# Distributed multiscale computing

Borgdorff, J.

# A

# An algorithm for a temporal conduit filter

The conduit filter used as a computational element in MML and described in Section 2.3.1 can alter the data flow between submodels. As such, it is good to explore what algorithm would be good to do this, and show that this algorithm does not introduce deadlocks.

Consider again, two submodels $A$ and $B$ with temporal regular scales $S_A(2\text{ s}, 1\text{ hr})$, $S_B(1\text{ s}, 1\text{ hr})$ that are coupled on a single domain. They have temporal scale overlap and use the interact coupling templates $\mathbf{O}_\mathbf{i}^A \rightarrow \mathbf{S}^B$ and $\mathbf{O}_\mathbf{i}^B \rightarrow \mathbf{S}^A$. To allow these submodels to have their own time step, while still handling the coupling correctly, a temporal conduit filter should be used on conduits between $A$ and $B$. Messages from $A$ to $B$ should be interpolated to give $B$ enough messages to process, and in the other direction messages should be aggregated to ensure $A$ does not receive too many, as shown in Algorithm 6.

The filter in the direction from $A$ to $B$ has to know the time series $\vartheta = (e_0, \ldots, e_n)$ of $B$, in order to send the correct number of messages, for instance by the modeller specifying it. It does not need to know the time step of $A$, as $A$ sends, with each message, both the timestamp of the message and the timestamp of the next event. Thus, the filter can decide whether the next event of $A$ will need to be aggregated, or, that a message can be sent to $B$ immediately.

**Algorithm 6:** Example of a temporal scale conduit filter, where $i$ is the index of the last event $e_i$ for which the receiving submodel already received a message.

**Input**: Time series $\vartheta = (e_0, \ldots, e_n)$ of receiving submodel

```
1  i ← 0;
2  while i < n do
3  |  m ← receive();
4  |  add m to messages;
5  |  t_m* ← time of message after m, or ∞ if m is the last message;
6  |  if t(e_n) ≤ t_m* then
7  |  |  i′ ← n;
8  |  else
9  |  |  i′ ← k such that t(e_k) ≤ t_m* < t(e_{k+1});
10 |  end
11 |  if i′ > i then
12 |  |  (s_{i+1}, ..., s_{i′}) ← interpolate within messages;
13 |  |  send(s_{i+1}), ..., send(s_{i′});
14 |  |  i ← i′;
15 |  end
16 end
```

The communication from $A$ to $B$ is correct if the filter sends the right number of messages at the right time. This statement can be assessed by three filter properties, with filter $f$ and events $e_1, \ldots, e_n$ of the receiving submodel:

1. for each event $e_i$ with $1 \leq i \leq n$, $f$ sends exactly one message;

2. if $f$ receives a message with time $t \geq t(e_i)$, $1 \leq i \leq n$, it has already sent a message for event $e_i$; and

3. if $f$ receives a message with time $t < t(e_i)$, $1 \leq i \leq n$, it has not yet sent a message for event $e_i$.

Due to the well-ordering of time series, messages received are in-order. Also given, is that a conduit filter may only send messages if a message is received. Then the first two properties combined prevent a deadlock, while the third property ensures that messages sent to $B$ can be meaningful, that is, it prevents the trivial algorithm where upon the first message received, $n$ messages are sent.

The first property is straightforward and prevents a deadlock for $B$, without considering other submodels or couplings. The second property prevents a deadlock that would occur if: the filter waits for a message from $A$; $B$ waits for a message from the filter; and $A$ waits for a message from $B$. Suppose that all properties above hold, and that a deadlock occurs due to the filter. Submodel $A$ and $B$ would have model time $t_A$ and $t_B$, respectively, which is, by their SEL, also the time of the next message they will send. Because of property 2, the conduit filter can not receive a message with time $t_A$ if $t_A \geq t_B$, so $t_A < t_B$. Now $A$ would only wait for a message from $B$ if $t_A > t_B$, following the definition of the interact coupling template in Table 2.2. Combined, these statements imply that $t_A > t_B$ and $t_A < t_B$, contradicting that a deadlock can occur under the stated conditions.

Take Algorithm 6 as filter $f$ and apply it to the conduit from $A$ to $B$. Assume that $A$ starts at model time $t_0 < t(e_1)$, with event $e_1$ of $B$. In words, $A$ must start before or at same model time as $B$, approximately, since a conduit filter can not produce output without input.

Before showing that $f$ satisfies the properties of a deadlock-free conduit filter, observe that lines 13 and 14, combined with $\vartheta$ being well-ordered and messages of $f$ being sent in-order, ensure that at the end of each loop, exactly $i$ messages are sent.

First, $A$ will send a last message since it has a finite number of time steps. At that point line 5 will assign $t_{m*} \leftarrow \infty$. After that, line 7 will ensure that $i = n$ at the end of the algorithm, fulfilling property 1. Second, starting time $t_0$ of $A$ has $t_0 < t(e_1)$ by assumption, so the first message $f$ receives will obey property 2 with time $t = t_0 < t(e_1)$. Suppose message $m$ with next event time $t_{m*}$ obeyed property 2, then the next message $m'$ has time $t_{m'} = t_{m*}$. Now suppose that $f$ violates property 2, by assuming that it sent a message for $e_k$ and that $t_{m'} > t(e_k)$. Then in the previous iteration of the while loop, $t_{m*} > t(e_k)$. Because time series $\vartheta$ is well-ordered, the conditions on line 6 and 9 will ensure that $i' \geq k$. Now line 13 ensures that a message has been sent for $e_k$ before $m'$ is received. This is a contradiction, showing that $f$ fulfils property 2. Finally, suppose $f$ has sent a message for event $e_k$. Then line 13 has been executed with $i' \geq k$, so lines 6 or 9 had $t_{m*} \geq t(e_k)$. Since $f$ receives messages in-order, this means it can not receive a message with time $t_m < t(e_k)$, fulfilling property 3.

This shows that the three properties for a deadlock-free filter have been satisfied by Algorithm 6.