On using evolutionary algorithms for solving the functional composition problem

Dennis Schwerdel, Bernd Reuther, Paul Müller
University of Kaiserslautern, Germany
{schwerdel, reuther, pmueller}@informatik.uni-kl.de

I. Introduction

The Internet has aged and outgrown its original purpose, today it has a lot of problems. In theory, most of these problems could be solved by modifying existing protocols, introducing new protocols or changing protocol interaction. The real problem of the current Internet architecture is that it offers no practical way to introduce changes into a deployed network. The available protocols and their functionality and interaction cannot be changed easily, because most changes require a global synchronization that is nearly impossible due to the size of the Internet.

A future Internet architecture needs mechanisms to support protocol evolution: New protocols or improved versions of existing protocols should be easy to introduce and should be automatically used for communication.

The next two subsections contain short introductions to the functional composition problem and evolutionary algorithms. Section 2 describes the proposed algorithm. A discussion of open issues is contained in section 3 and section 4 shows the next steps.

A. Functional composition

In a flexible networking architecture, networking functionality is often encapsulated as so-called building blocks. These blocks are independent and can be combined to form a more complex functionality like a communication stack. This combination of building blocks is called functional composition. Finding an optimal combination is a hard problem since the number of possibilities is exponential and there is no trivial way to reduce the size of the candidate set.

B. Evolutionary algorithms

Evolutionary algorithms are designed after the model for biological evolution, i.e. how a certain population advances under evolutionary pressure.

In computer science evolutionary algorithms are used to solve problems that are hard to solve analytically. In an evolutionary algorithm the individuals of the population are proposed solutions for a certain problem. An objective function is used to compare and rank all individuals of the population.

The evolutionary algorithms performs its calculation in steps called "generations". In each generation, the population is increased by adding new individuals and then again reduced to the normal population size by retaining only the best individuals. This way the population is enhancing over time.

Methods to create new individuals to increase the population normally modify exiting individuals by mutation or combination. Mutation slightly changes one individual by a certain factor, that can either be fixed, decreasing with time or depend on a random distribution. Combination uses qualities from two individuals with equal probability to form a new individual.

II. Approach

To calculate workflow graphs the following algorithm is proposed and has been implemented: In the beginning the population is filled using generators that do not require existing individuals. Then the following steps are repeated until a good workflow graph has been found or a time limit has been reached:

1) Increase the population using so called "generators".
2) Rank the individuals using the evaluation process.
3) Correct errors found during the evaluation using a component called "workflow-fixer" and re-evaluate the enhanced individuals.
4) Decrease the population by selecting only the best individuals based on the ranking obtained during evaluation.
5) Check the termination criteria.

A. Dependency model

To be able to communicate, the building blocks in this approach offer certain interaction points called "ports", which can be interconnected to allow communication between the building blocks. The graph that results from the building blocks, their ports and the connections between them is called "workflow".

Building blocks can define limits on minimal or maximal connections on their ports. The building blocks communicate with messages of arbitrary types, so a very basic requirement for a valid connection is that the data types must be compatible and that a port must be able to receive all messages that can be sent by any connected remote port. Also building blocks can require that remote ports must be able to send a certain message type.

To describe dependencies between building blocks, each port of a building block has attributes, which are named numeric values. These attribute values can be fixed values or can depend on attributes of connected remote ports or even remote ports connected to other ports of the same building block. When attributes depend on other attributes they must be dynamically calculated which requires the workflow to be complete.

Building blocks can specify constraints on the attributes a valid remote port must have.

An example for an attribute would be the MTU. A building block that adds a 20 byte header between the ports “in” and “out” would specify the value of MTU for the port “in” like that: $in.MTU = min(r.MTU - 20 \,|\, r \in remote(out))$. Additionally this building block could specify the following constraint: $\forall r \in remote(out) : r.MTU \geq 20$.

The semantic of an attribute is tied to the name and must be regarded by the building block designer when using or defining the attribute value.

B. Evaluation

The evaluation of a workflow has three steps:

Syntactical analysis: In this phase the workflow graph is checked for connectivity, it is checked that the workflow contains at least one data source and one data sink (e.g. application and network) and that paths between sources and sinks exist. Then all connections are checked for type mismatches and finally connection limits are checked. The result is the number of syntax errors.

Attribute constraints: This phase is only executed if no syntax errors have been detected. In this phase all attribute constraints are checked and the result is the sum of the number of constraint violations and the number of attribute calculation errors (e.g. dependency loops). In this phase the hard requirements of the user, the application and the network can be injected as attribute constraints of the data sources and sinks.

Objective function: This last phase is only executed when no errors have been detected so far. This phase calculates an objective value using the attributes present at the data sources and data sinks. An objective function combines those attributes using given weights that represent the soft requirements of the user, the application and the network.

The resulting 3-tuples in the form of (syntax errors, constraint violations, objective value) are compared from left to right, forming a complete order of all workflow graphs.

C. Generators

To create new workflow graphs for the population a lot of methods can be used:

Random generation: This generator creates completely random workflow graphs by selecting a random number of building blocks and creating random but type-matching connections between ports.

Cached results: This generator merely returns workflow graphs that have been the result of other (similar) calculations.

Pattern-based generation: This generator uses common patterns (either given by a designer or recognized in cached workflows) to form new workflows.

Mutation: This generator uses individuals from the population and slightly changes them by adding or removing connections and building blocks. The mutation factor decreases with time.

Combination: This generator uses two individuals and merges them.

For the initial population only the upper three generators can be used and during calculation mostly mutation and combination are used. The share that each generator has in creating new individuals can be configured and is open for further research.

Any other way to create workflow graphs can be plugged in easily as a generator which will enrich the population and might lead faster to better results.

D. Workflow-Fixer

As the random generators create invalid workflow graphs with a high probability, a special component has been introduced to the generation process that corrects small errors. The workflow-fixer searches for obvious errors in workflows, like message type mismatches or unconnected graph parts. These errors are corrected by modifying the graph accordingly.

Using the workflow-fixer a lot of workflow graphs can be enhanced significantly and thus the quality of the whole population is increased. It is important to note that the workflow-fixer works in an indeterministic way and does only change the graph slightly so the random nature of the evolutionary algorithm is preserved.

III. Discussion

An important point for functional composition is the resource consumption (i.e. computation time). This determines both at what time and on which hardware the composition can be executed. Current plans with the composition only regard end-systems and end-to-end communication so embedded or network hardware is currently not considered.

At this early stage of development the final resource consumption of the algorithm can not be estimated. Small changes to parameters can yield great performance increases. It might be possible to run the algorithm upon connection initialization.

This mainly depends on a few factors:

- How the building blocks are described (Lots of constraints vs. complex objective functions).
- How good the individuals in the initial population are (Good heuristics, cached results to similar requests).

Even if the performance does not allow the full algorithm to run on connection initialization it can still be used to slightly adapt a set of given solutions and select the best of them. Also it can be used at design time to pre-calculate workflows using much more resources than are available at connection initialization.

IV. Outlook

As stated in the previous sections the algorithm offers a lot of parameters that can be tweaked for performance. So one of the next tasks will be to find good values for those parameters in realistic use cases.

The approach will be tested using a protocol framework and some real protocol implementations in the next months.