Overview

aericci
2008/07/01 15:03
Table of Contents
Every paradigm, computational model, or language is based on some kind of driving abstract or high level metaphors, which are often are essential to define the main characteristics of the paradigm abstractions. simpA metaphors are taken from humans cooperative working environments, where "systems" are composed by individual autonomous entities (humans), encapsulating different kind of skills, which actively carry on some kind of task or goal oriented activities, both individually and cooperatively by interacting with other individuals and suitably exploiting resources and tools, artifacts to use a unique term, available in the environment.

simpA brings this metaphor down to computer programming, conceiving a software systems as one or multiple workspaces where ensemble of autonomous entities called agents execute their task-oriented activities and interact by co-constructing, sharing and using artifacts, analogously to the human case.

So, while classes and objects are still used to represent basic data structures and abstract data types, in simpA agents and artifacts are used as basic high-level building blocks to decompose and structure complex systems, composed by ensemble of interacting and coordinating parts.

In particular:

- **agents** can be used to model as first-class abstraction pro-active entities, i.e. entities programmed so as to autonomously execute some kind activity - composed by one or more activities - encapsulating the control of such activity;
- **artifacts** can be used to model as first-class abstraction what is used or constructed by agents during their activities, including resources, tools, devices, typically passive entities encapsulating some kind of functionality.
- **workspaces** are the logical place where agents and artifacts are immersed, used to give a topology to the overall activities, and then partition the application environment.

Objects and classes are used as basic abstractions to define data structures to build agents and artifacts.

We believe that this approach to computer programming can be a good candidate to for being a general-purpose easy to learn and use programming model for multi-core / concurrent / distributed programming, so as to make it easier the design and development of those software systems that more and more are target of mainstream software development. According to several computer scientists and engineers, concurrency is going to play a fundamental role in mainstream software development, both for distributed / network-based and standalone applications. Quoting Herb Sutter and James Larus, Microsoft (ACM Queue, 3(7), Sept. 2005), among the others:

"The biggest sea change in software development since the OO revolution is knocking at the door, and its name is Concurrency… The concurrency revolution is primarily a software revolution. The difficult problem is not building multicore hardware, but programming it in a way that lets mainstream applications benefit from the continued exponential growth in CPU performance… Today concurrent programming languages and tools are at a level comparable to sequential programming at the beginning of the structured programming era. Semaphores and coroutines are the assembler of concurrency, and locks and threads are the slightly higher-level structured constructs of concurrency. What we need is OO for concurrency - higher-level abstractions that help build concurrent programs, just as object-oriented abstractions help build large componentized programs."

Following this perspective, in simpA agents and artifacts abstractions can be used as higher-level building blocks to design and build concurrent applications, not providing just mechanisms but a new (methodological) perspective to be exploit for developing software systems.