Using Genetic Programming to Generate Protocol Adaptors for Interprocess Communication

Werner Van Belle*, Tom Mens**, and Theo D'Hondt

Programming Technology Lab, Vrije Universiteit Brussel,
Pleinlaan 2, 1050 Brussel, Belgium
{werner.van.belle,tom.mens,tjdhondt}@vub.ac.be
http://prog.vub.ac.be

Abstract. As mobile devices become more powerful, interprocess communication becomes increasingly more important. Unfortunately, this larger freedom of mobility gives rise to unknown environments. In these environments, processes that want to communicate with each other will be unable to do so because of protocol conflicts. Although conflicting protocols can be remedied by using adaptors, the number of possible combinations of different protocols increases dramatically. Therefore we propose a technique to generate protocol adaptors automatically. This is realised by means of genetically engineered classifier systems that use Petri nets as a specification for the underlying protocols. This paper reports on an experiment that validates this approach.

1 Introduction

In the field of evolvable computing, software (and hardware) is developed that adapts itself to new runtime environments as necessary. The runtime environments targetted in this paper are open distributed systems in which interprocess communication forms an essential problem. In these environments an application consists of processes that communicate with other processes to reach specific goals.

With the advent of mobile devices these processes do not necessarily know in which kind of runtime environments they will execute. Therefore they rely on standardised solutions, such as JINI, to find other processes offering a certain behaviour.

Once the other process is known, the real problems start. How can the requesting process communicate with the unknown offered process? Given the fact that those processes are developed by different organisations, the protocols provided and required can vary greatly. As a result protocol conflicts arise.

* Corresponding author. He is developing peer-to-peer embedded systems for a project funded by the Flemish Institute for Science and Technology (IWT).

** Tom Mens is a Postdoctoral Fellow of the Fund for Scientific Research - Flanders (Belgium)

A.M. Tyrrell, P.C. Haddow, and J. Torresen (Eds.): ICES 2003, LNCS 2606, pp. 422–433, 2003.

On first sight, a solution to this problem would be to offer protocol adaptors between every possible pair of processes. The problem with this approach is that the number of adaptors grows quadratic to the number of process protocols and as such it simply doesn't scale. The solution is to automate the generation of protocol adaptors between communicating processes.

As a potentially useful technique for this adaptor generation, we explored the research domain of adaptive systems. We found that the combination of genetic programming, classifier systems, and a formal specification in terms of Petri nets allowed us to automate the detection of protocol conflicts, as well as the creation of program code for adaptors that solve these conflicts. This paper reports on an experiment we performed to validate this claim.

2 Prerequisites of Interprocess Communication

Processes communicate with each other only by sending messages over a communication channel (similarily to CSP [1] and the π-calculus [2]). Communication channels are accessed by the process' ports. Processes communicate asynchronously and always copy their messages completely upon sending. The connections between processes are full duplex: every process can *send* and *receive* messages over a port. This brings us in a situation where a process *provides* a certain protocol and *requires* a protocol from another process. A process can have multiple communication channels: for every communication partner and for every provided/required protocol.

We imposed other requirements on the interprocess communication to allow us to generate adaptors:

1. *Implicit addressing.* No process can use an explicit address of another process. Processes work in a connection-oriented way. The connections are set up solely by one process: the connection broker. This connection broker will also evolve adaptors and place them upon the connections when necessary.
2. *Disciplined communication.* No process can communicate with other processes by other means than its ports. Otherwise, 'hidden' communication (e.g., over a shared memory) cannot be modified by the adaptor. This also means that all messages passed over a connection should be copied. Messages cannot be shared by processes (even if they are on the same host), because this would result in a massive amount of concurrency problems.
3. *Explicit protocol descriptions.* While humans prefer a protocol description written in natural language, computers need an explicit formal description of the protocol semantics. A simple syntactic description is no longer suitable.

3 Specifying Protocols

As a running example we choose a typical problem of communicating processes: how processes synchronise with each other. Typically, a server provides a concurrency protocol (often a transaction protocol) [3] that can be used by clients.