Kepler - Bug #4256

Change to AbstractReceiver prevents putArray() from being called

07/22/2009 10:18 AM - Christopher Brooks

Status:	Resolved	**Start date:**	07/22/2009
Priority:	Normal	**Due date:**	
Assignee:	Christopher Brooks	**% Done:**	0%
Category:	director	**Estimated time:**	0.00 hour
Target version:	2.0.0	**Spent time:**	0.00 hour
Bugzilla-Id:	4256		

Description

Here's a bug from 5/13.

I agree that the problem is that putArray()
is no longer being called, which is being overridden in the derived
classes. domains/modal/kernel/FSMReceiver.java also calls putArray(),
so this is relevant.

I think the right solution is to modify ArrayToSequence and remove
the change to to AbstractReceiver.java

I'd like to see this fixed before we roll out Ptolemy II 8.0.beta.

On 5/13, Bert Rodiers wrote:

> Hello Christopher,
>
> Another issue with this change is that putArray is no longer called, which
> could be overridden by derived classes. Now this functionality is bypassed.
> For example:
> CIReceiver has this implementation:
> public synchronized void putArray(Token[] tokenArray, int
> numberOfTokens)
> throws NoRoomException {
> for (int i = 0; i < numberOfTokens; i++) {
> _tokens.add(tokenArray[i]);
> }
>
> _notify();
> }
>
> FSMReceiver has this implementation:
> public void putArray(Token[] tokenArray, int numberOfTokens)
> throws NoRoomException, IllegalActionException {
> if (numberOfTokens != 1 || tokenArray.length < 1) {
> throw new IllegalActionException(getContainer(), "Receiver
> cannot accept more than one token.");
> }
> put(tokenArray[0]);
> }
>
> and so on...
>
> Regards,
> Bert

Edward wrote:

> If the only difference is the order the loops,
> I think we are OK with the change in AbstractReceiver.

Edward

2009/5/13 Christopher Brooks <cxh@eecs.berkeley.edu>

Hi Dan,
I folded in your change.

One thing that has me concerned is that if the model mutates
between when we check the containers and when we do
the conversion, we could have problems.

For example, when I run

```
$PTII/bin/vergil ~/ptII/ptolemy/domains/pn/kernel/test/auto/block.xml
```

putArrayToAll() is called:

```
ptolemy.actor.AbstractReceiver.putArrayToAll(AbstractReceiver.java:297)
at ptolemy.actor.IOPort.send(IOPort.java:2728)
at ptolemy.actor.TypedIOPort.send(TypedIOPort.java:524)
at ptolemy.domains.sdf.lib.ArrayToSequence.fire(ArrayToSequence.java:176)
at ptolemy.actor.process.ProcessThread.run(ProcessThread.java:217)
```

I don't see where a lock is held that would prevent the model
from mutating between when we check the containers and when
we use them.

Interestingly, the original code had a similar problem, so
this is not a new issue.

I'm probably overlooking something . . .

To address Edward's concern, while I feel that this is a fundamental
change, we do have a trivial example that needs the change and
we don't have a counter example that is broken by the change.

The difference between what we had before and what we have now isr
the order of the loops. Previously, we looped through
the receivers in the outer loop and the tokens in the inner loop.
Now we loop through the tokens in the outer loop and the
receivers in the inner loop.

Why should this make a difference?

_Christopher

Hi Bert and Christopher,

Thanks for clarifying the performance problem, and adding
my update and test case.

What do you think of the following putArrayToAll? I think
this is what Bert suggested; it does not make extra calls
to getContainer.

```
public void putArrayToAll(Token[] tokens, int numberOfTokens,
Receiver[] receivers) throws NoRoomException,
IllegalActionException {
if (numberOfTokens > tokens.length) {
IOPort container = getContainer();
throw new IllegalActionException(container,
"Not enough tokens supplied.");
}

// Cache the containers for each receiver to minimize
// the number of calls to getContainer.
IOPort[] containers = new IOPort[receivers.length];
for (int j = 0; j < receivers.length; j++) {
containers[j] = receivers[j].getContainer();
}

// Loop through the tokens on the outer loop and
// the receivers on the inner loop. See
// pn/kernel/test/block.xml for a test case
// (Bug fix proposed by Daniel Crawl.)
for(int i = 0; i < numberOfTokens; i++) {
for (int j = 0; j < receivers.length; j++ ) {
if (containers[j] == null) {
receivers[j].put(tokens[i]);
} else {
receivers[j].put(containers[j].convert(tokens[i]));
}
}
}
}
```

> Thanks,
>
> --dan

On Wed May 13 10:12:27 PDT 2009
Sean Riddle wrote:

> I agree that the correct approach would be to modify ArrayToSequence.
> I can't remember quite why, but I also needed this behavior from
> ArrayToSequence at some point, and the change is only one or two
> lines, plus you're guaranteed that this won't affect other parts of
> the system unexpectedly.
>
> - Sean

On Wed May 13 08:32:39 PDT 2009

Edward A. Lee wrote:

> I see.
>
> It seems to me that the "right" fix is to modify ArrayToSequence,
> not to modify the mechanism in AbstractReceiver. The reason for
> the mechanism in AbstractReceiver is to support bulk transfers
> efficiently. I'm not sure where else this is used, but it does
> seem that preserving this capability would be useful.
>
> I don't feel that strongly about it though...
>
> Edward
>
> Jianwu Wang wrote:
>
>> Hi Edward,
>>
>> Dan and I found the problem from a real world workflow. The workflow
>> includes about 50 job files, their execution time varies from 1 hours to 5
>> hours. The jobs can be submitted by firing JobSubmitter actor 50 times with
>> corresponding inputs, and their status can be checked by JobStatus actor in
>> a loop. I hope once one job is finished, the following actors related to the
>> data of this job can be processed without waiting for other jobs. So that we
>> can enable pipeline parallel. The workflow has the structure similar with
>> the attached test case. With default PN director configuration,
>> ArrayToSequence actor tries to send all array data to one next actor before

sending data to another connected actor. But the Equals actor need the data from the two Expressions to be fired. So the Equals actor will increase its capacity and not be fired since there is still running actor (job actors). The workflow has to wait all jobs are finished before processing the following actors, which is very inefficient for my case.

With Dan's modification, the ArrayToSequence actor can transfer the first token of the array to all the following actors before transfer the second token. Then the Equals actor can be fired timely and do not need to increase capacity. So the same workflow can realize real pipeline parallel, namely the following actors can be fired immediately with corresponding data once one job is finished.

Best wishes

Sincerely yours

Jianwu Wang
jianwu@sdsc.edu
http://users.sdsc.edu/~jianwu/ <http://users.sdsc.edu/%7Ejianwu/>

Scientific Workflow Automation Technologies (SWAT) Laboratory
San Diego Supercomputer Center University of California, San Diego
San Diego, CA, U.S.A.

Edward A. Lee wrote:

> I don't really understand the problem being solved here.
> The deadlock is a consequence of setting the maximum queue capacity to one. Why would you want to do that?
>
> Edward
>
> Daniel Crawl wrote:
>
>> Hi Edward,
>>
>> Attached is the test case. I set the max queue capacity to one so that an exception is thrown instead of an artificial deadlock occurring. With my change, the exception does not occur and the model finishes.
>>
>> Is there a test case demonstrating the performance problem? In both versions, put is called (with a single token) the same number of times, so it's not clear how my change could hurt efficiency.
>>
>> Thanks,
>>
>> --dan
>>
>> Edward A. Lee wrote:
>>
>>> Dan,
>>>
>>> Are you sure the deadlock is artificial?
>>> I would like to see the test case. Maybe the model isn't using the right actors?
>>>
>>> The point of the methods you changed was to improve efficiency, and sending tokens one at a time nullifies that point. There is really not point in even having these methods, I think.
>>>
>>> Edward
>>>
>>> Daniel Crawl wrote:

Hi Christopher,

I made this update to prevent unnecessary artificial deadlocks
in PN under certain circumstances. I can add a test case that
demonstrates the problem.

If the convert is performed, is the update ok? Note that no
tests failed in ptolemy/actor/test/ due to this change...
Since calling convert is essential, I can also add a test case
for this.

There were effectively two nested loops before, so I do not see
how this change could degrade performance. If it is measurably
different, it is improved since the outer loop no longer calls
a method.

Thanks,

--dan

Christopher Brooks wrote:

> Yep, I went ahead and reverted the change.
>
> _Christopher
>
> Edward A. Lee wrote:
>
>> The call to convert is essential.
>>
>> Without it, we'll get some very esoteric and difficult to track
>> type system bugs. A likely manifestation is that actors will
>> start throwing ClassCastException because they have declared
>> an input to be double, so they cast incoming tokens to DoubleToken.
>> Without the call to convert(), they may get, say, an IntToken.
>> This will be a very unfriendly error...
>>
>> Edward
>>
>> Christopher Brooks wrote:
>>
>>> Hi Daniel,
>>> I'm concerned that this is a performance hit because we
>>> have two nested loops. Can you tell me more about why this
>>> change is necessary? Do you have a test case that illustrates
>>> the bug? Without a test case, it is not likely that the fix will
>>> persist, though the comment should help.
>>>
>>> The entire method is:
>>> /** Put a sequence of tokens to all receivers in the specified
>>> array.
>>>
>>> - Implementers will assume that all such receivers
>>> - are of the same class.
>>> - @param tokens The sequence of token to put.
>>> - @param numberOfTokens The number of tokens to put (the
>>>   array might
>>> - be longer).
>>> - @param receivers The receivers.
>>> - @exception NoRoomException If there is no room for the
>>>   token.
>>> - @exception IllegalActionException If the token is not
>>>   acceptable
>>> - to one of the ports (e.g., wrong type), or if the tokens
>>>   array
>>> - does not have at least the specified number of tokens.
>>>   */

```
public void putArrayToAll(Token[] tokens, int numberOfTokens,
Receiver[] receivers) throws NoRoomException,
IllegalActionException {
if (numberOfTokens > tokens.length) {
IOPort container = getContainer();
throw new IllegalActionException(container,
"Not enough tokens supplied.");
}
```

```
// Put a single token at a time for each receiver instead
of
// putting the entire array. In the latter case, we may
block
// on a receiver while other receiver(s) starve.
for(int i = 0; i < numberOfTokens; i++) {
for (int j = 0; j < receivers.length; j++ ) {
receivers[j].put(tokens[i]);
}
}
}
```

I do see how this could be a problem with blocking though.

Your change is to call put() instead of putArray().
AbstractReceiver.putArray() looks like:

```
public void putArray(Token[] tokenArray, int numberOfTokens)
throws NoRoomException, IllegalActionException {
IOPort container = getContainer();

// If there is no container, then perform no conversion.
if (container == null) {
for (int i = 0; i < numberOfTokens; i++) {
put(tokenArray[i]);
}
} else {
for (int i = 0; i < numberOfTokens; i++) {
put(container.convert(tokenArray[i]));
}
}
}
```

It looks like your change is ok when the container is null, but
in the AbstractReceiver base class it does not handle the call
to convert()? I'm not sure if this is important or not.

I'm fairly certain that putArrayToAll() will be called when we
call
IOPort.broadcast.

What do you think?

_Christopher

Daniel Crawl wrote:

> Author: crawl
> Date: 2009-05-06 14:45:46 -0700 (Wed, 06 May 2009)
> New Revision: 53516
>
> Modified:
> trunk/ptolemy/actor/AbstractReceiver.java
> Log:
> Put a single token at a time for each receiver in
> putArrayToAll().
>
> Modified: trunk/ptolemy/actor/AbstractReceiver.java

==
============

```
--- trunk/ptolemy/actor/AbstractReceiver.java    2009-05-06
21:13:26 UTC (rev 53515)
++ trunk/ptolemy/actor/AbstractReceiver.java    2009-05-06
21:45:46 UTC (rev 53516)
@ -300,8 +300,13 @
"Not enough tokens supplied.");
}
-        for (int j = 0; j < receivers.length; j+) {
-            receivers[j].putArray(tokens, numberOfTokens);
+        // Put a single token at a time for each receiver
instead of
+        // putting the entire array. In the latter case, we may
block
+        // on a receiver while other receiver(s) starve.
+        for(int i = 0; i < numberOfTokens; i++) {
+            for (int j = 0; j < receivers.length; j++ ) {
+                receivers[j].put(tokens[i]);
+            }
}
}
```

____
Ptexternal-cvs mailing list
Ptexternal-cvs@chess.eecs.berkeley.edu
http://chess.eecs.berkeley.edu/ptexternal/listinfo/ptexternal-cvs

____
Ptolemy maillist - Ptolemy@chess.eecs.berkeley.edu
http://chess.eecs.berkeley.edu/ptolemy/listinfo/ptolemy

Kepler-dev mailing list
Kepler-dev@kepler-project.org
http://mercury.nceas.ucsb.edu/kepler/mailman/listinfo/kepler-dev

____
Ptolemy maillist - Ptolemy@chess.eecs.berkeley.edu
http://chess.eecs.berkeley.edu/ptolemy/listinfo/ptolemy

--
Christopher Brooks (cxh at eecs berkeley edu) University of California
CHESS Executive Director   US Mail: 337 Cory Hall
Programmer/Analyst CHESS/Ptolemy/Trust   Berkeley, CA 94720-1774
ph: 510.643.9841 fax:510.642.2718   (Office: 545Q Cory)
home: (F-Tu) 707.665.0131 (W-F) 510.655.5480

____
Ptolemy maillist - Ptolemy@chess.eecs.berkeley.edu
http://chess.eecs.berkeley.edu/ptolemy/listinfo/ptolemy

History

#2 - 07/23/2009 04:23 PM - Christopher Brooks

Bert Rodiers fixed this, the change has been merged into the 8.0.beta branch.

The fix was to revert the change in AbstractReceiver and fix ArrayToSequence.
I also added a test: pn/auto/PNAbstractReceiverTest.xml

#3 - 03/27/2013 02:26 PM - Redmine Admin

Original Bugzilla ID was 4256

Files

block.xml	5.43 KB	07/22/2009	Christopher Brooks