Tribal Programming for a Better World

Dr. David Hales
Dept. of Computer Science, University of Bologna, Italy

Abstract
A number of recent agent-based simulation models, inspired by sociological concepts, claim to demonstrate a novel approach for constructing co-operative and robust self-organising groupings (or tribes). But these models tend to be highly abstract, focusing on simple games (like the Prisoner's Dilemma). Even application orientated work using similar approaches presents simulations related to specific application domains. If we are to take this "tribal approach" seriously as a way of constructing self-organising distributed information systems we need a larger and more general vision coupled with actual implementations of useful applications that work for real "in the wild". In this talk we outline some of the previous work based on "tags" and evolving networks and then sketch our more general "tribal vision". Finally we discuss on-going work towards implementations and argue that some existing and successful applications embody aspects of the vision - though not constructed with this in mind.

General overview paper - tribal approach:

-- Rationality – what should I do next to achieve my goals?

individual rationality (ind. optimal nash eq.) methodological individualism
"what's in it for me?"

collective or social rationality (social optimal) functionalism (Durkhiem, Parsons)
"what's in it for everyone?"

problems with these approaches…

So a new kind of rationality is needed that is:

Easy to code, local info, scalable, robust, opportunistic, allows for agent autonomy.

Tribal rationality – (subgroup within the collective) Conflict theories (Marx, Weber)
"what's in it for us?"
"us and them" – defining the boundary

-- Building tribes with minimal code / overheads for "simple" tasks

Easy way is to use a bounded evolutionary-like rationality using only local info.
Simple heuristics (kind of base-case)

"if I see someone doing better than me I copy their behavior"
"sometimes I try out something new and see what happens"

Turns out this can build tribes if the behaviors incorporate ways to define tribe boundaries (tags = minimal way, links in a social network = another minimal way)

Using these methods we can already build simple P2P protocols that produce tribes creating efficient, self-organizing, scalable and robust solutions to some problems (file-sharing, tribe specialization) – developed initially with the PD game.

-- Evolutionary process that creates dynamic tribes that are internally altruistic

Some tag examples…

Examples in the “wild” – bittorrent = tracker gives tribe = swarm

-- seeders, leaders, leechers, managers

To build a productive tribe for some tasks requires a redistribution of resources to make peers stick around

-- Code as culture / let system select it’s own code, dynamically

Because distributed always on systems under constant changing demand offer no deterministic results - what works today, might not work tomorrow...

we have an "ecology" of software and systems interacting and changing all the time.

traditional testing is hard to do eg. planetlab... and even then, its not really very good. Simulations help but are limited. The best way to test things is to release them into "the wild" and see what happens.

But then, even if things seem to work, we have no assurance they will continue to work.

traditional programming involves designing from the top down certain kinds of deterministic processes considering all possible conditions at each stage. Testing involves trying out as many of those conditions we can reasonably expect.

This wont work.

Programmers tend to see their code as "controlling" what’s going on.

But perhaps we need to move away from this idea and see code as no more than some set of changing resources that processors (or nodes) may decide to utilize and drop to achieve their goals or "service levels".

Open source development shows us that programmers can develop and package code for deployment in a highly collaborative yet distributed way and that via take-up of that code, a kind of evaluation is performed.

Essentially users download software and execute it, if this produces a perceived improvement in achieving goals or required levels of performance the code "sticks" and possibly spreads.

In this vision, code is no more than a pool of packages from which nodes can select. They spread like memes / viral programming – kind of embody an artificial culture within the tribe.

One of the easiest ways for a node to know what packages to initially download and install is to simply point at another node which seems to be doing what is required - "I want to be like that node".

This kind of process follows and evolutionary logic and can be, at least partially, automated into protocols

--

New Computer Science, New Social Science (MABS)

Since social science isn't really a science you can't just read-off accepted theories into code. In order to focus the social theory toward engineering issues computer scientists need to ENGAGE and shape social science itself.

What we are talking about, of course, is a new kind of computer science influenced by social science AND a new kind of social science influenced by computer science. It often appears to me that the latter domain has made more progress than the former - interesting to question why?

- just constructing new computer models of social phenomena becomes social science research, there are many open areas within that domain.

- the computer science side seems less developed maybe because of entrenched methodologies, a focus on formalisms rather than practical problem solving and the culture divide - historically comp. sci. people don't think about problems sociologically, often focus on classical game theory approaches - generally of little value due to unrealistic assumptions of rational actors.