RESOLVER

BOOLEAN SATISFIABILITY SOLVER BASED ON A GENETIC ALGORITHM

Architectural Documentation

Team Imperium

Dewald de Jager
Ernst Eksteen
Vignesh Iyer
Regan Koopmans
Craig van Heerden

October 17, 2017

Contents

1 Description of the Type of System and Subsystem(s)

1.1 Transformational

The core functionality of the software, namely the genetic local search algorithm, is part of a *Transformational System*. The user of the system is expected to specify/adjust various control parameters and the formula file as input for the the implementation of the algorithm. This is followed a variety of time-intensive computations - a population of individuals representing possible assignments (candidate solutions) to the formula (problem), selection of parent individuals, application of cross-over to produce offspring, tabu-search to improve the offspring and condition-based insertion of the offspring into the population.

These procedures will be performed iteratively until the formula is satisfied or an upper-bound on the number of iterations is reached without a solution. During the processing, the user receives progress-based and statistically-based information from the system from the system, but the link is uni-directional (the user cannot communicate to the system during the processing).

1.2 Interactive and Event-Driven

The system is event-driven and interactive in that it will wait until it receives a request from the user until it begins solving when in "server" mode. This means that it receives events and controls from an external entity in the form of the user-interface. The user can also prematurely terminate a solving instance, and thus interrupt the usual flow of the program.

2 Description of the Architectural Styles

2.1 Client-Server

The client-server architectural design sits at the core of our architectural design. We have made the clear distinction between presentation and operation, which has naturally lead to the emergence of the client-server architecture in our project.

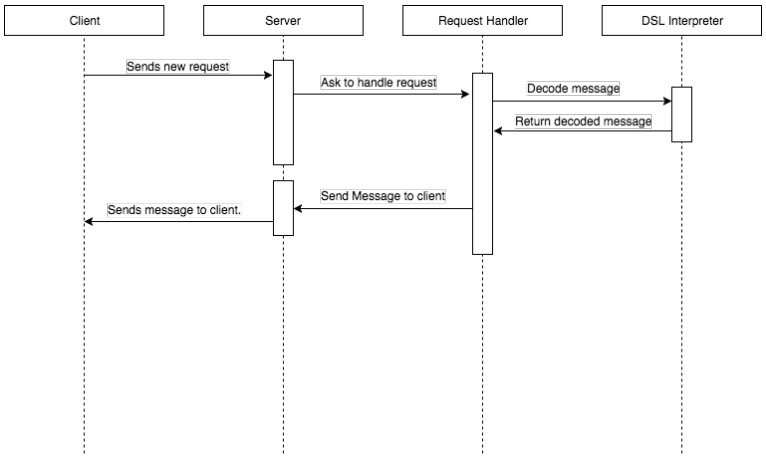

Figure 1: Sequence Diagram of Request Chain

2.2 N-Tier

The system is designed to be organized logically into levels of specificity. That is, on the very lowest level, we have strings of bits which represent candidates in the , which is encapsulated by the Individual module.

At the very highest level, there is simply the problem (i.e the formula) and the mechanism to solve it (the GASAT function exposed by the Genetic Algorithm module).

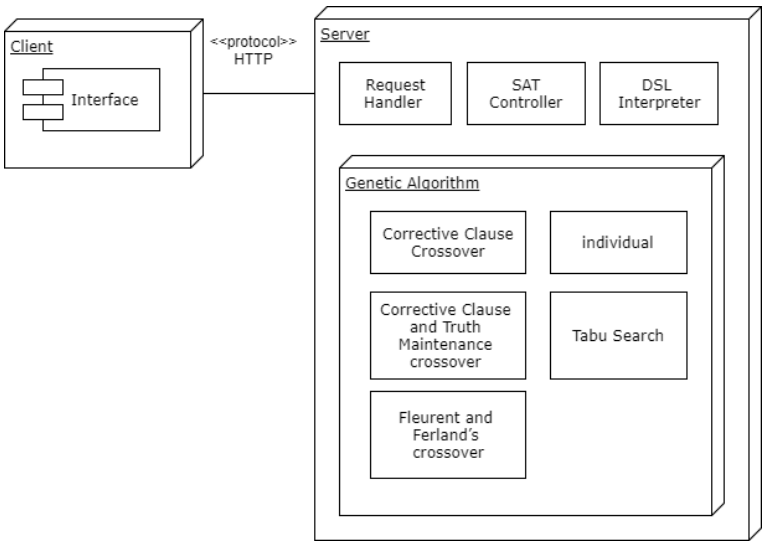

Figure 2: Deployment Diagram of System

3 User Interface Design

The solving core makes no assumptions about the nature of an interface, and therefore places no constraints other than conforming the domain-specific language (DSL) we have proposed for this task specifically. This liberates the design of the user interface, and allows us to focus on usability without fundamental concern of meeting self-imposed restraints.

We wish to construct the primary user-interface into the system using hierarchical information structures. In other words, the interface can relay information to the degree that the user desires. If, for instance, a user . Expanding menu options and tool-tips.

4 Design Patterns

Design Patterns play an important role in the design of the proposed software. In this section we will investigate the important design patterns that are exhibited in the design of our system.

4.1 GRASP Patterns

4.1.1 Controller

RequestHandler is a Creator. RequestHandler initiates the construction of the GA instance using information obtained through the server

4.1.2 Creator

SATController is a creator in our system. SATController creates. This is because.

4.2 GoF Patterns

4.2.1 Observer

SATController is an Observable class from which interfaces can observe. These interface observers may attach at any point. Information regarding the state of solving is regularly pushed to the observers.

4.2.2 Interpreter

RequestHandler is an Interpreter, in that it obtains and a request in the form a DSL made for server-client communication, and understands how to fulfill this request in terms of concrete operations in the solving system.

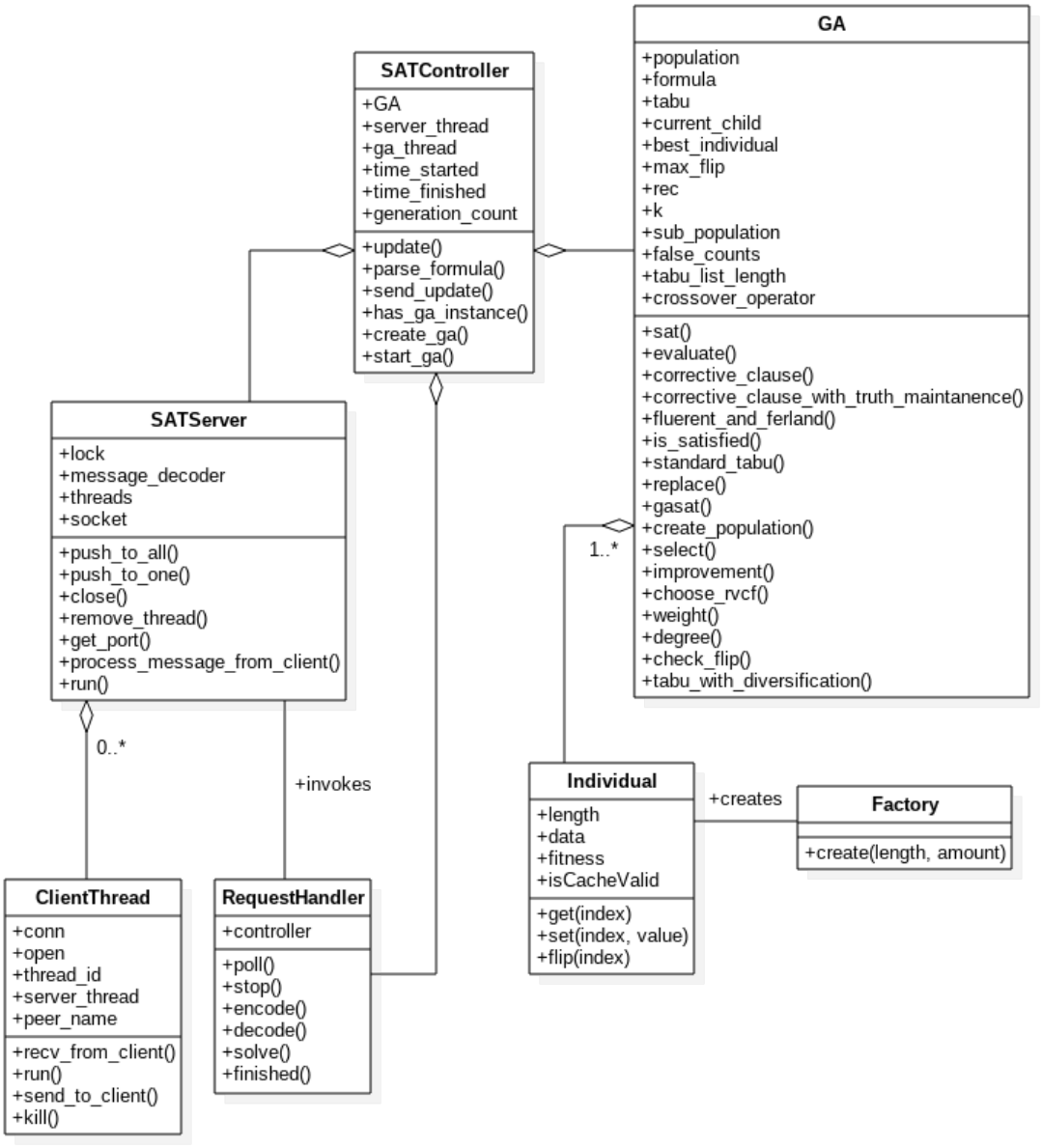

Figure 3: Class Diagram of System

4.2.3 Factory

The individual modules defines a helper class, Factory, for the creation of populations of Individuals. Although a very simple class, it offers an established structured way to create sets of individuals, removing this responsibility from any class that wishes to do this.

4.2.4 Singleton

SATController is a Singleton. This ensures that all modules (in particular GA and SATServer) in the running program have reference to the same running SATController instance.

4.2.5 Template Method

The cross-over method in our system is a Template Method. On a high level, the cross-over function is given two parents and it yields a single new candidate solution. The particular cross-over function (i.e Corrective Clause, Fleurent & Ferland etc.) need not be known to the rest of

the system, and indeed, it is encapsulated in the variable *crossover_operator* for the rest of the system.

5 Software Design Principles

5.1 Separation of Concerns

In our architectural design of the system, we have incorporated separation of concerns by splitting code in explicit modules, grouped by general function. Internal to the modules, our methods have been designed to be largely independent of another, and are often static, needing only the information that is passed as direct parameters.

5.2 Design for Change

The system has been designed to facilitate and adapt to anticipated changes. Researchers using this tool may be inclined to experiment and have a sufficient level of control to create novel scientific findings. The system must therefore be abstract enough to allow for paradigm shifts in the understanding of boolean satisfiability, and still be able to provide useful results.

5.3 Low Coupling

The system design exhibits low coupling, particularly in the relationship between the interface and the core SAT functionality. The interface, by knowing only the simple shared DSL and making standard network calls, can take full advantage of the core features and receive all relevant information in return. Further, there is no specification on how this returning information is meant to be displayed.

This outcomes together mean that it is incredibly easy to write and modify a novel interface, which might offer a greatly different perspective than the interface currently in development. This low coupling also has the consequence that the interface and the running SAT instance can be geographically separated, and running on disparately powerful machines. This opens up opportunities for running the SAT instance on a more powerful, remote machine, and having an interface monitor and interact with it over simple network calls.

5.4 Simplicity

We have taking great care to make the system as simple as possible. Our project specifications, although comprehensive, has been vague at times. In these cases, we have always chosen the most simple interpretation of our instructions, to avoid convolution of what is essentially a very simple mathematical problem.

The architecture described in this paper is one that is minimally specific, while being precise enough to have well-defined partitioned responsibilities for modules and classes.

6 Non-Functional Requirements

6.1 Availability

The server which hosts the logic of the Genetic Algorithm must be available to connect locally, in addition to being operational on a cluster, to cover the case where a user is unable to connect to the cluster.

6.2 Modifiability

The system must allow for easy structural changes of the code, particularly by means of implementing design patterns for easy modification to the GASAT algorithm without having to re-design the system.

6.3 Extensibility

New features and upgrades, such as the introduction of new cross-over and selection operators, must easily be addable in the future. The architectural design of the system should support the ability to introduce these software changes and not disrupt software artifacts for which these changes do not concern.

6.4 Usability

The system must be cater to the needs of both researchers, problem-solvers and beginners by providing a balance between easy learn-ability, satisfaction in the user interface but .

6.5 Portability

The software should ideally function uniformly across multiple architectures and operating systems, so as to appeal to the widest possible audience of researchers, problem-solvers and students. The system may also be seen to be portable in the sense that the Conjunctive Normal Form DIMACS file-format remains constant across platforms.

The client-server architecture is valuable once again in the perspective of portability. Indeed it is true that a solving core may be running on one operating system, and that the interface may be running on another, separated geographically and communicating over a network. The user may also alternatively interact with the solving core through a web-interface through a browser, in which case the interface is platform-less, and the details of the solving core are unknown.

6.6 Correctness

The proposed software is, first and foremost, a tool for researchers of boolean satisfiability. To this end, the SAT Solver must fulfill the internal mathematical consistency properties by application of the SOFL method for design. The system must be proven to function correctly, both empirically and experimentally.

6.7 Reliability

Although the functionality of the genetic algorithm . The functioning of the SAT solver should also not significantly deteriorate as the input formula becomes larger and more complex.