Subject: NEW: PortQueue (M$ CCR port to u++)
Posted by kohait00 on Mon, 26 Jul 2010 12:47:38 GMT
View Forum Message <> Reply to Message

hi guys

here comes my first launch, to forum first (to clarify some M$ related questions first, before uploading it to bazaar. and it needs testing)

in
http://www.ultimatepp.org/forum/index.php?t=msg&th=5314& amp;start=0&

i've been outlining a parallel / distributed data flow design pattern implemented and used by M$ in their M$ Robotics Development Studio. i had the chance to use / study it in one of my practice semesters. and it really impressed me. so i try to make this technology available to upp.

basicly said there are only few things:

a port queue can receive data elements. it has attached a receiver, which can spawn user code associated with it. this is done in parallel, *not* sequential (this is the advantage actually). parent arbiters can impose conditions on when a receiver can spawn its task.

the source implementation from M# is based on .NET, which i was able to look at using Reflector. so i have ported CCR for now, and not everything, only the most important things, for now:

Port<T>: a strongtype elements queue, provided as data source for a Receiver<T>

Receiver<T>: a capsulation for the user Task (delegate in C#), fed by a Port<T> with elements and spawning for each incoming data element own independant parallel task. a receiver can be a onetime only or a reissue.

Choice: a parent Arbiter, that lets execute only one of N Receivers, which ever comes first, the others are teared down safely.

JoinReceiver: a parent Arbiter, that issues the user Task only when items can be consumed on all ports, handling them over to the user task itself.

Interleave: a parent arbiter that arbitrates teardown receivers, exclusive receivers and concurrent receivers. a typical applications would use it to setup how the internal application state should be handled (i.e. reading state can be done in concurrent mode, modifying in exclusive, teardown receivers bring down the arbiter (is exclusive as well), used to shutdown applications usually).

since JoinReceiver is a Receiver, it can be nested inside a Choice as well.

at M$, this design approach is paired with a distributed software service (DSS) environment, which is a runtime for services design with CCR, alltogether yielding a powerfull application design that reliefs the pain of thinking in mutex, semaphore, atomic etc.. (DSS is not yet ported, will come next

so take a look, it is not that much code. i do not know how much the licence issue from M$ point of view impacts here, since i used UPP to realize it. but clearely ideas came from CCR . decide yourself.

the PortQueueTest package is the one to run. it has 4 szenarios in the ctor..separated with if's. choose only 1 of them for easiness.
first is independant paralell receivers on own ports
second is choice receivers, executing only one of them
third is join receiver, executing only when all items are available
forth is the interleave, executing the receivers according to group pertenance.

i need to provide some better, more obvious testcases ofcorse, some that more clearly show the parallel aspect of the whole thing. but it's a starting point.

RFC

File Attachments

1) PortQueue.rar, downloaded 280 times