hundreds of classes and matching them against every constraint because we ignore where are the patterns. (Smalltalk code is used to implement declarative-like constraints, but that means a *#forall:* is actually a loop that is always re-executed for every domain item, and this is a problem in *our* context). Finally, important aspects of the program are not available as fragments: method code is opaque to the tool, which doesn’t find things like message sends (this is planned as future work) and thus there is no behavioral conformance. This is not as critical for their work, but in ours, ignoring the behavior will typically produce large amounts of false hits.

2.3.4.2 Pattern-based reverse-engineering of design components

Rudolf Keller, Reinhard Schauer, Sébastien Robitaille and Patrick Pagé [Keller] follow many similar goals to ours, working in their SPOOL environment that is the best description we have found of an ideal world for design pattern detection. SPOOL includes reverse engineering featuring UML 1.1 support, a design repository where the UML metamodel is mapped to an OODB, and a powerful graphical user interface where results of their findings can be analyzed in multiple ways.

This is a different kind of environment: rather than visual tools and tracking, it is an environment to enable better detection tools to be created, and to integrate with each other, to further multiply their capabilities.

There is a strong stress in the fact that completely automatic design recovery is not always possible, and even when it is possible, the number of hits can be so big that users need help to find the really relevant ones (e.g. template methods are found in the thousands).

One interesting counterpoint to the difficulties of automatic detection is found when they conclude, “*the effort for manual recovery of a significant number of design patterns in large-scale systems is infeasible, even with the use of state-of-the art software comprehension tools*”. As a justification, “*it is these patterns of thought that comprise the rationale of many pieces of an existing software system, and to comprehend the software, we need to recover these patterns, be it automatically or manually*”.

A few of their plans for future works have been addressed by our research:

- *"First, we will continue extending our repository to capture all major constructs of C++ and to cover additional programming languages*". They also have some extensions of UML to better support C++. We thrived to extract as much detail as possible from the programs, and we have no language dependencies beyond the parsers.
- "*The schema of the repository will be based on multiple logical layers, each increasing the level of abstraction of the source code models*". We have done a multi-level, generic design in a pervasive manner, including representations and implementation.

2.3.5 "Hard" pattern detection tools

This is where our own research is focused: automatic detection of patterns in arbitrary program code, in a non-ideal environment.

While we do not pretend, and our experience as well as previous works deny, to obtain perfect, fully-automatic detection of every pattern, we consider critical to do *as much as possible* in an automatic manner and without aid that is not always available. We would ideally have a full environment and facilities for manual filtering, but we should minimize the dependency on the environment and the burden on the user.

It's also worth notice that environment-aided approach may not apply to code that was not developed in the said environments in the first place; automatic detection, as hard as it can be, may often be the only option available.

2.3.5.1 Design recovery by automated search of structural design patterns in object-oriented software

Christian Krämer and Lutz Prechelt [Krämer] produced the PAT system, one main inspiration for our work. This is a tool that detects some structural GoF patterns in C++ source code: Adapter, Bridge, Composite, Decorator and Proxy. A commercial CASE tool is used to reverse engineer sources into OMT models; the patterns are defined as OMT diagrams as well, and everything is exported to a Prolog inference engine. The patterns become rules and the sources become facts, and Prolog's query evaluator will match the facts to the rules and find patterns. User intervention may be needed to filter false hits. They generate LaTeX output so existing, popular, and standard formatting software can be used to visualize it. Finally, they also offer a good analysis of the evaluation and metrics issues for pattern detection.

The authors mention limitations: "*One important limitation of the approach is that some characteristics of design patterns require too much semantic information about the behavior of the methods to be modeled by a CASE tool, let alone to be extracted from source code automatically today. An example is the design pattern Composite which requires operations for adding and deleting elements and for iterating over all of these elements. For the same reason it is also difficult to find behavioral patterns instead of structural ones.*"

These paragraphs roughly define the current state of pattern detection:

1. Working with structure is not good enough;
2. Working with behavior is hard. Existing tools do not provide them. The mass of information may be excessive and make analysis impractical.
3. Structural patterns are the "practical" ones because they only need structure.

Their work also summarizes the field and provides the starting point for this thesis: reverse engineering, rule-based inference engines, representation of patterns and code, are the fundamental pieces.

2.3.5.1.1 CASE tool / reverse engineering limitations

We have also used a number of CASE tools, both commercial and public domain, and the "state of the art" is class diagrams. The best ones can do roundtrip engineering or integrate with the developer's environment, i.e. they combine forward and reverse engineering, keep track of manual changes and synchronize everything.

Nothing seems to be available to recover behavior, though. A few tools and research are recently doing sequence diagrams (from sources) or state diagrams (from profiling / coverage execution), but we need to look inside methods and have as much detail as possible.

Our decision was starting from scratch and designing a reverse engineering tool that would be ideal for this work.

2.3.5.1.2 Inference engine

Working with an inference engine can be quite involved, because it is likely that not all of the work is done there. PAT uses helper utilities written in the CASE tool's scripting language to convert pattern models to rules and code models to facts.

Given that the CASE tool and the Prolog software are separate environments, there is likely some work in the bridging. One interesting possibility is using an *embedded* inference engine, i.e. one that runs as a library, or component, controlled by the main program. This enables easy collaboration of the AI work with other tasks, so powerful strategies may be implemented without difficulty.

For details on our approach, see 3.3.4: Inference engine.

2.3.5.1.3 Knowledge model

We need as much relevant information about the sources as possible. Another important issue is how to organize it. [Florijn] created a "fragment model" in order to formalize the patterns and their instantiations. [Krämer] uses the Prolog syntax to define patterns.

This is an interesting idea. Notations like Horn clauses can serve as a surprisingly efficient formal notation for code artifacts. They are simple, precise and readable, and lead easily to algebraic reasoning or graph / sets theories. And they can be executed with no semantic gap[1]. We eventually decided to not need to invent yet another formal language, to define patterns.

[1] Except that inference engines can be tricky, and transformations may be necessary to make the rules more efficient (and less readable).

2.3.5.1.4 Differences in our approach

Because the work on PAT laid out the foundations for our own work, we should mention a few relevant differences at this point. See also 3.1.1: Justification.

- Instead of OMT diagrams coming from some tool that may have weaknesses in its reverse engineering, we decided to do the full UML 1.3, because we felt that it would be rich and extensible enough to support anything we could do without coming up with "unclean" extensions or language dependencies.
- By doing our own reverse engineering software, we could also obtain information about behavior, such as message sends, that is missing in PAT; their rules are purely structural and they report a relatively high percentage of false hits that they attribute to this lack of information.
- Instead of creating a standalone tool that does one thing, we decided to be more ambitious and design a general framework for reasoning about code, for which detecting patterns would be only one possible use. (This issue is not even mentioned by the authors of most research.)

2.3.5.2 Requirements for integrating software architecture and reengineering models: CORUM II

Rick Kazman, Steven G. Woods and S. Jeromy Carrière [Kazman] present CORUM II, "*…a generic framework for the integration of architectural and code-based reengineering tools. This framework is needed because there is a variety of standalone reengineering tools that operate at different levels of abstraction ranging from 'code-level' to software architecture. (...)*".

The integration and the tools discussed include creation and manipulation of Abstract Syntax Trees (ASTs), Control Flow Graphs (CFGs) and Data Flow Graphs (DFGs). Their work is mostly concerned with bridging the gap between multiple tools and multiple levels of abstraction, and they present a detailed model for architecture representation and transitions, standardize syntactic and semantic elements so they can be exchanged and the strengths of several tools can be combined. The entire framework is not directly relevant to this thesis; they neither mention patterns, nor focus reverse engineering in particular.

The fundamental concepts are still important. Working with multiple isolated systems – like CASE tools, AI shells, and custom programs written in "conventional" languages – is likely to happen here, and some care and planning should take place. There is a big abstraction ladder going from source code to design patterns.

The CORUM paper mentions data like ASTs. Some tools do manipulations and transformations in the ASTs, and this has advantages:

- It is easy – parser generators do most of the hard work.
- No information is lost, because the AST will contain all information in the sources.

On the other hand, ASTs are completely language-dependent. Tools typically need to be rewritten for different languages, and updated significantly should the language evolve. All the effort of doing a tool is also diluted, because it only applies to one language. General know-how is also hard to extract from experience, because it is too mangled with language-specific details. Finally, ASTs are very low-level; they are like Assembly programming for architectural work.

We decided to do every effort to make this research as general as possible, independent of particular languages or other traits. A number of design ideas helped realized this aim, the major being the use of UML-compliant models extensively (*not* for simple import / export / visualization of models).

2.4 Conclusion

Automatic detection of design patterns in arbitrary source code is hard. No commercial tools are able that even try to do it, and the research is in early steps.

Detection of design patterns starts where conventional reverse engineering ends: moving from sources to structural diagrams. The detection will likely use some sort of Artificial Intelligence strategy: the patterns are far from straightforward to identify; the identification work may be modeled as pattern matching.

There are a fair number of variants in this research. Tools may be integrated to forward engineering environments or not. They may benefit of repositories or not. The design documentation may need to be built from zero, or incrementally patched after manual changes of sources previously synchronized with some pattern definition.

Of these scenarios, it seems we've chosen one of the most difficult but also interesting cases: fully automatic discovery of design patterns in raw program code, without any special metadata or environment.

3 A Framework for pattern detection

Basic research is what I'm doing when I don't know what I am doing.

– Wernher von Braun

The strongest arguments prove nothing so long as the conclusions are not verified by experience. Experimental science is the queen of sciences and the goal of all speculation.

– Roger Bacon

3.1 Introduction

This section will present a framework for reverse engineering and reasoning about code, oriented for pattern detection, which was developed as part of this thesis. This software:

- is a very complete, general-use reverse engineering tool;
- in addition to doing the work specific to our research, is an extensible, reusable framework for several kinds of tasks;
- implements a methodology that we developed for pattern detection;
- demonstrates the concepts explained in section 4: A Method for Pattern Detection.

3.1.1 Justification

One valid question at this point is why doing all this implementation. CASE software and reverse engineering tools, at least, are available in variety.

Pattern-detection software is not available, except for a very small number of research prototypes. We described them in the previous section, and we identified a number of shortcomings that should have had impact on their efficiency:

- **Dependency on weak tools**

Using a commercial CASE tool is an easy way to obtain model information from source code; the tool does the structural reverse engineering and all there is to do is importing its files.

Such tools may have limitations in the models, or perhaps in their reverse engineering facility, which affect the pattern detection job. For one thing, only structural data will be

available, while the kind of reasoning we want to do about the code demands at least some of the behavior to be analyzed (e.g. message sends or instantiation).

And then, there are other tools that will make the behavior available, but will be weak in the structure, or in the support for OO-specific features… working with abstract syntax tree or procedural call graphs is a bad start. To summarize, we weren't able to find anything that would fulfill our functional requirements.

There is a catch, too: importing models from CASE tools may turn out to be as hard, or harder, than implementing the equivalent reverse engineering (structural-only); this is true, for instance, for inspection of Java bytecode: both reflection and class files make inspection extremely easy.

- **Use of simple, custom models**

Both code and patterns should be represented in some way so we can work with them. Some previous works adopt the models provided by tools they rely on. Others create their own meta-representations. In the former cases, the models may be not ideal for the task at hand; in the latter ones, the models can be too simple (to minimize effort) or they can be too customized (and make hard any integration, reuse, or extensions of the tool).

We considered a good idea to adopt the standard UML, the Unified Modeling Language, to represent code. The advantages are clear: the UML is a standard metamodel, and it is a very sophisticated and very well designed one, so it is much better than anything we would be likely to invent for the same purpose. The UML is a high-level spec that will enable tools to be language independent – although we've selected the Java language, we would like to produce something that is of a more universal value than peephole detection strategies that work only for Java.

- **Restrictions on freedom, integration, experimentation**

This is undiscovered country, and we felt that having control over every part of the process, and having everything integrated, would make possible to address any problem much better than if we were relying on somebody else's (closed-source) tools.

This point proved important in more ways than expected. Because we adopted an open-source, embedded inference engine, we could identify some issues that impacted the performance of our tool, and suggest improvements that were adopted by the inference engine's author.

To summarize, we hope to do a better work by using what we consider to be an ideal environment for what we are doing; and we had to create this environment because it was not available. This environment contains a "foundation" that we describe here, and it supports additional code to do the actual finding of patterns, which we describe in section 4: A Method for Pattern Detection. The driving idea of this research is not any really innovative method to detect patterns; we rather aim to incrementally improve over previous attempts by designing a better toolset and detailing the accompanying methodological aspects. It is our expectation that we can achieve improvements not only because the tools will be powerful, but because they will be very flexible and make experimentation easy.

3.2 Implementing the Unified Modeling Language

The Unified Modeling Language [UML] is a recent standard. There are a number of tools produced by the research community to work with modelization, like the OMT-tool; but there is nothing for UML that we could use[2]. And we need the metamodel, not visual tools or files, so the products of CASE tools supporting the UML wouldn't help.

We produced an implementation of the UML 1.3, which is actually the first contribution of this thesis. This implementation is built as an independent, self-contained library, and designed to support a black-box form of reuse.

The implementation is partial: it contains the Core (Backbone, Classifiers, Relationships, Dependencies); Extension Mechanisms; Data Types (Data Types, Expressions), and Model Management. There is an additional Util package with extensions. The other packages are not available as we didn't need them (Common Behavior, Collaborations, Use Cases, State Machines, Activity Graphs), but they are planned: see 6.2: Future work.

3.2.1 The specification

The good part about implementing the UML is the availability of the full model. We decided to work with the UML 1.3 specification, which was in final draft form at the time of our work. Still, the specification needs to be complemented, mapped and interpreted in a number of places.

To give one example, the spec includes OCL constraints (UML's Object Constraint Language) for Well-Formedness Rules. These are a highly desirable property of the metamodel. Models can be very complex, and we are going to have a new reverse engineering tool that creates them, so they are likely to be ill formed as effect of bugs in the producer (the client of the UML package). The constraints are, then, a major lifesaver. This is so important that it is one of the main reasons why we decided to do our own implementation of the UML. OCL code cannot be included as is in a Java program; then we need an equivalent code, and we need a number of other elements as well.

3.2.2 Main design decisions

The UML is big, and it defines a very large graph of complex objects. While thinking on how to manage this, we eventually decided on a number of relevant strategies.

3.2.2.1 Type system

The UML is very complex, on the abstract data type (ADT) viewpoint. It contains several uses of multiple inheritance, some "implicit" (non-modeled) types, recursion and cyclic structures, extension facilities, and large service interfaces when everything is combined.

We decided to use Java interfaces massively. Almost everything in the UML spec is mapped to interfaces instead of classes, and there are implementation classes, which are all isolated in "private" packages. A factory was introduced and should be used to create *all* metamodel objects. This also allows us to have two public interfaces in the classes:

[2] There is Argo/UML, an open-source CASE tool project [Argo], which contains an implementation of the UML metamodel. But we considered this implementation not ideal for our work.

the client interface (methods appearing in the Java interface) and the internal interface (methods that are public in the implementation classes, but do not implement any operation defined by the corresponding interface). We wound that this is better than package-level visibility, because the UML is split in several subpackages but there are strong relationships between types inside different subpackages.

We introduced some interfaces that are not part of the spec. The most important is the *TypedElement* interface, defining any metamodel objects that has a type (a *Classifier*). Typed elements are *StructuralFeature* and *Parameter* – their common denominator is *ModelElement*, where a *type* relationship could not be put. And the *TypedElement* would be needed to handle local variables, in the reverse engineering of code, so we have a third typed element outside of the UML, the *LocalVar* class.

3.2.2.2 Extension

The UML contains a complete extension mechanism (stereotypes, tagged values) but that is aimed to extending the metamodel. This is different from extending the metamodel's *implementation*. We need to do this second form of extension in order to adapt the UML objects to the implementation of most non-trivial tasks.

Our implementation implements the Visitor pattern [GoF] that enables clients of the UML package to extend its behavior in a clean manner.

We found that we would want to extend structure too. Inheritance could be used, but it was not desired in our specific scenario (storing unique IDs and detailed information about behavior in metamodel objects). A Decorator was not desired as it has the side effect of changing the types that we are using, and changing the creation procedure. We developed a mini-pattern that solves the problem in a slightly different way.

3.2.2.2.1 The Custom Data mini-pattern

Intent: Attaching new data to objects.

Motivation and Applicability: Insert new pieces of data in another object, without going to the trouble of creating derived classes or new decorated types that point to the primary object. Both force us to define new classes, and we may need to do them multiple times if we want to extend multiple objects without a common base. We could also have a hashtable mapping objects to the extended data, but this would be inefficient and fragile because its management must be manually coordinated.

Structure: Custom Data is the structural equivalent of a Visitor. There is an interface (or abstract class) called *CustomData*, defining at least two services – *getCustom()* to retrieve the "custom data" of the object, and *setCustom()* to store it. A more sophisticated *CustomData* may accept indexes to enable storing multiple items in a list, or otherwise store multiple items in some other form of collection.

Implementation: Trivial. The pattern also includes a standard implementation of this interface, the *CustomDataImpl* class, which can be used by extended objects with inheritance or delegation. Generic types or wrappers may be used in static-typed languages, to make the Custom Data facility type-safe.

Consequences: We can add pieces of data to existing objects in an easy and efficient manner. The cost for each extensible object when is one object reference, pointing to the custom data if any, or to a collection object that may be lazy-created.

Related Patterns: Decorator and Visitor [GoF].

3.2.2.3 Externalization

Even though we wouldn't have time to support integration with a CASE tool or other environment, it is desired that the produced models are in some way possible to visualize. These models can be quite large and complicated, so it may be very hard to track them in debuggers. Even inspector-style environments like Smalltalk's would offer a piecemeal view of the objects, making hard to fully browse it.

Instead of writing a new GUI browser or supporting some complex file format to communicate with existing tools, we decided to use the eXtended markup Language, XML [XML]. There is a standard XMI DTD in the new UML standard, but it is very complex too so it wouldn't help us a lot. We can put the solution in a new mini-pattern.

3.2.2.3.1 A debugging mini-pattern: Poor man's XML

Intent: Browsing graphs of objects of any complexity.

Motivation and Applicability: We need an easy-to-browse output of a complex graph of objects, for example for debugging purposes. But avoiding work is highly desirable. The result can also be useful for a crude, prototypical form of user interface or persistence purposes, depending on the implementation.

Structure: We can piggyback into available XML viewers. Every object in the problem should implement a method for dumping to "quick and dirty XML". One candidate is the standard *toString()* in Java, *#writeOn:* in Smalltalk, *operator<<* in C++ or equivalent.

Implementation: The definition of the XML to be used is simple:

- Enclose each object, of type MyType, with tags generated from its dynamic type's name: "`<MyType> myData </MyType>`". It is easy to do that in a generic way if the language supports type introspection.
- Have a small set of helper methods to automate work such as iterating over collections, and converting special characters in the data, like "<", to the proper HTML sequence, like ">".

Example of our implementation, for the class *AssociationEndImpl*:

Listing 1: Poor man's XML support code

```
public void writeOn (Writer wr) throws java.io.IOException
{
        super.writeOn(wr);
        Util.writeOn(wr, "isNavigable", navigable);
        Util.writeOn(wr, "ordering", ordering);
        Util.writeOn(wr, "aggregation", aggregation);
        Util.writeOn(wr, "targetScope", targetScope);
        Util.writeOn(wr, "multiplicity", multiplicity);
        Util.writeOn(wr, "changeability", changeability);
        Util.writeOn(wr, "visibility", visibility);
        if (type != null) Util.writeOn(wr, "type", type.getName());
        Util.writeOn(wr, "specification", specifications, false);
        Util.writeOn(wr, "qualifier", qualifiers, false);
}
```

We are using here a set of helper methods (*Util.writeOn()*) that handle multiple types, take care of special characters, introduce the tags automatically, and understand collection objects like *specifications* including options to do deep or shallow visit of these (we need to avoid cycles). The helper code is general and reusable, once done.

The resulting output may be dumped to a file and visualized with any XML browser, including Internet browsers. There are browsers that won't complain about the missing DTD, so we gain a sophisticated visualization of our data for free, with effort equivalent of the common *toString()* that many programmers use to pretty print their objects in debugging time. The advantage is enormous for a graph of thousands of objects. XML browsers typically allow expanding / retracting nodes, search, use syntax highlight etc.

Consequences: Easy visualization and comprehension of complex data structures, and relationships between objects. But we need to keep the formatting methods up-to-date after changing objects.

Related Patterns: The objects may be implemented as Composites, or they may support the Visitor [GoF]; both cases may lead to alternative implementations.

3.2.2.4 Well-Formedness Rules

Because Java doesn't support declarative programming, the OCL constraints must be transformed in conventional methods that we call when wanting to check if an object is well formed. This immediately introduces a problem: *When* to do that? Having some "check" method than analyses the object and throws an exception or somehow flags the error does not solve the consistency problem. We have no support for Design by Contract [Meyer]. If we are forced to call this method manually, this may be significant work and we may forget to do it. If the method is executed automatically (e.g. after any change to the object's state, or to its collaborators') the performance hit is big when we do a lot of manipulation in the same object (a single check in the end would be better), and the checks are not trivial.

Also, the object may *need* to be inconsistent for short periods of time, and it may be hard to avoid that between method calls. It is typical to use state variables to fool the class invariant.

We introduced an approach where automatic verification is available, but optional. Clients of the UML objects may turn automatic verification on and off – or they may even compiled the UML package without automatic verification code at all in "release" builds. In any event, the check may be invoked manually.

3.2.2.5 The UML Context

The UML defines a set of standard elements: Stereotypes, Tagged Values and Constraints that are predefined. Such objects are modified ass the UML packaged is used, because the standard elements will point to objects using them. This is a problem if we want to have multiple independent models in the application, and it is a problem for memory management too. Actually, the problem is the same of global variables, as that is what the standard elements are.

We introduced a new object called *UMLContext*, and extended *Model* to have one such context. Its use is transparent for the client, but the context is available should the client want it.

3.2.2.6 Type Proxies

Some tools that work with models have often the need to handle *unresolved types*. This may happen in a forward-engineering tool if the user defines some element (like a Parameter) but doesn't specify its type. It can happen in a reverse-engineering tool, if one class depends on another to define the types of some feature, but the other type wasn't yet discovered (and a recursive implementation may be undesired or not possible). Other possibility is when doing type inference work, e.g. if the language analyzed has no static types.

In such cases, we need to put a placeholder in lieu of the type, and replace it with the correct type object (some kind of Classifier) when it is found. This substitution is not trivial because we may ignore even the specific kind of classifier (e.g. in Java, object references can map to either Classes or Interfaces) and the replacement usually means pointing to an existing object and not cloning its contents.

The Proxy pattern [GoF] solves this problem. Our implementation of the UML package offers type-safe proxies for the entire hierarchy that ends in *Classifier*, and includes *GeneralizableElement*, *ModelElement*, *Element* and *Object*.

Additionally, we have a *ProxyKillerVisitor* that scans a model and removes all proxies, and puts the pointed classifiers where the proxies are found. We can then remove the overhead of proxies as soon as all types are resolved in the model (or a sub-tree of it).

As effect of the interface-based design, the proxies look like normal model elements and they are transparent for both clients and the UML package's internals. The exception is that some well-formedness rules must detect proxies and skip verification of objects that may have incorrect kinds of classifiers because they are still unresolved.

3.3 Reverse Engineering

We decided to do a full reverse engineering engine because none was available that would be ideal: generating highly detailed information about the code, supporting the UML metamodel, being open to allow any customization. The task turned out to be a sub-project with its own share of research and development.

3.3.1 The Java language

The Java platform [JLS] [JVMSpec] was selected as an initial target for a number of reasons:

- Java is a modern Object-Oriented language, and we feel that patterns are quite popular in the Java community.
- Java is a statically typed language, something that makes our work much easier because type inference is not needed (as much).
- Java supports enough metaprogramming to allow using this method for a good prototype.
- Java is simple enough; its grammar is tractable although not small, and it lacks a preprocessor to destroy semantics in source code. Our previous experience with roundtrip engineering CASE tools supporting Java hinted that Java is a very friendly language for that.

All constraints about easiness are useful so we wouldn't waste too much time in implementation details that do not contribute to our main objectives (since they are not tied to any language). It would be otherwise more useful to have support for a language for which a very large amount of legacy code exists, such as C++.

Java was also selected for all implementation.

3.3.2 Multi-layered reverse engineering

It is easy enough to write a parser for some language that emits interesting bits of code, such as class definitions or method calls, to an inference engine. It is also useless in the long run. Our strategy includes many levels of genericity.

3.3.2.1 The Generic layer of reverse engineering

3.3.2.1.1 The *GenericReader*

This contains the core reverse engineering that is language-independent. It contains a few template methods to bind to the language-specific code, and some job management and other bureaucracy.

The second part is a lot of job management and type management. Dealing with types (i.e. classifiers) is quite complex, because there are diverse complicated situations: Types may be needed before they are available and recursive resolution may be a bad idea, and we need to use placeholders for "unresolved types" which may contain or not knowledge of the specific classifier class. These missing types may be defined later so we should fix references to them (this is solved using proxies). We should enable

support for primitive types and special cases in the type system of any language, so there are hooks for these.

These special cases in the type system are very annoying and we created for them this definition.

Definition 1: Bizarre types

Types neither known at compile-time, nor explicitly defined by code in libraries or user's source are known as ***Bizarre Types****.*

Bizarre Types are anomalous and ugly things produced by the compilers or runtimes in an on-demand fashion. In the Java language, arrays are bizarre types. Types like *int[]* or *MyWindow[][]* do not exist in the language, and they cannot be defined by Java sources, i.e. by a class (even though it will represent a class and its instances will be objects). They do not even exist in bytecode form (other than the name mangling used to refer to them). The source-to-bytecode compilers have special handling for these types (they assume the existence of a *length* public attribute, and emit special bytecodes for uses like creation). The Virtual machines also have special handling: they create the corresponding classes dynamically, when they are first needed (so even reflection is available; they look like normal objects).

A reverse engineering tool should do the same thing: create the bizarre type the first time it is needed. We opted to put these types in a special package where primitive types also live, so we can avoid all of them easily.

Arrays are actually half-baked generic types. Proper generic types could be handled using the same mechanism, except that we would store the generated classifiers in the namespace of their realization (Java arrays are not packaged).

3.3.2.1.2 Model post-processing

We aimed to remove as much complex and reusable work as possible from the language-specific code, so we defined a system where that code is allowed to produce a very "raw" model of the code. For example, the complex binding of operations required by UML is not necessary. Certain issues can only be solved after all related classes are scanned, and not in a per-class basis.

There are a number of post-processing operations that will fix the raw models. The *GenericReader* invokes them after the scanning job queue is empty. All of them are implemented as Visitors over the model elements.

UML requires that classes have all operations defined by the interfaces they implement, so we should copy the operations *unless* the classes either define or inherit them; the *InterfaceCollectVisitor* handles this transformation. Multiple methods that override each other in a polymorphic chain should be bound to a single operation.) The *MethodBindVisitor* implements this.

Then we hit the problem of resolving Associations. This is not an exact science, and we do the possible here. The raw model will contain Attributes for each and every class or instance variable found; the *RelationshipBuilderVisitor* will decide if they are attributes or if they should be transformed into some form of relationship. The current strategy is transforming attributes into associations when their type is neither primitive nor bizarre. We resolve 1-N multiplicity associations when the information about the element type is provided by the language-specific code (true for Java arrays or homogeneous collections like C++ template instances would be). We didn't attack the

identification of aggregations and other details, like whether a pair of associations A→B and B→A should be transformed into one A↔B; this would be quite hard and we notices that even high-end commercial CASE tools do not try to do any better. It would probably pay for the pattern detection, though; see 6.2: Future work.

3.3.2.2 The Language layer

This should be reimplemented for each language, and will contain all code that depends on the language but not on the introspection mechanism (such as source parsing or bytecode interpretation).

Our *JavaReader* class implements this. The first thing it does is filling the roles defined by the *GenericReader*: supporting primitives and bizarre types (arrays). Also provided here are more trivia like our entire command-line user interface.

The other major work in this layer is building of UML objects. The introspection mechanism will find the information necessary to define something; for example, to produce a method we need to know its enclosing classifier, its name, modifiers (like visibility), names and types of all parameters (if any), and type of return (if any). After the introspection code (like a parser) collects all this brute data, it calls a service from the language layer to produce the entire mesh of UML metamodel objects that map the method.

3.3.2.3 The Mechanism layer

This is the low-level layer of the language-specific code, and it deals with introspection of the physical target. We can think on four different ways to inspect code: reflection, bytecode disassembling, parsing sources, and profiling.

Multiple mechanisms are implemented and they are interchangeable, realizing the Strategy pattern [GoF]. There is an interesting extension of the Strategy that we call *fall-through*. If one Concrete Strategy fails to accomplish its task, it may define a Fall-Through Strategy that will retry it. We use this in the Source mechanism; if it fails because sources for a class are not available, the systems falls through the Class File mechanism.

3.3.2.3.1 Reflection

This method was our first prototype because it is very easy; we can dynamically-load the classes and query their properties. Unfortunately, in Java the class loading has side effects (compilation and execution of class initializers, and no guarantee of cleanup) and the reflection gives no access to method code.

3.3.2.3.2 Bytecode `.class` Files

This is very similar to reflection but without the limitations, and it turned out to be *easier* because we found an excellent library from IBM's alphaWorks [CFParse] that provides powerful, easy reading and even writing access to class files, from structure to method op-codes.

We started to implement decompilation of bytecode but left it unfinished, as using sources seemed a better idea for having full access to behavior.

3.3.2.3.3 Sources

This is the most complete and powerful way to obtain static information about code. It is also the harder by far (as noticed by [Bowman]). The Java grammar turned out to be more complex than our expectation.

We used a JavaCC / JJTree-generated parser [JavaCC], based on code from the "VTransformer" example of JavaCC that fully supports version 1.1 of the language. This tool-generated code produces an Abstract Syntax Tree with support for the Visitor pattern included, so our analysis of the sources is one big Visitor. It is actually too big, so it was split into many classes bound by inheritance. ("Categories" a la Smalltalk would have helped; the problem is the excessive number of methods in a single class.)

At this point, we started to write a compiler – something really not planned, but the elements of compilers just multiplied. We needed to do a lot of type resolution, including management of imports (with a separated class that piggybacks on the class file introspector because we may not have sources of imported types) and symbol tables. We had to do automatic generation of code elements such as default constructors and super-call of parent's constructors, just like compilers do. And we had to generate a quasi-complete intermediary code for the method behavior, so there is a bit of semantic work too.

One additional visitor is provided to generate XML from the AST, so we can visualize it easily. This code (and probably more) will actually work for any language, provided that we use JavaCC / JJTree to do the other parsers. See 6.2: Future work.

3.3.2.3.4 Profiling

This is the only mechanism that we did not implemented. This possibility could be valuable to add information that is missing from the sources; one good example would be clarifying the nature of associations (like those using Java collections).

3.3.3 Language-neutral code generation

We have the desire to extract from code not only the structure of classifiers, features and relationships, but also the semantics of their method code, in as much detail as possible. The primary needs are:

- **Message Sends**. This is the most important item, essential to enable detection of collaborations.
- **Instantiations**. Some patterns may be interested in lifecycle; e.g. a Factory.
- **Local Variables**. These may be needed, for example, to do static dispatch of messages.
- **Field References**. Adds information about use of the objects' states.

The solution we decided to explore is inspecting the contents of Method's sources and, like a compiler would do, generate *intermediary code* that states the method's actions in a neutral, canonical form. This code should have a number of desirable characteristics:

- **Simple**. Working with ASTs is very hard, because a large number of items are involved in the construction of simple actions like a `new` statement. We prefer a straightforward New object with relevant data such as the type being instantiated (in the sources, even that may be unavailable before name resolution is done). In bytecodes, there are problems such as mapping variables to stack frames.
- **Neutral**. We do not want to be tied to the AST of any particular language.
- **Flexible**. We may want to do some transformation or complex analysis of the code (such as Data Flow / Control Flow Analysis).

The resulting design is a set of code artifact objects. We decided to use some of them during "compilation" but not in the knowledge base emission, because the overhead would be heavy and we wouldn't need them before some DFA/CFA could be done.

The listing (in the CLIPS language) shows the code artifacts used in the detection:

Listing 2: Code templates

```
(deftemplate Code extends Element
        (slot method)
        (slot owner)
)

(deftemplate Send extends Code
        (slot receiver)
        (slot message)
)

(deftemplate Ref extends Code
        (slot object)
        (slot field)
)

(deftemplate Ret extends Code
        (slot value)
)

(deftemplate Assign extends Code
        (slot lhs)
        (slot value)
)

(deftemplate New extends Code
        (slot type)
)

(deftemplate Cast extends Code
        (slot type)
        (slot value)
)

(deftemplate Local extends Code
        (slot type)
        (slot name)
)
```

There are equivalent classes in the "compiler", and additionally, we have Unary (*UnOp value*) and Binary Operations (*BinOp left right*), Literals (*Lit*) and Type References (*TypeRef type*). No support at all exists for control (such as labels and branches).

It is possible to enable debugging output of intermediary code to check what does it look like. Our test target is a method from the *JavaReader* class:

Listing 3: A method to compile

```
/**
 * Creates a Generalization from raw parsed data.
 * @param umlRoot Classifier who's inheriting something.
 * @param parentName Name of <code>umlRoot</code>'s parent class/interface.
 * @return The new generalization.
 */
protected Generalization makeGeneralization (Classifier umlRoot,
        String parentName)
{       Classifier umlSuperklass = getType(parentName, umlRoot.getClass());
        Generalization umlGeneralization = UML.newGeneralization((Name)null,
                (Name)null, umlRoot, umlSuperklass);
        assignID(umlGeneralization);
        umlRoot.getNamespace().addOwnedElement(umlGeneralization);
        return umlGeneralization;
}
```

Listing 4: Compiling

```
CODER: reeng.java.JavaReader: makeGeneralization()
e> Ref(<this>,getType)
 > Ref(<this>,parentName)
 ! Ref(<this>,parentName)
 > Ref(<this>,umlRoot)
 < Ref(<this>,umlRoot)
 > Ref(Ref(<this>,umlRoot),getClass)
 < Ref(Ref(<this>,umlRoot),getClass)
 > Send(Ref(<this>,umlRoot),getClass)
 ! Send(Ref(<this>,umlRoot),getClass)
e< Ref(<this>,getType)
e> Send(<this>,getType)
e! Send(<this>,getType)
e> Local(uml.core.Classifier,umlSuperklass)
 > TypeRef(uml.util.UML)
 < TypeRef(uml.util.UML)
 > Ref(uml.util.UML,newGeneralization)
 > LIT
 < LIT
 > Cast(uml.datatype.Name,LIT)
 ! Cast(uml.datatype.Name,LIT)
 > LIT
 < LIT
 > Cast(uml.datatype.Name,LIT)
 ! Cast(uml.datatype.Name,LIT)
 > Ref(<this>,umlRoot)
 ! Ref(<this>,umlRoot)
 > Ref(<this>,umlSuperklass)
 ! Ref(<this>,umlSuperklass)
 < Ref(uml.util.UML,newGeneralization)
 > Send(uml.util.UML,newGeneralization)
 ! Send(uml.util.UML,newGeneralization)
 > Local(uml.core.relationship.Generalization,umlGeneralization)
 > Ref(<this>,assignID)
 > Ref(<this>,umlGeneralization)
 ! Ref(<this>,umlGeneralization)
 < Ref(<this>,assignID)
 > Send(<this>,assignID)
 > Ref(<this>,umlRoot)
 < Ref(<this>,umlRoot)
 > Ref(Ref(<this>,umlRoot),getNamespace)
 < Ref(Ref(<this>,umlRoot),getNamespace)
 > Send(Ref(<this>,umlRoot),getNamespace)
 < Send(Ref(<this>,umlRoot),getNamespace)
 > Ref(Send(Ref(<this>,umlRoot),getNamespace),addOwnedElement)
 > Ref(<this>,umlGeneralization)
 ! Ref(<this>,umlGeneralization)
 < Ref(Send(Ref(<this>,umlRoot),getNamespace),addOwnedElement)
 > Send(Send(Ref(<this>,umlRoot),getNamespace),addOwnedElement)
 > Ref(<this>,umlGeneralization)
 < Ref(<this>,umlGeneralization)
 > Ret(Ref(<this>,umlGeneralization))
Local(uml.core.Classifier,umlSuperklass)
Local(uml.core.relationship.Generalization,umlGeneralization)
Send(<this>,assignID)
Send(Send(Ref(<this>,umlRoot),getNamespace),addOwnedElement)
Ret(Ref(<this>,umlGeneralization))
ARG: Ref(<this>,parentName)
ARG: Send(Ref(<this>,umlRoot),getClass)
ARG: Send(<this>,getType)
ARG: Cast(uml.datatype.Name,LIT)
ARG: Cast(uml.datatype.Name,LIT)
ARG: Ref(<this>,umlRoot)
ARG: Ref(<this>,umlSuperklass)
ARG: Send(uml.util.UML,newGeneralization)
ARG: Ref(<this>,umlGeneralization)
ARG: Ref(<this>,umlGeneralization)
```

```
e : empty stack
> : push
< : pop
! : retire to arguments
```

The emphasized lines are the code that will be later emitted; all lines before it are a sequence of operations in the code stack, as the parser proceeds gradually sending details about statements in the method. The code inside argument lists is isolated and handled separately; the code inside arguments also becomes input for inference. While this listing shows multiple trees of nested code artifacts, the generated facts will be cross-indexed.

There are a good number of possibilities here that we couldn't explore. In addition to DFA & CFA (completely language-independent), we could do some preprocessing to make the inference more efficient.

3.3.4 Inference engine

This is the final step; it contains, and depends on, the method for elaborating pattern detection, that will be covered in the next chapter.

We based our work on [JESS], the Java Expert System Shell. This is derived from previous tools, OPS5 and CLIPS [CLIPS], and provides backwards compatibility with most of CLIPS in the language, while adding a number of new features, including Java specific support (e.g. reflection of Java objects).

JESS can be used in many ways. Its inference engine can be run as a stand-alone shell where the user gets a prompt to write CLIPS code and interact with the environment, or it can be used as a library of components, having a client Java program to control JESS and interact with it in many ways. We can write part of the application in Java, part in JESS, and communicate freely in both directions. The CLIPS code is able to access Java objects, the rules can match them, the Java code can manipulate rules and facts and every aspect of the inference engine, etc. The easiest or most efficient alternative can be used for any task.

In our implementation, there is a *FactCreatorVisitor* that scans the model, generate facts corresponding to interesting structural data and asserts these facts into JESS; our *InferenceEngine* is actually a wrapper for JESS's. We also read the compiled code stored in Methods and produce facts for the code.

Once this step of fact generation is done, the entire model data is discarded and JESS is executed to activate rules over the facts. These rules are explained in section 4: A Method for Pattern Detection.

3.3.4.1 Mapping UML to knowledge

We decided to not do a straight mapping of the UML metamodel to CLIPS fact templates.

Although CLIPS supports single inheritance of *deftemplates* (the models for facts), there is a semantic gap between objects and facts in a rule-based expert system. The latter are a best fit for the relational model, and not abiding to this reality would adversely impact performance and complexity.

Discussing this issue with the Ernest Friedman-Hill, author of JESS, would confirm that: "*It's very similar to RDBMSs. In fact, there was (and is) a lot of overlap between the people who developed this sort of ES technology and who developed heavy-duty relational database technology. Many folks (David Miranker is a good example) continue to blur the distinction.*" [private e-mail].

Our *deftemplates* for the UML are a product of the following recipe, very equivalent to OO-to-Relational mapping:

1. Start with the structure of the UML metamodel objects; write initial templates with one CLIPS slot per attribute.
2. Remove multiple inheritance by selecting "main" superclass and duplicating slots from secondary superclasses.
3. Add a numerical unique-ID slot to everything. This is the *(slot code)*.
4. Map references to other objects with the numerical IDs.

(We can have fact-references but these are hard to use, because cyclic dependencies in the UML graph prevent us from generating all facts in a single top-down scan, and facts cannot be modified; they can be retracted and replaced by updated facts, but these will have different fact-IDs.)

5. Map collections as reverse pointers; e.g. if *Classifier* contains a list of *Feature*, there will be no information about features in the *Classifier* template, but the *Feature* template will have a back-pointer to its *Classifier*. All such pointers will be *owner* if they don't yet exist in the UML model.
6. Denormalization: Destroy tiny facts that are uninteresting to manipulate as separate entities, by inlining their slots into the body of any other template using them. For example, the template *MultiplicityRange* would have only slots *upper* and *lower*, so we replace any range slots in other templates by two slots named *range.upper* and *range.lower*.
7. Comment out slots or entire templates that will not be useful for pattern detection, as well as the code to generate them.

The resulting definitions are sampled below, including comments that clarify some transformations.

Listing 5: Templates for the UML (partial)

```
; uml.core ---------------------------------------

(deftemplate Element
        (slot code)
)

(deftemplate ModelElement extends Element
;       (slot clientDependencies)
        (slot name) ; (Name.body)
;       (slot namespace)
;       (slot namespace-visibility)    ; (ElementOwnership.visibility)
;       (slot sourceFlows) -- not used
        (slot stereotype) ; (Stereotype.name)
;       (slot stereotypeConstraints) -- not used
;       (slot supplierDependencies)
;       (slot taggedValues)
;       (slot targetFlows) -- not used
)

(deftemplate Constraint extends ModelElement
;       (slot body) ; (BooleanExpression) -- not used
;       (slot constrainedElements) -- alternate
)

(deftemplate Feature extends ModelElement
        (slot owner)
        (slot ownerScope)
        (slot visibility)
)

(deftemplate BehavioralFeature extends Feature
;       (slot parameters)
;       (slot query) -- not supported by reeng
)

(deftemplate Method extends BehavioralFeature
;       (slot body) ; (ProcedureExpression.body) -- not used
        (slot specification)
)

(deftemplate Operation extends BehavioralFeature
        (slot abstract)
;       (slot concurrency)
        (slot leaf)
;       (slot root)
)

(deftemplate StructuralFeature extends Feature
        (slot changeability)
        (slot multiplicity)
        (slot targetScope)
        (slot type)
)

(deftemplate Attribute extends StructuralFeature
        (slot associationEnd)
        (slot initialValue)      ; (Expression.body)
)

(deftemplate GeneralizableElement extends ModelElement
        (slot abstract)
;       (slot generalizations)
        (slot leaf)
;       (slot root)
;       (slot specializations)
)
```

```
(deftemplate Namespace extends ModelElement
;       (slot ownedElements)
)

(deftemplate Package extends Namespace)

(deftemplate Model extends Package)

(deftemplate Classifier extends GeneralizableElement
;       (slot ownedElements)     ; Inherited (Namespace)
;       (slot features)
;       (slot participants)
)

(deftemplate Parameter extends ModelElement
        (slot owner)      ; new! -> BehavioralFeature
;       (slot defaultValue) -- not supported by reeng
        (slot kind)
        (slot type)
)

; Inlined in clients
; ------------------
; (deftemplate ElementOwnership extends Element
;       (slot visibility)
; )

; uml.classifier -----------------------------------
...cut...
; uml.core.relationship ----------------------------
...cut...
; uml.core.dependency ------------------------------
...cut...
; uml.datatype -------------------------------------
...cut...
 (deftemplate Multiplicity
        (slot name)              ; new!
;       (slot range)             ; Inlined below
        (slot range.upper); (MultiplicityRange)
        (slot range.lower); (MultiplicityRange)
)

(assert (Multiplicity (name ZERO_ONE) (range.lower 0) (range.upper 1)))
(assert (Multiplicity (name ONE)      (range.lower 1) (range.upper 1)))
(assert (Multiplicity (name ZERO_N)   (range.lower 0) (range.upper -1)))
(assert (Multiplicity (name ONE_N)    (range.lower 1) (range.upper -1)))

; Inlined in clients
; ------------------
; (deftemplate MultiplicityRange
;       (slot upper)
;       (slot lower)
; )
...cut...
; uml.datatype.expression --------------------------
...cut...
; uml.extension ------------------------------------
...cut...
```

3.4 Conclusion

Tools for parsing, reverse engineering, design language metamodels or expert systems are nothing new. Unfortunately, they often exist either as independent parts that are hard to get to work together or inside monolithic tools where customization and reuse are difficult – e.g. because the reverse engineering requires a customized metamodel of the code, or because handling of method behavior relies on language-specific information.

Our idea was having a framework that is very powerful, flexible, language-neutral and standards-compliant. This is a toolset aimed to research: attacking problems not yet well understood, or maybe even not yet well defined. In this context, the ease of change and experimentation is of utmost importance.

4 A Method for Pattern Detection

All of physics is either impossible or trivial. It is impossible until you understand it, and then it becomes trivial.

– Ernest Rutherford

4.1 Candidate Pattern

Investigation of the subject and experimentation with our tool led to the definition of a method for implementing detection of design patterns, or similar artifacts. We will explain this method with a case study.

The first step is to select a pattern that we desire to detect. We should identify

- If the pattern is possible to be detected;
- If the pattern is worth to be detected;
- If the pattern demands some special strategy to be detected.

4.1.1 Possible

Common wisdom and previous research tells that structural patterns are the easy ones, while others may be harder.

The first thing to clarify is the definition of "structural" here. In effect of [GoF]'s categorization, we could assume a "default" understanding; in their words, "*Structural patterns are concerned with how classes and objects are composed to form larger structures*".

Definition 2: Gamma's structural patterns
Structural Pattern *= A pattern concerned with structure.*

We reviewed previous works, such as [Krämer], making a point that structural patterns are the ones possible to reverse engineer. The term is misleading in this context, because we tend to think, for example, that a Proxy is easy to detect while a Template Method (labeled as Behavioral) could be harder.

We propose a different definition.

Definition 3: Structural patterns for reverse engineering

Structural Pattern *= A pattern that exhibits a standard structure.*

By "standard structure" we mean that the pattern leads naturally to an implementation which structure is very often the same, or has a tractable number of variations.

Additionally, "structure" is usually associated with class diagrams. If a pattern instance follows closely its model (which is a prototype), the pattern is easy to detect. But the issue here is that class diagrams are usually available and we can handle them in a precise way. This property is only exclusive to class diagrams if we depend on tools that only provide class diagrams. If any other information is available, more patterns become possible to detect. In some sense, we are also extending the definition of "structural" as we are going to work with structures (i.e. organized sets) of other artifacts, like message sends, object instantiations, use of particular commands, use of particular libraries / APIs, typing, and so on.

Definition 4: Patterns that are Possible to detect

Every pattern that can be identified by a set of artifacts that we can extract from code is ***Possible to Detect****.*

This covers most patterns, but not all. The extent of the inclusion may depend on the implementation environment. For example, if a pattern's characteristic traits depend on the availability of private methods in the language, this is a problem in a language that doesn't support that feature. The pattern will still be implemented, but the prohibition of invoking a method will be "implemented" informally, e.g. by documentation or naming standards, and the tool may be unable to extract this information if such conventions are not standardized. The pattern would yet be detected by weakened constraints, but that would be useless if this weakening produced a very large number of false hits.

Definition 5: Patterns that are Easy to detect

Every pattern that is Possible the Detect and has a small (tractable) number of variations of those artifacts, is ***Easy to Detect****.*

The exact meaning of *tractable* depends on a number of factors, such as environment constraints (see Definition 8), and the facilities of the detection tools. The tools should have powerful abstraction features to allow us to describe multiple combinations of some "pattern building blocks" without having to list each of them explicitly.

Definition 6: Patterns that are Practical to detect

Every pattern that is Easy to Detect and has a sufficient number of constraints to not produce a large number of misidentifications, is ***Practical to Detect****.*

If a pattern is too simple and thousands of objects will contain the characteristics that identify that pattern by coincidence, the pattern is Easy, but not Practical to Detect, because we should face a hard manual filtering of too much false hits. On the other hand, if we have powerful tools to filter the patterns, we can put more patterns in this category, as the threshold is higher.

4.1.2 Worth

This is a last question that we consider necessary.

A pattern that describes a too general concept and will produce an excessive number of matches is probably not worth detecting. Suppose in some environment (e.g. a particular language) the used of factory Methods is mandatory. Each and every objects should be created by a factory method, defined by the developer or automatically. It is worthless detecting the Factory Method because the detection will simply list every class in the inspected system.

On the other hand, we may *want* to detect obvious / ubiquitous patterns because we want to validate the code – perhaps the programmer *forgot* to write the mandatory factory method for some class! See 6.2: Future work.

Other patterns may be worthless to detect because they are too rare. One example is the Interpreter pattern. Very few programs contain interpreters, and in those using them, they are most likely single occurrences, very obvious to find and very explicit.

We can add a last item on the issue.

Definition 7: Patterns that are Worth to detect

Patterns that are Practical to Detect, and provide information about the code that is neither redundant, nor too rare or easy to find manually, are considered ***Worth to Detect****.*

If a pattern is Practical to Detect but not Worth to Detect, we have no much need for a detection tool, other than as an interesting exercise. This definition is dependent of what use we want to do of the tool: if the objective is validation, we can argue that all Practical cases are also Worth cases.

4.1.3 Strategy

Some patterns will be possible to detect after following a cookbook's recipe. Others may demand special tricks that are usually environment-dependent.

Although we focused on doing a research that is independent from specific idioms, we cannot ignore that being specific about languages, APIs, frameworks, or even particular "informal standards" can facilitate the detection of many patterns. This happens when *in a given environment*, there is a de facto standard way to implement a specific pattern. The cause may range from specific support from the environment (a particular variation of the pattern is so easy or efficient, perhaps produced by tools, and everybody does it that way even if there are other dozens of possible variations) to mandatory rules (the core libraries require a specific implementation).

This usually happens when the environment itself makes serious use of the design patterns. Applications are developed through reuse and probably extension of the system's core features, so they must abide to its rules. One example is the Observer pattern in the Smalltalk system. It is safe to assume that virtually all Smalltalk programs that need this functionality will use what is offered by *Object*; the detection tool will only need to identify the objects fulfilling some of the roles in the pattern.

Other easy example is the Iterator pattern. Languages like Java, Smalltalk and ANSI C++ contain standard collection classes that implement Iterators. In this case, the libraries implement *all* roles of the pattern (the collection classes are the Aggregates) so

there is no real detection work required (unless developers create new collections and iterators, but this is supposed to happen through extension of the libraries, so the detection is still easy). In any event, the detection is probably not Worth in such cases – probably not even for validation, as incorrect implementation are likely to not even compile, or to be too broken to be identified even as a partial pattern. It could still be Worth in other environments.

It is useful to introduce more formally the concept of Environment used here:

Definition 8: Environments for pattern instances

A Pattern is instantiated for a specific programming language, standard "core" libraries / APIs / frameworks, development tools, domain, and both de facto *and* de juris *standard programming practices. This is known as the pattern's* ***Environment****.*

Strategy may depend of several items of the environment.

- If there are *de facto* rules that are not enforced by compilers or libraries or tools, but agreed and largely followed by the programming community, this is good as compiler-enforced rules with respect to detection.

A good number of such rules come from programming paradigms. For example, in the Java language, classes that are closely related are placed in a common package. Even though one can write an entire application with 2,000 classes in a single package, this is not a reasonable scenario. This is a very useful heuristics because most design patterns imply a tight relationship between some roles (although not necessarily, and not probably, all roles). We can at the very least classify hits by likeliness, atributing higher grades to instances where the closely related objects are in the same package.

- The *de juris* rules are not only compiler- or library-imposed standards. Corporate-imposed development guidelines may be just as good.

Examples:

"*All persistence should be done with Oracle's libraries.*" A reverse engineering tool can match specific APIs to know which objects are persistent, and extract events about their lifecycle.

"*All distribution should be done with CORBA, and using the Naming Service is mandatory.*" The tool could detect collaborations between remote objects, instead of being confused by use of stubs and skeletons, or inserting needless indirection in the produced models.

"*The Model-View-Controller mechanism should be used for every GUI.*" If the target environment has standard support for the MVC framework, it is trivial to find any use of MVC. We should match for specific types and messages.

"*The guidelines from the Smalltalk with Style book should be followed.*" We can assume a few sanity rules to be granted in the sources, instead of trying to work around many possible [bad] variations of the code.

Large IT departments are used to produce and publish such guidelines, and they are used (and can afford) to customize tools.

- Domains are a different crosscut of the development community that we may also profit to consider.

We could perhaps focus on a particular domain; for example, "Agent Systems" or "Data Mining Applications" or "E-Commerce Applications". We could then identify possible common traits of these domains, which would help detection. Each particular development community defines, as their field matures, a core set of tools and practices (and beliefs) that we can trust to encounter in their software.

Pattern detection will be implemented by an expert system; we may benefit from interviews with the experts to acquire the knowledge about what are the standard practices (that are interesting to patterns) in the domain.

4.1.4 Picking the candidate

There is a wide range of paths we could follow; but we are going to concentrate in the basics and not attack such particular strategies. The selected case study is the Abstract Factory. It is fair to check if it passes the constraints and heuristics defined so far:

- **Possible**

The Abstract Factory contains factory and product types; both may have an abstract level (but this is not mandatory). If the *ConcreteFactory* is differentiated from the *AbstractFactory* (i.e., the roles are realized by different implementation classes), it should be a subtype of it. If the *ConcreteProduct* is differentiated from the Abstract Product, it should be a subtype of it. The Concrete Factory contains a *createProduct* method that should be accessible from clients (i.e. it should be public). The *createProduct* method contains code that instantiates the *ConcreteProduct* and returns it (so its return type should be the *AbstractProduct*). The *ConcreteProduct*'s creation mechanism should be private to the factory: the client should not be allowed to instantiate it directly, without using the factory – i.e., no public constructors.

The paragraph above refers to class structure (classes, methods, inheritance) and code artifacts (new, return), including some detail like the visibility of methods. All of these are easy to detect, as we developed a tool that obtains them all from sources.

- **Easy**

The recipe above does not pretend to describe all abstract factories with absolute precision. For example, the creation of the product may be done indirectly – using reflective operations that are hard to detect, or maybe through some middleware that contains object lifecycle facilities (a case where domain-specific strategies would help).

We will put our hope in the statement "99% of object creations use the language's `new` statement in a straight manner, so we won't miss many cases". This "reality check" is always important: First to not put *too much* confidence in a tool (and see it as a helper, but not as something definitive to assert critical things); second, to identify possible helper strategies (like supporting metaprogramming or middleware).

If we ignore these odd possibilities, we can be safe about the Abstract Pattern. Its implementation is straightforward in any OOPL – it is not like an Interpreter that depends on big algorithms[3]. The structure and collaborations are simple.

[3] But we could again use environmental heuristics: an Interpreter is very likely to be implemented using standard tools, like lex/yacc for C, JavaCC or ANTLR for Java; these tools typically produce very characteristic code. So we can be able to detect easily even a pattern that contains really complex code!

- **Practical**

The Abstract Factory depends on non-public construction, something that we can reasonably expect to be extremely uncommon in objects that are *not* part of an Abstract Factory[4]. The object would be useless otherwise… *unless* it exists solely to product some exotic side effect of its class initialization, but we can dismiss the possibility as rare and bad practice (and schedule it for a defect detection tool).

This particular element seems to be essential. Without it, we would match as Abstract Factories any set of classes where one class has a public method that creates and returns instances of other class. This is broad enough to include, for example, all *String toString()* methods in Java or *char* operator<<(ostream&)* in C++; all of them would be considered factories of Strings, as they typically create the strings being returned.

- **Worth**

We are targeting the Java platform in our implementation, and Java programs are by no means enforced to use abstract factories. The pattern is also not rare: our experience hints that it is common enough to be worthwhile to search for. (The detection may eventually dismiss our experience, but we should use the available heuristics when they seem reasonable.)

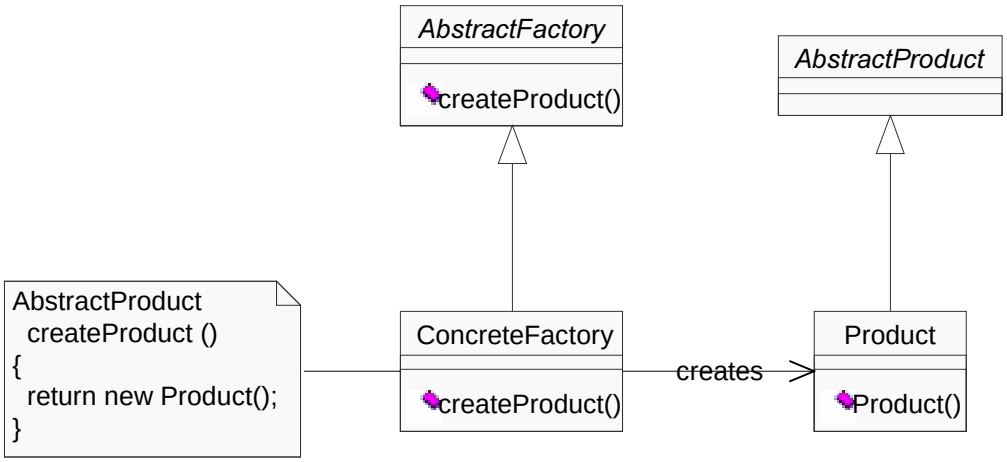
The Abstract Factory is not a very complex and visible thing. We can expect applications to include several instances of this pattern that are not very visible. We can expect this factory to be incorporated as just another role of classes doing many other things – e.g. a *Bank* class that takes care of many tasks, and also is a factory of *Agency*. The Abstract Factory role is "lost" in the namespace of the *Bank*'s many other attributions.

[4] Another possibility would be a Singleton.

4.2 Structural prototype

The second step is sketching a prototype of the pattern to be found. We will now start working with our tool. The patterns themselves are defined by

- The same elements we obtain from target source code through reverse engineering (knowledge about class structure and several code artifacts);
- Rules to constrain these elements, and maybe obtain additional, higher-level knowledge.



This leads to a prototype-like definition, where we start from one instance of the pattern to obtain the knowledge; the rules will be done later.

Listing 6: Prototype for Abstract Factory

```
abstract public class AbstractFactory
{
        abstract public AbstractProduct createProduct ();
}

public class ConcreteFactory extends AbstractFactory
{
        public AbstractProduct createProduct ()
        {
                return new Product();
        }
}

abstract public class AbstractProduct
{
}

public class Product extends AbstractProduct
{
        public Product ()
        {
        }
}
```

The prototype is a straightforward rendering of the pattern model in some language supported by the detection tool. But the prototype is not a perfect model of the pattern. It contains information that we do not need (like names of classes and methods), and it seems to be too rigid (it doesn't seem to support cases where a single class realizes the

Abstract and Concrete roles for either the Factory or the Product, or even all of them). But we will add the necessary flexibility with rules, after careful analysis of the prototype.

The prototype does not point, unfortunately, elements that *cannot* exist in the pattern. In the Abstract Factory, we could have this heuristics: "the *createProduct()* method cannot read any instance variable from the receiver"; this would filter out cases like accessors that must wrap the attributes returned into new objects, because the class doesn't store the attributes in a straightforward manner. We could write anti-prototypes for possible false hits, and have rules that match the *absence* of their features.

The prototype has the advantage of serving as a test case; the detection code will need to recognize the prototype as a valid instance of the pattern.

4.3 Knowledge

Given a prototype for a pattern, the next pass uses the detection tool itself to produce a set of *facts* that describe the pattern. The facts are lists of values that accord to the templates sampled by Listing 5.

We show the entire output of running the tool; the first half of the listing is not interesting because it contains data about basic elements such as the language's primitive types. This "header" is always the same for any execution. Everything we need to check is the second half.

Listing 7: Obtaining facts for the Abstract Factory (1/2)

```
T:\>sr  -preeng.test.AbstractFactory  reeng.test.AbstractFactory.ConcreteFactory
-rvfc -dS:\
SourceReader - (C)1999, Osvaldo Pinali Doederlein
Reading classes...
[1145K #6] s: reeng.test.AbstractFactory.Product
[1245K] Killing classifier proxies...
[1261K] Collecting interfaces into classes...
[1262K] Binding methods to operations...
Time: 2.454s - Heap: 744K used, 3223K of 3968K free.
[745K] Starting Inference Engine...
Facts: 52 in 0.07s - Heap: 1030K used, 2937K of 3968K free.
[962K] Thinking...
Inferencing took 0.01s - Heap: 964K used, 3003K of 3968K free.
Dumping facts...
f-0     (Multiplicity (name ZERO_ONE) (range.upper 1) (range.lower 0))
f-1     (Multiplicity (name ONE) (range.upper 1) (range.lower 1))
f-2     (Multiplicity (name ZERO_N) (range.upper -1) (range.lower 0))
f-3     (Multiplicity (name ONE_N) (range.upper -1) (range.lower 1))
f-4     (Model (code 0) (name Model) (namespace nil) (stereotype nil))
f-5     (Package (code 5) (name <primitives>) (namespace 0) (stereotype nil))
f-6     (Primitive (code 6) (name void) (namespace 5) (stereotype nil)
        (abstract FALSE) (leaf TRUE) (root TRUE))
f-7     (is-a (sub 6) (super 6) (inherits TRUE))
f-8     (Primitive (code 7) (name boolean) (namespace 5) (stereotype nil)
        (abstract FALSE) (leaf TRUE) (root TRUE))
f-9     (is-a (sub 7) (super 7) (inherits TRUE))
f-10    (Primitive (code 8) (name byte) (namespace 5) (stereotype nil)
        (abstract FALSE) (leaf TRUE) (root TRUE))
f-11    (is-a (sub 8) (super 8) (inherits TRUE))
f-12    (Primitive (code 9) (name char) (namespace 5) (stereotype nil)
        (abstract FALSE) (leaf TRUE) (root TRUE))
f-13    (is-a (sub 9) (super 9) (inherits TRUE))
f-14    (Primitive (code 10) (name short) (namespace 5) (stereotype nil)
        (abstract FALSE) (leaf TRUE) (root TRUE))
f-15    (is-a (sub 10) (super 10) (inherits TRUE))
f-16    (Primitive (code 11) (name int) (namespace 5) (stereotype nil)
        (abstract FALSE) (leaf TRUE) (root TRUE))
f-17    (is-a (sub 11) (super 11) (inherits TRUE))
f-18    (Primitive (code 12) (name long) (namespace 5) (stereotype nil)
        (abstract FALSE) (leaf TRUE) (root TRUE))
f-19    (is-a (sub 12) (super 12) (inherits TRUE))
f-20    (Primitive (code 13) (name float) (namespace 5) (stereotype nil)
        (abstract FALSE) (leaf TRUE) (root TRUE))
f-21    (is-a (sub 13) (super 13) (inherits TRUE))
f-22    (Primitive (code 14) (name double) (namespace 5) (stereotype nil)
        (abstract FALSE) (leaf TRUE) (root TRUE))
f-23    (is-a (sub 14) (super 14) (inherits TRUE))
```

Listing 8: Obtaining facts for the Abstract Factory (2/2)

```
f-24    (Package (code 15) (name reeng) (namespace 0) (stereotype nil))
f-25    (Package (code 16) (name reeng.test) (namespace 15) (stereotype nil))
f-26    (Package (code 17) (name reeng.test.AbstractFactory) (namespace 16)
        (stereotype nil))
f-27    (Class (code 18) (name reeng.test.AbstractFactory.ConcreteFactory)
        (namespace 17) (stereotype nil) (abstract FALSE) (leaf FALSE)
        (root FALSE) (active FALSE))
f-28    (Method (code 31) (name createProduct) (namespace 17) (stereotype nil)
        (owner 18) (ownerScope instance) (visibility public)
        (specification 24))
f-29    (New (code 1) (method 31) (owner 0) (type 35))
f-30    (Ret (code 0) (method 31) (owner nil) (value 1))
f-31    (Operation (code 33) (name reeng.test.AbstractFactory.ConcreteFactory)
        (namespace 17) (stereotype create) (owner 18) (ownerScope type)
        (visibility public) (abstract FALSE) (concurrency sequential)
        (leaf FALSE) (root FALSE))
f-32    (Method (code 34) (name reeng.test.AbstractFactory.ConcreteFactory)
```

```
        (namespace 17) (stereotype create) (owner 18) (ownerScope type)
        (visibility public) (specification 33))
f-33    (Send (code 2) (method 34) (owner nil) (receiver <super>)
        (message reeng.test.AbstractFactory.AbstractFactory))
f-34    (is-a (sub 18) (super 18) (inherits TRUE))
f-35    (is-a (sub 18) (super 22) (inherits TRUE))
f-36    (is-a (sub 18) (super 19) (inherits TRUE))
f-37    (Class (code 19) (name reeng.test.AbstractFactory.AbstractFactory)
        (namespace 17) (stereotype nil) (abstract TRUE) (leaf FALSE)
        (root FALSE) (active FALSE))
f-38    (Operation (code 24) (name createProduct) (namespace 17)
        (stereotype nil) (owner 19) (ownerScope instance) (visibility public)
        (abstract TRUE) (concurrency sequential) (leaf FALSE) (root FALSE))
f-39    (Parameter (code 26) (name nil) (namespace 17) (stereotype nil)
        (owner 24) (defaultValue nil) (kind return) (type 25))
f-40    (Operation (code 27) (name reeng.test.AbstractFactory.AbstractFactory)
        (namespace 17) (stereotype create) (owner 19) (ownerScope type)
        (visibility public) (abstract FALSE) (concurrency sequential)
        (leaf FALSE) (root FALSE))
f-41    (Method (code 28) (name reeng.test.AbstractFactory.AbstractFactory)
        (namespace 17) (stereotype create) (owner 19) (ownerScope type)
        (visibility public) (specification 27))
f-42    (Send (code 3) (method 28) (owner nil) (receiver <super>)
        (message java.lang.Object))
f-43    (is-a (sub 19) (super 22) (inherits TRUE))
f-44    (is-a (sub 19) (super 19) (inherits TRUE))
f-45    (Class (code 25) (name reeng.test.AbstractFactory.AbstractProduct)
        (namespace 17) (stereotype nil) (abstract TRUE) (leaf FALSE)
        (root FALSE) (active FALSE))
f-46    (Operation (code 37) (name reeng.test.AbstractFactory.AbstractProduct)
        (namespace 17) (stereotype create) (owner 25) (ownerScope type)
        (visibility public) (abstract FALSE) (concurrency sequential)
        (leaf FALSE) (root FALSE))
f-47    (Method (code 38) (name reeng.test.AbstractFactory.AbstractProduct)
        (namespace 17) (stereotype create) (owner 25) (ownerScope type)
        (visibility public) (specification 37))
f-48    (Send (code 4) (method 38) (owner nil) (receiver <super>)
        (message java.lang.Object))
f-49    (is-a (sub 25) (super 22) (inherits TRUE))
f-50    (is-a (sub 25) (super 25) (inherits TRUE))
f-51    (Class (code 35) (name reeng.test.AbstractFactory.Product)
        (namespace 17) (stereotype nil) (abstract FALSE) (leaf FALSE)
        (root FALSE) (active FALSE))
f-52    (Operation (code 40) (name reeng.test.AbstractFactory.Product)
        (namespace 17) (stereotype create) (owner 35) (ownerScope type)
        (visibility public) (abstract FALSE) (concurrency sequential)
        (leaf FALSE) (root FALSE))
f-53    (Method (code 41) (name reeng.test.AbstractFactory.Product)
        (namespace 17) (stereotype create) (owner 35) (ownerScope type)
        (visibility public) (specification 40))
f-54    (Send (code 5) (method 41) (owner nil) (receiver <super>)
        (message reeng.test.AbstractFactory.AbstractProduct))
f-55    (is-a (sub 35) (super 35) (inherits TRUE))
f-56    (is-a (sub 35) (super 22) (inherits TRUE))
f-57    (is-a (sub 35) (super 25) (inherits TRUE))
f-58    (Package (code 20) (name java) (namespace 0) (stereotype nil))
f-59    (Package (code 21) (name java.lang) (namespace 20) (stereotype nil))
f-60    (Class (code 22) (name java.lang.Object) (namespace 21)
        (stereotype nil) (abstract FALSE) (leaf FALSE) (root TRUE)
        (active FALSE))
f-61    (is-a (sub 22) (super 22) (inherits TRUE))
f-62    (private-instantiable 22)
```

Now, it is necessary to analyze this output. Because we didn't do any easy-to-use user interface, the task is manual and harder than it should be (we could make good use of some tool to present and manipulate this data, like SPOOL [Keller]).

The facts mapping UML objects are easy to understand when one knows the metamodel (and our mapping of it to CLIPS, presented in 3.3.4.1: Mapping UML to knowledge). The awkward part is the references between facts, which are realized by indexes and codes, like keys in databases.

We can see here one additional trait of our mapping: we discard Generalizations and Abstractions in favor of much more useful *is-a* facts to map them. The *is-a* facts are a closure of inheritance and implementation. They have *inherits* set to *FALSE* when they are generated from interface implementation, a not existent case in this example.

Having said that, the first thing to do is filtering the fact listing and removing every detail that is not going to be necessary for the Abstract Factory.

- **Fact IDs and is-a facts.** We do not use the IDs; we use the codes only. There may be several is-a facts for each class, but we should abstract them to the subtyping constraints that exist in the pattern.
- **Information not related to the pattern definition.** These are packages, most stereotypes, and others. For each fact and each slot, we consider if that information is relevant. For example, the slot *leaf = FALSE* in the Abstract Factory seems relevant because a leaf class cannot be extended, while the slot *root = FALSE* is irrelevant because we do not care if the Abstract Factory can have any superclasses or not. And the slot *abstract = TRUE* seems certainly relevant.
- **Redundancies in UML itself.** For example, constructor methods (those with *stereotype = create*) are always class methods (*ownerScope = TYPE*) and they cannot be abstract. They are also always associated to a Method in the same class, so we may be able to ignore one in the matching.
- **More subtle redundancies in the UML that we get to know from the details in the spec.** Methods are forced to have several attributes identical to those from their specification Operation – for example, the visibility should be the same (Well-Formedness Rule [3] for Method in UML 1.3). If we know that we will match both the Operation and the Method, we can avoid matching the same details in the Method (but we will only know this later).
- **Redundancies that we can extract from common sense.** For example, it is absurd having an abstract class that is a leaf. No well-formed model will be produced with such a case. In the Java language, for example, that would require a class to be both *abstract* and *final* – an illegal aberration, so we can safely ignore it. (Even if the language doesn't enforce the rule, it is reasonable to assume it.) We can thus remove the *leaf* slot whenever the class is abstract.
- **Additional types or features in the output**. The *java.lang.Object* class is included because it is a dependency of others, but we are not really interested on it. There are constructors not existent in the prototype sources, for example, for class *AbstractFactory*. Default public constructors are automatically generated to comply with the rules of the language when a class has no constructors; this applies even to abstract classes. There are rules to generate code for default constructors of concrete classes, too.

- **Redundant code**. We will match *createProduct()*'s return parameter to exist and have the *AbstractProduct* type. This means that the Ret code artifact is redundant, unless we want to do some dataflow analysis (we won't).
- **Not used names**. Names are usually irrelevant. The exception is method names (because of a limitation in the tool, we match message sends by name and not by identity with Operations). In this pattern there is no relevant message send, so we can eliminate all names.

We could even implement most of this checklist filters automatically, in our theoretical user-friendly front-end for the tool. And we can also reorder things because the fact listing may be heavily disordered.

All those codes, for the remaining elements, should be substituted by variables (identifiers prefixed by "*?*").

In any case, the result is the following listing, a cleaned-up listing containing only facts really relevant for the pattern:

Listing 9: Filtered facts for the Abstract Factory

```
(Class (code ?absFactory) (abstract TRUE))
(Operation (code ?createOp) (owner ?absFactory) (ownerScope instance)
        (visibility public) (abstract TRUE))
(Parameter (owner ?createOp) (kind return) (type ?product))
(Class (code ?factory) (abstract FALSE))
(is-a (sub ?factory) (super ?absFactory) (inherits TRUE))
(Method (code ?create) (owner ?factory) (ownerScope instance)
        (visibility public) (specification ?createOp))
(New (method ?create) (type ?product))
(Class (code ?absProduct) (abstract TRUE))
(Class (code ?product) (abstract FALSE))
(is-a (sub ?factory) (super ?absFactory) (inherits TRUE))
(Operation (stereotype create) (owner ?product) (visibility public))
```

We could already write a rule to detect the pattern: this sequence of facts with variables is the complete body of the rule.

Unfortunately, at this point, the rule would only detect patterns that are exactly equal to the prototype; this is highly undesired and makes the next step necessary.

4.4 Generalization

This is the least automatic and most important step of the method. We should think about the pattern, its possible variations, and how to adapt that list of facts so they are compatible with a more general Abstract Factory.

The way to do it is enumerating every major variation that this pattern could have, and deduce the consequences in the facts.

- A single class may realize the *AbstractFactory* and the *ConcreteFactory* roles. Because this class should be concrete, there is no *AbstractFactory*. *(Likewise for the Product.)*

Solution: We should not require that the *AbstractFactory* is abstract, or not a leaf. We should not require that the *ConcreteFactory* is not a root. We should remove these requirements, when existing. They are redundant anyway, because we can use the knowledge that *(is-a (sub ConcreteFactory) (super AbstractFactory))*. This case matches both cases, as

$$\forall T : isa(T,T)$$

- We could have additional classes in the inheritance path between *AbstractFactory* and *ConcreteFactory*. *(Likewise for the Product.)*

Solution: Already solved by the previous case, because

$$\forall T_1, T_2, T_3 : isa(T_1,T_2) \wedge isa(T_2,T_3) \rightarrow isa(T_1,T_3)$$

These two cases are exactly why we abandoned the Generalizations and Abstractions in favor of the *is-a* facts. They will be very useful in most patterns; virtually everywhere that we encounter abstract vs. concrete pairs of roles.

- The *createProduct()* method may be a class method; in this case, the factory class(es) could even be not instantiable!

Solution: Remove the constraint *ownerScope = instance*.

- The abstract roles could be realized by interfaces instead of classes, for example in the Java language.

Solution: Instead of requiring the *Class* template, we will require the *Classifier* template in the facts that match all abstract roles. (The CLIPS language supports inheritance and this is used by our fact templates to map the UML metamodel.)

We should also remove the matching *inherits = TRUE* in the *is-a* facts, because we are going to support interface implementation as well.

The new listing contains a much better set of knowledge that we could already use to detect many abstract factories:

Listing 10: Generalized facts for the Abstract Factory

```
(Classifier (code ?absFactory))
(Operation (code ?createOp) (owner ?absFactory) (ownerScope instance)
        (visibility public))
(Parameter (owner ?createOp) (kind return) (type ?product))
(Class (code ?factory) (abstract FALSE))
(is-a (sub ?factory) (super ?absFactory) (inherits TRUE))
(Method (code ?create) (owner ?factory) (ownerScope instance)
        (visibility public) (specification ?createOp))
(New (method ?create) (type ?product))
(Classifier (code ?absProduct))
(Class (code ?product) (abstract FALSE))
(is-a (sub ?factory) (super ?absFactory) (inherits TRUE))
(Operation (stereotype create) (owner ?product) (visibility public))
```

It is worth mention that so far, we have used the “Rapid Application Development” approach: It may be very slow compared to a competent person writing code.

All three first steps are only provided as a formalization of the method, and as a tutorial. In our experience, after some practice they become irrelevant and we can write at least the Filtered step immediately from the prototype sources. We could even have a tool to automatically generate the rules. But we would still need manual effort from the Generalization step forwards.

4.5 Optimization

Making inference engines perform efficiently sometimes seems a black art. Small variations in the rules can easily mean speed and memory differences of one order of magnitude, so we *need* to optimize the rule as much as possible.

The fundamental principles to optimize rule-based expert systems, in our context, are:

4.5.1 Efficient representation of the knowledge.

This is already done in the design of the UML fact templates (mostly, as we opted to keep things as resembling to the metamodel as possible). See 3.3.4.1: Mapping UML to knowledge.

4.5.2 Avoiding redundant work.

This part was mostly addressed in the previous steps, but it is not complete. We can find other places to cut. For example, if class A has any method with a new statement referring to class B, then the class B should have a public constructor; we can remove the match for this constructor.

One beautiful optimization is using polymorphism to remove subtype checks. If class A defines an operation O, and class B has a method M which specification is O, then B is a subtype of A and we can remove the *is-a* constraints!

$$\forall A, B, O, M : owner(O) = A \wedge owner(M) = B \wedge specification(M) = O \rightarrow isa(B, A)$$

4.5.3 Inference engine-specific tuning

Placing first the patterns that generate the least number of partial matches [JESS].

Basically, we need to reorder the patterns to minimize the explosion of joins between big sets of temporary results. This is similar to optimizing complex SQL queries. Sometimes it is a guess or trial-and-error procedure.

We can now present the complete rule, in its final form that also includes a header and the consequent to assert what was found, a template to define the pattern itself, and many comments about the optimizations.

Listing 11: Generalized, complete rule for the Abstract Factory

```
(deftemplate AbstractFactory
        (slot abstractFactory)
        (slot concreteFactory)
        (slot abstractProduct)
        (slot product)
        (slot createOperation)
)

(defrule find-abstract-factory
        ; Matches [Abstract]Factory
        ; implied: (Classifier (leaf FALSE) (code ?absFactory))
        (Operation (code ?createOp) (name ?createOpName)
                (visibility public) (owner ?absFactory))
        (Parameter (owner ?createOp) (kind return) (type ?absProduct))
        ; Matches the ConcreteFactory
        (Method (code ?create) (specification ?createOp) (owner ?factory))
        ; Matches instantiation of product by the factory
        (New (method ?create) (type ?product))
        ; implied: (Class (code ?factory))
        ; Matches [Abstract]Product and Product
        ; implied: (Classifier (code ?absProduct))
        ; implied by New: (Class (abstract FALSE) (code ?product))
        (is-a (sub ?product) (super ?absProduct))
        ; implied by New: (Method (owner ?product)
        ;  (stereotype create) (visibility public))
        ; Desambiguate: FactoryMethod is a special case of AbstractFactory
        (not (FactoryMethod (abstractCreator ?absFactory) (creator ?factory)
                (factoryMethod ?createOpName) (abstractProduct ?absProduct)
                (product ?product)))
=>
        (assert (AbstractFactory (abstractFactory ?absFactory)
                (concreteFactory ?factory) (createOperation ?createOpName)
                (abstractProduct ?absProduct) (product ?product)))
)
```

Additional work could happen depending on the pattern. We may want to factor out common knowledge that is useful for several patterns. One example is the *private-instantiable* rule, useful for patterns that require that some class cannot expose public constructors (we have only Singleton, but we have two rules for different variants of Singleton).

Listing 6. Factoring Rules

```
(defrule find-private-instantiable
        ; Candidate class
        (Class (abstract FALSE) (code ?cls))
        ; Cannot have any non-private constructor
        (not (Operation (owner ?cls) (stereotype create) (visibility public)))
        (not (Operation (owner ?cls) (stereotype create)
                (visibility protected)))
=>
        (assert (private-instantiable ?cls))
)
```

This kind of factoring is irrelevant for performance, because the JESS inference engine does it automatically. The motivation is rather making explicit any interesting “pattern building blocks” that we may find.

4.6 Validation

The rule is complete and can be executed. It should then pass by good tests to assert its efficiency:

- Detecting the original prototype;
- Detecting variants of the prototype considered in the Generalization step;
- Detecting other sources, which were *not* analyzed as input for the design of the pattern.

We should fix the rules if we find unexpected problems. For example, multiple patterns may be related. In the presented Listing 11, there is a clause to ignore this pattern if the same set of classes match for the Factory Method pattern, because the Abstract Factory is a special case of that (structurally) and only after implementing both patterns, we noticed the redundancy in the results.

A detailed analysis of a number of test cases is provided next, to evaluate the efficacy of our methods, and of its implementation.

4.7 Conclusion

While section 3 presented the bulk of the development work backing this research, that amounts only to infrastructure upon which we could execute our main objectives, and write our main implementation.

This implementation, the rules that match design patterns, is of minimal size and very high-level. It will detect patterns implemented in any object-oriented programming language. It will benefit from excellent, detailed knowledge about the code, down to traceable, complex method code.

We present a rationale to write the rules that match a given design pattern. Although the final code usually appears simple, there are a fair number of issues influencing its exact form. These issues range from obvious to obscure, but we consider valuable to specify and justify them all, in an attempt to formalize the whole process. This formalization would also help to build additional tools, to generate automatically (most of) the rules for new patterns from sample code, or even for other tools (like visualization).

The steps listed here, in their many details, do not pretend to be complete. Additional issues could surface in specific patterns. We hope though to have most generic parts of the method here. In section 5: Results, while analyzing our attempts to detect several patterns, several additional cases surface but nothing that demanded major revisions in the method.

5 Results

If an experiment requires statistical analysis to establish a result, then one should do a better experiment.

– *Ernest Rutherford*

The most exciting phrase to hear in science, the one that heralds new discoveries, is not "Eureka!" (I found it!) but "That's funny"...

– *Isaac Asimov*

5.1 Testing Principles

We have performed a number of tests in "real world" program sources, to validate the tool and the principles behind it.

Testing pattern detection tools / methods should ideally meet two criteria:

1. **Quantitative**: A large variety of software should have their sources inspected. The very nature of design patterns includes that they can usually be implemented in more than one way; but we can expect to have consistent style in all code from a single application, or even in all code from a single company where software practices are homogenized.

We risk finding that the tool is very good because we analyzed code from somebody who implements the patterns the way we want them, or finding that the tool is bad because we analyzed some odd software where even Singletons are implemented in a very uncommon way.

2. **Qualitative**: We should know beforehand, or inspect manually, all the source code that is tested, in order to identify misses and false hits precisely. This is unfortunately very hard to do, in effect of the large size of tests that satisfy the previous criterion.

The ideal criterion to analyze the tool's precision is not the ideal criterion to look at its usability. We are interested to know how good is the tool for somebody who is not willing to read all inspected sources, or even to manually verify all findings of the tool.

We can summarize the testing scenarios:

- **Blind Test**. The user does not know the inspected sources, and will not read the sources. The tool's output should be sufficient for any conclusions. The user will at best read an extremely small amount of the sources to manually verify a very few findings of the tool.
- **Black-box Test**. The user won't read the sources as well, but he or she does know the inspected code, as a client of that code. The user knows the code either from its documentation or from reusing it (libraries, frameworks).
- **White-box test**. The user will read the entire sources, or already knows them very well, perhaps because the user wrote the sources.

In the first two cases, it is reasonably safe to trust that classes and methods have names that convey they meaning, so we can just analyze the output from the detection tool and consider if each instance makes sense. The approach has proven good for criticizing the hits, but it doesn't cover misses.

5.2 Test Cases

The following code bases were used as test cases for the detection:

- **The `std` test: JDK core**

This is the set of all core classes[5] reachable from *java.lang.Object*, using the JDK 1.2.2's source code. A total of 341 classes are scanned[6].

The test code is not an application, but it is a very good case of framework analysis – we would like to know which design patterns are used by an OOPL's core libraries, because these patterns will likely affect the design of application code (for example, template methods).

This is a black-box test. We did not do any extensive reading of JDK's source code to interpret the results; we trusted in our knowledge of the libraries (i.e. what anybody who programs in Java is supposed to know), and inspected the sources sparingly, to verify some hypotheses.

- **The `self` test: The pattern detection tool itself**

This is where we test our own sources (total 328 classes scanned, but this includes shallow-scanned classes from the JDK). The test case has the advantage of using well-known sources, so validating the analysis is more precise.

This is a white-box test, because we know these sources very well.

- **The `argo` test: A CASE tool**

We used an open source CASE tool, Argo UML 0.7 [Argo]. The size of the test case (771 classes including those imported from the JDK), is near to the maximum we could afford to process in our test machine (heap use is ~100Mb[7]), and the application is fairly complex and complete.

Argo contains another implementation of the UML metamodel which is completely independent from ours, and that would provide an interesting case to see style differences. It also includes several interesting features like extensive GUI, code critics, or XML processing, so it should be a very good "real world application" doing a good diversity of work.

This is a blind test. We did not use Argo's code before this test, and we have only read a small fraction of one of its subsystems (the UML metamodel's backbone).

---

[5] We excluded from the deep scan the `sun.*` classes.

[6] This includes deep-scanned and shallow-scanned classes; the latter are part of exclusions but they are needed as dependencies, so we create the classifier but do not fill it with features, code, and its own dependencies except superclasses/interfaces. The same applies to other class counts.

[7] Most of this memory is due to the Rete network [Forgy] built by JESS to optimize inference; results of all intermediary inferences are maintained to speed up inferences. The process is similar to materialized views created by databases, but in this case the "indexes" are all in memory. We could solve the problem by running the tool multiple times, with only one pattern rule loaded each time. But it is quite possible that our rules are yet far from being as efficient as possible.

5.2.1 Interpreting the Results

Some patterns produce a relatively large number of hits, because they occur for each combination of the elements that form the pattern. For example, the Abstract Factory include these hits in the self test case:

Table 1: Multiple hits per pattern

	Abs. Factory	Conc. Factory	AProd	CProd	Create Operation
5	ClassifierImpl	ClassifierImpl	Set	HashSet	allOperations
6	ClassifierImpl	ClassifierImpl	Set	HashSet	allMethods
7	ClassifierImpl	ClassifierImpl	Set	HashSet	allAttributes
8	ClassifierImpl	ClassifierImpl	Set	HashSet	associations
9	ClassifierImpl	ClassifierImpl	Set	HashSet	oppositeAssociationEnds
10	ClassifierImpl	ClassifierImpl	Set	HashSet	specifications

There is actually only one factory class (*ClassifierImpl*), and only one product (*HashSet*), but there are multiple create operations. Other patterns, such as Template Method, can generate an even larger "explosion" of hits.

Another issue to consider is that patterns are not and end in themselves; we want patterns as a means to understand programs, and we'll see what interesting things we can learn about programs from the patterns found on them.

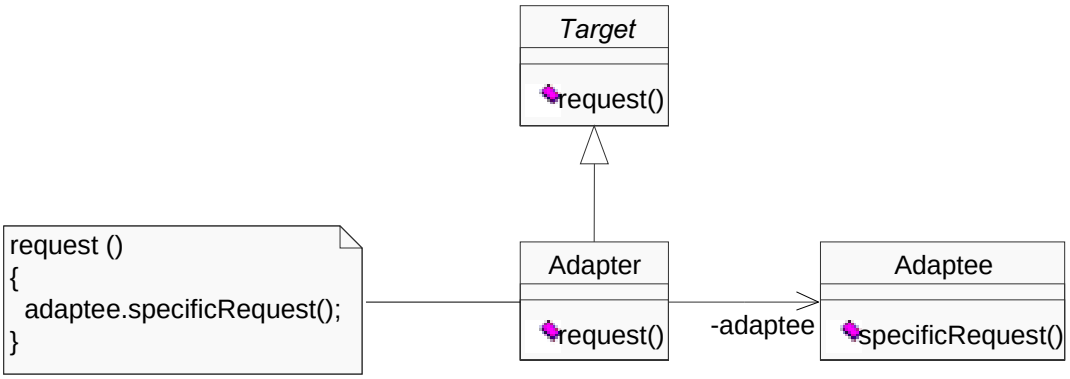
Every test will be discussed in great detail in the paragraphs labeled by the test case codes, and it may be necessary to consult the data listings in Appendix: Output from test cases.

The notation $X \rightarrow Y$ will be often used to mention pattern hits, where both X and Y may be an element of the pattern (e.g. *MyClass* or *myMethod()*) or a composition of elements (*MyClass.myMethod()*), and the arrow stands for a message. *ClassA* $\rightarrow$ *ClassB* means that some method of *ClassA* invokes some method of *ClassB*, without being specific (we may have multiple hits, or the exact methods may be irrelevant).

5.3 Structural Patterns

5.3.1 Adapter

Pattern abstract: The Adapter pattern is intended to convert an interface to another. This is useful, for example, as a tool to manage change. If the interface of the *Adaptee* changes, instead of updating every *Client* that uses it, we can just create (or change) an *Adapter* that points to the *Adaptee* and forwards a *request* coming from the client to a *specificRequest* implemented by the *Adaptee*, and defined by a *Target* type. Even when there's no change, the Adapter can bind together objects that prefer to work with different interfaces: For example, some *Adaptee* can expose a *setWeight(kilograms)* but a Client who prefers working with pounds would need to do conversions manually for every call. The solution is having an *Adapter* to do this conversion when relaying the messages and return values.



Results:

- `Std` (49 hits): There are some good hits, like *PermissionCollection.add()* → *Hashtable.put()*, but most hits seem to have no intention of Adapters. Several methods that rely on aggregate objects to do part of their work look like adapters. This happens, for example, in *StringTokenizer.nextToken()* → *String.substring()* because the *nextToken()* operation returns a substring of the string being parsed, which is part of the tokenizer's state.
- `Self` (0 hits): The software doesn't use adapters, so the tool is correct.
- `Argo` (88 hits): Again, we find a small number of good adapters: *Diagram.setName()* → *DiagramInfo.updateName()* is clear. *Editor.figs()* → *LayerManager.elements()* seems to be another good case. But again, a large number of false hits are reported.

We decided to inspect some bad hits and we found situations like this:

Listing 12: Unusual adapter

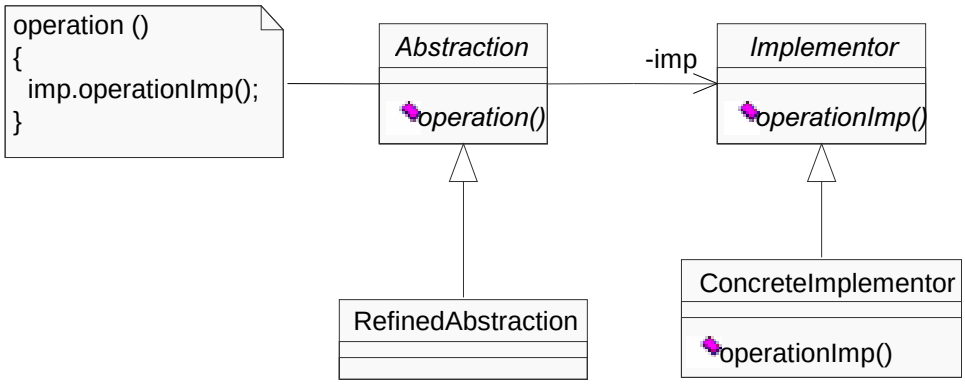
```
class Editor
{
...
/** Get the renderer object that decides how to display nodes */
  public GraphNodeRenderer getGraphNodeRenderer() {
    Layer active = _layerManager.getActiveLayer();
    if (active instanceof LayerPerspective)
      return ((LayerPerspective)active).getGraphNodeRenderer();
    else return null;
  }
...
}
```

The code is considered an adapter, where *Editor.getGraphNodeRenderer()* → *LayerManager.getActiveLayer()*. Unfortunately, this message send is an intermediary step, and the code seems more like a proxy.

Conclusion: The Adapter detection doesn't seem to miss Adapters, but its precision is unsatisfactory, there are too many false hits. Control and data flow analysis would solve this problem. We could also assume that all good adapters do a simple forwarding, i.e. all code in the *request* is a message send to the *specificRequest*. Unfortunately the tradeoff of strictness here would exist in missed patterns. Even with the predominance of false hits, the total number of hits is still small for large programs, the detection is still useful but it demands manual validation.

5.3.2 Bridge

Pattern abstract: The Bridge pattern is intended to decouple an *Abstraction* from its *ConcreteImplementor* so that the two can vary independently. The *Abstraction* will contain one *Implementor*, and several *ConcreteImplementors* may subclass it. The interfaces defined by the *Abstraction* and the *Implementor* will usually be similar, but this is not a condition.



Results:

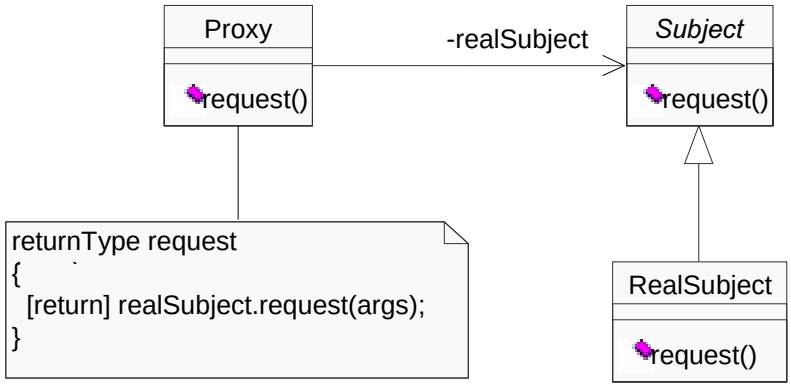
- `Std` (147 hits): Almost all hits (134) are in the streaming classes, e.g. *PrintWriter.write()* → *Writer.write()*. These reveal the layered architecture of the streams. A few hits appear in exception and reflection classes, all for *printStackTrace()*, and others in *Date* → *Calendar*.
- `Self` (4 hits): The small number of bridges here is accidental, and all would be avoided if the rules required identical names for *operation* and *operationImpl*.

- `Argo` (217 hits): Bridges are dominated (170 hits) by an abstraction called *FigEdge* having many concrete implementors (*FigState*, *FigText*, *FigLine*, *FigUseCase* and others), almost all of them extending the implementor *Fig*. The remaining hits don't look to be bridges, they rather seem like communication with aggregates – e.g. *TabDiagram.modeChange()* → *ToolBar.unpressAllButtons()*.

Conclusion: It was hard to find a good balance of strictness, because a relaxed Bridge becomes too simple. We have some false hits due to not enforcing the abstraction's *operation()* to look like the implementor's *operationImpl()*, although this happens in practice for most good bridges. This change could be made easily, but it would probably incur in some misses; unlike a Proxy, these methods are not enforced to have the same exact signature.

5.3.3 Proxy

Pattern abstract: The Proxy pattern isolates the client of an object from the object with an additional level of indirection, but in a transparent manner; the client is not aware that it is sending messages to the *Proxy* object, which implements the same interface of the pointed *Subject*. The *Proxy* stored a pointer to the *Subject*, so this can be changed easily; if multiple clients are linked to that *Subject* through the same *Proxy*, and the Subject should be replaced, it is easier to change one pointer than many.



Results:

- `Std` (78 hits): Again, a large number of hits in stream classes. All findings (66) show something like *ObjectInputStream* → *DataInputStream* where the message is always a low-level I/O operation, such as `readInt()`. More hits for formatting classes (*DateFormat* being a proxy for *Calendar*), and some cases in the security package, *ProtectionDomain* is proxy for multiple kinds of *PermissionCollection* (the same message in each case).
- `Self` (55 hits): Mapping of multiple inheritance shows 51 hits (e.g. *ClassifierImpl* → *NamespacePrivate*); all the *…Private* classes are delegates containing implementation that will be inherited as non-primary implementation bases. The other four hits are all for *Multiplicity* → *MultiplicityRange*. These cases are all *almost* good proxies. On the other hand, we didn't detect our *real* hierarchy of classifier proxies, because they use opaque references to the subjects and casts (we have a general *ObjectProxy* and many type-safe subclasses).

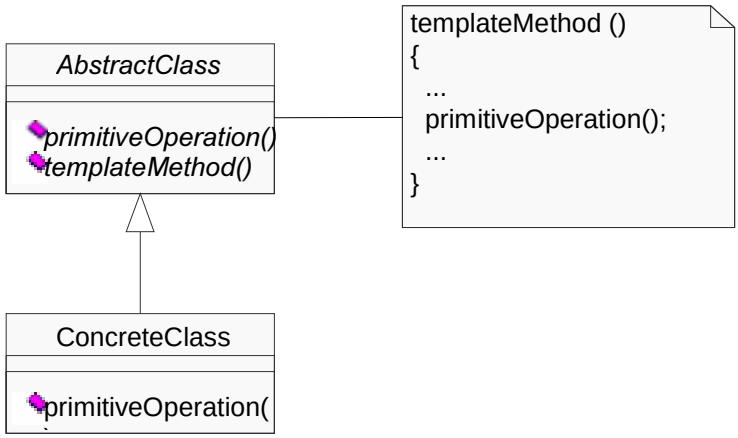
- `Argo` (127 hits): Similarly to the Adapter, the hits look very interesting but they are not exactly Proxies in intention. For example. *Selection.dispose()* → *Fig.dispose()*. The source shows that a Selection is created over one Figure, and works like a proxy for it; but the selection does additional things. It is actually a decorator.

Conclusion: The proxies detected may be accidental rather than something intended. The *Writer* and *Reader* objects in `std` are full of purpose, though; their very meaning is being proxies to the classes pointed as their subjects. The rules we used accept "partial proxies", which cover only part of the subject's interface – this may be desired or not, and in fact it would probably be a good idea to have both options in the detection. False hits could be eliminated; see 5.6: Pattern families.

5.4 Behavioral Patterns

5.4.1 Template Method

Pattern abstract: Template Methods are a powerful way to create reusable, extensible frameworks. An *AbstractClass* will contain a *templateMethod*, implementing some behavior which details may be missing or be desirable to customize. A given detail is handled by an abstract *primitiveOperation* behavior, called by the *templateMethod*. A derived *ConcreteClass* will implement the *primitiveOperation*. This works like a contract where the client of the *AbstractClass* is required to fill in the blanks, and the payoff is flexibility to customize the behavior of the template methods without having to override them. This is not a good idea if the method is complex and the changes to be done are "inside" it – overriding works better when one wants to extend, or totally replace, the existing implementation. Ultimately, the Template Method combats the cases where overriding does not work well (i.e. requiring copy-and-paste of code).



Results:

- `Std` (193 hits): The first flow of hits shows again the stream classes; fundamental classes like *InputStream* have Template Methods like *read(byte[])* calling abstract, Primitive Operations such as *read()* which should be implemented by all sorts of specific streams, as *FileInputStream*. (The full signatures don't appear in the output, so all *read* operations seem equal.) There is a similar case for the *Number* class and it's many primitive wrapper children, like *Integer*. Next, many hits spreading from *Object*: *toString()*, *hashCode()*. The collection classes also produce a fair number of primitive operations: *iterator()*, *size()*, *add()* and others.
- `Self` (25 hits): A few Template Methods, all of them good. It's worth notice that template methods are not used a lot in this software, because we have different mechanisms in place (like Visitors) and the whole UML package is planned for black-box reuse. We have a few very important template methods laid out by *GenericReader*, and they are missed because we provide default implementations for the primitive operations, so they are not abstract like the rules demand.

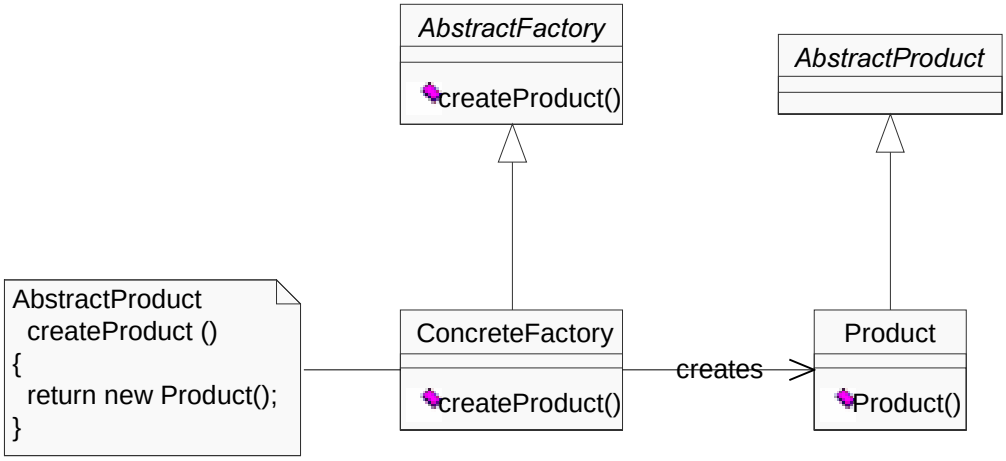
- `Argo` (381 hits): The very large number of Template Methods contain many interesting cases – event handling (*paint()*), command processing (*paintAtHead()* → *doIt(), actionPerformed()* → *translate()*), what appears to be part of constraint-based layout calculation (*connectionPoint ()* → *getGravityPoints()*), forward engineering (*GeneratorJava.generate()* → *generateOperation()*) and others.

Conclusion: The detection has no false hits, and it shows important parts of design. Argo seems to rely heavily on template methods and we learn a lot about it reading the tool's output. In every case, there are a very small number of Template Methods and Primitive Operations, and each generates a large number of hits; that is consistent with our expectation about use of this pattern. We miss some hits when default operations are provided, but this is probably not very good practice (in our own software, we realize that we could refactor the default implementations to an additional class).

5.5 Creational Patterns

5.5.1 Abstract Factory

Pattern abstract: The Abstract Factory pattern allows isolating clients of some *AbstractProcuct* from their implementation as a *ConcreteProduct* (or multiple of these). The problem here is that instantiation would require knowledge of the *ConcreteProduct*'s type, and hardwiring use of such types. The *AbstractFactory* defines an alternative creation protocol, the *createOperation*, implemented by one or more *ConcreteFactory*: the only object that needs to know the *ConcreteProduct*.



Results:

- `Std` (130 hits): "Good" factories (like many methods in the Collections class) are intermixed with cases that are factories accidentally. This happens a lot in immutable objects, like Java's String: something like *concat()* or *toLowerCase()* appear as factory methods, because new Strings are produced instead of changing the receiver. Methods for conversions (lots of *toString()*, *valueOf()* and others) also qualify. The oddest finding was some exceptions being considered products, in methods that return Object (like *Vector.get()*); this happens because we have done no DFA and we don't detect that the exception object being created is not the same being returned. This could even be addressed without any DFA, but with simple additional rules, if we had supported control statements like `throw`.
- `Self` (61 hits): Roughly, half of the findings are for the factory methods in our big UML factory, e.g. *UML.newClass()*. The other half shows UML's additional operations that have to create the collections they return (because these are calculated on the fly), like *ClassifierImpl.allMethods()*. There are only four other cases; only one is a false hit, due to non-existing flow analysis.
- `Argo` (192 hits): The program seems to have cases similar to our code; there's a large number of *get…()* methods that appear as factories. The *make…()* methods are better "on purpose" factories.

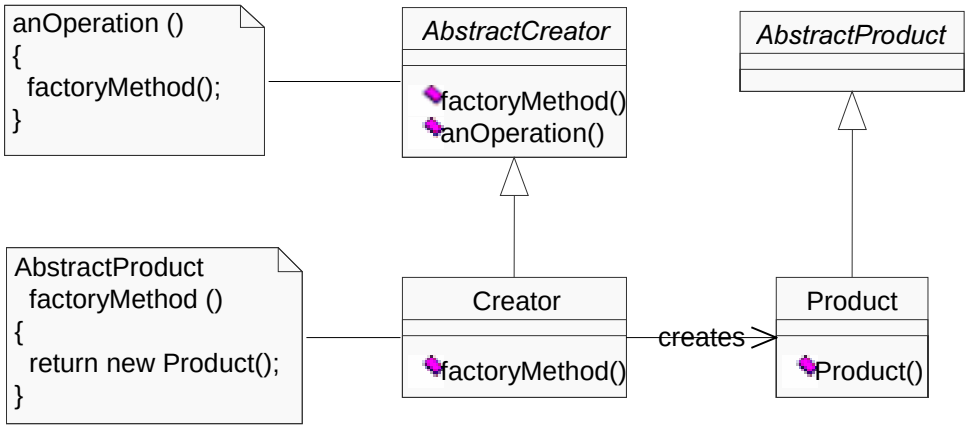
Conclusion: The detection is good, but factoring objects is something very common and could have some additional heuristics to know that producing some objects is the purpose of a method, and not a means to do something else. For example, an objects

that has attributes *x* and *y* and a method *getPosition() { return new Point(x,y); }* is not a real factory, it's only an accessor. It could be a good idea to not allow references to the receiver's fields.

Once again, we could have better detection by supporting additional patterns, e.g. Immutable Object, and considering them in the Abstract Factory's rules.

5.5.2 Factory Method

Pattern abstract: This is a fusion of the Abstract Factory and Template Method patterns. Like the former, the Factory Method is mostly useful in frameworks to make a type parametrizable. If the framework itself includes clients of the *AbstractProduct* but this class may (or should) be subclassed by the programmer, then the programmer also needs to tell the framework to use the new *Product* created. By also deriving a new *Creator* from the framework's *AbstractCreator*, the programmer is able to configure this behavior.



Results:

- `Std` (6 hits): Four good factory methods in collection classes and all are due to template methods for creating iterators to the collections. Two accidental ones in formatting classes; their semantics is a superset of returning a new *Product*.
- `Self` (0 hits): Correct, there is no use of factory methods here. (This is actually an oversight in our software and we plan to change it.)
- `Argo` (12 hits): Every hit is good, and they all have factory methods which names begin by *make* or *create*. The *Fig* class is the most popular abstract product, apparently having as concrete products all the low-level figure objects (*FigLine*, *FigText*, etc.) but not the high-level figures (such as *FigClass*).

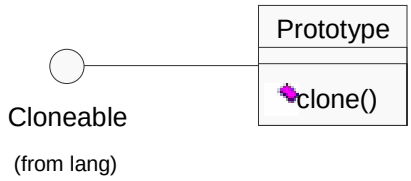
Conclusion: The detection seems precise; the few false hits can easily be removed by requiring that no other code exists in the method but the create and return operations. We usually relax this rule (in other patterns too) because the method may contain some "noise" bookkeeping (e.g. logging or assert operations, or reading a database to obtain data needed for the main operation).

5.5.3 Prototype

Pattern abstract: The Prototype defines a mechanism to create an instance using another instance as a template. The *Prototype* defines a *clone* operation, which is implemented by every *ConcretePrototype*. The client does not need to know the *ConcretePrototype*'s class, and they don't need to provide any parameters for initialization because *clone* will initialize the new instance with a copy of the receiver. The pattern can also be used to make easier the creation of many instances of some class with a complex initialization, when all instances are initialized to the same state.

The Prototype pattern was exceedingly easy to do because it is one instance of environment-aided pattern (see Definition 8). The Java programming language defines a standard mechanism for prototypes: The method *java.lang.Object.clone()* has a standard implementation that does a field-by-field copy of objects, and must be overridden (it is protected). Any class that realizes the *Prototype* role must implement the interface *java.lang.Cloneable*. Once some class implements *Cloneable* (or any other interface that extends it), all of its subclasses are forced to be cloneable too. Overriding the *clone()* method is not required.

We can thus write a rule that matches Prototype pattern instances with 100% of accuracy: all true instances will be found and no false hits will happen. This rule will be Java-specific.



Different strategies would be necessary for other languages. For example, in C++ we can attribute the Prototype qualifier to classes that have public *copy constructors* – including default copy constructors generated by the compiler in the absence of any. The number of hits would be probably very large, because in C++ the programmer has to explicitly disable cloning when it is not wanted (by writing a *private* copy constructor that prevents the compiler from generating a public one).

Results:

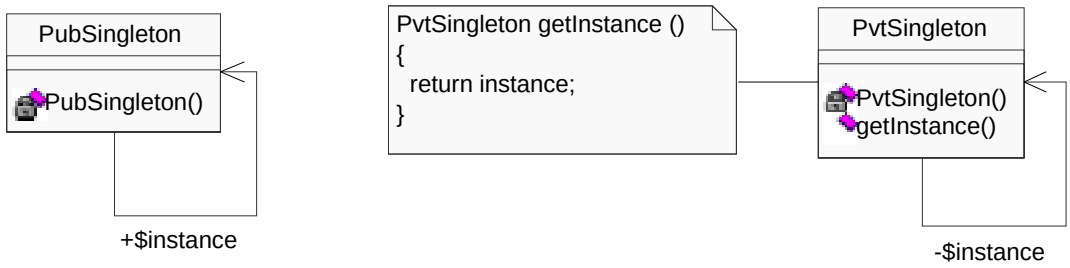
- `Std` (30 hits): We were surprised with the very small number of objects in the JDK are prototypes: only collections, and some date and formatting classes. The rationale is perhaps that applications usually extend these objects and they don't want to inherit the responsibility of being safely cloneable. But we ignore why *final* classes like *String* are not cloneable.
- `Self` (5 hits): All of the few hits are due to imported objets; our implementation has yet to address concerns like cloning, serialization or thread-safety (these are all in the wish list).
- `Argo` (151 hits): A lot of objects are prototypes, including fifty "Action" objects (e.g. *ActionPrint*), many "command" objects that seem related (e.g. *CmdPrint*) and other 50+ of "Figure" objects (low level ones like *FigRect*, and high level ones like *FigClass*).

Conclusion: The precision is perfect because the environment helps, and we could find important information about the design of all test cases. On the other hand, when the environment helps that much, we don't really need a pattern detection tool: any good class browser would suffice. This pattern would be classified, according to our Definition 7, as not worth to detect.

5.5.4 Singleton

Pattern abstract: The Singleton insures that only one instance of the *Singleton* class exists. This definition is too rigid if applied literally, because there is a good number of cases where we may want a multiple, but fixed number, of instances. A good example would be a *Boolean* class with two instances *TRUE* and *FALSE*. The important semantics of the Singleton is all there, if the client cannot instantiate new booleans, and the provided instances are immutable. We understand the Singleton as a contract about the lifecycle of its instances, where we consider good to relax the multiplicity constraint from "one" to "fixed number".

Two variants of the Singleton are considered in our rules: a "public" variant where the variable holding the instance is directly accessible (but read-only), and a "private" variant using accessors.



Results:

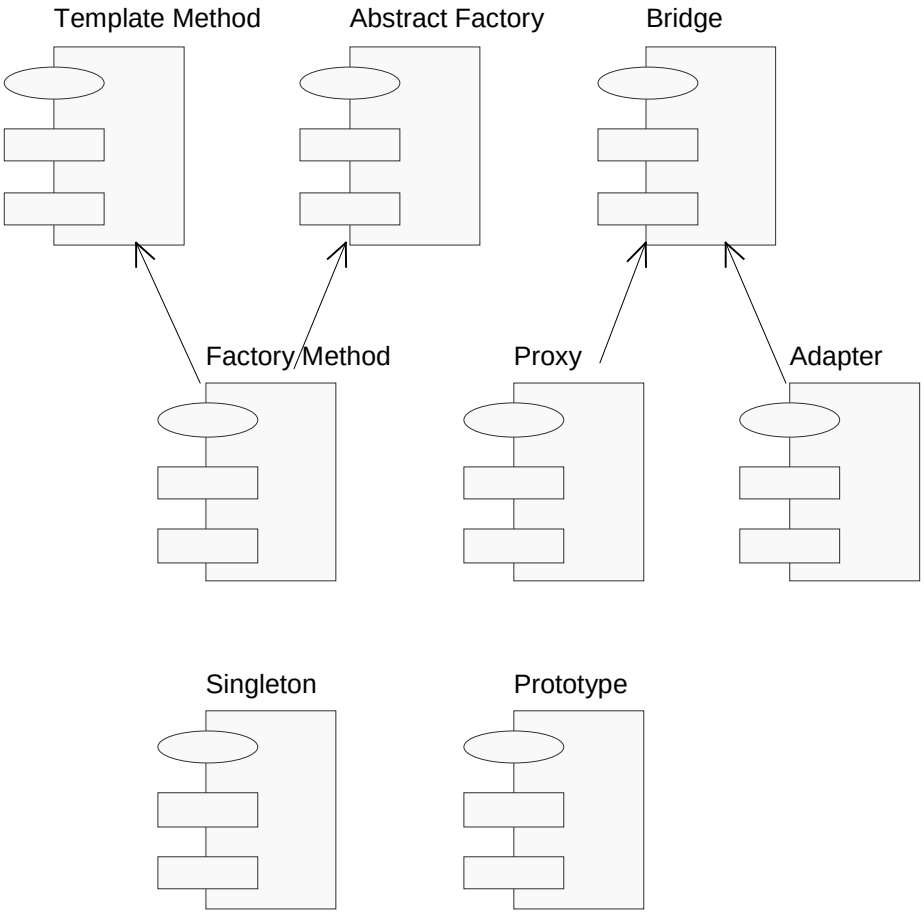
- `Std` (69 hits): Almost all hits go to class *java.lang.Character$UnicodeBlock*, which stores a large number of Unicode tables as singletons. We don't require Singletons to be the unique instances of their classes; we rather search for objects playing the role of Singleton. The 66 objects in this case are the 66 unique instances of their class; the client cannot create new *UnicodeBlock*s, and the existing instances cannot mutate. The important aspect is then the lifecycle of the class's extent. There are other singletons: the JVM's *Runtime* and the local host's *InetAddress* (appears in two matches). Both are extremely interesting cases of Singleton. Notice that *InetAddress.getLocalHost()* always returns the singleton, but *InetAddress.getByName()* can either return it or create (factory) a different address, depending on a parameter.
- `Self` (0 hits): No Singleton found. There is actually no singleton in the program, but we have some situations that could be interesting for further analysis. First, a few classes are just packages of utility methods (all *static*) and we could consider such classes Singletons. Second, some objects are not designed to be Singletons but they are instantiated only once in the program; for example *Model*. This is a side effect of generic, reusable code. If the UML implementation were tied to the rest of our software, we could be tempted to make *Model* (and *UMLContext*) hardwired Singletons.

- `Argo` (1 hit): The only Singleton found has the gratifying name of *SINGLETON* and it exists in the class *uci.uml.ui.EOElement*. This class is a Functor, or function object (a trick to encapsulate first-class methods in objects, made popular in C++ and apparently adopted by Argo's authors). Unfortunately, several other things named *SINGLETON* exist in Argo's sources, but they don't show up in the detection. We inspected one case and it didn't contain any constructor, which means that it contains the default, compiler-generated constructor which is public and disqualifies the singleton. We then scanned all sources and there was no other creation of the offending class, so we suppose the missing private constructor is an error.

Conclusion: The detection seems to work very well, but we could enhance it to find violations.

5.6 Pattern families

We could identify similarities between the structure of multiple patterns. This actually surfaced as a problem: several patterns would have a high correlation (large number of common hits) in preliminary testing.



The solution for the problem, once identified, is using *disambiguating rules* that favor the most specific pattern. Example:

Listing 13: Disambiguating rules

```
(defrule find-bridge
... matches the bridge ...
        ; Desambiguate
        (not (Adapter (request ?operationName)
                (specificRequest ?operationImpName)))
        (not (Proxy (proxy ?abstraction) (subject ?implementor)))
=>
        (assert (Bridge (abstraction ?abstraction)
                (implementor ?implementor)
                (concreteImplementor ?concImplementor)
                (operation ?operationName) (operationImp ?operationImpName)))
)
```

In the example, we consider a Bridge valid if the same elements won't also match for Proxy or Adapter, its more specific "sub-patterns". Similarly, there are rules in both Template Method and Abstract Factory to reject any cases that also match for Factory Method, who "inherits" both.

These rules helped us filter out a large number of false hits. Even when the specific pattern is still a false hit, it is "less false" than before, and easier to manually filter – because it is more close to the real meaning of the code, and because the number of hits to check is smaller.

In the Proxy detection, we found hits that appear to be actually Decorators. Because we did not implement the Decorator detection, we did not filter it from Proxy in the same way: a Decorator is (structurally) a Proxy that adds or changes some behavior, instead of simply relaying messages to the subject. A temporary solution would be inserting in the Proxy's rules all matches to reject elements of Decorator, but this would be a big confusion and would require similar work to implement the Decorator. We expect to have better precision in the *existing* patterns if we implement a larger number of new, related patterns, because we will incrementally make existing ones more precise by rejecting these cases.

One possible good side effect of this may be optimization: we could rewrite the rules for the most specific patterns to be extensions from their base patterns. For example, the Proxy rule would start selecting all Bridges as candidates, and would filter the hits satisfying constraints from the proxy. The problem is that we may have *some* Proxies who are not Bridges: this relationship is accidental, and it can even change as we enhance any of the rules to consider additional special cases or filter additional false hits. Added work would also be necessary to retract the produced Bridges and avoid them to be generated again by the inference engine. In the end, it is probably better to let the Rete network do automatically merge common rules, instead of trying to be smarter than it and producing unreadable and unmaintainable rules.

5.7 Conclusion

The detection of patterns examined in our test cases contains a wide spectrum of findings. Most patterns are very satisfying; in some cases we identify problems that we know how to solve; in a few cases we identify problems that may require some new approach. We still lack significant data about misses, but if the `self` test case is any representative, the tool seems very good in not missing pattern instances. Whether the results are correct or not, we could learn interesting facts about the process and even about the applications analyzed. Summarizing,

Pattern	Results (in respect to presence of false hits)
Adapter	Poor; Many false hits, but easy to fix (relaxed rule)
Bridge	Good; False hits include Composites (not implemented so we don't disambiguate); other cases fixable with DFA/CFA
Proxy	Good; Decorators included (not implemented so we don't disambiguate)
Template Method	Excellent
Abstract Factory	Very good; we should exclude Immutable Objects, and some accessors (requires CFA)
Factory Method	Excellent
Prototype	Perfect (but Java-specific)
Singleton	Excellent

All pattern detections seem OK in respect to inclusion of correct instances. We noticed a few misses of special cases not considered by the rules in Proxy. A more thorough test would be needed to assert inclusion quality, anyway. The amount of manual work to filter the findings is very small; we would benefit from sophisticated visualization tools (e.g. in Adapter) but our raw listings are usually good enough. They only require a very quick examination, usually under one minute for an entire test case / pattern to have a general view of the software and identify the probable good and bad hits.

The better part of the tests is actually realizing the amount of understanding that we can get easily from programs (without inspecting every hit in detail). Even if the number of hits is large, the sorting and the redundancy of patterns will greatly help to draw generalizations.

The fact that all examined software takes seriously the job of naming classes and methods, and organizing them into packages, is also very important. When we see an entry like (in Template Method) *AbstractCollection.addAll()* → *ArrayList.add()*, the semantics of these classes and methods is clear.

The Adapter has many false hits because it is very relaxed, and other rules may miss hits because they are very strict. This is a trial and error, incremental process to "tune" the rules until they detect enough good patterns, while yet not detecting too much bad instances to be annoying. We could certainly tune the Adapter to not report any false hit, but we would likely miss some good ones.

The ability to match method code proved crucial for precision, and it could be even better with use of data and control flow analysis (there is zero use of either).

In the debate about need of static types, we can report some findings:

- Static types of Java helped us to do the basic reverse engineering, but that is all. Type inference would be needed for languages without static types.
- Because of unfinished parts of our code generator, we couldn't put pointers to the operations in message sends (e.g. *(Send (receiver <this>) (operation 457))*). The solution was using names, as in *(Send (receiver <this>) (message doThis))*. We were worried about mistakes when multiple independent operations have the same names by accident (we don't even use full signatures, like *void doThis(int,String)*) but this turned out to be irrelevant. The matches of message sends are usually one or two inside a much bigger rule with many other constraints, and the other constraints filter out almost all the spurious hits.
- Even with static types, we have some need for inference in the reverse engineering step; namely, in collections (we don't know the element types because Java lacks generics) and in the precise reverse engineering of associations.

It seems that static typing is a good help in the fundamental reverse engineering, but less relevant in the higher-level detection task.

There are a number of places where we would greatly benefit from data and control flow analysis. This is something planned for the framework; all prerequisites are in place but time constraints prevented its realization.

The evaluation of the Java core libraries is probably ineffective if we want to look at it as a framework. Certain patterns, like Template Method and Factory Method, will only appear when the framework and an application are analyzed as a whole, because some roles are implemented only in the Application level. The framework may hardwire things when talking to itself, or the interesting objects may be unreachable from our scanning from *java.lang.Object*. In our `self` and `argo` test cases, the dependencies in the JDK were skipped for reasons of efficiency (we would lack memory to handle all details of a big application like Argo/UML, plus all Java core classes it uses). This complete analysis would depend on improvement in the tools, and would probably reveal a new set of interesting cross-layer patterns.

6 Conclusion

An education isn't how much you have committed to memory, or even how much you know. It's being able to differentiate between what you do know and what you don't.

– Anatole France

6.1 Summary

Design Patterns have well known advantages. Lutz Prechelt and Barbara Unger [Prechelt] propose several "claims" as targets for investigation: Developers learn better design skills fast; Using patterns improves designer (and maybe programmer) productivity; Both novices and experienced developers improve their designs; Communication among designers, programmers and maintainers improve. Testing these claims is not easy, but the authors report positive results in their experiments to confirm the claims about better communication.

Code understanding is not restricted to maintenance tasks. Our use of JDK's library source in test cases was quite revealing; it shows a wealth of information that is critical to the design of Java's all-important core libraries, and not explicitly available anywhere in Sun's official documentation (although we would find it in books). The information that is available is scattered, disaggregated. Programmers learning the Java platform are supposed to read the `javadoc` pages for multiple classes and methods, and deduce that they form, for example, a Template Method, which gives information on the semantics and reusability of the libraries. Detecting patterns is a potential tool for general system understanding.

We do an extensive effort to attack the problem of automatic detection of design patterns in program code. We contribute a set of tools that we consider necessary for the job; a general methodology driving the design and use of these tools, and providing the rationale of automatic detection; and a number of experiments to validate and explore the possibilities of the former items.

We have yet to find the silver bullet of architectural reverse engineering: completely automatic production of high-level design information for any given program code. This research reports incremental improvements over previous works in the field. We contribute a unique combination of state of the art reverse engineering, metamodeling, and method code analysis; genericity, openness, and integration; and a theoretical analysis of the entire task.

6.2 Future work

There are many points in our work to deserve additional effort. Some items in this wish list were investigated by others with successful results, while a few surface from our experience, usually when analyzing the reasons for unsatisfactory performance of a given detection.

1. **Persistent knowledge base for pattern knowledge**. Other pattern tools use this for forward engineering, or simply to store pattern definitions and their findings. We would rather like to add some machine learning to the expert system, so the detection could improve when manual validation adds exceptions and special cases, and manual detection could become less necessary with time.
2. **Complete the UML metamodel**. This is not useful for the pattern detection task (we already implement the subset of UML required for that), but our metamodel implementation is an independent project with other possible uses.
3. **Detection of "quasi" patterns, or antipatterns**. This is useful for refactoring or code critic tools. One possibility could be the use of an existing fuzzy logic extension to CLIPS; other is simply considering selected special (wrong) cases.
4. **Better reverse engineering**: We want to detect Aggregations, map Associations better, implement Permissions, perfect support for Java and add support for other languages, support analysis of method bytecodes. There is plenty room to improve.
5. **Integration with other tools**. We are having some dialog with the authors of Argo/UML and we hope to have some integration of features, as reverse engineering is in Argo's wish list and CASE tool integration is in ours.
6. **Visualization**. We also want custom visualization support for the detection results.

7 Bibliography

Read in order to live.

– *Gustave Flaubert*

[Argo] Jason Robbins, David Redmiles, David Hilbert and others. The Argo/UML CASE Tool. Available in the Internet at http://www.ics.uci.edu/pub/arch/uml/

[Borne] Isabelle Borne, Nicolas Revault. Comparaison d'outils de mise en œuvre de design patterns. Revue l'Objet, éditions Hermes, Volume 5, numéro 2 (to be published), 1999

[Bosch] J. Bosch. Design patterns & frameworks: On the issue of language support. Proceedings LSDF'97.

[Bowman] Ivan T. Bowman, Michael W. Godfrey, Richard C. Holt. Extracting source models from Java programs: Parse, Disassemble, or Profile? Submitted to the 1999 ACM SIGPLAN Workshop on Program Analysis for Software Tools and Engineering, Toulouse, France, September 6, 1999.

[Brand] M.G.J. van der Brand, P. Klint, C. Verhoef. Reverse engineering and system renovation: An annotated bibliography. Available in the Internet at http://adam.wins.uva.nl/~x/reeng/REanno.html

[Brooks] Frederick P. Brooks, Jr. The Mythical Man-Month: Essays on Software Engineering. Addison-Wesley, 1995 (anniversary edition).

[CFParse] Matt Greenwood. CFParse, a set of tools for class file editing. Available in the Internet at http://www.alphaworks.ibm.com/tech/cfparse/

[Chikofsky] E.J. Chikofsky and J.H. Cross. Reverse engineering and design recovery: A taxonomy. IEEE Software, 7(1):13-17, 1990.

[CLIPS] Robert Savely et al. A tool for building expert systems. Available in the Internet at http://www.ghgcorp.com/clips/CLIPS.html

[Florijn] Gert Florijn, Marco Meijers, Pieter van Winsen. Tool support for object-oriented patterns. In Mehmet Aksit, editor, 11th European Conference on Object-Oriented Programming (ECOOP), LNCS 1241, pages 472-495, Jyväskylä, Finland, June 1997. Springer Verlag.

[Forgy] Charles Forgy. The Rete Algorithm: A Fast Algorithm for the Many Pattern / Many Object Pattern Matching Problem. PhD thesis. Carnegie-Mellon University, 1979.

[Frédéric] Bernet Frédéric, Ceberio Martine, Heulot Olivier. Documenter la conception et l'architecture à l'aide de patterns: Comment détecter des patterns.

[GoF] Erich Gamma, Richard Helf, Ralph Johnson, John Vlissides. Design Patterns : Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[JavaCC] Sun Microsystems. Java Compiler Compiler, the Java Parser Generator. Available in the Internet at http://www.suntest.com/JavaCC/

[JESS] Ernest Friedman-Hill. The Java Expert System Shell. Available in the Internet at http://herzberg.ca.sandia.gov/jess/

[JLS] James Gosling, Bill Joy, Guy L., Jr. Steele. *The Java Language Specification*. Addison-Wesley, 1996. Also available in the Internet at http://java.sun.com/docs/books/jls/

[JVMSpec] Tim Lindholm, Frank Yellin. *The Java™ Virtual Machine specification, Second edition*. Addison-Wesley, 1999. Also available in the Internet at http://java.sun.com/docs/books/vmspec/

[Kazman] Rick Kazman, Steven G. Woods, S. Jeromy Carrière. Requirements for integrating software architecture and reengineering models: CORUM II. Proceedings of WCRE 98, (Honolulu, HI), October 1998, 154-163.

[Keller] Rudolf Keller, Reinhard Schauer, Sébastien Robitaille, Patrick Pagé. Pattern-based reverse-engineering of design components. In Proceedings of the 21st International Conference on Software Engineering, pages 226-235, Los Angeles, USA, May 1999. IEEE CS press.

[Krämer] Christian Krämer, Lutz Prechelt. Design recovery by automated search of structural design patterns in object-oriented software. Proc. Working Conf. on Reverse Engineering, IEEE CS press, Monterey, November 1996.

[Lieberherr] Karl Lieberherr and Ian Holland, Assuring good style for object-oriented programs. IEEE Software, September 1989, pages 38-48

[Meyer] Bertrand Meyer. Object-Oriented Software Construction. Prentice-Hall, 1997.

[Odenthal] Georg Odenthal, Klaus Quimberly-Cirkel. *Using patterns for design and documentation*. ???

[Prechelt] Lutz Prechelt, Barbara Unger. A series of controlled experiments on design patterns: Methodology and results. Proc. Softwaretechnik '98 GI Conference (Softwaretechnik-Trends 18(3)), pp. 53-60, Paderborn, September 7-9, 1998.

[Shull] Forrest Shull, Walcélio L. Melo, Victor R. Basili. An inductive method for discovering design patterns from object-oriented software systems. Technical Report CS-TR-3597, University of Maryland, Computer Science Dept., 1996.

[Systä] Tarja Systä. Dynamic reverse engineering of Java software. ECOOP'99 Workshop on Reverse Engineering.

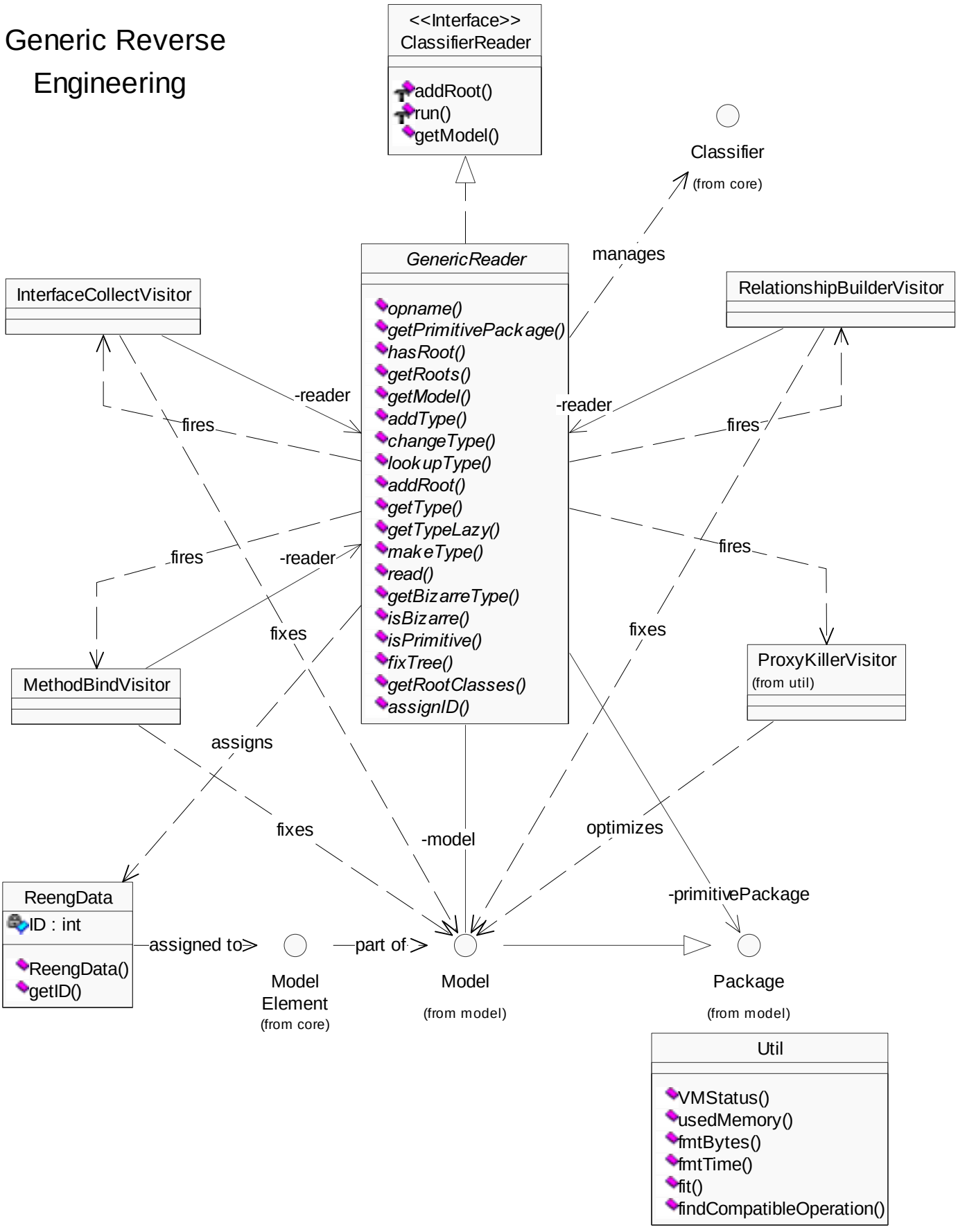
[UML] Object Management Group. OMG Unified Modeling Language specification. Version 1.3, June 1999. Available in the Internet at http://uml.systemhouse.mci.com/artifacts.htm

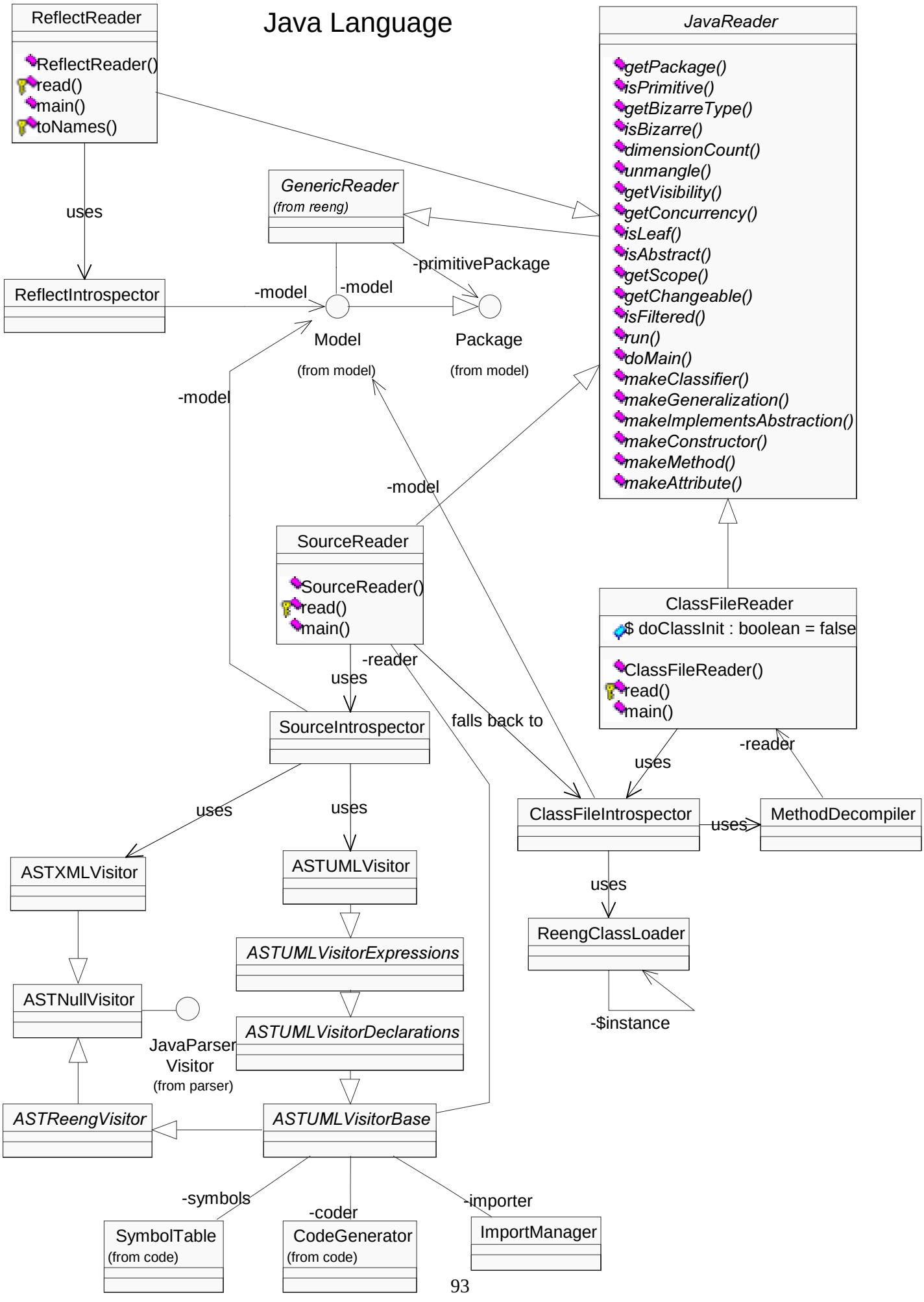
[XML] World Wide Web Consortium. Extensibe Markup Language. Available in the Internet at http://www.w3.org/XML/

8 Appendix: Models

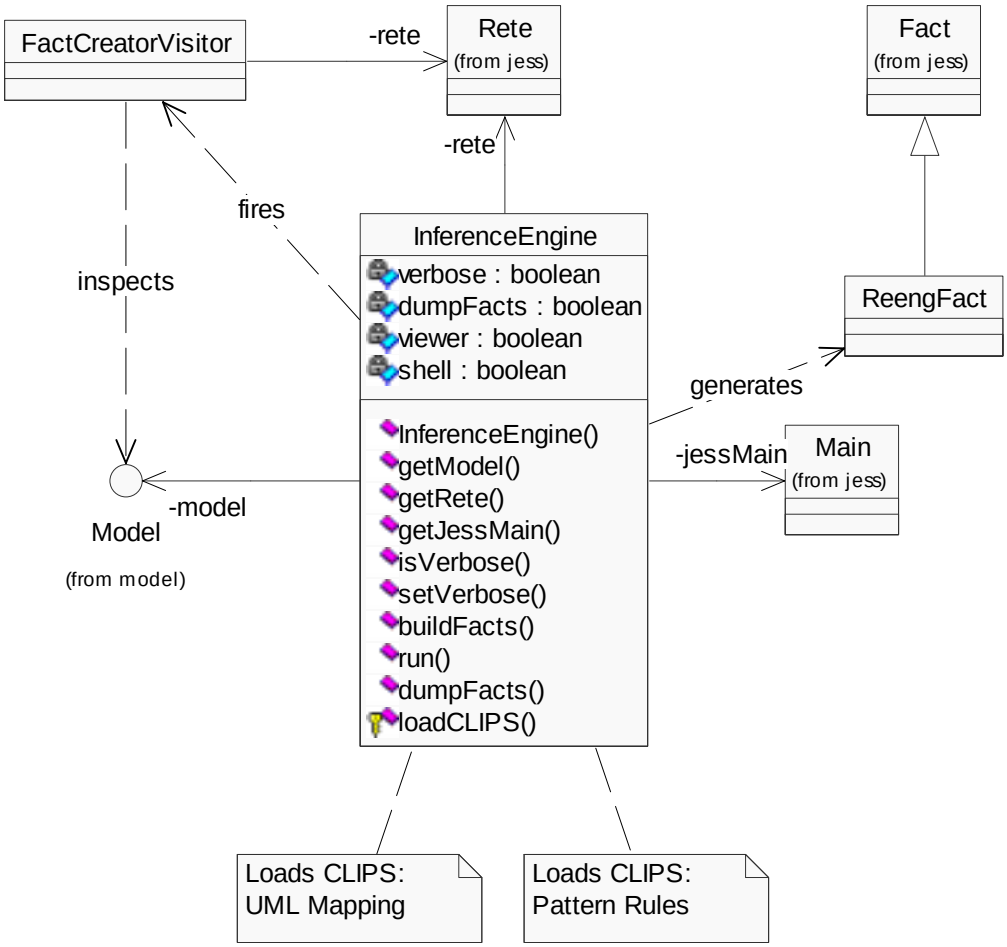
Included here are UML models for the entire implementation of the software developed for this research. The first diagrams model the reverse engineering framework, the latter are our interpretation of the UML 1.3.

Generic Reverse Engineering

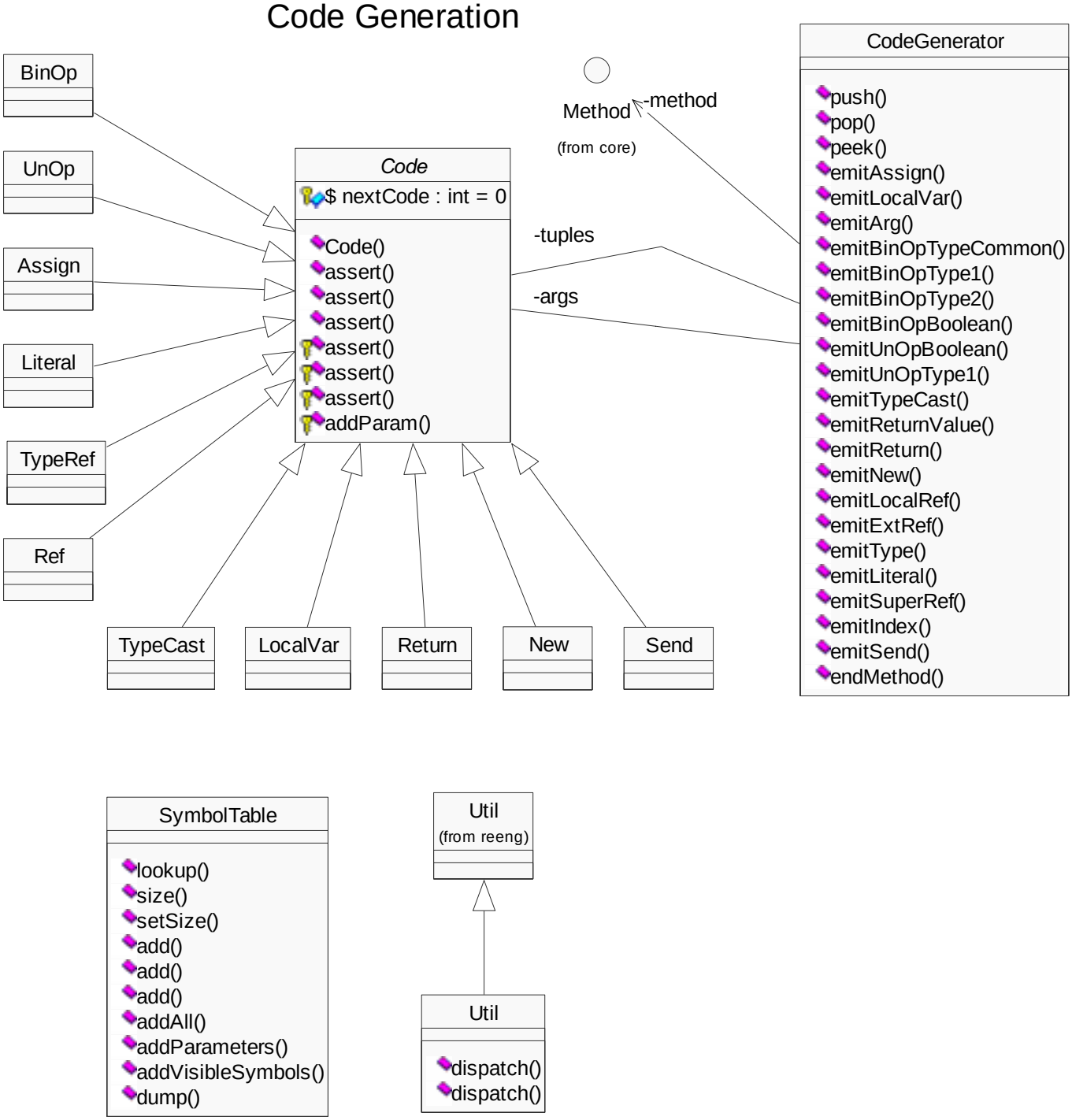


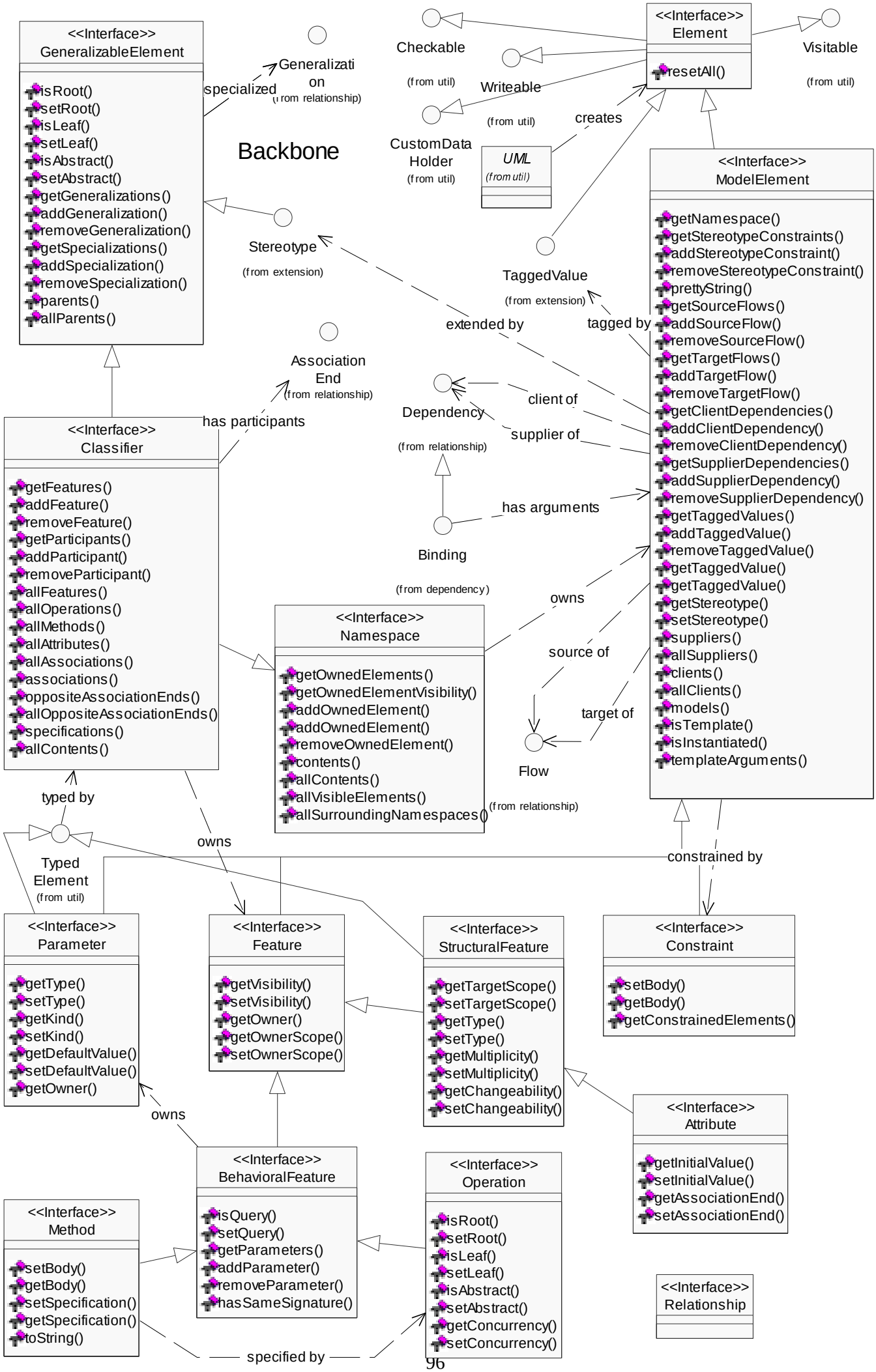

Java Language
ReflectReader
ReflectReader()
read()
main()
toNames()
uses
ReflectIntrospector
GenericReader
(from reeng)
-primitivePackage
-model
-model
Model
(from model)
Package
(from model)
-model
-model
JavaReader
getPackage()
isPrimitive()
getBizarreType()
isBizarre()
dimensionCount()
unmangle()
getVisibility()
getConcurrency()
isLeaf()
isAbstract()
getScope()
getChangeable()
isFiltered()
run()
doMain()
makeClassifier()
makeGeneralization()
makeImplementsAbstraction()
makeConstructor()
makeMethod()
makeAttribute()
SourceReader
SourceReader()
read()
main()
-reader
uses
SourceIntrospector
falls back to
ClassFileReader
$ doClassInit : boolean = false
ClassFileReader()
read()
main()
uses
-reader
uses
uses
ClassFileIntrospector
uses
MethodDecompiler
uses
ASTXMLVisitor
ASTUMLVisitor
ReengClassLoader
ASTUMLVisitorExpressions
ASTNullVisitor
ASTUMLVisitorDeclarations
-$instance
JavaParser
Visitor
(from parser)
ASTReengVisitor
ASTUMLVisitorBase
-symbols
-coder
-importer
SymbolTable
(from code)
CodeGenerator
(from code)
ImportManager

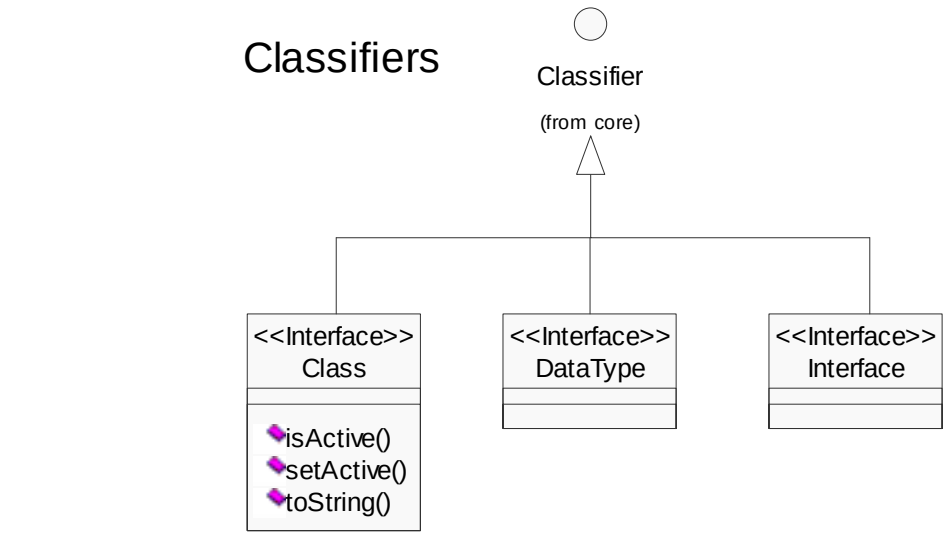
Inference Engine

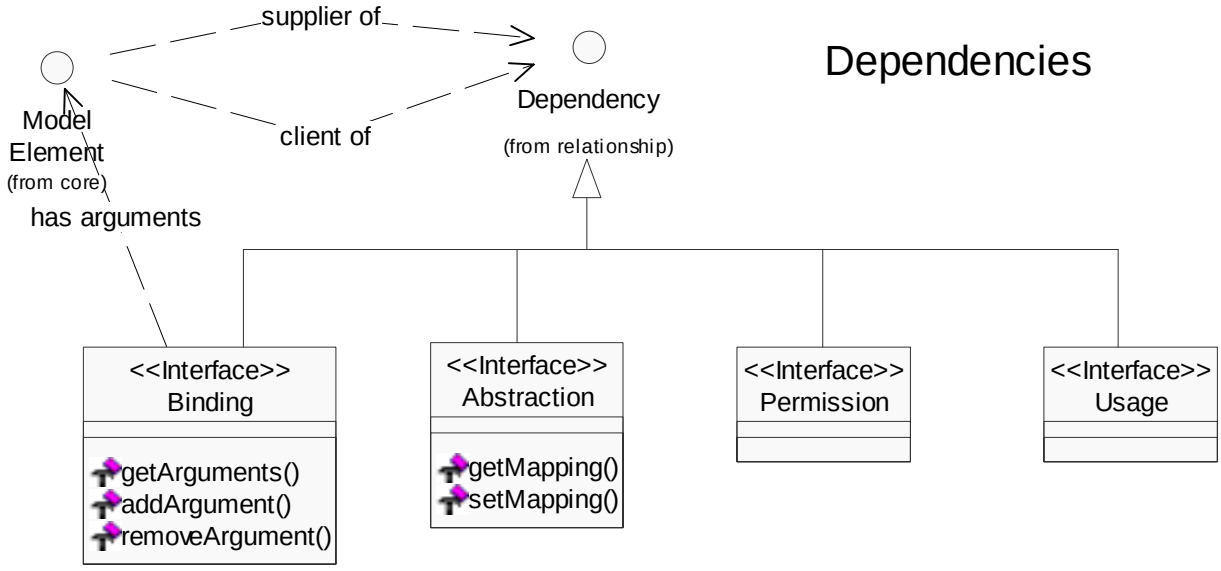


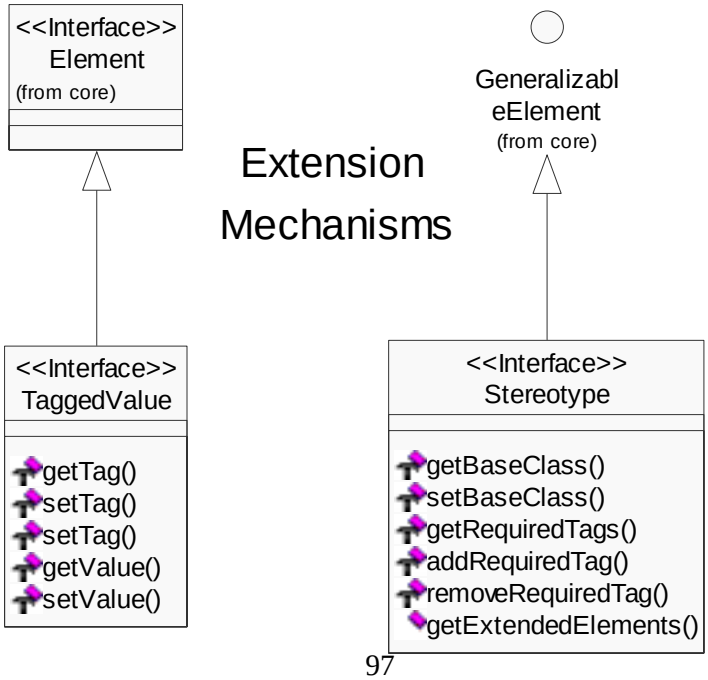
Code Generation

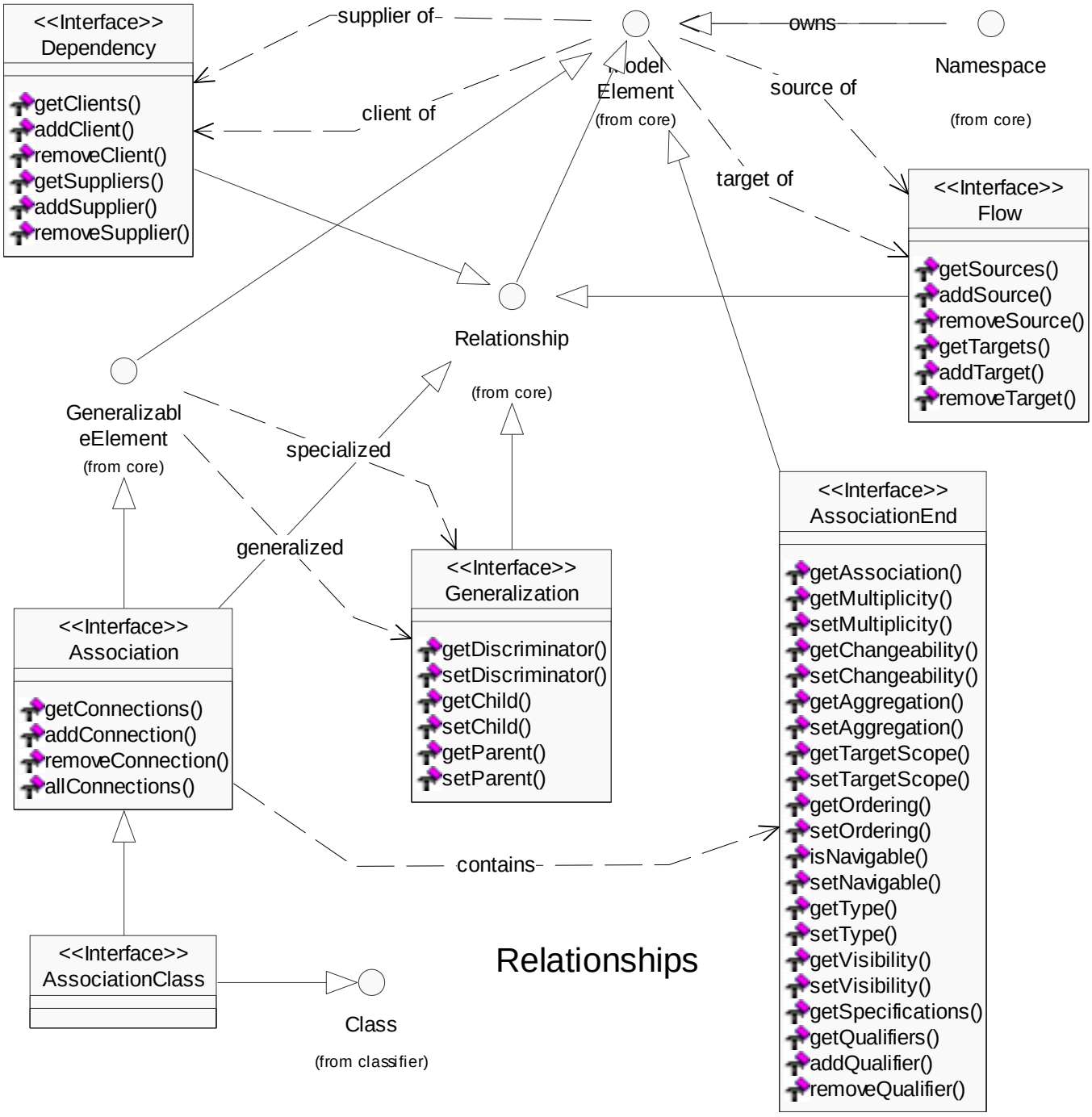


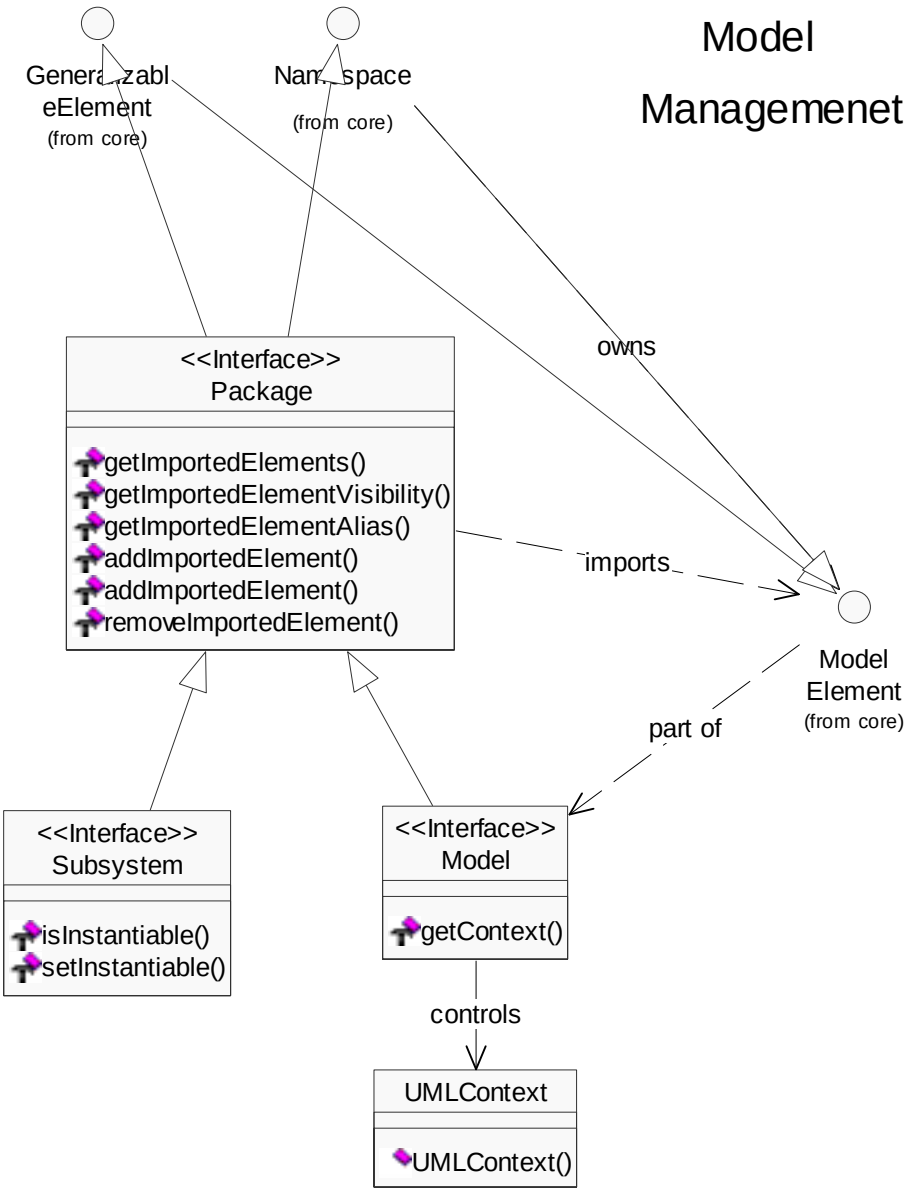

<<Interface>>
GeneralizableElement
isRoot()
setRoot()
isLeaf()
setLeaf()
isAbstract()
setAbstract()
getGeneralizations()
addGeneralization()
removeGeneralization()
getSpecializations()
addSpecialization()
removeSpecialization()
parents()
allParents()
specialized
Generalization
(from relationship)
Backbone
Checkable
(from util)
Writeable
(from util)
CustomData
Holder
(from util)
UML
(from util)
creates
<<Interface>>
Element
resetAll()
Visitable
(from util)
<<Interface>>
ModelElement
getNamespace()
getStereotypeConstraints()
addStereotypeConstraint()
removeStereotypeConstraint()
prettyString()
getSourceFlows()
addSourceFlow()
removeSourceFlow()
getTargetFlows()
addTargetFlow()
removeTargetFlow()
getClientDependencies()
addClientDependency()
removeClientDependency()
getSupplierDependencies()
addSupplierDependency()
removeSupplierDependency()
getTaggedValues()
addTaggedValue()
removeTaggedValue()
getTaggedValue()
getTaggedValue()
getStereotype()
setStereotype()
suppliers()
allSuppliers()
clients()
allClients()
models()
isTemplate()
isInstantiated()
templateArguments()
Stereotype
(from extension)
TaggedValue
(from extension)
extended by
tagged by
Association
End
(from relationship)
has participants
Dependency
(from relationship)
client of
supplier of
has arguments
Binding
(from dependency)
<<Interface>>
Classifier
getFeatures()
addFeature()
removeFeature()
getParticipants()
addParticipant()
removeParticipant()
allFeatures()
allOperations()
allMethods()
allAttributes()
allAssociations()
associations()
oppositeAssociationEnds()
allOppositeAssociationEnds()
specifications()
allContents()
<<Interface>>
Namespace
getOwnedElements()
getOwnedElementVisibility()
addOwnedElement()
addOwnedElement()
removeOwnedElement()
contents()
allContents()
allVisibleElements()
allSurroundingNamespaces()
(from relationship)
owns
source of
target of
Flow
typed by
Typed
Element
(from util)
owns
constrained by
<<Interface>>
Parameter
getType()
setType()
getKind()
setKind()
getDefaultValue()
setDefaultValue()
getOwner()
<<Interface>>
Feature
getVisibility()
setVisibility()
getOwner()
getOwnerScope()
setOwnerScope()
<<Interface>>
StructuralFeature
getTargetScope()
setTargetScope()
getType()
setType()
getMultiplicity()
setMultiplicity()
getChangeability()
setChangeability()
<<Interface>>
Constraint
setBody()
getBody()
getConstrainedElements()
owns
<<Interface>>
Attribute
getInitialValue()
setInitialValue()
getAssociationEnd()
setAssociationEnd()
<<Interface>>
BehavioralFeature
isQuery()
setQuery()
getParameters()
addParameter()
removeParameter()
hasSameSignature()
<<Interface>>
Operation
isRoot()
setRoot()
isLeaf()
setLeaf()
isAbstract()
setAbstract()
getConcurrency()
setConcurrency()
<<Interface>>
Method
setBody()
getBody()
setSpecification()
getSpecification()
toString()
specified by
<<Interface>>
Relationship

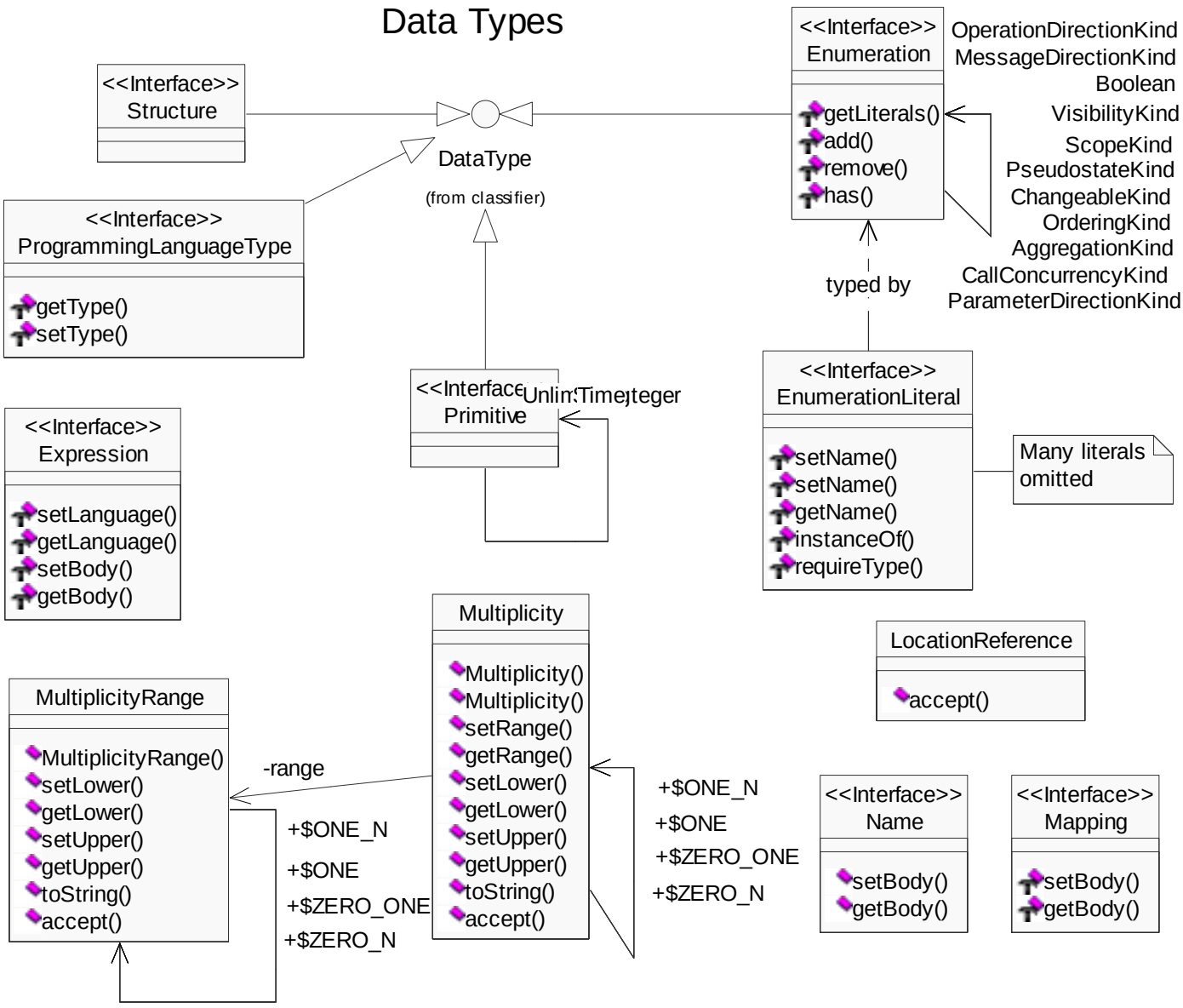

Classifiers
Classifier
(from core)
<<Interface>>
Class
isActive()
setActive()
toString()
<<Interface>>
DataType
<<Interface>>
Interface

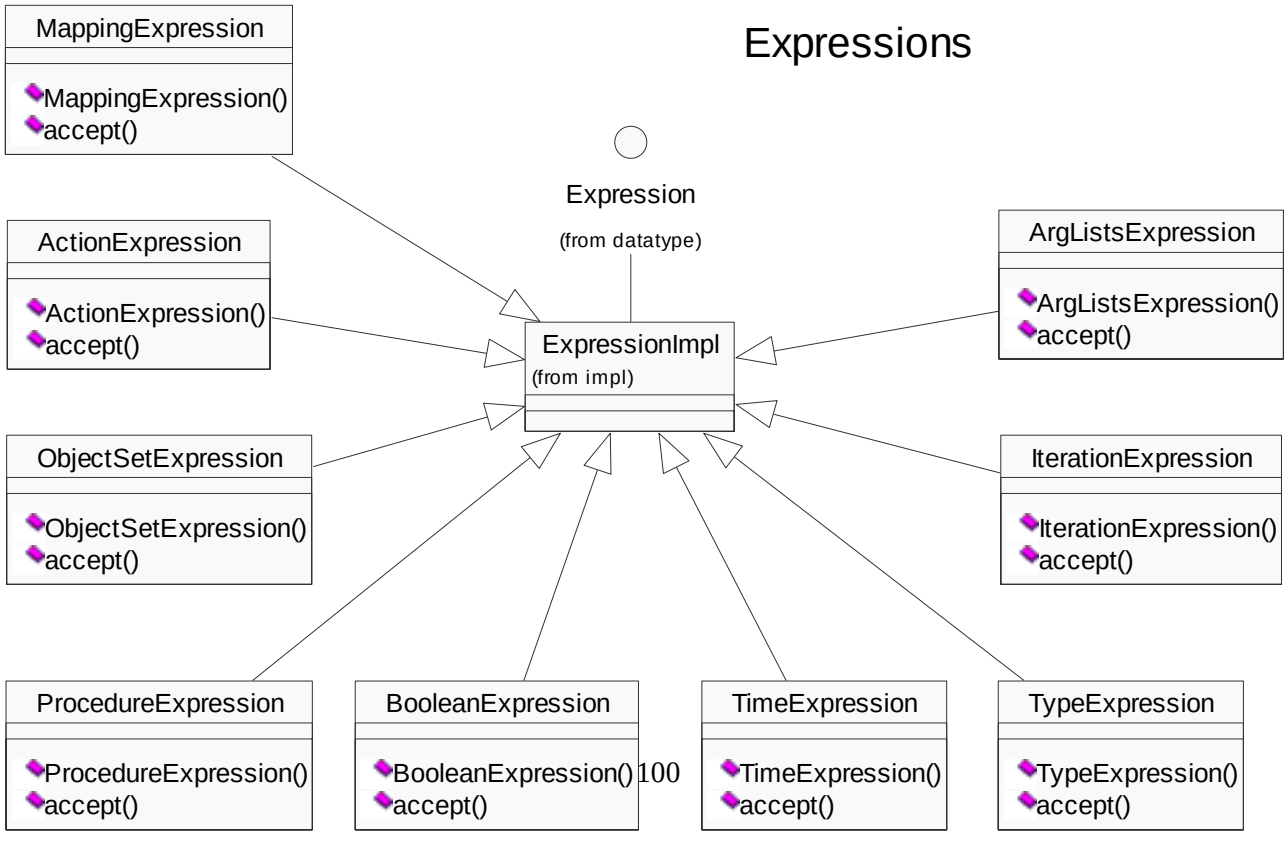

supplier of
client of
Model
Element
(from core)
has arguments
Dependency
(from relationship)
Dependencies
<<Interface>>
Binding
getArguments()
addArgument()
removeArgument()
<<Interface>>
Abstraction
getMapping()
setMapping()
<<Interface>>
Permission
<<Interface>>
Usage

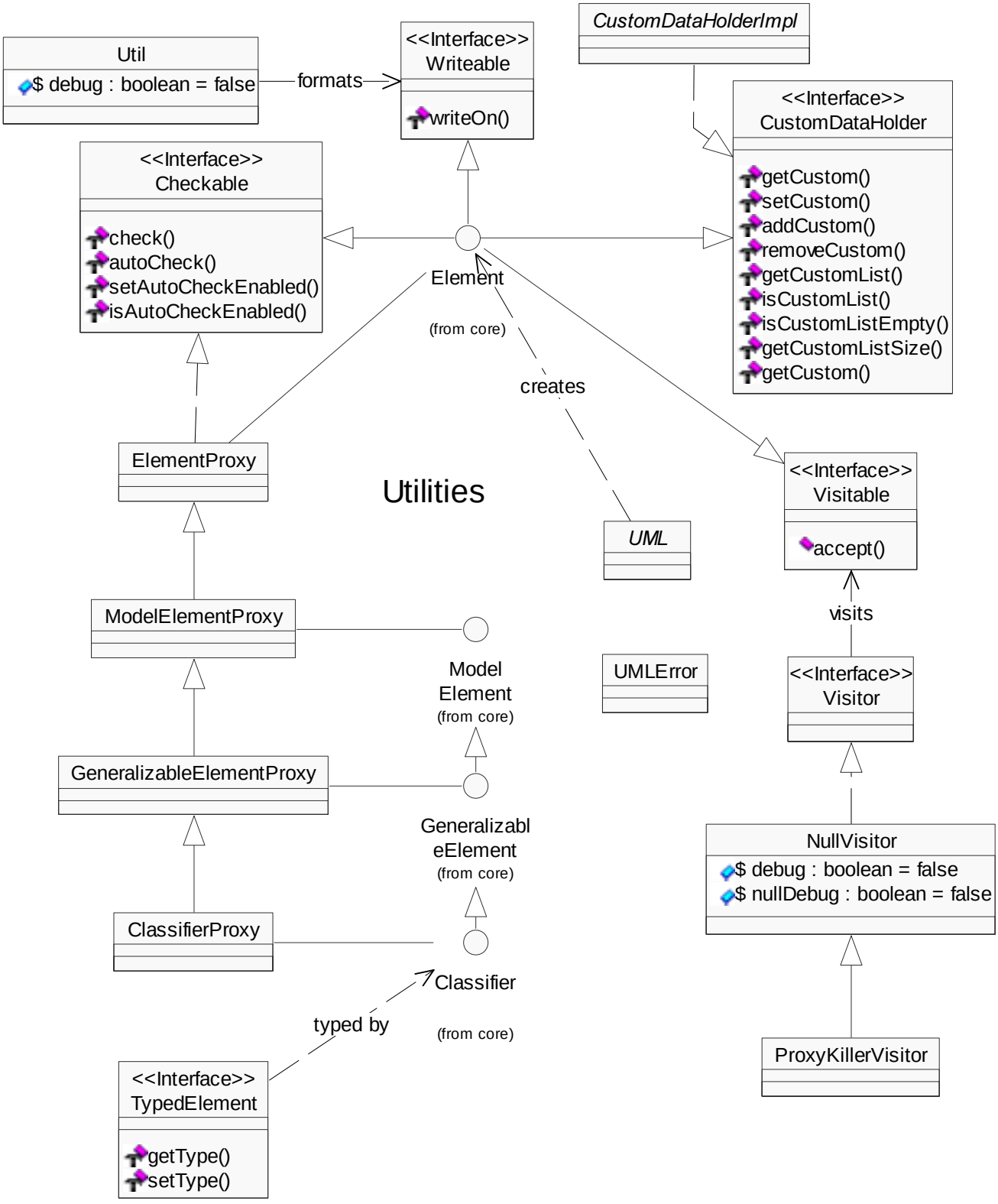

<<Interface>>
Element
(from core)
Extension
Mechanisms
Generalizabl
eElement
(from core)
<<Interface>>
TaggedValue
getTag()
setTag()
setTag()
getValue()
setValue()
<<Interface>>
Stereotype
getBaseClass()
setBaseClass()
getRequiredTags()
addRequiredTag()
removeRequiredTag()
getExtendedElements()


<<Interface>>
Dependency
getClients()
addClient()
removeClient()
getSuppliers()
addSupplier()
removeSupplier()
supplier of
client of
owns
odel
Element
(from core)
Namespace
(from core)
source of
target of
<<Interface>>
Flow
getSources()
addSource()
removeSource()
getTargets()
addTarget()
removeTarget()
Relationship
(from core)
Generalizabl
eElement
(from core)
specialized
generalized
<<Interface>>
Generalization
getDiscriminator()
setDiscriminator()
getChild()
setChild()
getParent()
setParent()
<<Interface>>
AssociationEnd
getAssociation()
getMultiplicity()
setMultiplicity()
getChangeability()
setChangeability()
getAggregation()
setAggregation()
getTargetScope()
setTargetScope()
getOrdering()
setOrdering()
isNavigable()
setNavigable()
getType()
setType()
getVisibility()
setVisibility()
getSpecifications()
getQualifiers()
addQualifier()
removeQualifier()
<<Interface>>
Association
getConnections()
addConnection()
removeConnection()
allConnections()
contains
<<Interface>>
AssociationClass
Class
(from classifier)
Relationships


Model
Managemenet
Generalizable
Element
(from core)
Namespace
(from core)
owns
<<Interface>>
Package
getImportedElements()
getImportedElementVisibility()
getImportedElementAlias()
addImportedElement()
addImportedElement()
removeImportedElement()
imports
Model
Element
(from core)
part of
<<Interface>>
Subsystem
isInstantiable()
setInstantiable()
<<Interface>>
Model
getContext()
controls
UMLContext
UMLContext()


Data Types
<<Interface>>
Structure
DataType
(from classifier)
<<Interface>>
Enumeration
getLiterals()
add()
remove()
has()
OperationDirectionKind
MessageDirectionKind
Boolean
VisibilityKind
ScopeKind
PseudostateKind
ChangeableKind
OrderingKind
AggregationKind
CallConcurrencyKind
ParameterDirectionKind
<<Interface>>
ProgrammingLanguageType
getType()
setType()
typed by
<<Interface>>
Primitive
UnlimitedNatural
Integer
<<Interface>>
EnumerationLiteral
setName()
setName()
getName()
instanceOf()
requireType()
Many literals omitted
<<Interface>>
Expression
setLanguage()
getLanguage()
setBody()
getBody()
Multiplicity
Multiplicity()
Multiplicity()
setRange()
getRange()
setLower()
getLower()
setUpper()
getUpper()
toString()
accept()
LocationReference
accept()
MultiplicityRange
MultiplicityRange()
setLower()
getLower()
setUpper()
getUpper()
toString()
accept()
-range
+$ONE_N
+$ONE
+$ZERO_ONE
+$ZERO_N
+$ONE_N
+$ONE
+$ZERO_ONE
+$ZERO_N
<<Interface>>
Name
setBody()
getBody()
<<Interface>>
Mapping
setBody()
getBody()


Expressions
MappingExpression
MappingExpression()
accept()
Expression
(from datatype)
ActionExpression
ActionExpression()
accept()
ArgListsExpression
ArgListsExpression()
accept()
ExpressionImpl
(from impl)
ObjectSetExpression
ObjectSetExpression()
accept()
IterationExpression
IterationExpression()
accept()
ProcedureExpression
ProcedureExpression()
accept()
BooleanExpression
BooleanExpression()
accept()
TimeExpression
TimeExpression()
accept()
TypeExpression
TypeExpression()
accept()


Util
$ debug : boolean = false
formats
<<Interface>>
Writeable
writeOn()
CustomDataHolderImpl
<<Interface>>
CustomDataHolder
getCustom()
setCustom()
addCustom()
removeCustom()
getCustomList()
isCustomList()
isCustomListEmpty()
getCustomListSize()
getCustom()
<<Interface>>
Checkable
check()
autoCheck()
setAutoCheckEnabled()
isAutoCheckEnabled()
Element
(from core)
creates
ElementProxy
Utilities
UML
<<Interface>>
Visitable
accept()
visits
ModelElementProxy
Model
Element
(from core)
UMLError
<<Interface>>
Visitor
GeneralizableElementProxy
GeneralizablE
eElement
(from core)
NullVisitor
$ debug : boolean = false
$ nullDebug : boolean = false
ClassifierProxy
Classifier
(from core)
typed by
ProxyKillerVisitor
<<Interface>>
TypedElement
getType()
setType()

9 Appendix: Output from test cases

Section 5: Results analyzes data present in the following listings.

Production of the listings begin by running our pattern detection tool, with the commands:

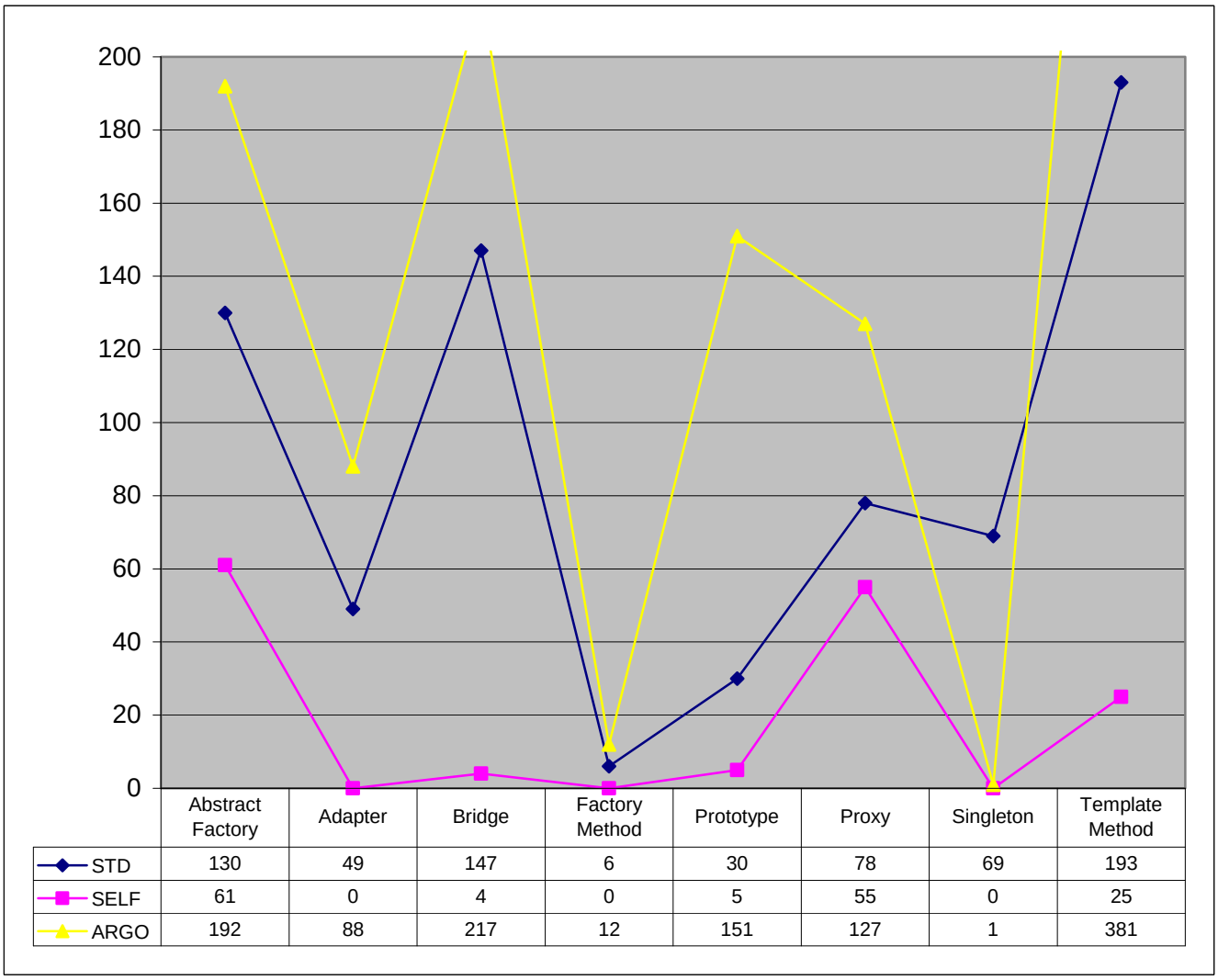
Listing 14: Executing test cases

```
reeng o std –ci
reeng o self –ci
reeng o argo –ci
```

This will produce three text files that can be imported by a spreadsheet supporting text files (Comma Separated); the additional grouping, sorting and layout should be done manually.

METRICS

	STD	SELF	ARGO
Abstract Factory	130	61	192
Adapter	49	0	88
Bridge	147	4	217
Factory Method	6	0	12
Prototype	30	5	151
Proxy	78	55	127
Singleton	69	0	1
Template Method	193	25	381



	Abstract Factory	Adapter	Bridge	Factory Method	Prototype	Proxy	Singleton	Template Method
STD	130	49	147	6	30	78	69	193
SELF	61	0	4	0	5	55	0	25
ARGO	192	88	217	12	151	127	1	381

ABSTRACT FACTORY: STD

	AbstractFactory	ConcreteFactory	AbstractProduct	ConcreteProduct	CreateOperation
1	java.io.ByteArrayOutputStream	java.io.ByteArrayOutputStream	java.lang.String	java.lang.String	toString
2	java.io.DataInputStream	java.io.DataInputStream	java.lang.String	java.lang.String	readUTF
3	java.io.File	java.io.File	java.net.URL	java.net.URL	toURL
4	java.io.File	java.io.File	java.io.File	java.io.File	createTempFile
5	java.io.File	java.io.File	java.io.File	java.io.File	getCanonicalFile
6	java.io.File	java.io.File	java.io.File	java.io.File	getAbsoluteFile
7	java.io.File	java.io.File	java.io.File	java.io.File	getParentFile
8	java.io.ObjectInputStream	java.io.ObjectInputStream	java.io.ObjectInputStream$GetField	java.io.ObjectInputStream$GetFieldImpl	readFields
9	java.io.ObjectOutputStream	java.io.ObjectOutputStream	java.io.ObjectOutputStream$PutField	java.io.ObjectOutputStream$PutFieldImpl	putFields
10	java.lang.Boolean	java.lang.Boolean	java.lang.Boolean	java.lang.Boolean	valueOf
11	java.lang.Byte	java.lang.Byte	java.lang.Byte	java.lang.Byte	valueOf
12	java.lang.Class	java.lang.Class	java.security.ProtectionDomain	java.security.ProtectionDomain	getProtectionDomain
13	java.lang.ClassLoader	java.lang.ClassLoader	java.util.Enumeration	sun.misc.CompoundEnumeration	getResources
14	java.lang.Double	java.lang.Double	java.lang.Double	java.lang.Double	valueOf
15	java.lang.Float	java.lang.Float	java.lang.Float	java.lang.Float	valueOf
16	java.lang.FloatingDecimal	java.lang.FloatingDecimal	java.lang.FloatingDecimal	java.lang.FloatingDecimal	readJavaFormatString
17	java.lang.FloatingDecimal	java.lang.FloatingDecimal	java.lang.String	java.lang.String	toJavaFormatString
18	java.lang.Integer	java.lang.Integer	java.lang.Integer	java.lang.Integer	valueOf
19	java.lang.Integer	java.lang.Integer	java.lang.String	java.lang.String	toString
20	java.lang.Long	java.lang.Long	java.lang.Long	java.lang.Long	valueOf
21	java.lang.Long	java.lang.Long	java.lang.String	java.lang.String	toString
22	java.lang.Object	java.io.ByteArrayOutputStream	java.lang.String	java.lang.String	toString
23	java.lang.Object	java.lang.FloatingDecimal	java.lang.String	java.lang.String	toString
24	java.lang.Object	java.lang.StringBuffer	java.lang.String	java.lang.String	toString
25	java.lang.Short	java.lang.Short	java.lang.Short	java.lang.Short	valueOf
26	java.lang.String	java.lang.String	java.lang.String	java.lang.String	valueOf
27	java.lang.String	java.lang.String	java.lang.String	java.lang.String	copyValueOf
28	java.lang.String	java.lang.String	java.lang.String	java.lang.String	toUpperCase
29	java.lang.String	java.lang.String	java.lang.String	java.lang.String	toLowerCase
30	java.lang.String	java.lang.String	java.lang.String	java.lang.String	replace
31	java.lang.String	java.lang.String	java.lang.String	java.lang.String	concat
32	java.lang.StringBuffer	java.lang.StringBuffer	java.lang.String	java.lang.String	substring
33	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	valueOf
34	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	add
35	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	subtract
36	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	multiply
37	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	divide
38	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	remainder

39	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	pow
40	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	gcd
41	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	negate
42	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	modPow
43	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	java.math.BigInteger	modInverse
44	java.net.InetAddress	java.net.InetAddress	java.net.InetAddress	java.net.InetAddress	getByName
45	java.net.URLConnection	java.net.URLConnection	java.security.Permission	java.security.AllPermission	getPermission
46	java.security.AccessController	java.security.AccessController	java.security.AccessControlContext	java.security.AccessControlContext	getContext
47	java.security.cert.CertificateFactory	java.security.cert.CertificateFactory	java.security.cert.CertificateFactory	java.security.cert.CertificateFactory	getInstance
48	java.security.MessageDigest	java.security.MessageDigest	java.security.MessageDigest	java.security.MessageDigest$Delegate	getInstance
49	java.security.Permission	java.io.FilePermission	java.security.PermissionCollection	java.io.FilePermissionCollection	newPermissionCollection
50	java.security.Permission	java.net.SocketPermission	java.security.PermissionCollection	java.net.SocketPermissionCollection	newPermissionCollection
51	java.security.Permission	java.security.BasicPermission	java.security.PermissionCollection	java.security.BasicPermissionCollection	newPermissionCollection
52	java.security.Permission	java.security.UnresolvedPermission	java.security.PermissionCollection	java.security.UnresolvedPermissionCollec	newPermissionCollection
53	java.security.Permission	java.util.PropertyPermission	java.security.PermissionCollection	java.util.PropertyPermissionCollection	newPermissionCollection
54	java.text.DateFormatSymbols	java.text.DateFormatSymbols	java.lang.String	java.lang.String	getLocalPatternChars
55	java.text.Format	java.text.Format	java.lang.Object	java.text.ParseException	parseObject
56	java.text.Format	java.text.Format	java.lang.Object	java.text.ParsePosition	parseObject
57	java.util.AbstractList	java.util.AbstractList	java.util.List	java.util.SubList	subList
58	java.util.AbstractList	java.util.AbstractList	java.util.ListIterator	java.util.AbstractList$ListItr	listIterator
59	java.util.AbstractList	java.util.AbstractList	java.lang.Object	java.lang.UnsupportedOperationExceptior	set
60	java.util.AbstractList	java.util.AbstractList	java.lang.Object	java.lang.UnsupportedOperationExceptior	remove
61	java.util.AbstractList	java.util.Collections$CopiesList	java.lang.Object	java.lang.IndexOutOfBoundsException	get
62	java.util.AbstractList	java.util.Collections$EmptyList	java.lang.Object	java.lang.IndexOutOfBoundsException	get
63	java.util.AbstractList	java.util.Vector	java.lang.Object	java.lang.ArrayIndexOutOfBoundsExceptic	get
64	java.util.AbstractList	java.util.Vector	java.lang.Object	java.lang.ArrayIndexOutOfBoundsExceptic	set
65	java.util.AbstractList	java.util.Vector	java.lang.Object	java.lang.ArrayIndexOutOfBoundsExceptic	remove
66	java.util.AbstractList$Itr	java.util.AbstractList$Itr	java.lang.Object	java.util.NoSuchElementException	next
67	java.util.AbstractList$ListItr	java.util.AbstractList$ListItr	java.lang.Object	java.util.NoSuchElementException	previous
68	java.util.AbstractMap	java.util.AbstractMap	java.lang.Object	java.lang.UnsupportedOperationExceptior	put
69	java.util.Arrays	java.util.Arrays	java.util.List	java.util.ArrayList	asList
70	java.util.Calendar	java.util.Calendar	java.util.Calendar	java.util.GregorianCalendar	getInstance
71	java.util.Calendar	java.util.Calendar	java.util.Date	java.util.Date	getTime
72	java.util.Collections	java.util.Collections	java.util.SortedMap	java.util.Collections$SynchronizedSortedM	synchronizedSortedMap
73	java.util.Collections	java.util.Collections	java.util.SortedSet	java.util.Collections$SynchronizedSortedS	synchronizedSortedSet
74	java.util.Collections	java.util.Collections	java.util.SortedMap	java.util.Collections$UnmodifiableSortedM	unmodifiableSortedMap
75	java.util.Collections	java.util.Collections	java.util.Set	java.util.Collections$SingletonSet	singleton
76	java.util.Collections	java.util.Collections	java.util.Set	java.util.Collections$SynchronizedSet	synchronizedSet
77	java.util.Collections	java.util.Collections	java.util.SortedSet	java.util.Collections$UnmodifiableSortedS	unmodifiableSortedSet
78	java.util.Collections	java.util.Collections	java.util.Set	java.util.Collections$UnmodifiableSet	unmodifiableSet
79	java.util.Collections	java.util.Collections	java.util.Map	java.util.Collections$SynchronizedMap	synchronizedMap

80	java.util.Collections	java.util.Collections	java.util.List	java.util.Collections$CopiesList	nCopies
81	java.util.Collections	java.util.Collections	java.util.Map	java.util.Collections$UnmodifiableMap	unmodifiableMap
82	java.util.Collections	java.util.Collections	java.util.List	java.util.Collections$SynchronizedList	synchronizedList
83	java.util.Collections	java.util.Collections	java.util.Collection	java.util.Collections$SynchronizedCollecti	synchronizedCollection
84	java.util.Collections	java.util.Collections	java.util.List	java.util.Collections$UnmodifiableList	unmodifiableList
85	java.util.Collections	java.util.Collections	java.util.Collection	java.util.Collections$UnmodifiableCollectio	unmodifiableCollection
86	java.util.Collections$SynchronizedList	java.util.Collections$SynchronizedList	java.util.List	java.util.Collections$SynchronizedList	subList
87	java.util.Collections$SynchronizedMap	java.util.Collections$SynchronizedMap	java.util.Set	java.util.Collections$SynchronizedSet	keySet
88	java.util.Collections$SynchronizedMap	java.util.Collections$SynchronizedMap	java.util.Set	java.util.Collections$SynchronizedSet	entrySet
89	java.util.Collections$SynchronizedMap	java.util.Collections$SynchronizedMap	java.util.Collection	java.util.Collections$SynchronizedCollecti	values
90	java.util.Collections$SynchronizedSortedM	java.util.Collections$SynchronizedSortedM	java.util.SortedMap	java.util.Collections$SynchronizedSortedM	subMap
91	java.util.Collections$SynchronizedSortedM	java.util.Collections$SynchronizedSortedM	java.util.SortedMap	java.util.Collections$SynchronizedSortedM	headMap
92	java.util.Collections$SynchronizedSortedM	java.util.Collections$SynchronizedSortedM	java.util.SortedMap	java.util.Collections$SynchronizedSortedM	tailMap
93	java.util.Collections$SynchronizedSortedS	java.util.Collections$SynchronizedSortedS	java.util.SortedSet	java.util.Collections$SynchronizedSortedS	subSet
94	java.util.Collections$SynchronizedSortedS	java.util.Collections$SynchronizedSortedS	java.util.SortedSet	java.util.Collections$SynchronizedSortedS	headSet
95	java.util.Collections$SynchronizedSortedS	java.util.Collections$SynchronizedSortedS	java.util.SortedSet	java.util.Collections$SynchronizedSortedS	tailSet
96	java.util.Collections$UnmodifiableList	java.util.Collections$UnmodifiableList	java.util.List	java.util.Collections$UnmodifiableList	subList
97	java.util.Collections$UnmodifiableList	java.util.Collections$UnmodifiableList	java.lang.Object	java.lang.UnsupportedOperationExceptior	set
98	java.util.Collections$UnmodifiableList	java.util.Collections$UnmodifiableList	java.lang.Object	java.lang.UnsupportedOperationExceptior	remove
99	java.util.Collections$UnmodifiableMap	java.util.Collections$UnmodifiableMap	java.util.Set	java.util.Collections$UnmodifiableMap$Ur	entrySet
100	java.util.Collections$UnmodifiableMap	java.util.Collections$UnmodifiableMap	java.lang.Object	java.lang.UnsupportedOperationExceptior	put
101	java.util.Collections$UnmodifiableMap	java.util.Collections$UnmodifiableMap	java.lang.Object	java.lang.UnsupportedOperationExceptior	remove
102	java.util.Collections$UnmodifiableMap$Ur	java.util.Collections$UnmodifiableMap$Ur	java.lang.Object	java.lang.UnsupportedOperationExceptior	setValue
103	java.util.Collections$UnmodifiableSortedM	java.util.Collections$UnmodifiableSortedM	java.util.SortedMap	java.util.Collections$UnmodifiableSortedM	subMap
104	java.util.Collections$UnmodifiableSortedM	java.util.Collections$UnmodifiableSortedM	java.util.SortedMap	java.util.Collections$UnmodifiableSortedM	headMap
105	java.util.Collections$UnmodifiableSortedM	java.util.Collections$UnmodifiableSortedM	java.util.SortedMap	java.util.Collections$UnmodifiableSortedM	tailMap
106	java.util.Collections$UnmodifiableSortedS	java.util.Collections$UnmodifiableSortedS	java.util.SortedSet	java.util.Collections$UnmodifiableSortedS	subSet
107	java.util.Collections$UnmodifiableSortedS	java.util.Collections$UnmodifiableSortedS	java.util.SortedSet	java.util.Collections$UnmodifiableSortedS	headSet
108	java.util.Collections$UnmodifiableSortedS	java.util.Collections$UnmodifiableSortedS	java.util.SortedSet	java.util.Collections$UnmodifiableSortedS	tailSet
109	java.util.Dictionary	java.util.Hashtable	java.util.Enumeration	java.util.Hashtable$Enumerator	keys
110	java.util.Dictionary	java.util.Hashtable	java.util.Enumeration	java.util.Hashtable$Enumerator	elements
111	java.util.Dictionary	java.util.Hashtable	java.lang.Object	java.lang.NullPointerException	put
112	java.util.GregorianCalendar	java.util.GregorianCalendar	java.util.Date	java.util.Date	getGregorianChange
113	java.util.HashMap$HashIterator	java.util.HashMap$HashIterator	java.lang.Object	java.util.ConcurrentModificationException	next
114	java.util.HashMap$HashIterator	java.util.HashMap$HashIterator	java.lang.Object	java.util.NoSuchElementException	next
115	java.util.Hashtable	java.util.Hashtable	java.util.Set	java.util.Hashtable$EntrySet	entrySet
116	java.util.Hashtable	java.util.Hashtable	java.util.Set	java.util.Hashtable$KeySet	keySet
117	java.util.Hashtable	java.util.Hashtable	java.util.Collection	java.util.Hashtable$ValueCollection	values
118	java.util.Hashtable$Entry	java.util.Hashtable$Entry	java.lang.Object	java.lang.NullPointerException	setValue
119	java.util.Hashtable$Enumerator	java.util.Hashtable$Enumerator	java.lang.Object	java.util.ConcurrentModificationException	next
120	java.util.Hashtable$Enumerator	java.util.Hashtable$Enumerator	java.lang.Object	java.util.NoSuchElementException	nextElement

121	java.util.jar.JarFile	java.util.jar.JarFile	java.util.jar.Manifest	java.util.jar.Manifest	getManifest
122	java.util.ResourceBundle	java.util.ResourceBundle	java.util.Locale	java.util.Locale	getLocale
123	java.util.ResourceBundle	java.util.ResourceBundle	java.lang.Object	java.util.MissingResourceException	getObject
124	java.util.Stack	java.util.Stack	java.lang.Object	java.util.EmptyStackException	peek
125	java.util.Vector	java.util.Vector	java.lang.Object	java.util.NoSuchElementException	firstElement
126	java.util.Vector	java.util.Vector	java.lang.Object	java.util.NoSuchElementException	lastElement
127	java.util.Vector	java.util.Vector	java.lang.Object	java.lang.ArrayIndexOutOfBoundsExcepti	elementAt
128	java.util.zip.ZipFile	java.util.jar.JarFile	java.util.zip.ZipEntry	java.util.jar.JarFile$JarFileEntry	getEntry
129	java.util.zip.ZipFile	java.util.jar.JarFile	java.io.InputStream	java.util.jar.JarVerifier$VerifierStream	getInputStream
130	java.util.zip.ZipFile	java.util.zip.ZipFile	java.util.zip.ZipEntry	java.util.zip.ZipEntry	getEntry

ABSTRACT FACTORY: SELF

	AbstractFactory	ConcreteFactory	AbstractProduct	ConcreteProduct	CreateOperation
1	reeng.GenericReader	reeng.GenericReader	uml.core.Classifier	uml.util.ClassifierProxy	getTypeLazy
2	reeng.GenericReader	reeng.GenericReader	java.util.List	java.util.ArrayList	getRootClasses
3	reeng.java.ASTNullVisitor	reeng.java.ASTUMLVisitorDeclarations	java.lang.Object	java.lang.Integer	visit
4	reeng.java.ASTNullVisitor	reeng.java.ASTUMLVisitorDeclarations	java.lang.Object	reeng.code.SymbolTable	visit
5	uml.core.impl.ClassifierImpl	uml.core.impl.ClassifierImpl	java.util.Set	java.util.HashSet	allOperations
6	uml.core.impl.ClassifierImpl	uml.core.impl.ClassifierImpl	java.util.Set	java.util.HashSet	allMethods
7	uml.core.impl.ClassifierImpl	uml.core.impl.ClassifierImpl	java.util.Set	java.util.HashSet	allAttributes
8	uml.core.impl.ClassifierImpl	uml.core.impl.ClassifierImpl	java.util.Set	java.util.HashSet	associations
9	uml.core.impl.ClassifierImpl	uml.core.impl.ClassifierImpl	java.util.Set	java.util.HashSet	oppositeAssociationEnds
10	uml.core.impl.ClassifierImpl	uml.core.impl.ClassifierImpl	java.util.Set	java.util.HashSet	specifications
11	uml.core.impl.GeneralizableElementImpl	uml.core.impl.GeneralizableElementImpl	java.util.Set	java.util.HashSet	parents
12	uml.core.impl.GeneralizableElementImpl	uml.core.impl.GeneralizableElementImpl	java.util.Set	java.util.HashSet	allParents
13	uml.core.impl.ModelElementImpl	uml.core.impl.ModelElementImpl	java.util.Set	java.util.HashSet	suppliers
14	uml.core.impl.ModelElementImpl	uml.core.impl.ModelElementImpl	java.util.Set	java.util.HashSet	allSuppliers
15	uml.core.impl.ModelElementImpl	uml.core.impl.ModelElementImpl	java.util.Set	java.util.HashSet	clients
16	uml.core.impl.ModelElementImpl	uml.core.impl.ModelElementImpl	java.util.Set	java.util.HashSet	allClients
17	uml.core.impl.ModelElementImpl	uml.core.impl.ModelElementImpl	java.util.Set	java.util.HashSet	models
18	uml.core.impl.ModelElementImpl	uml.core.impl.ModelElementImpl	java.util.Set	java.util.HashSet	templateArguments
19	uml.core.impl.ModelElementImpl	uml.core.impl.ModelElementImpl	java.util.List	java.util.ArrayList	getStereotypeConstraints
20	uml.core.impl.ModelElementImpl	uml.core.impl.ModelElementImpl	java.util.List	java.util.ArrayList	getSourceFlows
21	uml.core.impl.ModelElementImpl	uml.core.impl.ModelElementImpl	java.util.List	java.util.ArrayList	getTargetFlows
22	uml.core.impl.ModelElementImpl	uml.core.impl.ModelElementImpl	java.util.List	java.util.ArrayList	getClientDependencies
23	uml.core.impl.ModelElementImpl	uml.core.impl.ModelElementImpl	java.util.List	java.util.ArrayList	getSupplierDependencies
24	uml.core.impl.ModelElementImpl	uml.core.impl.ModelElementImpl	java.util.List	java.util.ArrayList	getTaggedValues
25	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	java.util.Set	java.util.HashSet	allVisibleElements
26	uml.core.relationship.impl.GeneralizationIı	uml.core.relationship.impl.GeneralizationIı	java.util.List	java.util.ArrayList	getChildren
27	uml.core.relationship.impl.GeneralizationIı	uml.core.relationship.impl.GeneralizationIı	java.util.List	java.util.ArrayList	getParents

28	uml.extension.impl.StereotypeImpl	uml.extension.impl.StereotypeImpl	java.util.List	java.util.ArrayList	getRequiredTags
29	uml.util.UML	uml.util.UML	uml.extension.TaggedValue	uml.extension.impl.TaggedValueImpl	newTaggedValue
30	uml.util.UML	uml.util.UML	uml.extension.Stereotype	uml.extension.impl.StereotypeImpl	newStereotype
31	uml.util.UML	uml.util.UML	uml.datatype.Structure	uml.datatype.impl.StructureImpl	newStructure
32	uml.util.UML	uml.util.UML	uml.datatype.ProgrammingLanguageType	uml.datatype.impl.ProgrammingLanguage	newProgrammingLanguageType
33	uml.util.UML	uml.util.UML	uml.datatype.Mapping	uml.datatype.impl.MappingImpl	newMapping
34	uml.util.UML	uml.util.UML	uml.datatype.Expression	uml.datatype.impl.ExpressionImpl	newExpression
35	uml.util.UML	uml.util.UML	uml.datatype.Primitive	uml.datatype.impl.PrimitiveImpl	newPrimitive
36	uml.util.UML	uml.util.UML	uml.datatype.Enumeration	uml.datatype.impl.EnumerationImpl	newEnumeration
37	uml.util.UML	uml.util.UML	uml.datatype.EnumerationLiteral	uml.datatype.impl.EnumerationLiteralImpl	newEnumerationLiteral
38	uml.util.UML	uml.util.UML	uml.datatype.Name	uml.datatype.impl.NameImpl	newName
39	uml.util.UML	uml.util.UML	uml.model.Subsystem	uml.model.impl.SubsystemImpl	newSubsystem
40	uml.util.UML	uml.util.UML	uml.model.Package	uml.model.impl.PackageImpl	newPackage
41	uml.util.UML	uml.util.UML	uml.model.Model	uml.model.impl.ModelImpl	newModel
42	uml.util.UML	uml.util.UML	uml.core.Parameter	uml.core.impl.ParameterImpl	newParameter
43	uml.util.UML	uml.util.UML	uml.core.Operation	uml.core.impl.OperationImpl	newOperation
44	uml.util.UML	uml.util.UML	uml.core.Namespace	uml.core.impl.NamespaceImpl	newNamespace
45	uml.util.UML	uml.util.UML	uml.core.Constraint	uml.core.impl.ConstraintImpl	newConstraint
46	uml.util.UML	uml.util.UML	uml.core.Method	uml.core.impl.MethodImpl	newMethod
47	uml.util.UML	uml.util.UML	uml.core.Attribute	uml.core.impl.AttributeImpl	newAttribute
48	uml.util.UML	uml.util.UML	uml.core.dependency.Permission	uml.core.dependency.impl.PermissionImp	newPermission
49	uml.util.UML	uml.util.UML	uml.core.dependency.Binding	uml.core.dependency.impl.BindingImpl	newBinding
50	uml.util.UML	uml.util.UML	uml.core.dependency.Abstraction	uml.core.dependency.impl.AbstractionImp	newAbstraction
51	uml.util.UML	uml.util.UML	uml.core.Classifier	uml.core.impl.ClassifierImpl	newClassifier
52	uml.util.UML	uml.util.UML	uml.core.classifier.Interface	uml.core.classifier.impl.InterfaceImpl	newInterface
53	uml.util.UML	uml.util.UML	uml.core.classifier.DataType	uml.core.classifier.impl.DataTypeImpl	newDataType
54	uml.util.UML	uml.util.UML	uml.core.classifier.Class	uml.core.classifier.impl.ClassImpl	newClass
55	uml.util.UML	uml.util.UML	uml.core.relationship.Generalization	uml.core.relationship.impl.GeneralizationI	newGeneralization
56	uml.util.UML	uml.util.UML	uml.core.relationship.Flow	uml.core.relationship.impl.FlowImpl	newFlow
57	uml.util.UML	uml.util.UML	uml.core.relationship.Dependency	uml.core.relationship.impl.DependencyIm	newDependency
58	uml.util.UML	uml.util.UML	uml.core.relationship.Association	uml.core.relationship.impl.AssociationImp	newAssociation
59	uml.util.UML	uml.util.UML	uml.core.relationship.AssociationEnd	uml.core.relationship.impl.AssociationEnd	newAssociationEnd
60	uml.util.UML	uml.util.UML	uml.core.relationship.AssociationClass	uml.core.relationship.impl.AssociationCla	newAssociationClass
61	uml.util.Util	uml.util.Util	java.util.HashSet	java.util.HashSet	newHashSet

ABSTRACT FACTORY: ARGO

	AbstractFactory	ConcreteFactory	AbstractProduct	ConcreteProduct	CreateOperation
1	uci.argo.checklist.CheckManager	uci.argo.checklist.CheckManager	uci.argo.checklist.ChecklistStatus	uci.argo.checklist.ChecklistStatus	getStatusFor
2	uci.argo.kernel.Agency	uci.argo.kernel.Agency	java.util.Vector	java.util.Vector	criticsForClass
3	uci.argo.kernel.Critic	uci.argo.kernel.CompoundCritic	java.util.Vector	java.util.Vector	getSupportedDecisions
4	uci.argo.kernel.Critic	uci.argo.kernel.CompoundCritic	java.util.Vector	java.util.Vector	getSupportedGoals
5	uci.argo.kernel.Critic	uci.argo.kernel.CompoundCritic	uci.util.VectorSet	uci.util.VectorSet	getKnowledgeTypes
6	uci.argo.kernel.Critic	uci.argo.kernel.Critic	uci.argo.kernel.ToDoItem	uci.argo.kernel.ToDoItem	toDoItem
7	uci.argo.kernel.Designer	uci.argo.kernel.Designer	uci.util.VectorSet	uci.util.VectorSet	getKnowledgeTypes
8	uci.argo.kernel.ToDoList	uci.argo.kernel.ToDoList	java.util.Vector	java.util.Vector	getDecisions
9	uci.argo.kernel.ToDoList	uci.argo.kernel.ToDoList	java.util.Vector	java.util.Vector	getGoals
10	uci.gef.ArrowHead	uci.gef.ArrowHead	java.awt.Point	java.awt.Point	pointAlongLine
11	uci.gef.CmdCopy$SimpleSelection	uci.gef.CmdCopy$SimpleSelection	java.lang.Object	java.awt.datatransfer.UnsupportedFlavorE	getTransferData
12	uci.gef.Diagram	uci.gef.Diagram	java.util.Vector	java.util.Vector	getNodes
13	uci.gef.Diagram	uci.gef.Diagram	java.util.Vector	java.util.Vector	getEdges
14	uci.gef.Fig	uci.gef.Fig	java.awt.Rectangle	java.awt.Rectangle	getBounds
15	uci.gef.Fig	uci.gef.Fig	java.awt.Rectangle	java.awt.Rectangle	routingRect
16	uci.gef.Fig	uci.gef.Fig	java.awt.Point	java.awt.Point	getLocation
17	uci.gef.Fig	uci.gef.Fig	java.awt.Point	java.awt.Point	getFirstPoint
18	uci.gef.Fig	uci.gef.Fig	java.awt.Point	java.awt.Point	getLastPoint
19	uci.gef.Fig	uci.gef.Fig	java.awt.Point	java.awt.Point	pointAlongPerimeter
20	uci.gef.Fig	uci.gef.Fig	java.awt.Point	java.awt.Point	center
21	uci.gef.Fig	uci.gef.Fig	java.awt.Point	java.awt.Point	connectionPoint
22	uci.gef.Fig	uci.gef.Fig	java.awt.Dimension	java.awt.Dimension	getSize
23	uci.gef.Fig	uci.gef.Fig	java.awt.Dimension	java.awt.Dimension	getMinimumSize
24	uci.gef.Fig	uci.gef.Fig	java.awt.Dimension	java.awt.Dimension	getPreferedSize
25	uci.gef.Fig	uci.gef.Fig	java.util.Vector	java.util.Vector	getPopUpActions
26	uci.gef.Fig	uci.gef.FigGroup	java.lang.Object	java.util.Vector	clone
27	uci.gef.Fig	uci.gef.FigLine	java.awt.Point	java.awt.Point	getPoints
28	uci.gef.Fig	uci.gef.FigPoly	java.awt.Point	java.awt.Point	getPoints
29	uci.gef.Fig	uci.gef.FigText	java.awt.Dimension	java.awt.Dimension	getMinimumSize
30	uci.gef.Fig	uci.uml.visual.FigActionState	java.awt.Dimension	java.awt.Dimension	getMinimumSize
31	uci.gef.Fig	uci.uml.visual.FigActor	java.awt.Dimension	java.awt.Dimension	getMinimumSize
32	uci.gef.Fig	uci.uml.visual.FigActor	uci.gef.Selection	uci.uml.visual.SelectionActor	makeSelection
33	uci.gef.Fig	uci.uml.visual.FigBranchState	uci.gef.Selection	uci.uml.visual.SelectionMoveClarifiers	makeSelection
34	uci.gef.Fig	uci.uml.visual.FigClass	java.awt.Dimension	java.awt.Dimension	getMinimumSize
35	uci.gef.Fig	uci.uml.visual.FigClass	uci.gef.Selection	uci.uml.visual.SelectionClass	makeSelection
36	uci.gef.Fig	uci.uml.visual.FigClassifierRole	java.awt.Dimension	java.awt.Dimension	getMinimumSize
37	uci.gef.Fig	uci.uml.visual.FigCompositeState	java.awt.Dimension	java.awt.Dimension	getMinimumSize
38	uci.gef.Fig	uci.uml.visual.FigCompositeState	uci.gef.Selection	uci.uml.visual.SelectionState	makeSelection

39	uci.gef.Fig	uci.uml.visual.FigEdgeModelElement	uci.gef.Selection	uci.uml.visual.SelectionEdgeClarifiers	makeSelection
40	uci.gef.Fig	uci.uml.visual.FigFinalState	uci.gef.Selection	uci.uml.visual.SelectionState	makeSelection
41	uci.gef.Fig	uci.uml.visual.FigHistoryState	uci.gef.Selection	uci.uml.visual.SelectionMoveClarifiers	makeSelection
42	uci.gef.Fig	uci.uml.visual.FigInitialState	uci.gef.Selection	uci.uml.visual.SelectionState	makeSelection
43	uci.gef.Fig	uci.uml.visual.FigInstance	java.awt.Dimension	java.awt.Dimension	getMinimumSize
44	uci.gef.Fig	uci.uml.visual.FigInterface	java.awt.Dimension	java.awt.Dimension	getMinimumSize
45	uci.gef.Fig	uci.uml.visual.FigInterface	uci.gef.Selection	uci.uml.visual.SelectionInterface	makeSelection
46	uci.gef.Fig	uci.uml.visual.FigMessage	java.awt.Dimension	java.awt.Dimension	getMinimumSize
47	uci.gef.Fig	uci.uml.visual.FigNodeModelElement	uci.gef.Selection	uci.uml.visual.SelectionNodeClarifiers	makeSelection
48	uci.gef.Fig	uci.uml.visual.FigPackage	java.awt.Dimension	java.awt.Dimension	getMinimumSize
49	uci.gef.Fig	uci.uml.visual.FigState	java.awt.Dimension	java.awt.Dimension	getMinimumSize
50	uci.gef.Fig	uci.uml.visual.FigState	uci.gef.Selection	uci.uml.visual.SelectionState	makeSelection
51	uci.gef.Fig	uci.uml.visual.FigUseCase	java.awt.Dimension	java.awt.Dimension	getMinimumSize
52	uci.gef.Fig	uci.uml.visual.FigUseCase	uci.gef.Selection	uci.uml.visual.SelectionUseCase	makeSelection
53	uci.gef.FigNode	uci.gef.FigNode	java.lang.Object	java.awt.Rectangle	hitPort
54	uci.gef.FigNode	uci.gef.FigNode	java.lang.Object	java.awt.Rectangle	deepHitPort
55	uci.gef.FigNode	uci.gef.FigNode	java.util.Vector	java.util.Vector	getPortFigs
56	uci.gef.FigPoly	uci.gef.FigPoly	java.awt.Polygon	java.awt.Polygon	getPolygon
57	uci.gef.FigPoly	uci.gef.FigPoly	java.util.Vector	java.util.Vector	getPointsVector
58	uci.gef.FigPoly	uci.gef.FigPoly	java.util.Vector	java.util.Vector	getPointsVectorNotFirst
59	uci.gef.FigText	uci.gef.FigText	java.lang.String	java.lang.String	deleteLastCharFromString
60	uci.gef.FigText	uci.gef.FigText	uci.gef.FigTextEditor	uci.gef.FigTextEditor	startTextEditor
61	uci.gef.Geometry	uci.gef.Geometry	java.awt.Point	java.awt.Point	ptClosestTo
62	uci.gef.Guide	uci.gef.Guide	java.awt.Point	java.awt.Point	snapTo
63	uci.gef.JGraph	uci.gef.JGraph	java.lang.Object	uci.gef.JGraph	clone
64	uci.gef.Layer	uci.gef.Layer	java.util.Vector	java.util.Vector	getContentsNoEdges
65	uci.gef.Layer	uci.gef.Layer	java.util.Vector	java.util.Vector	getContentsEdgesOnly
66	uci.gef.Layer	uci.gef.Layer	java.util.Enumeration	uci.util.EnumerationPredicate	elementsIn
67	uci.gef.Layer	uci.gef.Layer	java.util.Enumeration	uci.util.EnumerationPredicate	nodesIn
68	uci.gef.ModeBroom	uci.gef.ModeBroom	java.util.Vector	java.util.Vector	touching
69	uci.gef.PathConv	uci.gef.PathConv	java.awt.Point	java.awt.Point	getPoint
70	uci.gef.Prefs	uci.gef.Prefs	java.awt.Color	java.awt.Color	getRubberbandColor
71	uci.gef.Selection	uci.gef.Selection	java.awt.Rectangle	java.awt.Rectangle	getBounds
72	uci.gef.Selection	uci.uml.visual.SelectionWButtons	java.awt.Rectangle	java.awt.Rectangle	getBounds
73	uci.gef.SelectionManager	uci.gef.SelectionManager	java.awt.Rectangle	java.awt.Rectangle	getBounds
74	uci.gef.SelectionManager	uci.gef.SelectionManager	java.awt.Rectangle	java.awt.Rectangle	getContentBounds
75	uci.gef.SelectionManager	uci.gef.SelectionManager	java.util.Vector	java.util.Vector	getFigs
76	uci.gef.SelectionManager	uci.gef.SelectionManager	uci.gef.Selection	uci.gef.SelectionNoop	makeSelectionFor
77	uci.gef.SelectionManager	uci.gef.SelectionManager	uci.gef.Selection	uci.gef.SelectionMove	makeSelectionFor
78	uci.gef.SelectionManager	uci.gef.SelectionManager	uci.gef.Selection	uci.gef.SelectionLowerRight	makeSelectionFor
79	uci.gef.SelectionManager	uci.gef.SelectionManager	uci.gef.Selection	uci.gef.SelectionReshape	makeSelectionFor

80	uci.gef.SelectionManager	uci.gef.SelectionManager	uci.gef.Selection	uci.gef.SelectionResize	makeSelectionFor
81	uci.graph.MutableGraphSupport	uci.uml.visual.ClassDiagramGraphModel	java.lang.Object	uci.uml.Behavioral_Elements.Common_B	connect
82	uci.graph.MutableGraphSupport	uci.uml.visual.ClassDiagramGraphModel	java.lang.Object	uci.uml.Foundation.Core.Dependency	connect
83	uci.graph.MutableGraphSupport	uci.uml.visual.ClassDiagramGraphModel	java.lang.Object	uci.uml.Foundation.Core.Realization	connect
84	uci.graph.MutableGraphSupport	uci.uml.visual.ClassDiagramGraphModel	java.lang.Object	uci.uml.Foundation.Core.Association	connect
85	uci.graph.MutableGraphSupport	uci.uml.visual.ClassDiagramGraphModel	java.lang.Object	uci.uml.Foundation.Core.Generalization	connect
86	uci.graph.MutableGraphSupport	uci.uml.visual.CollabDiagramGraphModel	java.lang.Object	uci.uml.Behavioral_Elements.Collaboratio	connect
87	uci.graph.MutableGraphSupport	uci.uml.visual.StateDiagramGraphModel	java.lang.Object	uci.uml.Behavioral_Elements.State_Mach	connect
88	uci.graph.MutableGraphSupport	uci.uml.visual.UseCaseDiagramGraphMo	java.lang.Object	uci.uml.Foundation.Core.Association	connect
89	uci.graph.MutableGraphSupport	uci.uml.visual.UseCaseDiagramGraphMo	java.lang.Object	uci.uml.Foundation.Core.Generalization	connect
90	uci.ui.Swatch	uci.ui.Swatch	uci.ui.Swatch	uci.ui.Swatch	forColor
91	uci.ui.ToolBar	uci.ui.ToolBar	javax.swing.ButtonGroup	javax.swing.ButtonGroup	addRadioGroup
92	uci.ui.ToolBar	uci.ui.ToolBar	javax.swing.JButton	javax.swing.JButton	add
93	uci.ui.ToolBar	uci.ui.ToolBar	javax.swing.JToggleButton	javax.swing.JToggleButton	addToggle
94	uci.uml.critics.ChildGenUML	uci.uml.critics.ChildGenUML	java.util.Enumeration	uci.util.EnumerationComposite	gen
95	uci.uml.critics.ChildGenUML	uci.uml.critics.ChildGenUML	java.util.Enumeration	uci.util.EnumerationSingle	gen
96	uci.uml.Foundation.Core.ElementImpl	uci.uml.Foundation.Core.ElementImpl	java.util.Vector	java.util.Vector	getNamedProperty
97	uci.uml.Foundation.Core.ModelElementIm	uci.uml.Behavioral_Elements.State_Mach	java.util.Vector	java.util.Vector	alsoTrash
98	uci.uml.Foundation.Core.ModelElementIm	uci.uml.Behavioral_Elements.State_Mach	java.util.Vector	java.util.Vector	alsoTrash
99	uci.uml.Foundation.Core.ModelElementIm	uci.uml.Foundation.Core.Dependency	java.lang.Object	java.util.Vector	prepareForTrash
100	uci.uml.Foundation.Core.ModelElementIm	uci.uml.Foundation.Core.ModelElementIm	java.util.Vector	java.util.Vector	alsoTrash
101	uci.uml.Foundation.Core.NamespaceImpl	uci.uml.Foundation.Core.NamespaceImpl	java.util.Vector	java.util.Vector	getModelElements
102	uci.uml.Foundation.Data_Types.Expressic	uci.uml.Foundation.Data_Types.Expressic	java.util.Vector	java.util.Vector	getNamedProperty
103	uci.uml.generate.Parser	uci.uml.generate.Parser	uci.uml.Foundation.Data_Types.Uninterpr	uci.uml.Foundation.Data_Types.Uninterpr	parseUninterpreted
104	uci.uml.generate.Parser	uci.uml.generate.Parser	uci.uml.Foundation.Data_Types.Expressic	uci.uml.Foundation.Data_Types.Expressic	parseExpression
105	uci.uml.generate.Parser	uci.uml.generate.Parser	uci.uml.Foundation.Data_Types.Name	uci.uml.Foundation.Data_Types.Name	parseName
106	uci.uml.generate.Parser	uci.uml.generate.ParserDisplay	uci.uml.Behavioral_Elements.State_Mach	uci.uml.Behavioral_Elements.State_Mach	parseGuard
107	uci.uml.generate.Parser	uci.uml.generate.ParserDisplay	uci.uml.Behavioral_Elements.State_Mach	uci.uml.Behavioral_Elements.State_Mach	parseEvent
108	uci.uml.generate.Parser	uci.uml.generate.ParserDisplay	uci.uml.Behavioral_Elements.State_Mach	uci.uml.Behavioral_Elements.State_Mach	parseTransition
109	uci.uml.generate.Parser	uci.uml.generate.ParserDisplay	uci.uml.Foundation.Core.Parameter	uci.uml.Foundation.Core.Parameter	parseParameter
110	uci.uml.generate.Parser	uci.uml.generate.ParserDisplay	uci.uml.Foundation.Core.Operation	uci.uml.Foundation.Core.Operation	parseOperation
111	uci.uml.generate.Parser	uci.uml.generate.ParserDisplay	uci.uml.Foundation.Core.Attribute	uci.uml.Foundation.Core.Attribute	parseAttribute
112	uci.uml.generate.Parser	uci.uml.generate.ParserDisplay	uci.uml.Foundation.Data_Types.Multiplicit	uci.uml.Foundation.Data_Types.Multiplicit	parseMultiplicity
113	uci.uml.generate.ParserDisplay	uci.uml.generate.ParserDisplay	uci.uml.Behavioral_Elements.Common_B	uci.uml.Behavioral_Elements.Common_B	parseActions
114	uci.uml.ocl.OCLEvaluator	uci.uml.ocl.OCLEvaluator	java.util.Vector	java.util.Vector	eval
115	uci.uml.ocl.OCLEvaluator	uci.uml.ocl.OCLEvaluator	java.util.Vector	java.util.Vector	flatten
116	uci.uml.ocl.OCLExpander	uci.uml.ocl.OCLExpander	java.util.Vector	java.util.Vector	findTemplatesFor
117	uci.uml.ocl.TemplateReader	uci.uml.ocl.TemplateReader	java.util.Hashtable	java.util.Hashtable	read
118	uci.uml.ui.ChildGenFind	uci.uml.ui.ChildGenFind	java.util.Enumeration	uci.util.EnumerationComposite	gen
119	uci.uml.ui.ChildGenFind	uci.uml.ui.ChildGenFind	java.util.Enumeration	uci.util.EnumerationSingle	gen
120	uci.uml.ui.ChildGenRelated	uci.uml.ui.ChildGenRelated	java.util.Enumeration	uci.util.Enum	gen

121	uci.uml.ui.ChildGenRelated	uci.uml.ui.ChildGenRelated	java.util.Enumeration	uci.util.EnumerationComposite	gen
122	uci.uml.ui.ChildGenRelated	uci.uml.ui.ChildGenRelated	java.util.Enumeration	uci.util.EnumerationSingle	gen
123	uci.uml.ui.ClassGenerationDialog	uci.uml.ui.ClassGenerationDialog	java.awt.Dimension	java.awt.Dimension	getMaximumSize
124	uci.uml.ui.Menu	uci.uml.ui.Menu	javax.swing.JCheckBoxMenuItem	javax.swing.JCheckBoxMenuItem	addCheckItem
125	uci.uml.ui.MultiEditorPane	uci.uml.ui.MultiEditorPane	java.awt.Dimension	java.awt.Dimension	getMinimumSize
126	uci.uml.ui.MultiEditorPane	uci.uml.ui.MultiEditorPane	java.awt.Dimension	java.awt.Dimension	getPreferredSize
127	uci.uml.ui.NavigatorPane	uci.uml.ui.NavigatorPane	java.awt.Dimension	java.awt.Dimension	getMinimumSize
128	uci.uml.ui.NavigatorPane	uci.uml.ui.NavigatorPane	java.awt.Dimension	java.awt.Dimension	getPreferredSize
129	uci.uml.ui.Project	uci.uml.ui.Project	java.util.Vector	java.util.Vector	getStats
130	uci.uml.ui.Project	uci.uml.ui.Project	java.beans.VetoableChangeSupport	java.beans.VetoableChangeSupport	getVetoSupport
131	uci.uml.ui.Project	uci.uml.ui.Project	java.util.Vector	java.util.Vector	getDefinedTypesVector
132	uci.uml.ui.Project	uci.uml.ui.Project	java.net.URL	java.net.URL	findMemberURLInSearchPath
133	uci.uml.ui.Project	uci.uml.ui.Project	uci.uml.Foundation.Core.Classifier	uci.uml.Foundation.Core.MMClass	findType
134	uci.uml.ui.Project	uci.uml.ui.Project	uci.uml.ui.Project	uci.uml.ui.Project	makeEmptyProject
135	uci.uml.ui.SpacerPanel	uci.uml.ui.SpacerPanel	java.awt.Dimension	java.awt.Dimension	getSize
136	uci.uml.ui.SpacerPanel	uci.uml.ui.SpacerPanel	java.awt.Dimension	java.awt.Dimension	getPreferredSize
137	uci.uml.ui.SpacerPanel	uci.uml.ui.SpacerPanel	java.awt.Dimension	java.awt.Dimension	getMinimumSize
138	uci.uml.ui.TabSpawnable	uci.uml.ui.TabDiagram	java.lang.Object	uci.uml.ui.TabDiagram	clone
139	uci.uml.visual.ClassDiagramGraphModel	uci.uml.visual.ClassDiagramGraphModel	java.util.Vector	java.util.Vector	getPorts
140	uci.uml.visual.ClassDiagramGraphModel	uci.uml.visual.ClassDiagramGraphModel	java.util.Vector	java.util.Vector	getInEdges
141	uci.uml.visual.ClassDiagramGraphModel	uci.uml.visual.ClassDiagramGraphModel	java.util.Vector	java.util.Vector	getOutEdges
142	uci.uml.visual.ClassDiagramRenderer	uci.uml.visual.ClassDiagramRenderer	uci.gef.FigEdge	uci.uml.visual.FigDependency	getFigEdgeFor
143	uci.uml.visual.ClassDiagramRenderer	uci.uml.visual.ClassDiagramRenderer	uci.gef.FigEdge	uci.uml.visual.FigLink	getFigEdgeFor
144	uci.uml.visual.ClassDiagramRenderer	uci.uml.visual.ClassDiagramRenderer	uci.gef.FigNode	uci.uml.visual.FigInstance	getFigNodeFor
145	uci.uml.visual.ClassDiagramRenderer	uci.uml.visual.ClassDiagramRenderer	uci.gef.FigEdge	uci.uml.visual.FigRealization	getFigEdgeFor
146	uci.uml.visual.ClassDiagramRenderer	uci.uml.visual.ClassDiagramRenderer	uci.gef.FigEdge	uci.uml.visual.FigGeneralization	getFigEdgeFor
147	uci.uml.visual.ClassDiagramRenderer	uci.uml.visual.ClassDiagramRenderer	uci.gef.FigEdge	uci.uml.visual.FigAssociation	getFigEdgeFor
148	uci.uml.visual.ClassDiagramRenderer	uci.uml.visual.ClassDiagramRenderer	uci.gef.FigNode	uci.uml.visual.FigInterface	getFigNodeFor
149	uci.uml.visual.ClassDiagramRenderer	uci.uml.visual.ClassDiagramRenderer	uci.gef.FigNode	uci.uml.visual.FigPackage	getFigNodeFor
150	uci.uml.visual.ClassDiagramRenderer	uci.uml.visual.ClassDiagramRenderer	uci.gef.FigNode	uci.uml.visual.FigClass	getFigNodeFor
151	uci.uml.visual.CollabDiagramGraphModel	uci.uml.visual.CollabDiagramGraphModel	java.util.Vector	java.util.Vector	getPorts
152	uci.uml.visual.CollabDiagramGraphModel	uci.uml.visual.CollabDiagramGraphModel	java.util.Vector	java.util.Vector	getInEdges
153	uci.uml.visual.CollabDiagramGraphModel	uci.uml.visual.CollabDiagramGraphModel	java.util.Vector	java.util.Vector	getOutEdges
154	uci.uml.visual.CollabDiagramRenderer	uci.uml.visual.CollabDiagramRenderer	uci.gef.FigEdge	uci.uml.visual.FigAssociationRole	getFigEdgeFor
155	uci.uml.visual.CollabDiagramRenderer	uci.uml.visual.CollabDiagramRenderer	uci.gef.FigNode	uci.uml.visual.FigMessage	getFigNodeFor
156	uci.uml.visual.CollabDiagramRenderer	uci.uml.visual.CollabDiagramRenderer	uci.gef.FigNode	uci.uml.visual.FigClassifierRole	getFigNodeFor
157	uci.uml.visual.StateDiagramGraphModel	uci.uml.visual.StateDiagramGraphModel	java.util.Vector	java.util.Vector	getPorts
158	uci.uml.visual.StateDiagramGraphModel	uci.uml.visual.StateDiagramGraphModel	java.util.Vector	java.util.Vector	getInEdges
159	uci.uml.visual.StateDiagramGraphModel	uci.uml.visual.StateDiagramGraphModel	java.util.Vector	java.util.Vector	getOutEdges
160	uci.uml.visual.StateDiagramRenderer	uci.uml.visual.StateDiagramRenderer	uci.gef.FigEdge	uci.uml.visual.FigTransition	getFigEdgeFor
161	uci.uml.visual.StateDiagramRenderer	uci.uml.visual.StateDiagramRenderer	uci.gef.FigNode	uci.uml.visual.FigHistoryState	getFigNodeFor

162	uci.uml.visual.StateDiagramRenderer	uci.uml.visual.StateDiagramRenderer	uci.gef.FigNode	uci.uml.visual.FigJoinState	getFigNodeFor
163	uci.uml.visual.StateDiagramRenderer	uci.uml.visual.StateDiagramRenderer	uci.gef.FigNode	uci.uml.visual.FigForkState	getFigNodeFor
164	uci.uml.visual.StateDiagramRenderer	uci.uml.visual.StateDiagramRenderer	uci.gef.FigNode	uci.uml.visual.FigBranchState	getFigNodeFor
165	uci.uml.visual.StateDiagramRenderer	uci.uml.visual.StateDiagramRenderer	uci.gef.FigNode	uci.uml.visual.FigFinalState	getFigNodeFor
166	uci.uml.visual.StateDiagramRenderer	uci.uml.visual.StateDiagramRenderer	uci.gef.FigNode	uci.uml.visual.FigInitialState	getFigNodeFor
167	uci.uml.visual.StateDiagramRenderer	uci.uml.visual.StateDiagramRenderer	uci.gef.FigNode	uci.uml.visual.FigState	getFigNodeFor
168	uci.uml.visual.StateDiagramRenderer	uci.uml.visual.StateDiagramRenderer	uci.gef.FigNode	uci.uml.visual.FigCompositeState	getFigNodeFor
169	uci.uml.visual.StateDiagramRenderer	uci.uml.visual.StateDiagramRenderer	uci.gef.FigNode	uci.uml.visual.FigActionState	getFigNodeFor
170	uci.uml.visual.UseCaseDiagramGraphMo	uci.uml.visual.UseCaseDiagramGraphMo	java.util.Vector	java.util.Vector	getPorts
171	uci.uml.visual.UseCaseDiagramGraphMo	uci.uml.visual.UseCaseDiagramGraphMo	java.util.Vector	java.util.Vector	getInEdges
172	uci.uml.visual.UseCaseDiagramGraphMo	uci.uml.visual.UseCaseDiagramGraphMo	java.util.Vector	java.util.Vector	getOutEdges
173	uci.uml.visual.UseCaseDiagramRenderer	uci.uml.visual.UseCaseDiagramRenderer	uci.gef.FigNode	uci.uml.visual.FigUseCase	getFigNodeFor
174	uci.uml.visual.UseCaseDiagramRenderer	uci.uml.visual.UseCaseDiagramRenderer	uci.gef.FigNode	uci.uml.visual.FigActor	getFigNodeFor
175	uci.uml.visual.UseCaseDiagramRenderer	uci.uml.visual.UseCaseDiagramRenderer	uci.gef.FigEdge	uci.uml.visual.FigGeneralization	getFigEdgeFor
176	uci.uml.visual.UseCaseDiagramRenderer	uci.uml.visual.UseCaseDiagramRenderer	uci.gef.FigEdge	uci.uml.visual.FigAssociation	getFigEdgeFor
177	uci.util.EnumerationComposite	uci.util.EnumerationComposite	java.lang.Object	java.util.NoSuchElementException	nextElement
178	uci.util.EnumerationEmpty	uci.util.EnumerationEmpty	java.lang.Object	java.util.NoSuchElementException	nextElement
179	uci.util.EnumerationPredicate	uci.util.EnumerationPredicate	java.lang.Object	java.util.NoSuchElementException	nextElement
180	uci.util.EnumerationSingle	uci.util.EnumerationSingle	java.lang.Object	java.util.NoSuchElementException	nextElement
181	uci.util.PredicateStringMatch	uci.util.PredicateStringMatch	uci.util.Predicate	uci.util.PredicateEquals	create
182	uci.util.PredicateStringMatch	uci.util.PredicateStringMatch	uci.util.Predicate	uci.util.PredicateStringMatch	create
183	uci.util.PredicateType	uci.util.PredicateType	uci.util.PredicateType	uci.util.PredicateType	create
184	uci.util.Util	uci.util.Util	javax.swing.ImageIcon	javax.swing.ImageIcon	loadIconResource
185	uci.util.Util	uci.util.Util	java.net.URL	java.net.URL	fileToURL
186	uci.util.Util	uci.util.Util	java.net.URL	java.net.URL	fixURLExtension
187	uci.util.Util	uci.util.Util	java.io.File	java.io.File	URLToFile
188	uci.util.VectorSet	uci.util.VectorSet	uci.util.VectorSet	uci.util.VectorSet	reachable
189	uci.util.VectorSet	uci.util.VectorSet	uci.util.VectorSet	uci.util.VectorSet	transitiveClosure
190	uci.xml.DTDEntityResolver	uci.xml.DTDEntityResolver	org.xml.sax.InputSource	org.xml.sax.InputSource	resolveEntity
191	uci.xml.SAXParserBase	uci.xml.SAXParserBase	org.xml.sax.InputSource	org.xml.sax.InputSource	resolveEntity
192	uci.xml.xmi.XMIParser	uci.xml.xmi.XMIParser	uci.uml.Foundation.Data_Types.Multiplicit	uci.uml.Foundation.Data_Types.Multiplicit	parseMultiplicity

ADAPTER: STD

	Target	Adapter	Adaptee	Request	SpecificRequest
1	java.io.File	java.io.File	java.lang.String	toURL	replace
2	java.io.File	java.io.File	java.lang.String	getParent	substring
3	java.io.File	java.io.File	java.lang.String	getParent	lastIndexOf
4	java.io.File	java.io.File	java.lang.String	getName	substring
5	java.io.File	java.io.File	java.lang.String	getName	lastIndexOf
6	java.io.InputStream	java.util.jar.JarVerifier$VerifierStream	java.util.jar.JarVerifier	read	update
7	java.io.ObjectInputStream	java.io.ObjectInputStream	java.util.ArrayList	registerValidation	size
8	java.io.ObjectInputStream	java.io.ObjectInputStream	java.util.ArrayList	registerValidation	add
9	java.io.ObjectInputStream	java.io.ObjectInputStream	java.util.ArrayList	registerValidation	get
10	java.io.ObjectOutputStream	java.io.ObjectOutputStream	java.io.ObjectStreamClass	writeObject	forClass
11	java.io.ObjectOutputStream	java.io.ObjectOutputStream	java.io.ObjectStreamClass	defaultWriteObject	forClass
12	java.io.ObjectOutputStream$PutField	java.io.ObjectOutputStream$PutFieldImpl	java.io.ObjectStreamClass	put	getField
13	java.lang.Object	java.net.URL	java.lang.String	hashCode	toLowerCase
14	java.lang.Object	java.util.jar.Attributes$Name	java.lang.String	equals	equalsIgnoreCase
15	java.lang.Object	java.util.jar.Attributes$Name	java.lang.String	hashCode	toLowerCase
16	java.lang.Thread	java.lang.Thread	java.lang.ThreadGroup	setPriority	getMaxPriority
17	java.lang.ThreadGroup	java.lang.ThreadGroup	java.lang.ThreadGroup	getParent	checkAccess
18	java.security.PermissionCollection	java.security.UnresolvedPermissionCollec	java.util.Hashtable	add	get
19	java.security.PermissionCollection	java.security.UnresolvedPermissionCollec	java.util.Hashtable	add	put
20	java.text.DateFormat	java.text.SimpleDateFormat	java.lang.String	format	charAt
21	java.text.DateFormat	java.text.SimpleDateFormat	java.lang.String	parse	charAt
22	java.text.DecimalFormat	java.text.DecimalFormat	java.text.DecimalFormatSymbols	getDecimalFormatSymbols	clone
23	java.text.MessageFormat	java.text.MessageFormat	java.lang.String	toPattern	length
24	java.text.MessageFormat	java.text.MessageFormat	java.lang.String	parse	substring
25	java.text.NumberFormat	java.text.DecimalFormat	java.text.DecimalFormatSymbols	format	getNaN
26	java.text.NumberFormat	java.text.DecimalFormat	java.text.DecimalFormatSymbols	format	getInfinity
27	java.text.NumberFormat	java.text.DecimalFormat	java.text.DecimalFormatSymbols	parse	getNaN
28	java.text.NumberFormat	java.text.DecimalFormat	java.text.DigitList	format	set
29	java.text.NumberFormat	java.text.DecimalFormat	java.text.DigitList	parse	getDouble
30	java.text.NumberFormat	java.text.DecimalFormat	java.text.DigitList	parse	getLong
31	java.text.SimpleDateFormat	java.text.SimpleDateFormat	java.text.DateFormatSymbols	getDateFormatSymbols	clone
32	java.util.jar.JarVerifier	java.util.jar.JarVerifier	java.io.ByteArrayOutputStream	beginEntry	reset
33	java.util.jar.JarVerifier	java.util.jar.JarVerifier	java.io.ByteArrayOutputStream	update	write
34	java.util.jar.JarVerifier	java.util.jar.JarVerifier	java.util.Hashtable	getCerts	get
35	java.util.jar.Manifest	java.util.jar.Manifest	java.util.jar.Attributes	read	size
36	java.util.Locale	java.util.Locale	java.lang.String	getISO3Language	length
37	java.util.Locale	java.util.Locale	java.lang.String	getISO3Country	length
38	java.util.ResourceBundle$ResourceCache	java.util.ResourceBundle$ResourceCache	java.lang.String	setKeyValues	hashCode

39	java.util.StringTokenizer	java.util.StringTokenizer	java.lang.String	nextToken	charAt
40	java.util.StringTokenizer	java.util.StringTokenizer	java.lang.String	countTokens	charAt
41	java.util.StringTokenizer	java.util.StringTokenizer	java.lang.String	nextToken	substring
42	java.util.zip.InflaterInputStream	java.util.jar.JarInputStream	java.util.jar.JarVerifier	read	update
43	java.util.zip.InflaterInputStream	java.util.zip.InflaterInputStream	java.util.zip.Inflater	read	needsInput
44	java.util.zip.InflaterInputStream	java.util.zip.ZipInputStream	java.util.zip.CRC32	read	update
45	java.util.zip.ZipEntry	java.util.zip.ZipEntry	java.lang.String	isDirectory	endsWith
46	java.util.zip.ZipFile	java.util.zip.ZipFile	java.util.Vector	close	size
47	java.util.zip.ZipFile	java.util.zip.ZipFile	java.util.Vector	close	get
48	java.util.zip.ZipInputStream	java.util.jar.JarInputStream	java.util.jar.JarVerifier	getNextEntry	beginEntry
49	java.util.zip.ZipInputStream	java.util.zip.ZipInputStream	java.util.zip.CRC32	getNextEntry	reset

ADAPTER: SELF

Target	Adapter	Adaptee	Request	SpecificRequest

ADAPTER: ARGO

	Target	Adapter	Adaptee	Request	SpecificRequest
1	uci.argo.kernel.Critic	uci.argo.kernel.Critic	uci.util.VectorSet	addKnowledgeType	addElement
2	uci.argo.kernel.Critic	uci.argo.kernel.Critic	uci.util.VectorSet	containsKnowledgeType	contains
3	uci.argo.kernel.Designer	uci.argo.kernel.Designer	uci.argo.kernel.Agency	run	determineActiveCritics
4	uci.argo.kernel.Designer	uci.argo.kernel.Designer	uci.argo.kernel.ToDoList	addToDoItems	addAll
5	uci.argo.kernel.Designer	uci.argo.kernel.Designer	uci.argo.kernel.ToDoList	removeToDoItems	removeAll
6	uci.argo.kernel.Designer	uci.argo.kernel.Designer	uci.argo.kernel.ToDoList	nondisruptivelyWarn	addElement
7	uci.argo.kernel.Wizard	uci.uml.critics.WizMEName	uci.uml.ui.todo.WizStepTextField	doAction	getText
8	uci.argo.kernel.Wizard	uci.uml.critics.WizMEName	uci.uml.ui.todo.WizStepTextField	canGoNext	getText
9	uci.gef.Diagram	uci.uml.visual.UMLDiagram	uci.uml.visual.DiagramInfo	setName	updateName
10	uci.gef.Editor	uci.gef.Editor	uci.gef.Fig	mouseMoved	getTipString
11	uci.gef.Editor	uci.gef.Editor	uci.gef.LayerManager	getGridHidden	findLayerNamed
12	uci.gef.Editor	uci.gef.Editor	uci.gef.LayerManager	setGridHidden	findLayerNamed
13	uci.gef.Editor	uci.gef.Editor	uci.gef.LayerManager	clone	getActiveLayer
14	uci.gef.Editor	uci.gef.Editor	uci.gef.LayerManager	getGraphModel	getActiveLayer
15	uci.gef.Editor	uci.gef.Editor	uci.gef.LayerManager	setGraphModel	getActiveLayer
16	uci.gef.Editor	uci.gef.Editor	uci.gef.LayerManager	getGraphNodeRenderer	getActiveLayer
17	uci.gef.Editor	uci.gef.Editor	uci.gef.LayerManager	setGraphNodeRenderer	getActiveLayer
18	uci.gef.Editor	uci.gef.Editor	uci.gef.LayerManager	getGraphEdgeRenderer	getActiveLayer
19	uci.gef.Editor	uci.gef.Editor	uci.gef.LayerManager	setGraphEdgeRenderer	getActiveLayer
20	uci.gef.Editor	uci.gef.Editor	uci.gef.LayerManager	figs	elements
21	uci.gef.Editor	uci.gef.Editor	uci.gef.ModeManager	mode	push
22	uci.gef.Editor	uci.gef.Editor	uci.gef.ModeManager	finishMode	pop

23	uci.gef.Editor	uci.gef.Editor	uci.gef.SelectionManager	removed	deselect
24	uci.gef.Fig	uci.uml.visual.FigActionState	uci.gef.FigRRect	setBounds	setCornerRadius
25	uci.gef.Fig	uci.uml.visual.FigClass	uci.uml.visual.FigCompartment	getPopUpActions	isDisplayed
26	uci.gef.Fig	uci.uml.visual.FigClass	uci.uml.visual.FigCompartment	getMinimumSize	isDisplayed
27	uci.gef.Fig	uci.uml.visual.FigCompositeState	uci.gef.FigLine	setBounds	setShape
28	uci.gef.Fig	uci.uml.visual.FigInterface	uci.gef.FigText	setBounds	getMinimumSize
29	uci.gef.Fig	uci.uml.visual.FigState	uci.gef.FigLine	setBounds	setShape
30	uci.gef.JGraph	uci.gef.JGraph	uci.gef.Editor	setDiagram	getLayerManager
31	uci.gef.JGraph	uci.gef.JGraph	uci.gef.Editor	setDiagram	setGraphModel
32	uci.gef.JGraph	uci.gef.JGraph	uci.gef.Editor	setDiagram	damageAll
33	uci.gef.JGraph	uci.gef.JGraph	uci.gef.Editor	setDiagram	getSelectionManager
34	uci.gef.JGraph	uci.gef.JGraph	uci.gef.Editor	setVisible	setActiveTextEditor
35	uci.gef.JGraph	uci.gef.JGraph	uci.gef.Editor	selectByOwner	getLayerManager
36	uci.gef.JGraph	uci.gef.JGraph	uci.gef.Editor	selectByOwnerOrNoChange	getLayerManager
37	uci.gef.JGraph	uci.gef.JGraph	uci.gef.Editor	deselect	getSelectionManager
38	uci.gef.JGraph	uci.gef.JGraph	uci.gef.Editor	toggleItem	getSelectionManager
39	uci.gef.JGraph	uci.gef.JGraph	uci.gef.Editor	deselectAll	getSelectionManager
40	uci.gef.JGraph	uci.gef.JGraph	uci.gef.Editor	select	getSelectionManager
41	uci.gef.JGraph	uci.gef.JGraph	uci.gef.Editor	toggleItems	getSelectionManager
42	uci.gef.JGraph	uci.gef.JGraph	uci.gef.Editor	selectedFigs	getSelectionManager
43	uci.gef.Mode	uci.gef.ModeCreateEdge	uci.gef.FigNode	mousePressed	deepHitPort
44	uci.gef.Mode	uci.gef.ModeCreateEdge	uci.gef.FigNode	mousePressed	getPortFig
45	uci.gef.Mode	uci.gef.ModePlace	uci.gef.FigNode	mouseReleased	setEnclosingFig
46	uci.gef.Mode	uci.uml.visual.ModeCreateEdgeAndNode	uci.gef.FigNode	mouseReleased	setEnclosingFig
47	uci.gef.SelectionManager	uci.gef.SelectionManager	uci.gef.Editor	select	damaged
48	uci.gef.SelectionManager	uci.gef.SelectionManager	uci.gef.Editor	deselect	damaged
49	uci.gef.SelectionManager	uci.gef.SelectionManager	uci.gef.Editor	toggle	damaged
50	uci.gef.SelectionManager	uci.gef.SelectionManager	uci.gef.Editor	deselectAll	damaged
51	uci.uml.ui.ClassGenerationDialog	uci.uml.ui.ClassGenerationDialog	uci.uml.ui.TableModelClassChecks	actionPerformed	getChecked
52	uci.uml.ui.CriticBrowserDialog	uci.uml.ui.CriticBrowserDialog	uci.argo.kernel.Critic	setTarget	getHeadline
53	uci.uml.ui.CriticBrowserDialog	uci.uml.ui.CriticBrowserDialog	uci.argo.kernel.Critic	setTarget	getPriority
54	uci.uml.ui.CriticBrowserDialog	uci.uml.ui.CriticBrowserDialog	uci.argo.kernel.Critic	setTarget	getDescriptionTemplate
55	uci.uml.ui.CriticBrowserDialog	uci.uml.ui.CriticBrowserDialog	uci.argo.kernel.Critic	setTarget	getMoreInfoURL
56	uci.uml.ui.CriticBrowserDialog	uci.uml.ui.CriticBrowserDialog	uci.argo.kernel.Critic	setTargetHeadline	setHeadline
57	uci.uml.ui.CriticBrowserDialog	uci.uml.ui.CriticBrowserDialog	uci.argo.kernel.Critic	setTargetPriority	setPriority
58	uci.uml.ui.CriticBrowserDialog	uci.uml.ui.CriticBrowserDialog	uci.argo.kernel.Critic	setTargetMoreInfo	setMoreInfoURL
59	uci.uml.ui.CriticBrowserDialog	uci.uml.ui.CriticBrowserDialog	uci.argo.kernel.Critic	setTargetDesc	setDescription
60	uci.uml.ui.CriticBrowserDialog	uci.uml.ui.CriticBrowserDialog	uci.argo.kernel.Critic	actionPerformed	unsnooze
61	uci.uml.ui.EmailExpertDialog	uci.uml.ui.EmailExpertDialog	uci.argo.kernel.ToDoItem	setTarget	getHeadline
62	uci.uml.ui.EmailExpertDialog	uci.uml.ui.EmailExpertDialog	uci.argo.kernel.ToDoItem	setTarget	getPoster
63	uci.uml.ui.NavigatorPane	uci.uml.ui.NavigatorPane	uci.uml.ui.DisplayTextTree	setRoot	setModel

64	uci.uml.ui.ProjectBrowser	uci.uml.ui.ProjectBrowser	uci.uml.ui.DetailsPane	getDetailsTarget	getTarget
65	uci.uml.ui.ProjectBrowser	uci.uml.ui.ProjectBrowser	uci.uml.ui.DetailsPane	setDetailsTarget	setTarget
66	uci.uml.ui.ProjectBrowser	uci.uml.ui.ProjectBrowser	uci.uml.ui.DetailsPane	select	setTarget
67	uci.uml.ui.ProjectBrowser	uci.uml.ui.ProjectBrowser	uci.uml.ui.MultiEditorPane	jumpToDiagramShowing	getTarget
68	uci.uml.ui.ProjectBrowser	uci.uml.ui.ProjectBrowser	uci.uml.ui.NavigatorPane	setProject	forceUpdate
69	uci.uml.ui.ProjectBrowser	uci.uml.ui.ProjectBrowser	uci.uml.ui.NavigatorPane	setProject	setRoot
70	uci.uml.ui.ProjectBrowser	uci.uml.ui.ProjectBrowser	uci.uml.ui.Project	setTarget	setCurrentNamespace
71	uci.uml.ui.ProjectBrowser	uci.uml.ui.ProjectBrowser	uci.uml.ui.Project	setProject	getInitialTarget
72	uci.uml.ui.ProjectBrowser	uci.uml.ui.ProjectBrowser	uci.uml.ui.Project	updateTitle	getName
73	uci.uml.ui.TabDiagram	uci.uml.ui.TabDiagram	uci.gef.JGraph	setTarget	setDiagram
74	uci.uml.ui.TabDiagram	uci.uml.ui.TabDiagram	uci.ui.ToolBar	modeChange	unpressAllButtons
75	uci.uml.ui.TabResults	uci.uml.ui.TabResults	uci.uml.ui.PredicateFind	depthFirst	predicate
76	uci.uml.ui.TabResults	uci.uml.ui.TabResults	uci.uml.ui.table.TMResults	run	setTarget
77	uci.uml.ui.TabResults	uci.uml.ui.TabResults	uci.uml.ui.table.TMResults	valueChanged	setTarget
78	uci.uml.ui.TabResults	uci.uml.ui.TabResults	uci.uml.ui.table.TMResults	setResults	setTarget
79	uci.uml.ui.UpdateTreeHack	uci.uml.ui.UpdateTreeHack	uci.uml.ui.DisplayTextTree	run	forceUpdate
80	uci.uml.visual.FigClass	uci.uml.visual.FigClass	uci.uml.visual.FigCompartment	isOperationVisible	isDisplayed
81	uci.uml.visual.FigClass	uci.uml.visual.FigClass	uci.uml.visual.FigCompartment	isAttributeVisible	isDisplayed
82	uci.uml.visual.FigClass	uci.uml.visual.FigClass	uci.uml.visual.FigCompartment	setAttributeVisible	isDisplayed
83	uci.uml.visual.FigClass	uci.uml.visual.FigClass	uci.uml.visual.FigCompartment	setAttributeVisible	setDisplayed
84	uci.uml.visual.FigClass	uci.uml.visual.FigClass	uci.uml.visual.FigCompartment	setOperationVisible	isDisplayed
85	uci.uml.visual.FigClass	uci.uml.visual.FigClass	uci.uml.visual.FigCompartment	setOperationVisible	setDisplayed
86	uci.uml.visual.FigMessage	uci.uml.visual.FigMessage	uci.gef.FigPoly	setArrow	setPolygon
87	uci.uml.visual.StateDiagramGraphModel	uci.uml.visual.StateDiagramGraphModel	uci.uml.Behavioral_Elements.State_Mach	addNode	getTop
88	uci.xml.pgml.PGMLParser	uci.xml.pgml.PGMLParser	uci.gef.Diagram	handleElement	add

BRIDGE: STD

	Abstraction	Implementor	ConcreteImplementor	Operation	OperationImp
1	java.io.BufferedReader	java.io.Reader	java.io.InputStreamReader	close	close
2	java.io.BufferedWriter	java.io.Writer	java.io.OutputStreamWriter	write	write
3	java.io.BufferedWriter	java.io.Writer	java.io.OutputStreamWriter	flush	flush
4	java.io.BufferedWriter	java.io.Writer	java.io.OutputStreamWriter	close	close
5	java.io.BufferedWriter	java.io.Writer	java.io.PrintWriter	write	write
6	java.io.BufferedWriter	java.io.Writer	java.io.PrintWriter	flush	flush
7	java.io.BufferedWriter	java.io.Writer	java.io.PrintWriter	close	close
8	java.io.BufferedWriter	java.io.Writer	java.io.Writer	write	write
9	java.io.FilterInputStream	java.io.InputStream	java.io.BufferedInputStream	read	read
10	java.io.FilterInputStream	java.io.InputStream	java.io.BufferedInputStream	skip	skip
11	java.io.FilterInputStream	java.io.InputStream	java.io.BufferedInputStream	available	available
12	java.io.FilterInputStream	java.io.InputStream	java.io.BufferedInputStream	mark	mark
13	java.io.FilterInputStream	java.io.InputStream	java.io.BufferedInputStream	close	close
14	java.io.FilterInputStream	java.io.InputStream	java.io.BufferedInputStream	reset	reset
15	java.io.FilterInputStream	java.io.InputStream	java.io.BufferedInputStream	markSupported	markSupported
16	java.io.FilterInputStream	java.io.InputStream	java.io.ByteArrayInputStream	read	read
17	java.io.FilterInputStream	java.io.InputStream	java.io.ByteArrayInputStream	skip	skip
18	java.io.FilterInputStream	java.io.InputStream	java.io.ByteArrayInputStream	available	available
19	java.io.FilterInputStream	java.io.InputStream	java.io.ByteArrayInputStream	mark	mark
20	java.io.FilterInputStream	java.io.InputStream	java.io.ByteArrayInputStream	close	close
21	java.io.FilterInputStream	java.io.InputStream	java.io.ByteArrayInputStream	markSupported	markSupported
22	java.io.FilterInputStream	java.io.InputStream	java.io.ByteArrayInputStream	reset	reset
23	java.io.FilterInputStream	java.io.InputStream	java.io.DataInputStream	read	read
24	java.io.FilterInputStream	java.io.InputStream	java.io.FileInputStream	close	close
25	java.io.FilterInputStream	java.io.InputStream	java.io.FileInputStream	available	available
26	java.io.FilterInputStream	java.io.InputStream	java.io.FileInputStream	skip	skip
27	java.io.FilterInputStream	java.io.InputStream	java.io.FileInputStream	read	read
28	java.io.FilterInputStream	java.io.InputStream	java.io.InputStream	markSupported	markSupported
29	java.io.FilterInputStream	java.io.InputStream	java.io.InputStream	reset	reset
30	java.io.FilterInputStream	java.io.InputStream	java.io.InputStream	mark	mark
31	java.io.FilterInputStream	java.io.InputStream	java.io.InputStream	close	close
32	java.io.FilterInputStream	java.io.InputStream	java.io.InputStream	available	available
33	java.io.FilterInputStream	java.io.InputStream	java.io.InputStream	skip	skip
34	java.io.FilterInputStream	java.io.InputStream	java.io.InputStream	read	read
35	java.io.FilterInputStream	java.io.InputStream	java.io.ObjectInputStream	read	read
36	java.io.FilterInputStream	java.io.InputStream	java.io.ObjectInputStream	available	available
37	java.io.FilterInputStream	java.io.InputStream	java.io.ObjectInputStream	close	close
38	java.io.FilterInputStream	java.io.InputStream	java.io.PushbackInputStream	read	read

39	java.io.FilterInputStream	java.io.InputStream	java.io.PushbackInputStream	skip	skip
40	java.io.FilterInputStream	java.io.InputStream	java.io.PushbackInputStream	available	available
41	java.io.FilterInputStream	java.io.InputStream	java.io.PushbackInputStream	close	close
42	java.io.FilterInputStream	java.io.InputStream	java.io.PushbackInputStream	markSupported	markSupported
43	java.io.FilterInputStream	java.io.InputStream	java.util.jar.JarVerifier$VerifierStream	read	read
44	java.io.FilterInputStream	java.io.InputStream	java.util.jar.JarVerifier$VerifierStream	close	close
45	java.io.FilterInputStream	java.io.InputStream	java.util.jar.JarVerifier$VerifierStream	available	available
46	java.io.FilterInputStream	java.io.InputStream	java.util.jar.Manifest$FastInputStream	read	read
47	java.io.FilterInputStream	java.io.InputStream	java.util.jar.Manifest$FastInputStream	skip	skip
48	java.io.FilterInputStream	java.io.InputStream	java.util.jar.Manifest$FastInputStream	available	available
49	java.io.FilterInputStream	java.io.InputStream	java.util.jar.Manifest$FastInputStream	close	close
50	java.io.FilterInputStream	java.io.InputStream	java.util.zip.InflaterInputStream	read	read
51	java.io.FilterInputStream	java.io.InputStream	java.util.zip.InflaterInputStream	skip	skip
52	java.io.FilterInputStream	java.io.InputStream	java.util.zip.InflaterInputStream	available	available
53	java.io.FilterInputStream	java.io.InputStream	java.util.zip.InflaterInputStream	close	close
54	java.io.FilterInputStream	java.io.InputStream	java.util.zip.ZipFile$ZipFileInputStream	read	read
55	java.io.FilterInputStream	java.io.InputStream	java.util.zip.ZipFile$ZipFileInputStream	skip	skip
56	java.io.FilterInputStream	java.io.InputStream	java.util.zip.ZipFile$ZipFileInputStream	available	available
57	java.io.FilterInputStream	java.io.InputStream	java.util.zip.ZipInputStream	available	available
58	java.io.FilterInputStream	java.io.InputStream	java.util.zip.ZipInputStream	skip	skip
59	java.io.FilterInputStream	java.io.InputStream	java.util.zip.ZipInputStream	close	close
60	java.io.FilterOutputStream	java.io.OutputStream	java.io.BufferedOutputStream	write	write
61	java.io.FilterOutputStream	java.io.OutputStream	java.io.BufferedOutputStream	flush	flush
62	java.io.FilterOutputStream	java.io.OutputStream	java.io.ByteArrayOutputStream	write	write
63	java.io.FilterOutputStream	java.io.OutputStream	java.io.ByteArrayOutputStream	close	close
64	java.io.FilterOutputStream	java.io.OutputStream	java.io.DataOutputStream	write	write
65	java.io.FilterOutputStream	java.io.OutputStream	java.io.DataOutputStream	flush	flush
66	java.io.FilterOutputStream	java.io.OutputStream	java.io.FileOutputStream	write	write
67	java.io.FilterOutputStream	java.io.OutputStream	java.io.FileOutputStream	close	close
68	java.io.FilterOutputStream	java.io.OutputStream	java.io.ObjectOutputStream	close	close
69	java.io.FilterOutputStream	java.io.OutputStream	java.io.ObjectOutputStream	flush	flush
70	java.io.FilterOutputStream	java.io.OutputStream	java.io.ObjectOutputStream	write	write
71	java.io.FilterOutputStream	java.io.OutputStream	java.io.OutputStream	close	close
72	java.io.FilterOutputStream	java.io.OutputStream	java.io.OutputStream	flush	flush
73	java.io.FilterOutputStream	java.io.OutputStream	java.io.OutputStream	write	write
74	java.io.FilterOutputStream	java.io.OutputStream	java.io.PrintStream	close	close
75	java.io.FilterOutputStream	java.io.OutputStream	java.io.PrintStream	flush	flush
76	java.io.FilterOutputStream	java.io.OutputStream	java.io.PrintStream	write	write
77	java.io.FilterOutputStream	java.io.OutputStream	java.security.DigestOutputStream	write	write
78	java.io.ObjectInputStream	java.io.ObjectStreamClass	java.io.ObjectStreamClass	defaultReadObject	getFieldsNoCopy
79	java.io.ObjectOutputStream	java.io.ObjectOutputStream$PutField	java.io.ObjectOutputStream$PutFieldImpl	writeFields	write

80	java.io.ObjectOutputStream	java.io.ObjectStreamClass	java.io.ObjectStreamClass	writeObject	forClass
81	java.io.ObjectOutputStream	java.io.ObjectStreamClass	java.io.ObjectStreamClass	defaultWriteObject	forClass
82	java.io.ObjectOutputStream	java.io.ObjectStreamClass	java.io.ObjectStreamClass	defaultWriteObject	getFieldsNoCopy
83	java.io.ObjectOutputStream	java.io.ObjectStreamClass	java.io.ObjectStreamClass	writeObject	isReplaceable
84	java.io.ObjectOutputStream	java.io.OutputStream	java.io.BufferedOutputStream	flush	flush
85	java.io.ObjectOutputStream	java.io.OutputStream	java.io.ByteArrayOutputStream	close	close
86	java.io.ObjectOutputStream	java.io.OutputStream	java.io.DataOutputStream	flush	flush
87	java.io.ObjectOutputStream	java.io.OutputStream	java.io.FileOutputStream	close	close
88	java.io.ObjectOutputStream	java.io.OutputStream	java.io.FilterOutputStream	flush	flush
89	java.io.ObjectOutputStream	java.io.OutputStream	java.io.FilterOutputStream	close	close
90	java.io.ObjectOutputStream	java.io.OutputStream	java.io.OutputStream	flush	flush
91	java.io.ObjectOutputStream	java.io.OutputStream	java.io.OutputStream	close	close
92	java.io.ObjectOutputStream	java.io.OutputStream	java.io.PrintStream	flush	flush
93	java.io.ObjectOutputStream	java.io.OutputStream	java.io.PrintStream	close	close
94	java.io.ObjectOutputStream$PutFieldImpl	java.io.ObjectStreamClass	java.io.ObjectStreamClass	put	getField
95	java.io.PrintWriter	java.io.Writer	java.io.BufferedWriter	flush	flush
96	java.io.PrintWriter	java.io.Writer	java.io.BufferedWriter	close	close
97	java.io.PrintWriter	java.io.Writer	java.io.BufferedWriter	write	write
98	java.io.PrintWriter	java.io.Writer	java.io.OutputStreamWriter	flush	flush
99	java.io.PrintWriter	java.io.Writer	java.io.OutputStreamWriter	close	close
100	java.io.PrintWriter	java.io.Writer	java.io.OutputStreamWriter	write	write
101	java.io.PrintWriter	java.io.Writer	java.io.Writer	write	write
102	java.lang.ClassNotFoundException	java.lang.Throwable	java.lang.reflect.InvocationTargetExceptio	printStackTrace	printStackTrace
103	java.lang.ClassNotFoundException	java.lang.Throwable	java.lang.Throwable	printStackTrace	printStackTrace
104	java.lang.reflect.InvocationTargetExceptio	java.lang.Throwable	java.lang.ClassNotFoundException	printStackTrace	printStackTrace
105	java.lang.reflect.InvocationTargetExceptio	java.lang.Throwable	java.lang.Throwable	printStackTrace	printStackTrace
106	java.net.URL	java.net.URLStreamHandler	java.net.URLStreamHandler	toExternalForm	toExternalForm
107	java.security.DigestOutputStream	java.security.MessageDigest	java.security.MessageDigest	write	update
108	java.text.DecimalFormat	java.text.DigitList	java.text.DigitList	parse	fitsIntoLong
109	java.util.Date	java.util.Calendar	java.util.Calendar	setTime	setTimeInMillis
110	java.util.Date	java.util.Calendar	java.util.Calendar	getTimezoneOffset	setTimeInMillis
111	java.util.Date	java.util.Calendar	java.util.Calendar	getTimezoneOffset	getTimeInMillis
112	java.util.Date	java.util.Calendar	java.util.Calendar	getTimezoneOffset	setTimeZone
113	java.util.Date	java.util.Calendar	java.util.Calendar	getTimezoneOffset	getTimeZone
114	java.util.jar.Attributes$Name	java.lang.String	java.lang.String	equals	equalsIgnoreCase
115	java.util.jar.Attributes$Name	java.lang.String	java.lang.String	hashCode	toLowerCase
116	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.BufferedInputStream	read	read
117	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.BufferedInputStream	close	close
118	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.BufferedInputStream	available	available
119	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.ByteArrayInputStream	read	read
120	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.ByteArrayInputStream	close	close

121	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.ByteArrayInputStream	available	available
122	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.DataInputStream	read	read
123	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.FileInputStream	read	read
124	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.FileInputStream	close	close
125	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.FileInputStream	available	available
126	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.FilterInputStream	read	read
127	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.FilterInputStream	close	close
128	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.FilterInputStream	available	available
129	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.InputStream	read	read
130	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.InputStream	close	close
131	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.InputStream	available	available
132	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.ObjectInputStream	read	read
133	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.ObjectInputStream	close	close
134	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.ObjectInputStream	available	available
135	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.PushbackInputStream	read	read
136	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.PushbackInputStream	close	close
137	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.io.PushbackInputStream	available	available
138	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.util.jar.Manifest$FastInputStream	read	read
139	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.util.jar.Manifest$FastInputStream	close	close
140	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.util.jar.Manifest$FastInputStream	available	available
141	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.util.zip.InflaterInputStream	read	read
142	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.util.zip.InflaterInputStream	available	available
143	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.util.zip.InflaterInputStream	close	close
144	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.util.zip.ZipFile$ZipFileInputStream	read	read
145	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.util.zip.ZipFile$ZipFileInputStream	available	available
146	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.util.zip.ZipInputStream	available	available
147	java.util.jar.JarVerifier$VerifierStream	java.io.InputStream	java.util.zip.ZipInputStream	close	close

BRIDGE: SELF

	Abstraction	Implementor	ConcreteImplementor	Operation	OperationImp
1	reeng.java.ClassFileIntrospector	reeng.java.JavaReader	reeng.java.JavaReader	read	makeClassifier
2	reeng.java.ClassFileIntrospector	reeng.java.JavaReader	reeng.java.JavaReader	read	shouldRead
3	reeng.java.ImportManager	reeng.java.ASTUMLVisitorBase	reeng.java.ASTUMLVisitorBase	lookup	getEnclosingName
4	reeng.RelationshipBuilderVisitor	reeng.GenericReader	reeng.GenericReader	visitAttribute	getType

BRIDGE: ARGO

	Abstraction	Implementor	ConcreteImplementor	Operation	OperationImp
1	uci.argo.kernel.Agency	uci.argo.kernel.ControlMech	uci.argo.kernel.AndCM	determineActiveCritics	isRelevant
2	uci.argo.kernel.Agency	uci.argo.kernel.ControlMech	uci.argo.kernel.ControlMech	determineActiveCritics	isRelevant
3	uci.argo.kernel.Agency	uci.argo.kernel.ControlMech	uci.argo.kernel.CurDecisionCM	determineActiveCritics	isRelevant
4	uci.argo.kernel.Agency	uci.argo.kernel.ControlMech	uci.argo.kernel.DesignGoalsCM	determineActiveCritics	isRelevant
5	uci.argo.kernel.Agency	uci.argo.kernel.ControlMech	uci.argo.kernel.EnabledCM	determineActiveCritics	isRelevant
6	uci.argo.kernel.Agency	uci.argo.kernel.ControlMech	uci.argo.kernel.NotSnoozedCM	determineActiveCritics	isRelevant
7	uci.argo.kernel.Agency	uci.argo.kernel.ControlMech	uci.argo.kernel.OrCM	determineActiveCritics	isRelevant
8	uci.argo.kernel.DesignMaterial	uci.argo.kernel.ToDoList	uci.argo.kernel.ToDoList	inform	addElement
9	uci.argo.kernel.DesignMaterial	uci.argo.kernel.ToDoList	uci.argo.kernel.ToDoList	removePendingItems	removeAllElements
10	uci.argo.kernel.Wizard	uci.argo.kernel.ToDoItem	uci.argo.kernel.ToDoItem	next	changed
11	uci.argo.kernel.Wizard	uci.argo.kernel.ToDoItem	uci.argo.kernel.ToDoItem	back	changed
12	uci.argo.kernel.Wizard	uci.argo.kernel.ToDoItem	uci.argo.kernel.ToDoItem	finish	changed
13	uci.gef.Fig	uci.gef.Fig	uci.gef.FigEdge	propertyChange	propertyChange
14	uci.gef.Fig	uci.gef.Fig	uci.gef.FigNode	propertyChange	propertyChange
15	uci.gef.Fig	uci.gef.Fig	uci.uml.visual.FigEdgeModelElement	propertyChange	propertyChange
16	uci.gef.Fig	uci.gef.Fig	uci.uml.visual.FigNodeModelElement	propertyChange	propertyChange
17	uci.gef.Fig	uci.gef.Layer	uci.gef.Layer	setEnclosingFig	bringInFrontOf
18	uci.gef.Fig	uci.gef.Layer	uci.gef.Layer	delete	deleted
19	uci.gef.Fig	uci.gef.Layer	uci.gef.Layer	damage	damaged
20	uci.gef.Fig	uci.gef.Layer	uci.gef.LayerDiagram	setEnclosingFig	bringInFrontOf
21	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	setFig	setGroup
22	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	hitFig	hit
23	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	getBounds	getBounds
24	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	calcBounds	getBounds
25	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	calcBounds	calcBounds
26	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	contains	contains
27	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	intersects	intersects
28	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	getPerimeterLength	getPerimeterLength
29	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	hit	hit
30	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	stuffPointAlongPerimeter	stuffPointAlongPerimeter
31	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	setLineColor	setLineColor
32	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	getLineColor	getLineColor
33	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	setPoints	setPoints
34	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	setLineWidth	setLineWidth
35	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	getFirstPoint	getFirstPoint
36	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	getLastPoint	getLastPoint
37	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	getLineWidth	getLineWidth
38	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	setDashed	setDashed

39	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	getPoints	getPoints
40	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	getDashed	getDashed
41	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	getNumPoints	getNumPoints
42	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	translateEdge	translate
43	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	setNumPoints	setNumPoints
44	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	getXs	getXs
45	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	setXs	setXs
46	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	getYs	getYs
47	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	setYs	setYs
48	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	isResizable	isResizable
49	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	isReshapable	isReshapable
50	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	isRotatable	isRotatable
51	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	paint	paint
52	uci.gef.FigEdge	uci.gef.Fig	uci.gef.Fig	cleanUp	cleanUp
53	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigCircle	contains	contains
54	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigCircle	paint	paint
55	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigEdgePoly	setPoints	setPoints
56	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigEdgePoly	paint	paint
57	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigGroup	hitFig	hit
58	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigGroup	setLineColor	setLineColor
59	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigGroup	getLineColor	getLineColor
60	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigGroup	isResizable	isResizable
61	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigGroup	isReshapable	isReshapable
62	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigGroup	calcBounds	calcBounds
63	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigGroup	setLineWidth	setLineWidth
64	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigGroup	contains	contains
65	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigGroup	isRotatable	isRotatable
66	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigGroup	getLineWidth	getLineWidth
67	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigGroup	hit	hit
68	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigGroup	translateEdge	translate
69	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigGroup	paint	paint
70	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigInk	contains	contains
71	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigInk	setLineWidth	setLineWidth
72	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigLine	hitFig	hit
73	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigLine	calcBounds	calcBounds
74	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigLine	setPoints	setPoints
75	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigLine	getPerimeterLength	getPerimeterLength
76	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigLine	stuffPointAlongPerimeter	stuffPointAlongPerimeter
77	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigLine	getPoints	getPoints
78	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigLine	getNumPoints	getNumPoints
79	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigLine	isResizable	isResizable

80	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigLine	intersects	intersects
81	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigLine	isReshapable	isReshapable
82	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigLine	getXs	getXs
83	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigLine	isRotatable	isRotatable
84	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigLine	hit	hit
85	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigLine	getYs	getYs
86	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigLine	translateEdge	translate
87	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigLine	paint	paint
88	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigNode	hitFig	hit
89	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigNode	contains	contains
90	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigNode	hit	hit
91	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigNode	translateEdge	translate
92	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigNode	paint	paint
93	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigNode	cleanUp	cleanUp
94	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigPoly	hitFig	hit
95	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigPoly	calcBounds	calcBounds
96	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigPoly	contains	contains
97	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigPoly	hit	hit
98	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigPoly	getNumPoints	getNumPoints
99	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigPoly	getPerimeterLength	getPerimeterLength
100	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigPoly	stuffPointAlongPerimeter	stuffPointAlongPerimeter
101	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigPoly	setPoints	setPoints
102	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigPoly	getFirstPoint	getFirstPoint
103	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigPoly	getLastPoint	getLastPoint
104	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigPoly	translateEdge	translate
105	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigPoly	getPoints	getPoints
106	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigPoly	getXs	getXs
107	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigPoly	getYs	getYs
108	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigPoly	isResizable	isResizable
109	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigPoly	isReshapable	isReshapable
110	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigPoly	isRotatable	isRotatable
111	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigPoly	paint	paint
112	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigPoly	cleanUp	cleanUp
113	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigRect	paint	paint
114	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigRRect	paint	paint
115	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigSpline	translateEdge	translate
116	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigSpline	paint	paint
117	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigSpline	cleanUp	cleanUp
118	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigText	hitFig	hit
119	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigText	calcBounds	calcBounds
120	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigText	hit	hit

121	uci.gef.FigEdge	uci.gef.Fig	uci.gef.FigText	paint	paint
122	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigActionState	setLineColor	setLineColor
123	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigActionState	getLineColor	getLineColor
124	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigActionState	setLineWidth	setLineWidth
125	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigActionState	getLineWidth	getLineWidth
126	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigActor	setLineColor	setLineColor
127	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigActor	getLineColor	getLineColor
128	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigActor	setLineWidth	setLineWidth
129	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigActor	getLineWidth	getLineWidth
130	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigActor	isResizable	isResizable
131	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigBranchState	setLineColor	setLineColor
132	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigBranchState	getLineColor	getLineColor
133	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigBranchState	setLineWidth	setLineWidth
134	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigBranchState	getLineWidth	getLineWidth
135	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigBranchState	isResizable	isResizable
136	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigClass	translateEdge	translate
137	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigClassifierRole	setLineColor	setLineColor
138	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigClassifierRole	getLineColor	getLineColor
139	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigClassifierRole	setLineWidth	setLineWidth
140	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigClassifierRole	getLineWidth	getLineWidth
141	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigCompartment	paint	paint
142	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigCompositeState	setLineColor	setLineColor
143	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigCompositeState	getLineColor	getLineColor
144	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigCompositeState	setLineWidth	setLineWidth
145	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigCompositeState	getLineWidth	getLineWidth
146	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigFinalState	setLineColor	setLineColor
147	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigFinalState	getLineColor	getLineColor
148	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigFinalState	setLineWidth	setLineWidth
149	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigFinalState	getLineWidth	getLineWidth
150	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigFinalState	isResizable	isResizable
151	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigForkState	setLineColor	setLineColor
152	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigForkState	getLineColor	getLineColor
153	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigForkState	setLineWidth	setLineWidth
154	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigForkState	getLineWidth	getLineWidth
155	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigHistoryState	setLineColor	setLineColor
156	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigHistoryState	getLineColor	getLineColor
157	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigHistoryState	setLineWidth	setLineWidth
158	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigHistoryState	getLineWidth	getLineWidth
159	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigHistoryState	isResizable	isResizable
160	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigInitialState	setLineColor	setLineColor
161	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigInitialState	getLineColor	getLineColor

162	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigInitialState	setLineWidth	setLineWidth
163	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigInitialState	getLineWidth	getLineWidth
164	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigInitialState	isResizable	isResizable
165	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigInterface	setLineColor	setLineColor
166	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigInterface	setLineWidth	setLineWidth
167	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigJoinState	setLineColor	setLineColor
168	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigJoinState	getLineColor	getLineColor
169	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigJoinState	setLineWidth	setLineWidth
170	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigJoinState	getLineWidth	getLineWidth
171	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigMessage	setLineColor	setLineColor
172	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigMessage	getLineColor	getLineColor
173	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigMessage	setLineWidth	setLineWidth
174	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigMessage	getLineWidth	getLineWidth
175	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigNodeWithCompartments	calcBounds	calcBounds
176	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigPackage	setLineColor	setLineColor
177	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigPackage	getLineColor	getLineColor
178	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigPackage	setLineWidth	setLineWidth
179	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigPackage	getLineWidth	getLineWidth
180	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigState	setLineColor	setLineColor
181	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigState	getLineColor	getLineColor
182	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigState	setLineWidth	setLineWidth
183	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigState	getLineWidth	getLineWidth
184	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigUseCase	setLineColor	setLineColor
185	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigUseCase	getLineColor	getLineColor
186	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigUseCase	setLineWidth	setLineWidth
187	uci.gef.FigEdge	uci.gef.Fig	uci.uml.visual.FigUseCase	getLineWidth	getLineWidth
188	uci.gef.FigEdge	uci.gef.FigNode	uci.gef.FigNode	setSourceFigNode	removeFigEdge
189	uci.gef.FigEdge	uci.gef.FigNode	uci.gef.FigNode	setDestFigNode	removeFigEdge
190	uci.gef.FigEdge	uci.gef.FigNode	uci.gef.FigNode	delete	removeFigEdge
191	uci.gef.NetEdge	uci.gef.NetPort	uci.gef.NetPort	dispose	removeEdge
192	uci.gef.NetEdge	uci.gef.NetPort	uci.gef.NetPort	presentationFor	getParentNode
193	uci.gef.NetEdge	uci.gef.NetPort	uci.gef.NetPort	dispose	postDisconnect
194	uci.uml.ui.CriticBrowserDialog	uci.argo.kernel.Critic	uci.argo.kernel.Critic	setTarget	getPriority
195	uci.uml.ui.CriticBrowserDialog	uci.argo.kernel.Critic	uci.argo.kernel.Critic	setTarget	getDescriptionTemplate
196	uci.uml.ui.CriticBrowserDialog	uci.argo.kernel.Critic	uci.argo.kernel.Critic	setTarget	getMoreInfoURL
197	uci.uml.ui.CriticBrowserDialog	uci.argo.kernel.Critic	uci.argo.kernel.Critic	setTargetHeadline	setHeadline
198	uci.uml.ui.CriticBrowserDialog	uci.argo.kernel.Critic	uci.argo.kernel.Critic	setTargetPriority	setPriority
199	uci.uml.ui.CriticBrowserDialog	uci.argo.kernel.Critic	uci.argo.kernel.Critic	actionPerformed	unsnooze
200	uci.uml.ui.CriticBrowserDialog	uci.argo.kernel.Critic	uci.argo.kernel.Critic	setTargetMoreInfo	setMoreInfoURL
201	uci.uml.ui.CriticBrowserDialog	uci.argo.kernel.Critic	uci.argo.kernel.Critic	setTargetDesc	setDescription
202	uci.uml.ui.CriticBrowserDialog	uci.argo.kernel.Critic	uci.uml.critics.CrUML	setTarget	getMoreInfoURL

203	uci.uml.ui.EmailExpertDialog	uci.argo.kernel.ToDoItem	uci.argo.kernel.ToDoItem	setTarget	getHeadline
204	uci.uml.ui.EmailExpertDialog	uci.argo.kernel.ToDoItem	uci.argo.kernel.ToDoItem	setTarget	getPoster
205	uci.uml.ui.TabDiagram	uci.ui.ToolBar	uci.ui.ToolBar	modeChange	unpressAllButtons
206	uci.uml.ui.TabResults	uci.uml.ui.PredicateFind	uci.uml.ui.PredicateFind	depthFirst	predicate
207	uci.uml.ui.TabResults	uci.uml.ui.table.TMResults	uci.uml.ui.table.TMResults	setResults	setTarget
208	uci.uml.ui.TabResults	uci.uml.ui.table.TMResults	uci.uml.ui.table.TMResults	valueChanged	setTarget
209	uci.uml.ui.TabResults	uci.uml.ui.table.TMResults	uci.uml.ui.table.TMResults	run	setTarget
210	uci.uml.ui.UpdateTreeHack	uci.uml.ui.DisplayTextTree	uci.uml.ui.DisplayTextTree	run	forceUpdate
211	uci.uml.visual.ModeCreateEdgeAndNode	uci.gef.FigEdge	uci.gef.FigEdge	mouseReleased	setFig
212	uci.uml.visual.ModeCreateEdgeAndNode	uci.gef.FigEdge	uci.gef.FigEdge	mouseReleased	setSourcePortFig
213	uci.uml.visual.ModeCreateEdgeAndNode	uci.gef.FigEdge	uci.gef.FigEdge	mouseReleased	setDestPortFig
214	uci.uml.visual.ModeCreateEdgeAndNode	uci.gef.FigEdge	uci.gef.FigEdge	mouseReleased	setSourceFigNode
215	uci.uml.visual.ModeCreateEdgeAndNode	uci.gef.FigEdge	uci.gef.FigEdge	mouseReleased	setDestFigNode
216	uci.uml.visual.ModeCreateEdgeAndNode	uci.gef.FigEdge	uci.uml.visual.FigDependency	mouseReleased	setFig
217	uci.uml.visual.ModeCreateEdgeAndNode	uci.gef.FigEdge	uci.uml.visual.FigRealization	mouseReleased	setFig

FACTORY METHOD: STD

	AbstractCreator	Creator	AbstractProduct	Product	FactoryMethod
1	java.text.NumberFormat	java.text.ChoiceFormat	java.lang.Number	java.lang.Double	parse
2	java.text.NumberFormat	java.text.DecimalFormat	java.lang.Number	java.lang.Double	parse
3	java.util.AbstractCollection	java.util.AbstractList	java.util.Iterator	java.util.AbstractList$Itr	iterator
4	java.util.AbstractCollection	java.util.Hashtable$EntrySet	java.util.Iterator	java.util.Hashtable$Enumerator	iterator
5	java.util.AbstractCollection	java.util.Hashtable$KeySet	java.util.Iterator	java.util.Hashtable$Enumerator	iterator
6	java.util.AbstractCollection	java.util.Hashtable$ValueCollection	java.util.Iterator	java.util.Hashtable$Enumerator	iterator

FACTORY METHOD: SELF

	AbstractCreator	Creator	AbstractProduct	Product	FactoryMethod

FACTORY METHOD: ARGO

	AbstractCreator	Creator	AbstractProduct	Product	FactoryMethod
1	uci.argo.kernel.Wizard	uci.uml.critics.WizMEName	javax.swing.JPanel	uci.uml.ui.todo.WizStepTextField	makePanel
2	uci.gef.FigEdge	uci.gef.FigEdgePoly	uci.gef.Fig	uci.gef.FigPoly	makeEdgeFig
3	uci.gef.ModeCreate	uci.gef.ModeCreateEdge	uci.gef.Fig	uci.gef.FigLine	createNewItem
4	uci.gef.ModeCreate	uci.gef.ModeCreateFigCircle	uci.gef.Fig	uci.gef.FigCircle	createNewItem
5	uci.gef.ModeCreate	uci.gef.ModeCreateFigInk	uci.gef.Fig	uci.gef.FigInk	createNewItem
6	uci.gef.ModeCreate	uci.gef.ModeCreateFigLine	uci.gef.Fig	uci.gef.FigLine	createNewItem
7	uci.gef.ModeCreate	uci.gef.ModeCreateFigPoly	uci.gef.Fig	uci.gef.FigPoly	createNewItem
8	uci.gef.ModeCreate	uci.gef.ModeCreateFigRect	uci.gef.Fig	uci.gef.FigRect	createNewItem
9	uci.gef.ModeCreate	uci.gef.ModeCreateFigRRect	uci.gef.Fig	uci.gef.FigRRect	createNewItem
10	uci.gef.ModeCreate	uci.gef.ModeCreateFigSpline	uci.gef.Fig	uci.gef.FigSpline	createNewItem
11	uci.gef.ModeCreate	uci.gef.ModeCreateFigText	uci.gef.Fig	uci.gef.FigText	createNewItem
12	uci.gef.ModeCreate	uci.uml.visual.ModeCreateEdgeAndNode	uci.gef.Fig	uci.gef.FigPoly	createNewItem

PROTOTYPE: STD

	Prototype	ConcretePrototype
1	java.lang.Cloneable	java.io.ObjectOutputStream$Stack
2	java.lang.Cloneable	java.security.Provider
3	java.lang.Cloneable	java.text.ChoiceFormat
4	java.lang.Cloneable	java.text.DateFormat
5	java.lang.Cloneable	java.text.DateFormatSymbols
6	java.lang.Cloneable	java.text.DecimalFormat
7	java.lang.Cloneable	java.text.DecimalFormatSymbols
8	java.lang.Cloneable	java.text.DigitList
9	java.lang.Cloneable	java.text.Format
10	java.lang.Cloneable	java.text.MessageFormat
11	java.lang.Cloneable	java.text.NumberFormat
12	java.lang.Cloneable	java.text.SimpleDateFormat
13	java.lang.Cloneable	java.util.ArrayList
14	java.lang.Cloneable	java.util.Calendar
15	java.lang.Cloneable	java.util.Date
16	java.lang.Cloneable	java.util.GregorianCalendar
17	java.lang.Cloneable	java.util.HashMap
18	java.lang.Cloneable	java.util.Hashtable
19	java.lang.Cloneable	java.util.jar.Attributes
20	java.lang.Cloneable	java.util.jar.JarEntry
21	java.lang.Cloneable	java.util.jar.JarFile$JarFileEntry
22	java.lang.Cloneable	java.util.jar.Manifest
23	java.lang.Cloneable	java.util.Locale
24	java.lang.Cloneable	java.util.Properties
25	java.lang.Cloneable	java.util.ResourceBundle$ResourceCacheKey
26	java.lang.Cloneable	java.util.SimpleTimeZone
27	java.lang.Cloneable	java.util.Stack
28	java.lang.Cloneable	java.util.TimeZone
29	java.lang.Cloneable	java.util.Vector
30	java.lang.Cloneable	java.util.zip.ZipEntry

PROTOTYPE: SELF

	Prototype	ConcretePrototype
1	java.lang.Cloneable	java.util.Stack
2	java.lang.Cloneable	java.util.Vector
3	java.lang.Cloneable	jess.Fact
4	java.lang.Cloneable	jess.ValueVector
5	java.lang.Cloneable	reeng.jess.ReengFact

PROTOTYPE: ARGO

	Prototype	ConcretePrototype
1	java.lang.Cloneable	com.ibm.xml.parser.Child
2	java.lang.Cloneable	com.ibm.xml.parser.Parent
3	java.lang.Cloneable	com.ibm.xml.parser.TXElement
4	java.lang.Cloneable	java.awt.Dimension
5	java.lang.Cloneable	java.awt.geom.Dimension2D
6	java.lang.Cloneable	java.awt.geom.Point2D
7	java.lang.Cloneable	java.awt.geom.Rectangle2D
8	java.lang.Cloneable	java.awt.geom.RectangularShape
9	java.lang.Cloneable	java.awt.image.ImageFilter
10	java.lang.Cloneable	java.awt.image.RGBImageFilter
11	java.lang.Cloneable	java.awt.Point
12	java.lang.Cloneable	java.awt.Rectangle
13	java.lang.Cloneable	javax.swing.AbstractAction
14	java.lang.Cloneable	uci.gef.Cmd
15	java.lang.Cloneable	uci.gef.CmdAdjustGrid
16	java.lang.Cloneable	uci.gef.CmdAdjustGuide
17	java.lang.Cloneable	uci.gef.CmdAdjustPageBreaks
18	java.lang.Cloneable	uci.gef.CmdAlign
19	java.lang.Cloneable	uci.gef.CmdCopy
20	java.lang.Cloneable	uci.gef.CmdDistribute
21	java.lang.Cloneable	uci.gef.CmdGroup
22	java.lang.Cloneable	uci.gef.CmdInsertPoint
23	java.lang.Cloneable	uci.gef.CmdNudge
24	java.lang.Cloneable	uci.gef.CmdPaste
25	java.lang.Cloneable	uci.gef.CmdPrint
26	java.lang.Cloneable	uci.gef.CmdRemovePoint
27	java.lang.Cloneable	uci.gef.CmdReorder
28	java.lang.Cloneable	uci.gef.CmdSaveGIF
29	java.lang.Cloneable	uci.gef.CmdSelectAll
30	java.lang.Cloneable	uci.gef.CmdSelectInvert
31	java.lang.Cloneable	uci.gef.CmdSelectNear
32	java.lang.Cloneable	uci.gef.CmdSelectNext
33	java.lang.Cloneable	uci.gef.CmdSetMode
34	java.lang.Cloneable	uci.gef.CmdUngroup
35	java.lang.Cloneable	uci.gef.Fig
36	java.lang.Cloneable	uci.gef.FigCircle
37	java.lang.Cloneable	uci.gef.FigEdge
38	java.lang.Cloneable	uci.gef.FigEdgePoly
39	java.lang.Cloneable	uci.gef.FigGroup
40	java.lang.Cloneable	uci.gef.FigInk
41	java.lang.Cloneable	uci.gef.FigLine
42	java.lang.Cloneable	uci.gef.FigNode
43	java.lang.Cloneable	uci.gef.FigPoly
44	java.lang.Cloneable	uci.gef.FigRect
45	java.lang.Cloneable	uci.gef.FigRRect
46	java.lang.Cloneable	uci.gef.FigSpline
47	java.lang.Cloneable	uci.gef.FigText
48	java.lang.Cloneable	uci.gef.JGraph
49	java.lang.Cloneable	uci.gef.TransFilter
50	java.lang.Cloneable	uci.uml.ui.ActionAboutArgoUML
51	java.lang.Cloneable	uci.uml.ui.ActionActivityDiagram
52	java.lang.Cloneable	uci.uml.ui.ActionAddAttribute
53	java.lang.Cloneable	uci.uml.ui.ActionAddInternalTrans
54	java.lang.Cloneable	uci.uml.ui.ActionAddMessage
55	java.lang.Cloneable	uci.uml.ui.ActionAddOperation
56	java.lang.Cloneable	uci.uml.ui.ActionAddTopLevelPackage
57	java.lang.Cloneable	uci.uml.ui.ActionAggregation
58	java.lang.Cloneable	uci.uml.ui.ActionAutoCritique
59	java.lang.Cloneable	uci.uml.ui.ActionClassDiagram
60	java.lang.Cloneable	uci.uml.ui.ActionCollaborationDiagram
61	java.lang.Cloneable	uci.uml.ui.ActionCompartmentDisplay
62	java.lang.Cloneable	uci.uml.ui.ActionCopy
63	java.lang.Cloneable	uci.uml.ui.ActionCreateMultiple
64	java.lang.Cloneable	uci.uml.ui.ActionCut
65	java.lang.Cloneable	uci.uml.ui.ActionDeleteFromDiagram
66	java.lang.Cloneable	uci.uml.ui.ActionEmailExpert
67	java.lang.Cloneable	uci.uml.ui.ActionEmptyTrash
68	java.lang.Cloneable	uci.uml.ui.ActionExit
69	java.lang.Cloneable	uci.uml.ui.ActionFind
70	java.lang.Cloneable	uci.uml.ui.ActionFlatToDo
71	java.lang.Cloneable	uci.uml.ui.ActionGenerateAll
72	java.lang.Cloneable	uci.uml.ui.ActionGenerateOne
73	java.lang.Cloneable	uci.uml.ui.ActionGoToCritique
74	java.lang.Cloneable	uci.uml.ui.ActionGoToDetails
75	java.lang.Cloneable	uci.uml.ui.ActionGotoDiagram
76	java.lang.Cloneable	uci.uml.ui.ActionGoToEdit

77	java.lang.Cloneable	uci.uml.ui.ActionMoreInfo
78	java.lang.Cloneable	uci.uml.ui.ActionMultiplicity
79	java.lang.Cloneable	uci.uml.ui.ActionNavBack
80	java.lang.Cloneable	uci.uml.ui.ActionNavConfig
81	java.lang.Cloneable	uci.uml.ui.ActionNavForw
82	java.lang.Cloneable	uci.uml.ui.ActionNew
83	java.lang.Cloneable	uci.uml.ui.ActionNewToDoItem
84	java.lang.Cloneable	uci.uml.ui.ActionNextDetailsTab
85	java.lang.Cloneable	uci.uml.ui.ActionNextEditTab
86	java.lang.Cloneable	uci.uml.ui.ActionOpenCritics
87	java.lang.Cloneable	uci.uml.ui.ActionOpenDecisions
88	java.lang.Cloneable	uci.uml.ui.ActionOpenGoals
89	java.lang.Cloneable	uci.uml.ui.ActionOpenProject
90	java.lang.Cloneable	uci.uml.ui.ActionPaste
91	java.lang.Cloneable	uci.uml.ui.ActionPrint
92	java.lang.Cloneable	uci.uml.ui.ActionProperties
93	java.lang.Cloneable	uci.uml.ui.ActionRedo
94	java.lang.Cloneable	uci.uml.ui.ActionRemoveFromModel
95	java.lang.Cloneable	uci.uml.ui.ActionResolve
96	java.lang.Cloneable	uci.uml.ui.ActionSaveGIF
97	java.lang.Cloneable	uci.uml.ui.ActionSaveProject
98	java.lang.Cloneable	uci.uml.ui.ActionSaveProjectAs
99	java.lang.Cloneable	uci.uml.ui.ActionShowRapidButtons
100	java.lang.Cloneable	uci.uml.ui.ActionSnooze
101	java.lang.Cloneable	uci.uml.ui.ActionStateDiagram
102	java.lang.Cloneable	uci.uml.ui.ActionUndo
103	java.lang.Cloneable	uci.uml.ui.ActionUseCaseDiagram
104	java.lang.Cloneable	uci.uml.ui.nav.NavPerspective
105	java.lang.Cloneable	uci.uml.ui.nav.TreeModelComposite
106	java.lang.Cloneable	uci.uml.ui.props.PropPanel
107	java.lang.Cloneable	uci.uml.ui.props.PropPanelAssociation
108	java.lang.Cloneable	uci.uml.ui.props.PropPanelClass
109	java.lang.Cloneable	uci.uml.ui.props.PropPanelInterface
110	java.lang.Cloneable	uci.uml.ui.props.PropPanelTwoEnds
111	java.lang.Cloneable	uci.uml.ui.style.SPFigEdgeModelElement
112	java.lang.Cloneable	uci.uml.ui.style.StylePanel
113	java.lang.Cloneable	uci.uml.ui.style.StylePanelFig
114	java.lang.Cloneable	uci.uml.ui.style.StylePanelFigClass
115	java.lang.Cloneable	uci.uml.ui.TabDiagram
116	java.lang.Cloneable	uci.uml.ui.TabResults
117	java.lang.Cloneable	uci.uml.ui.TabSpawnable
118	java.lang.Cloneable	uci.uml.ui.TabToDo
119	java.lang.Cloneable	uci.uml.ui.todo.ToDoPerspective
120	java.lang.Cloneable	uci.uml.ui.ToDoItemAction
121	java.lang.Cloneable	uci.uml.ui.UMLAction
122	java.lang.Cloneable	uci.uml.ui.UMLChangeAction
123	java.lang.Cloneable	uci.uml.visual.FigActionState
124	java.lang.Cloneable	uci.uml.visual.FigActor
125	java.lang.Cloneable	uci.uml.visual.FigAssociation
126	java.lang.Cloneable	uci.uml.visual.FigAssociationRole
127	java.lang.Cloneable	uci.uml.visual.FigBranchState
128	java.lang.Cloneable	uci.uml.visual.FigClass
129	java.lang.Cloneable	uci.uml.visual.FigClassifierRole
130	java.lang.Cloneable	uci.uml.visual.FigCompartment
131	java.lang.Cloneable	uci.uml.visual.FigCompositeState
132	java.lang.Cloneable	uci.uml.visual.FigDependency
133	java.lang.Cloneable	uci.uml.visual.FigEdgeModelElement
134	java.lang.Cloneable	uci.uml.visual.FigFinalState
135	java.lang.Cloneable	uci.uml.visual.FigForkState
136	java.lang.Cloneable	uci.uml.visual.FigGeneralization
137	java.lang.Cloneable	uci.uml.visual.FigHistoryState
138	java.lang.Cloneable	uci.uml.visual.FigInitialState
139	java.lang.Cloneable	uci.uml.visual.FigInstance
140	java.lang.Cloneable	uci.uml.visual.FigInterface
141	java.lang.Cloneable	uci.uml.visual.FigJoinState
142	java.lang.Cloneable	uci.uml.visual.FigLink
143	java.lang.Cloneable	uci.uml.visual.FigMessage
144	java.lang.Cloneable	uci.uml.visual.FigNodeModelElement
145	java.lang.Cloneable	uci.uml.visual.FigNodeWithCompartments
146	java.lang.Cloneable	uci.uml.visual.FigPackage
147	java.lang.Cloneable	uci.uml.visual.FigRealization
148	java.lang.Cloneable	uci.uml.visual.FigState
149	java.lang.Cloneable	uci.uml.visual.FigStateVertex
150	java.lang.Cloneable	uci.uml.visual.FigTransition
151	java.lang.Cloneable	uci.uml.visual.FigUseCase

PROXY: STD

	Proxy	Subject	RealSubject	Request
1	java.io.InputStreamReader	java.io.InputStream	java.io.BufferedInputStream	close
2	java.io.InputStreamReader	java.io.InputStream	java.io.ByteArrayInputStream	close
3	java.io.InputStreamReader	java.io.InputStream	java.io.FileInputStream	close
4	java.io.InputStreamReader	java.io.InputStream	java.io.FilterInputStream	close
5	java.io.InputStreamReader	java.io.InputStream	java.io.InputStream	close
6	java.io.InputStreamReader	java.io.InputStream	java.io.ObjectInputStream	close
7	java.io.InputStreamReader	java.io.InputStream	java.io.PushbackInputStream	close
8	java.io.InputStreamReader	java.io.InputStream	java.util.jar.JarVerifier$VerifierStream	close
9	java.io.InputStreamReader	java.io.InputStream	java.util.jar.Manifest$FastInputStream	close
10	java.io.InputStreamReader	java.io.InputStream	java.util.zip.InflaterInputStream	close
11	java.io.InputStreamReader	java.io.InputStream	java.util.zip.ZipInputStream	close
12	java.io.ObjectInputStream	java.io.DataInputStream	java.io.DataInputStream	readBoolean
13	java.io.ObjectInputStream	java.io.DataInputStream	java.io.DataInputStream	readByte
14	java.io.ObjectInputStream	java.io.DataInputStream	java.io.DataInputStream	readUnsignedByte
15	java.io.ObjectInputStream	java.io.DataInputStream	java.io.DataInputStream	readShort
16	java.io.ObjectInputStream	java.io.DataInputStream	java.io.DataInputStream	readUnsignedShort
17	java.io.ObjectInputStream	java.io.DataInputStream	java.io.DataInputStream	readChar
18	java.io.ObjectInputStream	java.io.DataInputStream	java.io.DataInputStream	readInt
19	java.io.ObjectInputStream	java.io.DataInputStream	java.io.DataInputStream	readLong
20	java.io.ObjectInputStream	java.io.DataInputStream	java.io.DataInputStream	readFully
21	java.io.ObjectInputStream	java.io.DataInputStream	java.io.DataInputStream	readFloat
22	java.io.ObjectInputStream	java.io.DataInputStream	java.io.DataInputStream	readDouble
23	java.io.ObjectInputStream	java.io.DataInputStream	java.io.DataInputStream	skipBytes
24	java.io.ObjectInputStream	java.io.DataInputStream	java.io.DataInputStream	readLine
25	java.io.ObjectInputStream	java.io.DataInputStream	java.io.DataInputStream	readUTF
26	java.io.ObjectInputStream	java.io.InputStream	java.io.BufferedInputStream	read
27	java.io.ObjectInputStream	java.io.InputStream	java.io.ByteArrayInputStream	read
28	java.io.ObjectInputStream	java.io.InputStream	java.io.DataInputStream	read
29	java.io.ObjectInputStream	java.io.InputStream	java.io.FileInputStream	read
30	java.io.ObjectInputStream	java.io.InputStream	java.io.FilterInputStream	read
31	java.io.ObjectInputStream	java.io.InputStream	java.io.InputStream	read
32	java.io.ObjectInputStream	java.io.InputStream	java.io.PushbackInputStream	read
33	java.io.ObjectInputStream	java.io.InputStream	java.util.jar.JarVerifier$VerifierStream	read
34	java.io.ObjectInputStream	java.io.InputStream	java.util.jar.Manifest$FastInputStream	read
35	java.io.ObjectInputStream	java.io.InputStream	java.util.zip.InflaterInputStream	read
36	java.io.ObjectInputStream	java.io.InputStream	java.util.zip.ZipFile$ZipFileInputStream	read
37	java.io.ObjectOutputStream	java.io.DataOutputStream	java.io.DataOutputStream	writeShort
38	java.io.ObjectOutputStream	java.io.DataOutputStream	java.io.DataOutputStream	writeChar

39	java.io.ObjectOutputStream	java.io.DataOutputStream	java.io.DataOutputStream	writeInt
40	java.io.ObjectOutputStream	java.io.DataOutputStream	java.io.DataOutputStream	writeLong
41	java.io.ObjectOutputStream	java.io.DataOutputStream	java.io.DataOutputStream	writeFloat
42	java.io.ObjectOutputStream	java.io.DataOutputStream	java.io.DataOutputStream	writeDouble
43	java.io.ObjectOutputStream	java.io.DataOutputStream	java.io.DataOutputStream	writeBytes
44	java.io.ObjectOutputStream	java.io.DataOutputStream	java.io.DataOutputStream	writeChars
45	java.io.ObjectOutputStream	java.io.DataOutputStream	java.io.DataOutputStream	writeUTF
46	java.io.OutputStreamWriter	java.io.OutputStream	java.io.BufferedOutputStream	write
47	java.io.OutputStreamWriter	java.io.OutputStream	java.io.BufferedOutputStream	flush
48	java.io.OutputStreamWriter	java.io.OutputStream	java.io.ByteArrayOutputStream	write
49	java.io.OutputStreamWriter	java.io.OutputStream	java.io.ByteArrayOutputStream	close
50	java.io.OutputStreamWriter	java.io.OutputStream	java.io.DataOutputStream	write
51	java.io.OutputStreamWriter	java.io.OutputStream	java.io.DataOutputStream	flush
52	java.io.OutputStreamWriter	java.io.OutputStream	java.io.FileOutputStream	write
53	java.io.OutputStreamWriter	java.io.OutputStream	java.io.FileOutputStream	close
54	java.io.OutputStreamWriter	java.io.OutputStream	java.io.FilterOutputStream	write
55	java.io.OutputStreamWriter	java.io.OutputStream	java.io.FilterOutputStream	flush
56	java.io.OutputStreamWriter	java.io.OutputStream	java.io.FilterOutputStream	close
57	java.io.OutputStreamWriter	java.io.OutputStream	java.io.ObjectOutputStream	write
58	java.io.OutputStreamWriter	java.io.OutputStream	java.io.ObjectOutputStream	flush
59	java.io.OutputStreamWriter	java.io.OutputStream	java.io.ObjectOutputStream	close
60	java.io.OutputStreamWriter	java.io.OutputStream	java.io.OutputStream	write
61	java.io.OutputStreamWriter	java.io.OutputStream	java.io.OutputStream	flush
62	java.io.OutputStreamWriter	java.io.OutputStream	java.io.OutputStream	close
63	java.io.OutputStreamWriter	java.io.OutputStream	java.io.PrintStream	write
64	java.io.OutputStreamWriter	java.io.OutputStream	java.io.PrintStream	flush
65	java.io.OutputStreamWriter	java.io.OutputStream	java.io.PrintStream	close
66	java.io.OutputStreamWriter	java.io.OutputStream	java.security.DigestOutputStream	write
67	java.security.ProtectionDomain	java.security.PermissionCollection	java.io.FilePermissionCollection	implies
68	java.security.ProtectionDomain	java.security.PermissionCollection	java.net.SocketPermissionCollection	implies
69	java.security.ProtectionDomain	java.security.PermissionCollection	java.security.AllPermissionCollection	implies
70	java.security.ProtectionDomain	java.security.PermissionCollection	java.security.BasicPermissionCollection	implies
71	java.security.ProtectionDomain	java.security.PermissionCollection	java.security.PermissionsHash	implies
72	java.security.ProtectionDomain	java.security.PermissionCollection	java.security.UnresolvedPermissionCollection	implies
73	java.security.ProtectionDomain	java.security.PermissionCollection	java.util.PropertyPermissionCollection	implies
74	java.text.DateFormat	java.util.Calendar	java.util.Calendar	setTimeZone
75	java.text.DateFormat	java.util.Calendar	java.util.Calendar	getTimeZone
76	java.text.DateFormat	java.util.Calendar	java.util.Calendar	setLenient
77	java.text.DateFormat	java.util.Calendar	java.util.Calendar	isLenient
78	java.util.jar.Manifest	java.util.jar.Attributes	java.util.jar.Attributes	clear

PROXY: SELF

	Proxy	Subject	RealSubject	Request
1	uml.core.impl.ClassifierImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	writeOn
2	uml.core.impl.ClassifierImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	check
3	uml.core.impl.ClassifierImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	getOwnedElements
4	uml.core.impl.ClassifierImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	getOwnedElementVisibility
5	uml.core.impl.ClassifierImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	addOwnedElement
6	uml.core.impl.ClassifierImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	removeOwnedElement
7	uml.core.impl.ClassifierImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	contents
8	uml.core.impl.ClassifierImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	allVisibleElements
9	uml.core.impl.ClassifierImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	allSurroundingNamespaces
10	uml.core.impl.NamespaceImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	check
11	uml.core.impl.NamespaceImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	writeOn
12	uml.core.impl.NamespaceImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	getOwnedElements
13	uml.core.impl.NamespaceImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	getOwnedElementVisibility
14	uml.core.impl.NamespaceImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	addOwnedElement
15	uml.core.impl.NamespaceImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	removeOwnedElement
16	uml.core.impl.NamespaceImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	contents
17	uml.core.impl.NamespaceImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	allContents
18	uml.core.impl.NamespaceImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	allVisibleElements
19	uml.core.impl.NamespaceImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	allSurroundingNamespaces
20	uml.core.relationship.impl.AssociationClassImpl	uml.core.relationship.impl.AssociationPrivate	uml.core.relationship.impl.AssociationPrivate	getConnections
21	uml.core.relationship.impl.AssociationClassImpl	uml.core.relationship.impl.AssociationPrivate	uml.core.relationship.impl.AssociationPrivate	addConnection
22	uml.core.relationship.impl.AssociationClassImpl	uml.core.relationship.impl.AssociationPrivate	uml.core.relationship.impl.AssociationPrivate	removeConnection
23	uml.core.relationship.impl.AssociationClassImpl	uml.core.relationship.impl.AssociationPrivate	uml.core.relationship.impl.AssociationPrivate	allConnections
24	uml.core.relationship.impl.AssociationClassImpl	uml.core.relationship.impl.AssociationPrivate	uml.core.relationship.impl.AssociationPrivate	writeOn
25	uml.core.relationship.impl.AssociationClassImpl	uml.core.relationship.impl.AssociationPrivate	uml.core.relationship.impl.AssociationPrivate	check
26	uml.core.relationship.impl.AssociationImpl	uml.core.relationship.impl.AssociationPrivate	uml.core.relationship.impl.AssociationPrivate	getConnections
27	uml.core.relationship.impl.AssociationImpl	uml.core.relationship.impl.AssociationPrivate	uml.core.relationship.impl.AssociationPrivate	addConnection
28	uml.core.relationship.impl.AssociationImpl	uml.core.relationship.impl.AssociationPrivate	uml.core.relationship.impl.AssociationPrivate	removeConnection
29	uml.core.relationship.impl.AssociationImpl	uml.core.relationship.impl.AssociationPrivate	uml.core.relationship.impl.AssociationPrivate	allConnections
30	uml.core.relationship.impl.AssociationImpl	uml.core.relationship.impl.AssociationPrivate	uml.core.relationship.impl.AssociationPrivate	writeOn
31	uml.core.relationship.impl.AssociationImpl	uml.core.relationship.impl.AssociationPrivate	uml.core.relationship.impl.AssociationPrivate	check
32	uml.datatype.Multiplicity	uml.datatype.MultiplicityRange	uml.datatype.MultiplicityRange	setLower
33	uml.datatype.Multiplicity	uml.datatype.MultiplicityRange	uml.datatype.MultiplicityRange	getLower
34	uml.datatype.Multiplicity	uml.datatype.MultiplicityRange	uml.datatype.MultiplicityRange	setUpper
35	uml.datatype.Multiplicity	uml.datatype.MultiplicityRange	uml.datatype.MultiplicityRange	getUpper
36	uml.model.impl.PackageImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	writeOn
37	uml.model.impl.PackageImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	getOwnedElements
38	uml.model.impl.PackageImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	getOwnedElementVisibility

39	uml.model.impl.PackageImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	addOwnedElement
40	uml.model.impl.PackageImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	removeOwnedElement
41	uml.model.impl.PackageImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	contents
42	uml.model.impl.PackageImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	allContents
43	uml.model.impl.PackageImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	allVisibleElements
44	uml.model.impl.PackageImpl	uml.core.impl.NamespacePrivate	uml.core.impl.NamespacePrivate	allSurroundingNamespaces
45	uml.model.impl.PackageImpl	uml.model.impl.PackagePrivate	uml.model.impl.PackagePrivate	writeOn
46	uml.model.impl.PackageImpl	uml.model.impl.PackagePrivate	uml.model.impl.PackagePrivate	getImportedElements
47	uml.model.impl.PackageImpl	uml.model.impl.PackagePrivate	uml.model.impl.PackagePrivate	getImportedElementVisibility
48	uml.model.impl.PackageImpl	uml.model.impl.PackagePrivate	uml.model.impl.PackagePrivate	addImportedElement
49	uml.model.impl.PackageImpl	uml.model.impl.PackagePrivate	uml.model.impl.PackagePrivate	getImportedElementAlias
50	uml.model.impl.PackageImpl	uml.model.impl.PackagePrivate	uml.model.impl.PackagePrivate	removeImportedElement
51	uml.model.impl.SubsystemImpl	uml.model.impl.PackagePrivate	uml.model.impl.PackagePrivate	getImportedElements
52	uml.model.impl.SubsystemImpl	uml.model.impl.PackagePrivate	uml.model.impl.PackagePrivate	getImportedElementVisibility
53	uml.model.impl.SubsystemImpl	uml.model.impl.PackagePrivate	uml.model.impl.PackagePrivate	addImportedElement
54	uml.model.impl.SubsystemImpl	uml.model.impl.PackagePrivate	uml.model.impl.PackagePrivate	getImportedElementAlias
55	uml.model.impl.SubsystemImpl	uml.model.impl.PackagePrivate	uml.model.impl.PackagePrivate	removeImportedElement

PROXY: ARGO

	Proxy	Subject	RealSubject	Request
1	uci.argo.kernel.Designer	uci.argo.kernel.Agency	uci.argo.kernel.Agency	determineActiveCritics
2	uci.argo.kernel.Designer	uci.argo.kernel.DecisionModel	uci.argo.kernel.DecisionModel	getDecisions
3	uci.argo.kernel.Designer	uci.argo.kernel.DecisionModel	uci.argo.kernel.DecisionModel	isConsidering
4	uci.argo.kernel.Designer	uci.argo.kernel.DecisionModel	uci.argo.kernel.DecisionModel	setDecisionPriority
5	uci.argo.kernel.Designer	uci.argo.kernel.DecisionModel	uci.argo.kernel.DecisionModel	defineDecision
6	uci.argo.kernel.Designer	uci.argo.kernel.DecisionModel	uci.argo.kernel.DecisionModel	startConsidering
7	uci.argo.kernel.Designer	uci.argo.kernel.DecisionModel	uci.argo.kernel.DecisionModel	stopConsidering
8	uci.argo.kernel.Designer	uci.argo.kernel.GoalModel	uci.argo.kernel.GoalModel	getGoals
9	uci.argo.kernel.Designer	uci.argo.kernel.GoalModel	uci.argo.kernel.GoalModel	hasGoal
10	uci.argo.kernel.Designer	uci.argo.kernel.GoalModel	uci.argo.kernel.GoalModel	setGoalPriority
11	uci.argo.kernel.Designer	uci.argo.kernel.GoalModel	uci.argo.kernel.GoalModel	startDesiring
12	uci.argo.kernel.Designer	uci.argo.kernel.GoalModel	uci.argo.kernel.GoalModel	stopDesiring
13	uci.argo.kernel.ToDoItem	uci.argo.kernel.Wizard	uci.argo.kernel.Wizard	getProgress
14	uci.gef.Editor	uci.gef.Guide	uci.gef.GuideGrid	snap
15	uci.gef.Editor	uci.gef.LayerManager	uci.gef.LayerManager	preSave
16	uci.gef.Editor	uci.gef.LayerManager	uci.gef.LayerManager	postSave
17	uci.gef.Editor	uci.gef.LayerManager	uci.gef.LayerManager	postLoad
18	uci.gef.Editor	uci.gef.ModeManager	uci.gef.ModeManager	paint
19	uci.gef.Editor	uci.gef.ModeManager	uci.gef.ModeManager	mouseClicked
20	uci.gef.Editor	uci.gef.ModeManager	uci.gef.ModeManager	mousePressed

21	uci.gef.Editor	uci.gef.ModeManager	uci.gef.ModeManager	addModeChangeListener
22	uci.gef.Editor	uci.gef.ModeManager	uci.gef.ModeManager	removeModeChangeListener
23	uci.gef.Editor	uci.gef.ModeManager	uci.gef.ModeManager	mouseReleased
24	uci.gef.Editor	uci.gef.ModeManager	uci.gef.ModeManager	mouseDragged
25	uci.gef.Editor	uci.gef.ModeManager	uci.gef.ModeManager	mouseEntered
26	uci.gef.Editor	uci.gef.ModeManager	uci.gef.ModeManager	mouseMoved
27	uci.gef.Editor	uci.gef.ModeManager	uci.gef.ModeManager	keyTyped
28	uci.gef.Editor	uci.gef.ModeManager	uci.gef.ModeManager	keyPressed
29	uci.gef.Editor	uci.gef.SelectionManager	uci.gef.SelectionManager	paint
30	uci.gef.Editor	uci.gef.SelectionManager	uci.gef.SelectionManager	mouseClicked
31	uci.gef.Editor	uci.gef.SelectionManager	uci.gef.SelectionManager	addGraphSelectionListener
32	uci.gef.Editor	uci.gef.SelectionManager	uci.gef.SelectionManager	removeGraphSelectionListener
33	uci.gef.Editor	uci.gef.SelectionManager	uci.gef.SelectionManager	mousePressed
34	uci.gef.Editor	uci.gef.SelectionManager	uci.gef.SelectionManager	mouseReleased
35	uci.gef.Editor	uci.gef.SelectionManager	uci.gef.SelectionManager	mouseDragged
36	uci.gef.Editor	uci.gef.SelectionManager	uci.gef.SelectionManager	mouseMoved
37	uci.gef.Editor	uci.gef.SelectionManager	uci.gef.SelectionManager	keyTyped
38	uci.gef.Editor	uci.gef.SelectionManager	uci.gef.SelectionManager	keyPressed
39	uci.gef.JGraph	uci.gef.Editor	uci.gef.Editor	clone
40	uci.gef.JGraph	uci.gef.Editor	uci.gef.Editor	setGraphModel
41	uci.gef.JGraph	uci.gef.Editor	uci.gef.Editor	getGraphModel
42	uci.gef.JGraph	uci.gef.Editor	uci.gef.Editor	setGraphNodeRenderer
43	uci.gef.JGraph	uci.gef.Editor	uci.gef.Editor	getGraphNodeRenderer
44	uci.gef.JGraph	uci.gef.Editor	uci.gef.Editor	setGraphEdgeRenderer
45	uci.gef.JGraph	uci.gef.Editor	uci.gef.Editor	getGraphEdgeRenderer
46	uci.gef.LayerManager	uci.gef.Layer	uci.gef.Layer	add
47	uci.gef.LayerManager	uci.gef.Layer	uci.gef.Layer	remove
48	uci.gef.LayerManager	uci.gef.Layer	uci.gef.Layer	removeAll
49	uci.gef.LayerManager	uci.gef.Layer	uci.gef.Layer	sendToBack
50	uci.gef.LayerManager	uci.gef.Layer	uci.gef.Layer	bringForward
51	uci.gef.LayerManager	uci.gef.Layer	uci.gef.Layer	sendBackward
52	uci.gef.LayerManager	uci.gef.Layer	uci.gef.Layer	bringToFront
53	uci.gef.LayerManager	uci.gef.Layer	uci.gef.Layer	reorder
54	uci.gef.LayerManager	uci.gef.Layer	uci.gef.LayerDiagram	add
55	uci.gef.LayerManager	uci.gef.Layer	uci.gef.LayerDiagram	remove
56	uci.gef.LayerManager	uci.gef.Layer	uci.gef.LayerDiagram	removeAll
57	uci.gef.LayerManager	uci.gef.Layer	uci.gef.LayerDiagram	sendToBack
58	uci.gef.LayerManager	uci.gef.Layer	uci.gef.LayerDiagram	bringForward
59	uci.gef.LayerManager	uci.gef.Layer	uci.gef.LayerDiagram	sendBackward
60	uci.gef.LayerManager	uci.gef.Layer	uci.gef.LayerDiagram	bringToFront
61	uci.gef.LayerManager	uci.gef.Layer	uci.gef.LayerDiagram	reorder

62	uci.gef.ModeCreate	uci.gef.Fig	uci.gef.Fig	paint
63	uci.gef.ModeCreate	uci.gef.Fig	uci.gef.FigCircle	paint
64	uci.gef.ModeCreate	uci.gef.Fig	uci.gef.FigEdge	paint
65	uci.gef.ModeCreate	uci.gef.Fig	uci.gef.FigEdgePoly	paint
66	uci.gef.ModeCreate	uci.gef.Fig	uci.gef.FigGroup	paint
67	uci.gef.ModeCreate	uci.gef.Fig	uci.gef.FigLine	paint
68	uci.gef.ModeCreate	uci.gef.Fig	uci.gef.FigNode	paint
69	uci.gef.ModeCreate	uci.gef.Fig	uci.gef.FigPoly	paint
70	uci.gef.ModeCreate	uci.gef.Fig	uci.gef.FigRect	paint
71	uci.gef.ModeCreate	uci.gef.Fig	uci.gef.FigRRect	paint
72	uci.gef.ModeCreate	uci.gef.Fig	uci.gef.FigSpline	paint
73	uci.gef.ModeCreate	uci.gef.Fig	uci.gef.FigText	paint
74	uci.gef.ModeCreate	uci.gef.Fig	uci.uml.visual.FigCompartment	paint
75	uci.gef.ModeCreateEdge	uci.gef.FigNode	uci.gef.FigNode	mousePressed
76	uci.gef.ModeCreateEdge	uci.gef.FigNode	uci.uml.visual.FigClass	mousePressed
77	uci.gef.Selection	uci.gef.Fig	uci.gef.Fig	damage
78	uci.gef.Selection	uci.gef.Fig	uci.gef.Fig	getLocation
79	uci.gef.Selection	uci.gef.Fig	uci.gef.Fig	delete
80	uci.gef.Selection	uci.gef.Fig	uci.gef.Fig	dispose
81	uci.gef.Selection	uci.gef.Fig	uci.gef.Fig	reorder
82	uci.gef.Selection	uci.gef.Fig	uci.gef.Fig	translate
83	uci.gef.Selection	uci.gef.Fig	uci.gef.Fig	contains
84	uci.gef.Selection	uci.gef.Fig	uci.gef.Fig	hit
85	uci.gef.Selection	uci.gef.Fig	uci.gef.FigCircle	contains
86	uci.gef.Selection	uci.gef.Fig	uci.gef.FigEdge	contains
87	uci.gef.Selection	uci.gef.Fig	uci.gef.FigEdge	hit
88	uci.gef.Selection	uci.gef.Fig	uci.gef.FigEdge	delete
89	uci.gef.Selection	uci.gef.Fig	uci.gef.FigGroup	translate
90	uci.gef.Selection	uci.gef.Fig	uci.gef.FigGroup	contains
91	uci.gef.Selection	uci.gef.Fig	uci.gef.FigGroup	hit
92	uci.gef.Selection	uci.gef.Fig	uci.gef.FigInk	contains
93	uci.gef.Selection	uci.gef.Fig	uci.gef.FigLine	translate
94	uci.gef.Selection	uci.gef.Fig	uci.gef.FigLine	hit
95	uci.gef.Selection	uci.gef.Fig	uci.gef.FigNode	delete
96	uci.gef.Selection	uci.gef.Fig	uci.gef.FigNode	dispose
97	uci.gef.Selection	uci.gef.Fig	uci.gef.FigNode	contains
98	uci.gef.Selection	uci.gef.Fig	uci.gef.FigNode	translate
99	uci.gef.Selection	uci.gef.Fig	uci.gef.FigNode	hit
100	uci.gef.Selection	uci.gef.Fig	uci.gef.FigPoly	translate
101	uci.gef.Selection	uci.gef.Fig	uci.gef.FigPoly	contains
102	uci.gef.Selection	uci.gef.Fig	uci.gef.FigPoly	hit

103	uci.gef.Selection	uci.gef.Fig	uci.gef.FigSpline	translate
104	uci.gef.Selection	uci.gef.Fig	uci.gef.FigText	hit
105	uci.gef.Selection	uci.gef.Fig	uci.uml.visual.FigAssociationRole	dispose
106	uci.gef.Selection	uci.gef.Fig	uci.uml.visual.FigClass	translate
107	uci.gef.Selection	uci.gef.Fig	uci.uml.visual.FigClassifierRole	dispose
108	uci.gef.Selection	uci.gef.Fig	uci.uml.visual.FigMessage	dispose
109	uci.graph.DefaultGraphModel	uci.gef.NetList	uci.gef.NetList	getNodes
110	uci.graph.DefaultGraphModel	uci.gef.NetList	uci.gef.NetList	getEdges
111	uci.graph.DefaultGraphModel	uci.gef.NetList	uci.gef.NetList	removeNode
112	uci.graph.DefaultGraphModel	uci.gef.NetList	uci.gef.NetList	addNode
113	uci.graph.DefaultGraphModel	uci.gef.NetList	uci.gef.NetList	addEdge
114	uci.graph.DefaultGraphModel	uci.gef.NetList	uci.gef.NetList	removeEdge
115	uci.uml.ui.NavigatorPane	uci.uml.ui.DisplayTextTree	uci.uml.ui.DisplayTextTree	forceUpdate
116	uci.uml.ui.ProjectBrowser	uci.uml.ui.MultiEditorPane	uci.uml.ui.MultiEditorPane	getTarget
117	uci.uml.ui.ProjectBrowser	uci.uml.ui.MultiEditorPane	uci.uml.ui.MultiEditorPane	setTarget
118	uci.uml.ui.ProjectBrowser	uci.uml.ui.MultiEditorPane	uci.uml.ui.MultiEditorPane	select
119	uci.uml.ui.ProjectBrowser	uci.uml.ui.NavigatorPane	uci.uml.ui.NavigatorPane	getCurPerspective
120	uci.uml.ui.ProjectBrowser	uci.uml.ui.NavigatorPane	uci.uml.ui.NavigatorPane	setCurPerspective
121	uci.uml.ui.ProjectBrowser	uci.uml.ui.NavigatorPane	uci.uml.ui.NavigatorPane	getPerspectives
122	uci.uml.ui.ProjectBrowser	uci.uml.ui.NavigatorPane	uci.uml.ui.NavigatorPane	setPerspectives
123	uci.uml.ui.TabDiagram	uci.gef.JGraph	uci.gef.JGraph	removeGraphSelectionListener
124	uci.uml.ui.TabDiagram	uci.gef.JGraph	uci.gef.JGraph	removeModeChangeListener
125	uci.uml.ui.ToDoTreeRenderer	uci.uml.ui.UMLTreeCellRenderer	uci.uml.ui.UMLTreeCellRenderer	getTreeCellRendererComponent
126	uci.uml.visual.FigEdgeModelElement	uci.gef.FigText	uci.gef.FigText	keyPressed
127	uci.uml.visual.FigNodeModelElement	uci.gef.FigText	uci.gef.FigText	keyPressed

SINGLETON: STD

	Singleton	Accessor
1	java.lang.Character$UnicodeBlock	ALPHABETIC_PRESENTATION_FORMS
2	java.lang.Character$UnicodeBlock	ARABIC
3	java.lang.Character$UnicodeBlock	ARABIC_PRESENTATION_FORMS_A
4	java.lang.Character$UnicodeBlock	ARABIC_PRESENTATION_FORMS_B
5	java.lang.Character$UnicodeBlock	ARMENIAN
6	java.lang.Character$UnicodeBlock	ARROWS
7	java.lang.Character$UnicodeBlock	BASIC_LATIN
8	java.lang.Character$UnicodeBlock	BENGALI
9	java.lang.Character$UnicodeBlock	BLOCK_ELEMENTS
10	java.lang.Character$UnicodeBlock	BOPOMOFO
11	java.lang.Character$UnicodeBlock	BOX_DRAWING
12	java.lang.Character$UnicodeBlock	CJK_COMPATIBILITY
13	java.lang.Character$UnicodeBlock	CJK_COMPATIBILITY_FORMS
14	java.lang.Character$UnicodeBlock	CJK_COMPATIBILITY_IDEOGRAPHS
15	java.lang.Character$UnicodeBlock	CJK_SYMBOLS_AND_PUNCTUATION
16	java.lang.Character$UnicodeBlock	CJK_UNIFIED_IDEOGRAPHS
17	java.lang.Character$UnicodeBlock	COMBINING_DIACRITICAL_MARKS
18	java.lang.Character$UnicodeBlock	COMBINING_HALF_MARKS
19	java.lang.Character$UnicodeBlock	COMBINING_MARKS_FOR_SYMBOLS
20	java.lang.Character$UnicodeBlock	CONTROL_PICTURES
21	java.lang.Character$UnicodeBlock	CURRENCY_SYMBOLS
22	java.lang.Character$UnicodeBlock	CYRILLIC
23	java.lang.Character$UnicodeBlock	DEVANAGARI
24	java.lang.Character$UnicodeBlock	DINGBATS
25	java.lang.Character$UnicodeBlock	ENCLOSED_ALPHANUMERICS
26	java.lang.Character$UnicodeBlock	ENCLOSED_CJK_LETTERS_AND_MONTHS
27	java.lang.Character$UnicodeBlock	GENERAL_PUNCTUATION
28	java.lang.Character$UnicodeBlock	GEOMETRIC_SHAPES
29	java.lang.Character$UnicodeBlock	GEORGIAN
30	java.lang.Character$UnicodeBlock	GREEK
31	java.lang.Character$UnicodeBlock	GREEK_EXTENDED
32	java.lang.Character$UnicodeBlock	GUJARATI
33	java.lang.Character$UnicodeBlock	GURMUKHI
34	java.lang.Character$UnicodeBlock	HALFWIDTH_AND_FULLWIDTH_FORMS
35	java.lang.Character$UnicodeBlock	HANGUL_COMPATIBILITY_JAMO
36	java.lang.Character$UnicodeBlock	HANGUL_JAMO
37	java.lang.Character$UnicodeBlock	HANGUL_SYLLABLES
38	java.lang.Character$UnicodeBlock	HEBREW
39	java.lang.Character$UnicodeBlock	HIRAGANA
40	java.lang.Character$UnicodeBlock	IPA_EXTENSIONS
41	java.lang.Character$UnicodeBlock	KANBUN
42	java.lang.Character$UnicodeBlock	KANNADA
43	java.lang.Character$UnicodeBlock	KATAKANA
44	java.lang.Character$UnicodeBlock	LAO
45	java.lang.Character$UnicodeBlock	LATIN_1_SUPPLEMENT
46	java.lang.Character$UnicodeBlock	LATIN_EXTENDED_A
47	java.lang.Character$UnicodeBlock	LATIN_EXTENDED_ADDITIONAL
48	java.lang.Character$UnicodeBlock	LATIN_EXTENDED_B
49	java.lang.Character$UnicodeBlock	LETTERLIKE_SYMBOLS
50	java.lang.Character$UnicodeBlock	MALAYALAM
51	java.lang.Character$UnicodeBlock	MATHEMATICAL_OPERATORS
52	java.lang.Character$UnicodeBlock	MISCELLANEOUS_SYMBOLS
53	java.lang.Character$UnicodeBlock	MISCELLANEOUS_TECHNICAL
54	java.lang.Character$UnicodeBlock	NUMBER_FORMS
55	java.lang.Character$UnicodeBlock	OPTICAL_CHARACTER_RECOGNITION
56	java.lang.Character$UnicodeBlock	ORIYA
57	java.lang.Character$UnicodeBlock	PRIVATE_USE_AREA
58	java.lang.Character$UnicodeBlock	SMALL_FORM_VARIANTS
59	java.lang.Character$UnicodeBlock	SPACING_MODIFIER_LETTERS
60	java.lang.Character$UnicodeBlock	SPECIALS
61	java.lang.Character$UnicodeBlock	SUPERSCRIPTS_AND_SUBSCRIPTS
62	java.lang.Character$UnicodeBlock	SURROGATES_AREA
63	java.lang.Character$UnicodeBlock	TAMIL
64	java.lang.Character$UnicodeBlock	TELUGU
65	java.lang.Character$UnicodeBlock	THAI
66	java.lang.Character$UnicodeBlock	TIBETAN
67	java.lang.Runtime	getRuntime
68	java.net.InetAddress	getByName
69	java.net.InetAddress	getLocalHost

SINGLETON: SELF

Singleton	Accessor

SINGLETON: ARGO

	Singleton	Accessor
1	uci.uml.ui.EOElement	SINGLETON

TEMPLATE METHOD: STD

	AbstractClass	ConcreteClass	TemplateMethod	PrimitiveOperation
1	java.io.InputStream	java.io.BufferedInputStream	read	read
2	java.io.InputStream	java.io.BufferedInputStream	skip	read
3	java.io.InputStream	java.io.ByteArrayInputStream	read	read
4	java.io.InputStream	java.io.ByteArrayInputStream	skip	read
5	java.io.InputStream	java.io.DataInputStream	read	read
6	java.io.InputStream	java.io.DataInputStream	skip	read
7	java.io.InputStream	java.io.FileInputStream	read	read
8	java.io.InputStream	java.io.FileInputStream	skip	read
9	java.io.InputStream	java.io.FilterInputStream	read	read
10	java.io.InputStream	java.io.FilterInputStream	skip	read
11	java.io.InputStream	java.io.ObjectInputStream	read	read
12	java.io.InputStream	java.io.ObjectInputStream	skip	read
13	java.io.InputStream	java.io.PushbackInputStream	read	read
14	java.io.InputStream	java.io.PushbackInputStream	skip	read
15	java.io.InputStream	java.util.jar.JarVerifier$VerifierStream	read	read
16	java.io.InputStream	java.util.jar.JarVerifier$VerifierStream	skip	read
17	java.io.InputStream	java.util.jar.Manifest$FastInputStream	read	read
18	java.io.InputStream	java.util.jar.Manifest$FastInputStream	skip	read
19	java.io.InputStream	java.util.zip.InflaterInputStream	read	read
20	java.io.InputStream	java.util.zip.InflaterInputStream	skip	read
21	java.io.InputStream	java.util.zip.ZipFile$ZipFileInputStream	read	read
22	java.io.InputStream	java.util.zip.ZipFile$ZipFileInputStream	skip	read
23	java.io.OutputStream	java.io.BufferedOutputStream	write	write
24	java.io.OutputStream	java.io.ByteArrayOutputStream	write	write
25	java.io.OutputStream	java.io.DataOutputStream	write	write
26	java.io.OutputStream	java.io.FileOutputStream	write	write
27	java.io.OutputStream	java.io.FilterOutputStream	write	write
28	java.io.OutputStream	java.io.ObjectOutputStream	write	write
29	java.io.OutputStream	java.io.PrintStream	write	write
30	java.io.OutputStream	java.security.DigestOutputStream	write	write
31	java.io.Reader	java.io.BufferedReader	read	read
32	java.io.Reader	java.io.BufferedReader	skip	read
33	java.io.Reader	java.io.InputStreamReader	read	read
34	java.io.Reader	java.io.InputStreamReader	skip	read
35	java.io.Writer	java.io.BufferedWriter	write	write
36	java.io.Writer	java.io.OutputStreamWriter	write	write
37	java.io.Writer	java.io.PrintWriter	write	write
38	java.lang.Number	java.lang.Byte	byteValue	intValue

39	java.lang.Number	java.lang.Byte	shortValue	intValue
40	java.lang.Number	java.lang.Double	byteValue	intValue
41	java.lang.Number	java.lang.Double	shortValue	intValue
42	java.lang.Number	java.lang.Float	byteValue	intValue
43	java.lang.Number	java.lang.Float	shortValue	intValue
44	java.lang.Number	java.lang.Integer	byteValue	intValue
45	java.lang.Number	java.lang.Integer	shortValue	intValue
46	java.lang.Number	java.lang.Long	byteValue	intValue
47	java.lang.Number	java.lang.Long	shortValue	intValue
48	java.lang.Number	java.lang.Short	byteValue	intValue
49	java.lang.Number	java.lang.Short	shortValue	intValue
50	java.lang.Number	java.math.BigInteger	byteValue	intValue
51	java.lang.Number	java.math.BigInteger	shortValue	intValue
52	java.lang.Object	java.io.File	toString	hashCode
53	java.lang.Object	java.lang.Boolean	toString	hashCode
54	java.lang.Object	java.lang.Byte	toString	hashCode
55	java.lang.Object	java.lang.Character	toString	hashCode
56	java.lang.Object	java.lang.Character$Subset	toString	hashCode
57	java.lang.Object	java.lang.Double	toString	hashCode
58	java.lang.Object	java.lang.Float	toString	hashCode
59	java.lang.Object	java.lang.Integer	toString	hashCode
60	java.lang.Object	java.lang.Long	toString	hashCode
61	java.lang.Object	java.lang.Package	toString	hashCode
62	java.lang.Object	java.lang.reflect.Constructor	toString	hashCode
63	java.lang.Object	java.lang.reflect.Field	toString	hashCode
64	java.lang.Object	java.lang.reflect.Method	toString	hashCode
65	java.lang.Object	java.lang.Short	toString	hashCode
66	java.lang.Object	java.lang.String	toString	hashCode
67	java.lang.Object	java.math.BigInteger	toString	hashCode
68	java.lang.Object	java.net.InetAddress	toString	hashCode
69	java.lang.Object	java.net.URL	toString	hashCode
70	java.lang.Object	java.security.AccessControlContext	toString	hashCode
71	java.lang.Object	java.security.cert.Certificate	toString	hashCode
72	java.lang.Object	java.security.CodeSource	toString	hashCode
73	java.lang.Object	java.text.ChoiceFormat	toString	hashCode
74	java.lang.Object	java.text.DateFormat	toString	hashCode
75	java.lang.Object	java.text.DateFormatSymbols	toString	hashCode
76	java.lang.Object	java.text.DecimalFormat	toString	hashCode
77	java.lang.Object	java.text.DecimalFormatSymbols	toString	hashCode
78	java.lang.Object	java.text.DigitList	toString	hashCode
79	java.lang.Object	java.text.FieldPosition	toString	hashCode

80	java.lang.Object	java.text.MessageFormat	toString	hashCode
81	java.lang.Object	java.text.NumberFormat	toString	hashCode
82	java.lang.Object	java.text.ParsePosition	toString	hashCode
83	java.lang.Object	java.text.SimpleDateFormat	toString	hashCode
84	java.lang.Object	java.util.AbstractMap	toString	hashCode
85	java.lang.Object	java.util.Calendar	toString	hashCode
86	java.lang.Object	java.util.Collections$SynchronizedList	toString	hashCode
87	java.lang.Object	java.util.Collections$SynchronizedMap	toString	hashCode
88	java.lang.Object	java.util.Collections$SynchronizedSet	toString	hashCode
89	java.lang.Object	java.util.Collections$UnmodifiableList	toString	hashCode
90	java.lang.Object	java.util.Collections$UnmodifiableMap	toString	hashCode
91	java.lang.Object	java.util.Collections$UnmodifiableMap$Unmodifia	toString	hashCode
92	java.lang.Object	java.util.Collections$UnmodifiableSet	toString	hashCode
93	java.lang.Object	java.util.Date	toString	hashCode
94	java.lang.Object	java.util.GregorianCalendar	toString	hashCode
95	java.lang.Object	java.util.HashMap$Entry	toString	hashCode
96	java.lang.Object	java.util.Hashtable	toString	hashCode
97	java.lang.Object	java.util.Hashtable$Entry	toString	hashCode
98	java.lang.Object	java.util.jar.Attributes	toString	hashCode
99	java.lang.Object	java.util.jar.Attributes$Name	toString	hashCode
100	java.lang.Object	java.util.jar.Manifest	toString	hashCode
101	java.lang.Object	java.util.Locale	toString	hashCode
102	java.lang.Object	java.util.ResourceBundle$ResourceCacheKey	toString	hashCode
103	java.lang.Object	java.util.SimpleTimeZone	toString	hashCode
104	java.lang.Object	java.util.zip.ZipEntry	toString	hashCode
105	java.lang.Throwable	java.io.InvalidClassException	getLocalizedMessage	getMessage
106	java.lang.Throwable	java.io.WriteAbortedException	getLocalizedMessage	getMessage
107	java.security.MessageDigestSpi	java.security.MessageDigest$Delegate	engineDigest	engineDigest
108	java.security.PermissionCollection	java.io.FilePermissionCollection	toString	elements
109	java.security.PermissionCollection	java.net.SocketPermissionCollection	toString	elements
110	java.security.PermissionCollection	java.security.AllPermissionCollection	toString	elements
111	java.security.PermissionCollection	java.security.BasicPermissionCollection	toString	elements
112	java.security.PermissionCollection	java.security.PermissionsHash	toString	elements
113	java.security.PermissionCollection	java.security.UnresolvedPermissionCollection	toString	elements
114	java.security.PermissionCollection	java.util.PropertyPermissionCollection	toString	elements
115	java.text.DateFormat	java.text.SimpleDateFormat	format	format
116	java.text.DateFormat	java.text.SimpleDateFormat	parse	parse
117	java.text.DateFormat	java.text.SimpleDateFormat	parseObject	parse
118	java.text.Format	java.text.DateFormat	format	format
119	java.text.Format	java.text.DateFormat	parseObject	parseObject
120	java.text.Format	java.text.MessageFormat	format	format

121	java.text.Format	java.text.MessageFormat	parseObject	parseObject
122	java.text.Format	java.text.NumberFormat	format	format
123	java.text.Format	java.text.NumberFormat	parseObject	parseObject
124	java.text.NumberFormat	java.text.ChoiceFormat	format	format
125	java.text.NumberFormat	java.text.DecimalFormat	format	format
126	java.util.AbstractCollection	java.util.AbstractList	addAll	add
127	java.util.AbstractCollection	java.util.ArrayList	addAll	add
128	java.util.AbstractCollection	java.util.ArrayList	toArray	size
129	java.util.AbstractCollection	java.util.Arrays$ArrayList	toArray	size
130	java.util.AbstractCollection	java.util.Collections$CopiesList	toArray	size
131	java.util.AbstractCollection	java.util.Collections$EmptyList	toArray	size
132	java.util.AbstractCollection	java.util.Collections$EmptySet	clear	iterator
133	java.util.AbstractCollection	java.util.Collections$EmptySet	contains	iterator
134	java.util.AbstractCollection	java.util.Collections$EmptySet	remove	iterator
135	java.util.AbstractCollection	java.util.Collections$EmptySet	removeAll	iterator
136	java.util.AbstractCollection	java.util.Collections$EmptySet	retainAll	iterator
137	java.util.AbstractCollection	java.util.Collections$EmptySet	toArray	iterator
138	java.util.AbstractCollection	java.util.Collections$EmptySet	toArray	size
139	java.util.AbstractCollection	java.util.Collections$EmptySet	toString	iterator
140	java.util.AbstractCollection	java.util.Collections$SingletonSet	clear	iterator
141	java.util.AbstractCollection	java.util.Collections$SingletonSet	contains	iterator
142	java.util.AbstractCollection	java.util.Collections$SingletonSet	remove	iterator
143	java.util.AbstractCollection	java.util.Collections$SingletonSet	removeAll	iterator
144	java.util.AbstractCollection	java.util.Collections$SingletonSet	retainAll	iterator
145	java.util.AbstractCollection	java.util.Collections$SingletonSet	toArray	iterator
146	java.util.AbstractCollection	java.util.Collections$SingletonSet	toArray	size
147	java.util.AbstractCollection	java.util.Collections$SingletonSet	toString	iterator
148	java.util.AbstractCollection	java.util.Hashtable$EntrySet	toArray	size
149	java.util.AbstractCollection	java.util.Hashtable$KeySet	toArray	size
150	java.util.AbstractCollection	java.util.Hashtable$ValueCollection	toArray	size
151	java.util.AbstractCollection	java.util.SubList	clear	iterator
152	java.util.AbstractCollection	java.util.SubList	contains	iterator
153	java.util.AbstractCollection	java.util.SubList	remove	iterator
154	java.util.AbstractCollection	java.util.SubList	removeAll	iterator
155	java.util.AbstractCollection	java.util.SubList	retainAll	iterator
156	java.util.AbstractCollection	java.util.SubList	toArray	iterator
157	java.util.AbstractCollection	java.util.SubList	toArray	size
158	java.util.AbstractCollection	java.util.SubList	toString	iterator
159	java.util.AbstractCollection	java.util.Vector	addAll	add
160	java.util.AbstractCollection	java.util.Vector	toArray	size
161	java.util.AbstractList	java.util.ArrayList	add	add

162	java.util.AbstractList	java.util.ArrayList	addAll	add
163	java.util.AbstractList	java.util.SubList	add	add
164	java.util.AbstractList	java.util.SubList	addAll	add
165	java.util.AbstractList	java.util.SubList	equals	listIterator
166	java.util.AbstractList	java.util.SubList	indexOf	listIterator
167	java.util.AbstractList	java.util.SubList	lastIndexOf	listIterator
168	java.util.AbstractList	java.util.SubList	listIterator	listIterator
169	java.util.AbstractList	java.util.Vector	add	add
170	java.util.AbstractList	java.util.Vector	addAll	add
171	java.util.AbstractMap	java.util.HashMap	clear	entrySet
172	java.util.AbstractMap	java.util.HashMap	containsKey	entrySet
173	java.util.AbstractMap	java.util.HashMap	containsValue	entrySet
174	java.util.AbstractMap	java.util.HashMap	equals	entrySet
175	java.util.AbstractMap	java.util.HashMap	get	entrySet
176	java.util.AbstractMap	java.util.HashMap	hashCode	entrySet
177	java.util.AbstractMap	java.util.HashMap	putAll	put
178	java.util.AbstractMap	java.util.HashMap	remove	entrySet
179	java.util.AbstractMap	java.util.HashMap	size	entrySet
180	java.util.AbstractMap	java.util.HashMap	toString	entrySet
181	java.util.Calendar	java.util.GregorianCalendar	complete	computeFields
182	java.util.Calendar	java.util.GregorianCalendar	getActualMaximum	getMaximum
183	java.util.Calendar	java.util.GregorianCalendar	getActualMaximum	getLeastMaximum
184	java.util.Calendar	java.util.GregorianCalendar	getActualMinimum	getMinimum
185	java.util.Calendar	java.util.GregorianCalendar	getActualMinimum	getGreatestMinimum
186	java.util.Calendar	java.util.GregorianCalendar	roll	roll
187	java.util.Calendar	java.util.GregorianCalendar	setTimeInMillis	computeFields
188	java.util.Hashtable	java.security.Provider	equals	entrySet
189	java.util.Hashtable	java.security.Provider	hashCode	entrySet
190	java.util.Hashtable	java.security.Provider	toString	entrySet
191	java.util.TimeZone	java.util.SimpleTimeZone	getDisplayName	getRawOffset
192	java.util.zip.InflaterInputStream	java.util.jar.JarInputStream	skip	read
193	java.util.zip.InflaterInputStream	java.util.zip.ZipInputStream	skip	read

TEMPLATE METHOD: SELF

	AbstractClass	ConcreteClass	TemplateMethod	PrimitiveOperation
1	reeng.code.Code	reeng.code.Assign	assert	assert
2	reeng.code.Code	reeng.code.LocalVar	assert	assert
3	reeng.code.Code	reeng.code.New	assert	assert
4	reeng.code.Code	reeng.code.Return	assert	assert
5	reeng.code.Code	reeng.code.Send	assert	assert

6	reeng.code.Code	reeng.code.TypeCast	assert	assert
7	reeng.GenericReader	reeng.java.ClassFileReader	readWithStatus	read
8	reeng.GenericReader	reeng.java.ReflectReader	readWithStatus	read
9	reeng.GenericReader	reeng.java.SourceReader	readWithStatus	read
10	reeng.java.ASTNullVisitor	reeng.java.ASTUMLVisitorBase	visit	visitDefault
11	reeng.java.ASTNullVisitor	reeng.java.ASTXMLVisitor	visit	visitDefault
12	uml.core.impl.ElementImpl	uml.core.classifier.impl.ClassImpl	autoCheck	check
13	uml.core.impl.ElementImpl	uml.core.classifier.impl.DataTypeImpl	autoCheck	check
14	uml.core.impl.ElementImpl	uml.core.classifier.impl.InterfaceImpl	autoCheck	check
15	uml.core.impl.ElementImpl	uml.core.impl.BehavioralFeatureImpl	autoCheck	check
16	uml.core.impl.ElementImpl	uml.core.impl.ClassifierImpl	autoCheck	check
17	uml.core.impl.ElementImpl	uml.core.impl.ConstraintImpl	autoCheck	check
18	uml.core.impl.ElementImpl	uml.core.impl.GeneralizableElementImpl	autoCheck	check
19	uml.core.impl.ElementImpl	uml.core.impl.MethodImpl	autoCheck	check
20	uml.core.impl.ElementImpl	uml.core.impl.NamespaceImpl	autoCheck	check
21	uml.core.impl.ElementImpl	uml.core.impl.StructuralFeatureImpl	autoCheck	check
22	uml.core.impl.ElementImpl	uml.core.relationship.impl.AssociationClassImpl	autoCheck	check
23	uml.core.impl.ElementImpl	uml.core.relationship.impl.AssociationEndImpl	autoCheck	check
24	uml.core.impl.ElementImpl	uml.core.relationship.impl.AssociationImpl	autoCheck	check
25	uml.core.impl.ElementImpl	uml.core.relationship.impl.GeneralizationImpl	autoCheck	check

TEMPLATE METHOD: ARGO

	AbstractClass	ConcreteClass	TemplateMethod	PrimitiveOperation
1	uci.argo.kernel.Critic	uci.argo.kernel.CompoundCritic	critique	toDoItem
2	uci.argo.kernel.Critic	uci.argo.kernel.CompoundCritic	isRelevantToDecisions	getSupportedDecisions
3	uci.argo.kernel.Critic	uci.argo.kernel.CompoundCritic	setKnowledgeTypes	addKnowledgeType
4	uci.argo.kernel.Critic	uci.argo.kernel.CompoundCritic	stillValid	toDoItem
5	uci.argo.kernel.Critic	uci.uml.critics.CrUML	critique	predicate
6	uci.argo.kernel.Critic	uci.uml.critics.CrUML	getMoreInfoURL	getMoreInfoURL
7	uci.argo.kernel.Critic	uci.uml.critics.CrUML	stillValid	predicate
8	uci.argo.kernel.Wizard	uci.uml.critics.WizMEName	doAction	doAction
9	uci.argo.kernel.Wizard	uci.uml.critics.WizMEName	finish	getNumSteps
10	uci.argo.kernel.Wizard	uci.uml.critics.WizMEName	finish	doAction
11	uci.argo.kernel.Wizard	uci.uml.critics.WizMEName	next	doAction
12	uci.argo.kernel.Wizard	uci.uml.critics.WizMEName	redoAction	doAction
13	uci.gef.ArrowHead	uci.gef.ArrowHeadDiamond	paintAtHead	paint
14	uci.gef.ArrowHead	uci.gef.ArrowHeadDiamond	paintAtTail	paint
15	uci.gef.ArrowHead	uci.gef.ArrowHeadGreater	paintAtHead	paint
16	uci.gef.ArrowHead	uci.gef.ArrowHeadGreater	paintAtTail	paint
17	uci.gef.ArrowHead	uci.gef.ArrowHeadNone	paintAtHead	paint

18	uci.gef.ArrowHead	uci.gef.ArrowHeadNone	paintAtTail	paint
19	uci.gef.ArrowHead	uci.gef.ArrowHeadTriangle	paintAtHead	paint
20	uci.gef.ArrowHead	uci.gef.ArrowHeadTriangle	paintAtTail	paint
21	uci.gef.Cmd	uci.gef.CmdAdjustGrid	actionPerformed	doIt
22	uci.gef.Cmd	uci.gef.CmdAdjustGuide	actionPerformed	doIt
23	uci.gef.Cmd	uci.gef.CmdAdjustPageBreaks	actionPerformed	doIt
24	uci.gef.Cmd	uci.gef.CmdAlign	actionPerformed	doIt
25	uci.gef.Cmd	uci.gef.CmdCopy	actionPerformed	doIt
26	uci.gef.Cmd	uci.gef.CmdDistribute	actionPerformed	doIt
27	uci.gef.Cmd	uci.gef.CmdGroup	actionPerformed	doIt
28	uci.gef.Cmd	uci.gef.CmdInsertPoint	actionPerformed	doIt
29	uci.gef.Cmd	uci.gef.CmdNudge	actionPerformed	doIt
30	uci.gef.Cmd	uci.gef.CmdPaste	actionPerformed	doIt
31	uci.gef.Cmd	uci.gef.CmdPrint	actionPerformed	doIt
32	uci.gef.Cmd	uci.gef.CmdRemovePoint	actionPerformed	doIt
33	uci.gef.Cmd	uci.gef.CmdReorder	actionPerformed	doIt
34	uci.gef.Cmd	uci.gef.CmdSaveGIF	actionPerformed	doIt
35	uci.gef.Cmd	uci.gef.CmdSelectAll	actionPerformed	doIt
36	uci.gef.Cmd	uci.gef.CmdSelectInvert	actionPerformed	doIt
37	uci.gef.Cmd	uci.gef.CmdSelectNear	actionPerformed	doIt
38	uci.gef.Cmd	uci.gef.CmdSelectNext	actionPerformed	doIt
39	uci.gef.Cmd	uci.gef.CmdSetMode	actionPerformed	doIt
40	uci.gef.Cmd	uci.gef.CmdUngroup	actionPerformed	doIt
41	uci.gef.Fig	uci.gef.FigCircle	contains	contains
42	uci.gef.Fig	uci.gef.FigCircle	print	paint
43	uci.gef.Fig	uci.gef.FigEdge	align	getBounds
44	uci.gef.Fig	uci.gef.FigEdge	center	getBounds
45	uci.gef.Fig	uci.gef.FigEdge	contains	contains
46	uci.gef.Fig	uci.gef.FigEdge	delete	setOwner
47	uci.gef.Fig	uci.gef.FigEdge	dispose	delete
48	uci.gef.Fig	uci.gef.FigEdge	getClosestPoint	getBounds
49	uci.gef.Fig	uci.gef.FigEdge	getTrapRect	getBounds
50	uci.gef.Fig	uci.gef.FigEdge	pointAlongPerimeter	stuffPointAlongPerimeter
51	uci.gef.Fig	uci.gef.FigEdge	print	paint
52	uci.gef.Fig	uci.gef.FigEdge	setBounds	getBounds
53	uci.gef.Fig	uci.gef.FigEdge	setPoints	setPoints
54	uci.gef.Fig	uci.gef.FigEdge	translate	getBounds
55	uci.gef.Fig	uci.gef.FigEdgePoly	print	paint
56	uci.gef.Fig	uci.gef.FigEdgePoly	setPoints	setPoints
57	uci.gef.Fig	uci.gef.FigGroup	align	translate
58	uci.gef.Fig	uci.gef.FigGroup	contains	contains

59	uci.gef.Fig	uci.gef.FigGroup	createDrag	setBounds
60	uci.gef.Fig	uci.gef.FigGroup	print	paint
61	uci.gef.Fig	uci.gef.FigGroup	setBounds	setBounds
62	uci.gef.Fig	uci.gef.FigGroup	setHeight	setBounds
63	uci.gef.Fig	uci.gef.FigGroup	setLocation	translate
64	uci.gef.Fig	uci.gef.FigGroup	setSize	setBounds
65	uci.gef.Fig	uci.gef.FigGroup	setWidth	setBounds
66	uci.gef.Fig	uci.gef.FigGroup	setX	setBounds
67	uci.gef.Fig	uci.gef.FigGroup	setY	setBounds
68	uci.gef.Fig	uci.gef.FigInk	contains	contains
69	uci.gef.Fig	uci.gef.FigLine	align	translate
70	uci.gef.Fig	uci.gef.FigLine	createDrag	setBounds
71	uci.gef.Fig	uci.gef.FigLine	pointAlongPerimeter	stuffPointAlongPerimeter
72	uci.gef.Fig	uci.gef.FigLine	print	paint
73	uci.gef.Fig	uci.gef.FigLine	setBounds	setBounds
74	uci.gef.Fig	uci.gef.FigLine	setHeight	setBounds
75	uci.gef.Fig	uci.gef.FigLine	setLocation	translate
76	uci.gef.Fig	uci.gef.FigLine	setPoints	setPoints
77	uci.gef.Fig	uci.gef.FigLine	setSize	setBounds
78	uci.gef.Fig	uci.gef.FigLine	setWidth	setBounds
79	uci.gef.Fig	uci.gef.FigLine	setX	setBounds
80	uci.gef.Fig	uci.gef.FigLine	setY	setBounds
81	uci.gef.Fig	uci.gef.FigNode	align	translate
82	uci.gef.Fig	uci.gef.FigNode	contains	contains
83	uci.gef.Fig	uci.gef.FigNode	createDrag	setBounds
84	uci.gef.Fig	uci.gef.FigNode	delete	setOwner
85	uci.gef.Fig	uci.gef.FigNode	dispose	delete
86	uci.gef.Fig	uci.gef.FigNode	print	paint
87	uci.gef.Fig	uci.gef.FigNode	setBounds	setBounds
88	uci.gef.Fig	uci.gef.FigNode	setHeight	setBounds
89	uci.gef.Fig	uci.gef.FigNode	setLocation	translate
90	uci.gef.Fig	uci.gef.FigNode	setSize	setBounds
91	uci.gef.Fig	uci.gef.FigNode	setWidth	setBounds
92	uci.gef.Fig	uci.gef.FigNode	setX	setBounds
93	uci.gef.Fig	uci.gef.FigNode	setY	setBounds
94	uci.gef.Fig	uci.gef.FigPoly	align	translate
95	uci.gef.Fig	uci.gef.FigPoly	connectionPoint	getClosestPoint
96	uci.gef.Fig	uci.gef.FigPoly	connectionPoint	getGravityPoints
97	uci.gef.Fig	uci.gef.FigPoly	contains	contains
98	uci.gef.Fig	uci.gef.FigPoly	createDrag	setBounds
99	uci.gef.Fig	uci.gef.FigPoly	pointAlongPerimeter	stuffPointAlongPerimeter

100	uci.gef.Fig	uci.gef.FigPoly	print	paint
101	uci.gef.Fig	uci.gef.FigPoly	setBounds	setBounds
102	uci.gef.Fig	uci.gef.FigPoly	setHeight	setBounds
103	uci.gef.Fig	uci.gef.FigPoly	setLocation	translate
104	uci.gef.Fig	uci.gef.FigPoly	setPoints	setPoints
105	uci.gef.Fig	uci.gef.FigPoly	setSize	setBounds
106	uci.gef.Fig	uci.gef.FigPoly	setWidth	setBounds
107	uci.gef.Fig	uci.gef.FigPoly	setX	setBounds
108	uci.gef.Fig	uci.gef.FigPoly	setY	setBounds
109	uci.gef.Fig	uci.gef.FigRect	print	paint
110	uci.gef.Fig	uci.gef.FigRRect	print	paint
111	uci.gef.Fig	uci.gef.FigSpline	align	translate
112	uci.gef.Fig	uci.gef.FigSpline	print	paint
113	uci.gef.Fig	uci.gef.FigSpline	setLocation	translate
114	uci.gef.Fig	uci.gef.FigText	print	paint
115	uci.gef.Fig	uci.uml.visual.FigActionState	createDrag	setBounds
116	uci.gef.Fig	uci.uml.visual.FigActionState	delete	setOwner
117	uci.gef.Fig	uci.uml.visual.FigActionState	setBounds	setBounds
118	uci.gef.Fig	uci.uml.visual.FigActionState	setHeight	setBounds
119	uci.gef.Fig	uci.uml.visual.FigActionState	setSize	setBounds
120	uci.gef.Fig	uci.uml.visual.FigActionState	setWidth	setBounds
121	uci.gef.Fig	uci.uml.visual.FigActionState	setX	setBounds
122	uci.gef.Fig	uci.uml.visual.FigActionState	setY	setBounds
123	uci.gef.Fig	uci.uml.visual.FigActor	createDrag	setBounds
124	uci.gef.Fig	uci.uml.visual.FigActor	delete	setOwner
125	uci.gef.Fig	uci.uml.visual.FigActor	setBounds	setBounds
126	uci.gef.Fig	uci.uml.visual.FigActor	setHeight	setBounds
127	uci.gef.Fig	uci.uml.visual.FigActor	setSize	setBounds
128	uci.gef.Fig	uci.uml.visual.FigActor	setWidth	setBounds
129	uci.gef.Fig	uci.uml.visual.FigActor	setX	setBounds
130	uci.gef.Fig	uci.uml.visual.FigActor	setY	setBounds
131	uci.gef.Fig	uci.uml.visual.FigBranchState	delete	setOwner
132	uci.gef.Fig	uci.uml.visual.FigClass	align	translate
133	uci.gef.Fig	uci.uml.visual.FigClass	delete	setOwner
134	uci.gef.Fig	uci.uml.visual.FigClass	setLocation	translate
135	uci.gef.Fig	uci.uml.visual.FigClassifierRole	createDrag	setBounds
136	uci.gef.Fig	uci.uml.visual.FigClassifierRole	delete	setOwner
137	uci.gef.Fig	uci.uml.visual.FigClassifierRole	setBounds	setBounds
138	uci.gef.Fig	uci.uml.visual.FigClassifierRole	setHeight	setBounds
139	uci.gef.Fig	uci.uml.visual.FigClassifierRole	setSize	setBounds
140	uci.gef.Fig	uci.uml.visual.FigClassifierRole	setWidth	setBounds

141	uci.gef.Fig	uci.uml.visual.FigClassifierRole	setX	setBounds
142	uci.gef.Fig	uci.uml.visual.FigClassifierRole	setY	setBounds
143	uci.gef.Fig	uci.uml.visual.FigCompartment	print	paint
144	uci.gef.Fig	uci.uml.visual.FigCompositeState	createDrag	setBounds
145	uci.gef.Fig	uci.uml.visual.FigCompositeState	delete	setOwner
146	uci.gef.Fig	uci.uml.visual.FigCompositeState	setBounds	setBounds
147	uci.gef.Fig	uci.uml.visual.FigCompositeState	setHeight	setBounds
148	uci.gef.Fig	uci.uml.visual.FigCompositeState	setSize	setBounds
149	uci.gef.Fig	uci.uml.visual.FigCompositeState	setWidth	setBounds
150	uci.gef.Fig	uci.uml.visual.FigCompositeState	setX	setBounds
151	uci.gef.Fig	uci.uml.visual.FigCompositeState	setY	setBounds
152	uci.gef.Fig	uci.uml.visual.FigEdgeModelElement	delete	setOwner
153	uci.gef.Fig	uci.uml.visual.FigFinalState	delete	setOwner
154	uci.gef.Fig	uci.uml.visual.FigForkState	createDrag	setBounds
155	uci.gef.Fig	uci.uml.visual.FigForkState	delete	setOwner
156	uci.gef.Fig	uci.uml.visual.FigForkState	setBounds	setBounds
157	uci.gef.Fig	uci.uml.visual.FigForkState	setHeight	setBounds
158	uci.gef.Fig	uci.uml.visual.FigForkState	setSize	setBounds
159	uci.gef.Fig	uci.uml.visual.FigForkState	setWidth	setBounds
160	uci.gef.Fig	uci.uml.visual.FigForkState	setX	setBounds
161	uci.gef.Fig	uci.uml.visual.FigForkState	setY	setBounds
162	uci.gef.Fig	uci.uml.visual.FigHistoryState	delete	setOwner
163	uci.gef.Fig	uci.uml.visual.FigInitialState	delete	setOwner
164	uci.gef.Fig	uci.uml.visual.FigInstance	createDrag	setBounds
165	uci.gef.Fig	uci.uml.visual.FigInstance	delete	setOwner
166	uci.gef.Fig	uci.uml.visual.FigInstance	setBounds	setBounds
167	uci.gef.Fig	uci.uml.visual.FigInstance	setHeight	setBounds
168	uci.gef.Fig	uci.uml.visual.FigInstance	setSize	setBounds
169	uci.gef.Fig	uci.uml.visual.FigInstance	setWidth	setBounds
170	uci.gef.Fig	uci.uml.visual.FigInstance	setX	setBounds
171	uci.gef.Fig	uci.uml.visual.FigInstance	setY	setBounds
172	uci.gef.Fig	uci.uml.visual.FigInterface	createDrag	setBounds
173	uci.gef.Fig	uci.uml.visual.FigInterface	delete	setOwner
174	uci.gef.Fig	uci.uml.visual.FigInterface	setBounds	setBounds
175	uci.gef.Fig	uci.uml.visual.FigInterface	setHeight	setBounds
176	uci.gef.Fig	uci.uml.visual.FigInterface	setSize	setBounds
177	uci.gef.Fig	uci.uml.visual.FigInterface	setWidth	setBounds
178	uci.gef.Fig	uci.uml.visual.FigInterface	setX	setBounds
179	uci.gef.Fig	uci.uml.visual.FigInterface	setY	setBounds
180	uci.gef.Fig	uci.uml.visual.FigJoinState	createDrag	setBounds
181	uci.gef.Fig	uci.uml.visual.FigJoinState	delete	setOwner

182	uci.gef.Fig	uci.uml.visual.FigJoinState	setBounds	setBounds
183	uci.gef.Fig	uci.uml.visual.FigJoinState	setHeight	setBounds
184	uci.gef.Fig	uci.uml.visual.FigJoinState	setSize	setBounds
185	uci.gef.Fig	uci.uml.visual.FigJoinState	setWidth	setBounds
186	uci.gef.Fig	uci.uml.visual.FigJoinState	setX	setBounds
187	uci.gef.Fig	uci.uml.visual.FigJoinState	setY	setBounds
188	uci.gef.Fig	uci.uml.visual.FigMessage	createDrag	setBounds
189	uci.gef.Fig	uci.uml.visual.FigMessage	setBounds	setBounds
190	uci.gef.Fig	uci.uml.visual.FigMessage	setHeight	setBounds
191	uci.gef.Fig	uci.uml.visual.FigMessage	setSize	setBounds
192	uci.gef.Fig	uci.uml.visual.FigMessage	setWidth	setBounds
193	uci.gef.Fig	uci.uml.visual.FigMessage	setX	setBounds
194	uci.gef.Fig	uci.uml.visual.FigMessage	setY	setBounds
195	uci.gef.Fig	uci.uml.visual.FigNodeModelElement	delete	setOwner
196	uci.gef.Fig	uci.uml.visual.FigNodeWithCompartments	createDrag	setBounds
197	uci.gef.Fig	uci.uml.visual.FigNodeWithCompartments	setBounds	setBounds
198	uci.gef.Fig	uci.uml.visual.FigNodeWithCompartments	setHeight	setBounds
199	uci.gef.Fig	uci.uml.visual.FigNodeWithCompartments	setSize	setBounds
200	uci.gef.Fig	uci.uml.visual.FigNodeWithCompartments	setWidth	setBounds
201	uci.gef.Fig	uci.uml.visual.FigNodeWithCompartments	setX	setBounds
202	uci.gef.Fig	uci.uml.visual.FigNodeWithCompartments	setY	setBounds
203	uci.gef.Fig	uci.uml.visual.FigPackage	createDrag	setBounds
204	uci.gef.Fig	uci.uml.visual.FigPackage	delete	setOwner
205	uci.gef.Fig	uci.uml.visual.FigPackage	setBounds	setBounds
206	uci.gef.Fig	uci.uml.visual.FigPackage	setHeight	setBounds
207	uci.gef.Fig	uci.uml.visual.FigPackage	setSize	setBounds
208	uci.gef.Fig	uci.uml.visual.FigPackage	setWidth	setBounds
209	uci.gef.Fig	uci.uml.visual.FigPackage	setX	setBounds
210	uci.gef.Fig	uci.uml.visual.FigPackage	setY	setBounds
211	uci.gef.Fig	uci.uml.visual.FigState	createDrag	setBounds
212	uci.gef.Fig	uci.uml.visual.FigState	delete	setOwner
213	uci.gef.Fig	uci.uml.visual.FigState	setBounds	setBounds
214	uci.gef.Fig	uci.uml.visual.FigState	setHeight	setBounds
215	uci.gef.Fig	uci.uml.visual.FigState	setSize	setBounds
216	uci.gef.Fig	uci.uml.visual.FigState	setWidth	setBounds
217	uci.gef.Fig	uci.uml.visual.FigState	setX	setBounds
218	uci.gef.Fig	uci.uml.visual.FigState	setY	setBounds
219	uci.gef.Fig	uci.uml.visual.FigUseCase	createDrag	setBounds
220	uci.gef.Fig	uci.uml.visual.FigUseCase	delete	setOwner
221	uci.gef.Fig	uci.uml.visual.FigUseCase	setBounds	setBounds
222	uci.gef.Fig	uci.uml.visual.FigUseCase	setHeight	setBounds

223	uci.gef.Fig	uci.uml.visual.FigUseCase	setSize	setBounds
224	uci.gef.Fig	uci.uml.visual.FigUseCase	setWidth	setBounds
225	uci.gef.Fig	uci.uml.visual.FigUseCase	setX	setBounds
226	uci.gef.Fig	uci.uml.visual.FigUseCase	setY	setBounds
227	uci.gef.FigPoly	uci.gef.FigSpline	setEndPoints	moveVertex
228	uci.gef.FigPoly	uci.gef.FigSpline	setPoints	moveVertex
229	uci.gef.Guide	uci.gef.GuideGrid	snapTo	snap
230	uci.gef.Layer	uci.gef.LayerDiagram	elementsIn	elements
231	uci.gef.Layer	uci.gef.LayerDiagram	getContentsEdgesOnly	getContents
232	uci.gef.Layer	uci.gef.LayerDiagram	getContentsNoEdges	getContents
233	uci.gef.Layer	uci.gef.LayerDiagram	nodesIn	elements
234	uci.gef.Layer	uci.gef.LayerDiagram	paint	paintContents
235	uci.gef.Layer	uci.gef.LayerDiagram	paintGrayContents	paintContents
236	uci.gef.Layer	uci.gef.LayerGrid	getContentsEdgesOnly	getContents
237	uci.gef.Layer	uci.gef.LayerGrid	getContentsNoEdges	getContents
238	uci.gef.Layer	uci.gef.LayerGrid	paint	paintContents
239	uci.gef.Layer	uci.gef.LayerGrid	paintGrayContents	paintContents
240	uci.gef.Layer	uci.gef.LayerPageBreaks	getContentsEdgesOnly	getContents
241	uci.gef.Layer	uci.gef.LayerPageBreaks	getContentsNoEdges	getContents
242	uci.gef.Layer	uci.gef.LayerPageBreaks	paint	paintContents
243	uci.gef.Layer	uci.gef.LayerPageBreaks	paintGrayContents	paintContents
244	uci.gef.Mode	uci.gef.ModeBroom	print	paint
245	uci.gef.Mode	uci.gef.ModeCreate	print	paint
246	uci.gef.Mode	uci.gef.ModeCreate	setEditor	getInitialCursor
247	uci.gef.Mode	uci.gef.ModeModify	print	paint
248	uci.gef.Mode	uci.gef.ModePlace	print	paint
249	uci.gef.Mode	uci.gef.ModePlace	setEditor	getInitialCursor
250	uci.gef.Mode	uci.gef.ModeSelect	print	paint
251	uci.gef.PathConv	uci.gef.PathConvPercent	getPoint	stuffPoint
252	uci.gef.PathConv	uci.gef.PathConvPercentPlusConst	getPoint	stuffPoint
253	uci.gef.Selection	uci.gef.SelectionLowerRight	contains	hitHandle
254	uci.gef.Selection	uci.gef.SelectionLowerRight	hit	hitHandle
255	uci.gef.Selection	uci.gef.SelectionLowerRight	hitHandle	hitHandle
256	uci.gef.Selection	uci.gef.SelectionMove	contains	hitHandle
257	uci.gef.Selection	uci.gef.SelectionMove	hit	hitHandle
258	uci.gef.Selection	uci.gef.SelectionMove	hitHandle	hitHandle
259	uci.gef.Selection	uci.gef.SelectionNoop	contains	hitHandle
260	uci.gef.Selection	uci.gef.SelectionNoop	hit	hitHandle
261	uci.gef.Selection	uci.gef.SelectionNoop	hitHandle	hitHandle
262	uci.gef.Selection	uci.gef.SelectionReshape	contains	hitHandle
263	uci.gef.Selection	uci.gef.SelectionReshape	hit	hitHandle

264	uci.gef.Selection	uci.gef.SelectionReshape	hitHandle	hitHandle
265	uci.gef.Selection	uci.gef.SelectionResize	contains	hitHandle
266	uci.gef.Selection	uci.gef.SelectionResize	hit	hitHandle
267	uci.gef.Selection	uci.gef.SelectionResize	hitHandle	hitHandle
268	uci.gef.Selection	uci.uml.visual.SelectionActor	contains	hitHandle
269	uci.gef.Selection	uci.uml.visual.SelectionActor	hit	hitHandle
270	uci.gef.Selection	uci.uml.visual.SelectionActor	hitHandle	hitHandle
271	uci.gef.Selection	uci.uml.visual.SelectionClass	contains	hitHandle
272	uci.gef.Selection	uci.uml.visual.SelectionClass	hit	hitHandle
273	uci.gef.Selection	uci.uml.visual.SelectionClass	hitHandle	hitHandle
274	uci.gef.Selection	uci.uml.visual.SelectionInterface	contains	hitHandle
275	uci.gef.Selection	uci.uml.visual.SelectionInterface	hit	hitHandle
276	uci.gef.Selection	uci.uml.visual.SelectionInterface	hitHandle	hitHandle
277	uci.gef.Selection	uci.uml.visual.SelectionState	contains	hitHandle
278	uci.gef.Selection	uci.uml.visual.SelectionState	hit	hitHandle
279	uci.gef.Selection	uci.uml.visual.SelectionState	hitHandle	hitHandle
280	uci.gef.Selection	uci.uml.visual.SelectionUseCase	contains	hitHandle
281	uci.gef.Selection	uci.uml.visual.SelectionUseCase	hit	hitHandle
282	uci.gef.Selection	uci.uml.visual.SelectionUseCase	hitHandle	hitHandle
283	uci.uml.Foundation.Core.ElementImpl	uci.uml.Behavioral_Elements.Collaborations.Assc	addCharacteristic	fireVetoableChange
284	uci.uml.Foundation.Core.ElementImpl	uci.uml.Behavioral_Elements.Collaborations.Assc	addTaggedValue	fireVetoableChange
285	uci.uml.Foundation.Core.ElementImpl	uci.uml.Behavioral_Elements.Collaborations.Assc	fireVetoableChange	fireVetoableChange
286	uci.uml.Foundation.Core.ElementImpl	uci.uml.Behavioral_Elements.Collaborations.Assc	removeCharacteristic	fireVetoableChange
287	uci.uml.Foundation.Core.ElementImpl	uci.uml.Behavioral_Elements.Collaborations.Assc	removeTaggedValue	fireVetoableChange
288	uci.uml.Foundation.Core.ElementImpl	uci.uml.Behavioral_Elements.Collaborations.Assc	setClassification	fireVetoableChange
289	uci.uml.Foundation.Core.ElementImpl	uci.uml.Behavioral_Elements.Collaborations.Assc	setName	fireVetoableChange
290	uci.uml.Foundation.Core.ElementImpl	uci.uml.Behavioral_Elements.Common_Behavior.	dbgString	getOCLTypeStr
291	uci.uml.Foundation.Core.ElementImpl	uci.uml.Behavioral_Elements.Common_Behavior.	dbgString	getOCLTypeStr
292	uci.uml.Foundation.Core.ElementImpl	uci.uml.Foundation.Core.AssociationEnd	addCharacteristic	fireVetoableChange
293	uci.uml.Foundation.Core.ElementImpl	uci.uml.Foundation.Core.AssociationEnd	addTaggedValue	fireVetoableChange
294	uci.uml.Foundation.Core.ElementImpl	uci.uml.Foundation.Core.AssociationEnd	fireVetoableChange	fireVetoableChange
295	uci.uml.Foundation.Core.ElementImpl	uci.uml.Foundation.Core.AssociationEnd	removeCharacteristic	fireVetoableChange
296	uci.uml.Foundation.Core.ElementImpl	uci.uml.Foundation.Core.AssociationEnd	removeTaggedValue	fireVetoableChange
297	uci.uml.Foundation.Core.ElementImpl	uci.uml.Foundation.Core.AssociationEnd	setClassification	fireVetoableChange
298	uci.uml.Foundation.Core.ElementImpl	uci.uml.Foundation.Core.AssociationEnd	setName	fireVetoableChange
299	uci.uml.Foundation.Core.ElementImpl	uci.uml.Foundation.Core.Feature	addCharacteristic	fireVetoableChange
300	uci.uml.Foundation.Core.ElementImpl	uci.uml.Foundation.Core.Feature	addTaggedValue	fireVetoableChange
301	uci.uml.Foundation.Core.ElementImpl	uci.uml.Foundation.Core.Feature	fireVetoableChange	fireVetoableChange
302	uci.uml.Foundation.Core.ElementImpl	uci.uml.Foundation.Core.Feature	removeCharacteristic	fireVetoableChange
303	uci.uml.Foundation.Core.ElementImpl	uci.uml.Foundation.Core.Feature	removeTaggedValue	fireVetoableChange
304	uci.uml.Foundation.Core.ElementImpl	uci.uml.Foundation.Core.Feature	setClassification	fireVetoableChange

305	uci.uml.Foundation.Core.ElementImpl	uci.uml.Foundation.Core.Feature	setName	fireVetoableChange
306	uci.uml.generate.Generator	uci.uml.generate.GeneratorDisplay	generate	generateOperation
307	uci.uml.generate.Generator	uci.uml.generate.GeneratorDisplay	generate	generateAttribute
308	uci.uml.generate.Generator	uci.uml.generate.GeneratorDisplay	generate	generateParameter
309	uci.uml.generate.Generator	uci.uml.generate.GeneratorDisplay	generate	generatePackage
310	uci.uml.generate.Generator	uci.uml.generate.GeneratorDisplay	generate	generateClassifier
311	uci.uml.generate.Generator	uci.uml.generate.GeneratorDisplay	generate	generateStereotype
312	uci.uml.generate.Generator	uci.uml.generate.GeneratorDisplay	generate	generateTaggedValue
313	uci.uml.generate.Generator	uci.uml.generate.GeneratorDisplay	generate	generateAssociation
314	uci.uml.generate.Generator	uci.uml.generate.GeneratorDisplay	generate	generateAssociationEnd
315	uci.uml.generate.Generator	uci.uml.generate.GeneratorDisplay	generate	generateMultiplicity
316	uci.uml.generate.Generator	uci.uml.generate.GeneratorDisplay	generate	generateState
317	uci.uml.generate.Generator	uci.uml.generate.GeneratorDisplay	generate	generateTransition
318	uci.uml.generate.Generator	uci.uml.generate.GeneratorDisplay	generate	generateAction
319	uci.uml.generate.Generator	uci.uml.generate.GeneratorDisplay	generate	generateGuard
320	uci.uml.generate.Generator	uci.uml.generate.GeneratorJava	generate	generateOperation
321	uci.uml.generate.Generator	uci.uml.generate.GeneratorJava	generate	generateAttribute
322	uci.uml.generate.Generator	uci.uml.generate.GeneratorJava	generate	generateParameter
323	uci.uml.generate.Generator	uci.uml.generate.GeneratorJava	generate	generatePackage
324	uci.uml.generate.Generator	uci.uml.generate.GeneratorJava	generate	generateClassifier
325	uci.uml.generate.Generator	uci.uml.generate.GeneratorJava	generate	generateStereotype
326	uci.uml.generate.Generator	uci.uml.generate.GeneratorJava	generate	generateTaggedValue
327	uci.uml.generate.Generator	uci.uml.generate.GeneratorJava	generate	generateAssociation
328	uci.uml.generate.Generator	uci.uml.generate.GeneratorJava	generate	generateAssociationEnd
329	uci.uml.generate.Generator	uci.uml.generate.GeneratorJava	generate	generateMultiplicity
330	uci.uml.generate.Generator	uci.uml.generate.GeneratorJava	generate	generateState
331	uci.uml.generate.Generator	uci.uml.generate.GeneratorJava	generate	generateTransition
332	uci.uml.generate.Generator	uci.uml.generate.GeneratorJava	generate	generateAction
333	uci.uml.generate.Generator	uci.uml.generate.GeneratorJava	generate	generateGuard
334	uci.uml.ui.ProjectMember	uci.uml.ui.ProjectMemberDiagram	getName	getFileExtension
335	uci.uml.ui.ProjectMember	uci.uml.ui.ProjectMemberModel	getName	getFileExtension
336	uci.uml.ui.ProjectMember	uci.uml.ui.ProjectMemberModel	getURL	getName
337	uci.uml.ui.props.PropPanel	uci.uml.ui.props.PropPanelAssociation	removeUpdate	insertUpdate
338	uci.uml.ui.props.PropPanel	uci.uml.ui.props.PropPanelClass	removeUpdate	insertUpdate
339	uci.uml.ui.props.PropPanel	uci.uml.ui.props.PropPanelInterface	removeUpdate	insertUpdate
340	uci.uml.ui.style.StylePanel	uci.uml.ui.style.SPFigEdgeModelElement	removeUpdate	insertUpdate
341	uci.uml.ui.style.StylePanel	uci.uml.ui.style.SPFigEdgeModelElement	setTarget	refresh
342	uci.uml.ui.style.StylePanel	uci.uml.ui.style.StylePanelFig	removeUpdate	insertUpdate
343	uci.uml.ui.style.StylePanel	uci.uml.ui.style.StylePanelFig	setTarget	refresh
344	uci.uml.ui.style.StylePanel	uci.uml.ui.style.StylePanelFigClass	removeUpdate	insertUpdate
345	uci.uml.ui.style.StylePanel	uci.uml.ui.style.StylePanelFigClass	setTarget	refresh

346	uci.uml.ui.TabSpawnable	uci.uml.ui.TabDiagram	spawn	clone
347	uci.uml.ui.todo.WizStep	uci.uml.ui.todo.WizDescription	refresh	setTarget
348	uci.uml.ui.UMLAction	uci.uml.ui.ActionActivityDiagram	updateEnabled	shouldBeEnabled
349	uci.uml.ui.UMLAction	uci.uml.ui.ActionAddAttribute	updateEnabled	shouldBeEnabled
350	uci.uml.ui.UMLAction	uci.uml.ui.ActionAddInternalTrans	updateEnabled	shouldBeEnabled
351	uci.uml.ui.UMLAction	uci.uml.ui.ActionAddMessage	updateEnabled	shouldBeEnabled
352	uci.uml.ui.UMLAction	uci.uml.ui.ActionAddOperation	updateEnabled	shouldBeEnabled
353	uci.uml.ui.UMLAction	uci.uml.ui.ActionAggregation	updateEnabled	shouldBeEnabled
354	uci.uml.ui.UMLAction	uci.uml.ui.ActionCopy	updateEnabled	shouldBeEnabled
355	uci.uml.ui.UMLAction	uci.uml.ui.ActionCut	updateEnabled	shouldBeEnabled
356	uci.uml.ui.UMLAction	uci.uml.ui.ActionDeleteFromDiagram	updateEnabled	shouldBeEnabled
357	uci.uml.ui.UMLAction	uci.uml.ui.ActionEmptyTrash	updateEnabled	shouldBeEnabled
358	uci.uml.ui.UMLAction	uci.uml.ui.ActionGenerateOne	updateEnabled	shouldBeEnabled
359	uci.uml.ui.UMLAction	uci.uml.ui.ActionGoToCritique	updateEnabled	shouldBeEnabled
360	uci.uml.ui.UMLAction	uci.uml.ui.ActionGoToEdit	updateEnabled	shouldBeEnabled
361	uci.uml.ui.UMLAction	uci.uml.ui.ActionNavForw	updateEnabled	shouldBeEnabled
362	uci.uml.ui.UMLAction	uci.uml.ui.ActionPaste	updateEnabled	shouldBeEnabled
363	uci.uml.ui.UMLAction	uci.uml.ui.ActionProperties	updateEnabled	shouldBeEnabled
364	uci.uml.ui.UMLAction	uci.uml.ui.ActionRemoveFromModel	updateEnabled	shouldBeEnabled
365	uci.uml.ui.UMLAction	uci.uml.ui.ActionSaveProject	updateEnabled	shouldBeEnabled
366	uci.uml.ui.UMLAction	uci.uml.ui.ActionStateDiagram	updateEnabled	shouldBeEnabled
367	uci.uml.ui.UMLAction	uci.uml.ui.ActionUndo	updateEnabled	shouldBeEnabled
368	uci.uml.visual.SelectionWButtons	uci.uml.visual.SelectionActor	mouseReleased	buttonClicked
369	uci.uml.visual.SelectionWButtons	uci.uml.visual.SelectionActor	paint	paintButtons
370	uci.uml.visual.SelectionWButtons	uci.uml.visual.SelectionClass	mouseReleased	buttonClicked
371	uci.uml.visual.SelectionWButtons	uci.uml.visual.SelectionClass	paint	paintButtons
372	uci.uml.visual.SelectionWButtons	uci.uml.visual.SelectionInterface	mouseReleased	buttonClicked
373	uci.uml.visual.SelectionWButtons	uci.uml.visual.SelectionInterface	paint	paintButtons
374	uci.uml.visual.SelectionWButtons	uci.uml.visual.SelectionState	mouseReleased	buttonClicked
375	uci.uml.visual.SelectionWButtons	uci.uml.visual.SelectionState	paint	paintButtons
376	uci.uml.visual.SelectionWButtons	uci.uml.visual.SelectionUseCase	mouseReleased	buttonClicked
377	uci.uml.visual.SelectionWButtons	uci.uml.visual.SelectionUseCase	paint	paintButtons
378	uci.uml.visual.UMLDiagram	uci.uml.visual.UMLActivityDiagram	initialize	setNamespace
379	uci.uml.visual.UMLDiagram	uci.uml.visual.UMLClassDiagram	initialize	setNamespace
380	uci.uml.visual.UMLDiagram	uci.uml.visual.UMLCollaborationDiagram	initialize	setNamespace
381	uci.uml.visual.UMLDiagram	uci.uml.visual.UMLStateDiagram	initialize	setNamespace