CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2010; **22**:965–980
Published online 11 March 2010 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1433

A process-oriented architecture for complex system modelling[‡]

C. G. Ritson and P. H. Welch[*,†]

Computing Laboratory, University of Kent, Canterbury, Kent, CT2 7NF, England

SUMMARY

A fine-grained massively parallel and process-oriented architecture for the modelling of complex systems is presented. We propose that the concurrency in the model simplifies its design and construction by directly reflecting the processes in the natural world. The architecture is based on CSP, extended with mechanisms for process mobility from the pi-calculus; implementations are presented using the occam-pi language. A case study, modelling platelets (possibly artificial) within a blood vessel, is described. The aim for this model is to engineer emergent behaviour: the clotting of platelets in response to a wound in the blood vessel wall and the staunching of blood loss. A three-dimensional model is constructed, along with mechanisms for visualization and interaction. Its expressiveness and efficiency relies strongly on the dynamic and mobile capabilities of occam-pi. General principles for the design of large and complex system models are drawn. The described case study runs to millions of processes engaged in ever-changing communication topologies. It is free from deadlock, livelock, race hazards and starvation *by design*, employing a small set of synchronization patterns for which we have proven safety theorems. Compiled occam-π codes automatically and efficiently exploit all cores in a shared-memory multiprocessor system. They are also straightforward to distribute over standard cluster architectures. Copyright © 2007 CG Ritson & PH Welch.

Received 4 February 2009; Accepted 10 February 2009

KEY WORDS: occam-pi; concurrency; CSP; complex systems

1. INTRODUCTION

In this paper, a process-oriented architecture for simulating a complex environment and mobile agents is described (Section 2). The environment is modelled by a fixed topology of stateful

[*]Correspondence to: P. H. Welch, Computing Laboratory, University of Kent, Canterbury, Kent, CT2 7NF, England.
[†]E-mail: phw@kent.ac.uk
[‡]Revised version of Ritson CG, Welch PH. A process-oriented architecture for complex system modelling. In McEwan AA, Schneider S, Ifill W, Welch PH (eds). Communicating Process Architectures 2007. IOS Press: Amsterdam. Published with permission from IOS Press.

Contract/grant sponsor: UK EPSRC; contract/grant numbers: EP/E053505/1, EP/E049419/1

processes, one for each unit of space. State held includes the strength of specific environmental factors (e.g. chemicals), local forces and the presence of agents. Agents are mobile processes interacting directly with the space processes in their immediate neighbourhood and, when they sense their presence, other agents. Mechanisms for dynamically structuring hierarchies among agents are also introduced, allowing them to display complex group behaviours. The architecture combines deadlock-free communications patterns with (phased barrier controlled) shared state, maintaining freedom from race hazards and high efficiency. We have used occam-π [1,2] as our implementation language.

The biological accuracy of the blood clotting case study (Section 3) is very approximate. However, even with the current system, simple experiments are possible that have scientific interest (e.g. the effect of platelet density on the success of the clotting mechanism in stemming blood flow: too high or too low and the process fails). Section 4 discusses the emergent behaviours observed, the performance figures obtained and Section 5 gives directions for future work. For the latter, the process-oriented nature enables simple refinement (through the addition of processes modelling different stimulants/inhibitors of the clotting reaction, different platelet types and other participating organelles) to greater and greater realism. This paper presents a simple and flexible architecture. It shows promise for generic application across a broad range of complex system.

This paper is a revision of one presented at CPA-2007 [3]. The research was started as part of the (EPSRC) TUNA project [4–10] at the universities of York, Surrey and Kent, which seeks to explore simple and formal models of emergent behaviour. It continues as part of the CoSMoS project [11] on complex systems modelling, patterns and simulation, funded by EPSRC grant EP/E053505/1.

2. ARCHITECTURE

The simulation architecture is constructed in layers. At the bottom lie the *site* processes, representing distinct points (or regions) in the simulated space and managing information associated with that locality. Each site is a pure *server* process, handling requests on the server-end of a channel bundle (unique for each site). It will have a dynamically changing set of *client* processes (mobile agents), competing with each other to access the client-end of its channel bundle. Each channel bundle contains two channels used in opposite directions: one from a client to the server (*request*) and one from the server to a client (*response*). All communication is initiated by one of the clients successfully laying claim to its end of the channel bundle and making a request. Once accepted, the server and this client engage in a bounded conversation over the channel bundle, honouring some pre-agreed protocol. So long as no closed cycle of such *client-server* relationships exists across the whole process network, such communication patterns have been proven to be deadlock free [12,13].

2.1. Space modelling

To model *connected* space, each site has reference to the client-ends of the channel bundles serviced by its immediate *neighbours*. These references are only used for forwarding to visiting clients—so that they can explore their neighbourhood and, possibly, move. Sites must never directly communicate with other sites, since that could introduce client–server cycles and run the risk of deadlock. The inter-site references define the *topology* of the simulation world. For standard Euclidean space,

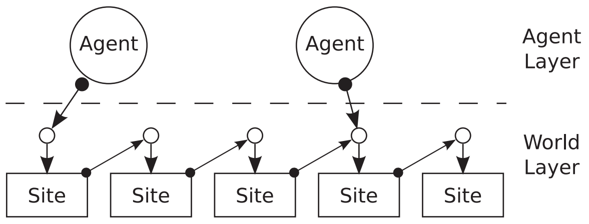

Figure 1. A simplified representation of sites and agents in a world where agents may only move right. Each site services an exclusive channel bundle for communicating with visiting agents. Agents obtain connections with their next site from references held by their current site.

these neighbourhood connections are fixed. For example, each site in a 3D *cubic* world might have access to the sites that are immediately above/below, left/right or in-front/behind it. In a more fully connected world, each site might have access to all 26 neighbours in the $3 \times 3 \times 3$ cube of which it forms the centre. Other interesting worlds might have *wormholes* or allow dynamic topologies which change in response to simulation events.

2.2. Mobile channels and processes

As previously stated, our simulation architecture is constructed in layers. The world layer is homogeneous—only sites. The agent layer is heterogeneous. There can be many kinds of agent processes, visiting and engaging with sites as they move around their world. Agent-site protocols fall into three categories: querying and modifying the current site state, obtaining access to neighbouring sites and moving between sites. Agents move through the simulated world registering and de-registering their presence in sites (commonly by depositing free channel-ends through which they may be contacted), using environmental information (held in the sites) to make decisions as they go and, possibly, modifying some environmental factors. An agent only needs to hold the channel-end of its current site and, when relevant, the next site it wishes to enter. For all this, the occam-π concept of channel-end mobility [14], derived from the π-calculus [15], is essential.

Figure 1 shows a one-dimensional world where each site has access only to the neighbour immediately to its right. In this world, agents can only move in one direction. The arrows with circles on their bases represent client–server relations (pointing to the server). The client-ends of these connections are *shared* between other sites and agents (shown by the arrows with solid disc bases). Recall that these connections do provide two-way communications.

2.3. Barriers and phases

Agents use *barriers* [16,17] to coordinate access to the sites into time-distinct *phases*. An occam-π `BARRIER` is (almost) the same as a multiway synchronization event in CSP [5,18–20]: *all* enrolled processes must reach (synchronize upon) the barrier in order for *all* of them to pass. The resulting phases ensure that they maintain a consistent view of their environment, and keep to the same simulation step rate. To prevent agents from viewing the world while it is in flux, at least two phases are required: a *discovery* phase where agents observe the world and make decisions, and a *modify* phase where agents change the world by implementing those decisions (e.g. by moving and/or updating environmental parameters). The basic agent logic, using occam-π *pseudo-code*, is:

```
WHILE alive
  SEQ
    SYNC discovery
    ... observe my neighbourhood
    SYNC modify
    ... change my neighbourhood
```

where `discovery` and `modify` are the coordinating barriers.

2.4. Site occupancy and agent movement

In a typical simulation, space is quantized in units the size of a single cell and only one agent will be allowed to occupy a given site at any point in time. Within our architecture, sites enforce this constraint. If two agents attempt to enter a site in the same simulation cycle, the decision can be left to chance (and the first agent to arrive enters), or made using an election algorithm (the *best* candidate is picked). In the case of an election algorithm, the modify phase should be sub-divided:

(a) agents request to enter the site providing some sort of candidacy information (e.g. mass, aggressiveness or unique ID). When the site receives a new candidate, it compares it with the exiting one and overwrites that if the new candidate is *better*.
(b) all agents query the site(s) they attempted to enter again, asking who *won*? On receiving the first of these queries, the site installs its current *best* candidate as the new occupier and passes those details back to the asker and to any subsequent queries.

However, an optimization can be made by including the first *modify* sub-phase in the *discovery* phase, increasing task parallelism. Only offers to move are made—no world state change is detectable by the agents in this phase. The second *modify* sub-phase simply goes into the *modify* phase. This optimization saves a whole barrier synchronization and we employ it (Section 3.5).

2.5. Agent–agent interaction

Some agents in the same locality may need to communicate with each other. To enable this, they deposit in their current site the client-end of a channel bundle that they will service. This client-end

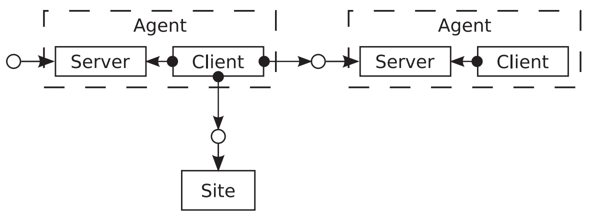

Figure 2. Agents are composed from client and server sub-processes to prevent client–server loops and maintain deadlock freedom.

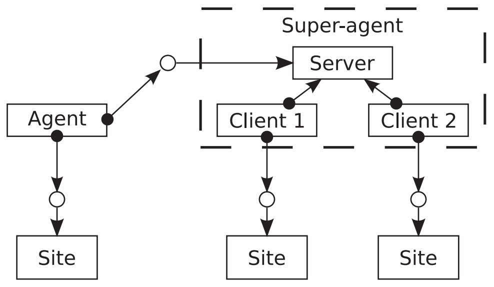

Figure 3. Super-agents as a layered composition of processes.

will be visible to other agents (observing from a neighbouring site). However, agents must take care how they communicate with each other in order to avoid client–server cycles and deadlock. A simple way to achieve this is to compose each agent from at least two sub-processes: a server to deal with inter-agent transactions and a client to deal with site processes and initiate inter-agent calls.

In Figure 2, the agent *server* process manages agent state: its clients are the *client* processes of its own and other agents. The agent *client* process drives all communication between the agent and the rest of its environment (the sites over which it roams, other agents in the neighbourhood and higher level agents to which it reports—Section 2.6). Technically, it would be safe for the agent *server* also to communicate with the sites.

2.6. Layers of agents

So far, agents have occupied a single site. Complex agents (e.g. a blood clot) may grow larger than the region represented by a single site and would need to span many, registering with all those that it occupies. This may be done from a single agent process (as above) or by composing it from many sub-processes (one *client* part per site). We view the latter approach as building up a *super-agent* (with more complex behaviour) from many lower level agents (with simpler behaviour and responsibilities). It introduces a third layer of processes.

In Figure 3, clients 1 and 2 share a higher level server process, holding information from both that enables them to act in a coordinated manner. Agents outside the super-agent just see a single server off a single agent. Such sharing of higher level servers allows us to create groups of arbitrarily large coordinated agents. The approach can be continued hierarchically to create ever more complex groups, while keeping the complexity of each process manageable. There are no client–server cycles and the pure clients (the lowest level agents) are the initiators of all activity.

3. HUMAN BLOOD CLOTTING SIMULATION

We have introduced the principle components of the simulation architecture: a hierarchical client–server network of sites, agents and super-agents. We now look at how this has been applied to simulate the clotting of platelets in the human blood stream [9].

Haemostasis is the response to blood vessel damage, whereby platelets are stimulated to become *sticky* and aggregate to form blood clots that seal small wounds, stemming blood loss and allowing healing. Platelets are non-living agents present in certain concentrations in blood; they are continually formed in bone marrow and have a half-life of around 10 days. Normally, they are inactive. They are triggered into becoming sticky by a complex range of chemical stimuli, moderated by a similarly complex range of inhibitors to prevent a lethal chain reaction. When sticky, they combine with each other (and proteins like fibrin) to form physically entangled clots. Extensive details can be found in [21].

The work presented in this paper employs a highly simplified model of haemostasis. We model the smooth and sticky states of platelets, with transition triggered by encountering a sufficient amount of a single chemical *factor* released by a simulated wound to the blood vessel wall. We model no inhibition of clotting, instead focusing only on the initial reaction to a wound.

Clots form when sticky platelets bump together and, with some degree of probability, become permanently entangled. The speed of an individual clot decreases with respect to the rate of blood flow as its size increases. We are not modelling other factors for the clotting material (such as fibrin). Nevertheless, even with this very simple model, we have reached the stage where emergent behaviours (the formation of blood clots and the sealing of wounds) are observed and simple experiments are possible that have scientific interest.

3.1. Sites

Sites define the space of the simulated environment. Our sites are arranged into cubic three-dimensional space (giving each site 26 neighbours). Sites are pure server processes, responding to agent (client) offers of, or requests for, information. They operate independently, engaging in no barrier synchronizations.

Interacting with the sites, the lowest level agents are blood *platelets* and chemical *factors* (which, when accumulated in the sites above a certain threshold, can switch passing platelets into their *sticky* state). Blood clots are super-agents, composed of many stuck-together platelets.

The sites allow one platelet to be resident at a time and store a unique `ID` number, stickiness, size (of the blood clot, if any, of which it is a part) and transaction channel-end (for later agent–agent communications). Sites use the (clot) size and unique `ID` to pick the best candidate during the entry elections described in Section 2.4.

In addition to platelet/clot information, the sites also store a clotting chemical factor level (obtained from passing factor processes), a unit vector (indicating the direction of blood flow) and a *blocking* flag (indicating whether the site is part of the blood vessel wall—in which case agents are denied entry).

Although using agents to simulate the wall would also be possible, we choose to implement it as a feature of space to save the memory overhead of having more agents (with very trivial behaviour).

Finally, each site has access to a *voxel* (a byte from a shared 3D-array), which it is responsible for maintaining. Whenever the site changes, it computes a transfer function over its state to set this voxel. The voxel itself is used to visualize the simulation via volume rendering techniques.

3.2. Platelets (agents)

Our simulation agents model individual platelets in the blood. Platelets are pure clients and do not communicate directly with each other. However, they are clients to their clot super-agent and it is this that keeps them together. A platelet may be in one of two states:

- *Non-sticky*: The platelet queries its local site and reports the blood-flow direction and clotting factor level to its super-agent. It then initiates any movement as instructed by the super-agent. The clot's size and unique `ID` are used to register presence in the sites.
- *Sticky*: In addition to the above non-sticky behaviour, the platelet searches neighbouring sites for other sticky platelets, and passes their details to its super-agent.

Platelets, along with the chemical factor processes (Section 3.3), move and update their environment. Together with the processes generating them and the processes controlling visualization, they are enrolled and synchronize on the *discovery* and *modify* barriers—dividing the timeline into those respective phases (Sections 2.3 and 3.5.1).

Note: for programming simplicity, *all* platelets in our current model have a clot process—even when they are not sticky or part of any clot. We may optimize those clot processes away later, introducing them only when a platelet becomes sticky. Most platelets in most simulations will not be sticky!

3.3. Clots (super-agents)

Clots coordinate groups of platelets. They accumulate the blood-flow vectors from their platelets' sites and make a decision on the direction of movement. That decision also depends on the size of clots, with larger clots moving more slowly. They also change platelets from non-sticky to sticky if sufficient levels of clotting factor are encountered (these accumulate over many simulation steps).

When two or more clots encounter each other, if they contain sticky platelets they *may* become stuck together and merge. One of the clots takes over as super-agent for all sets of platelets in the bump group—the other clots terminate.

In [17], a clotting model for a one-dimensional blood stream was presented (as an illustration of mobile channels and barriers). In that system, deciding which clot process takes over is simple. Only two clots can ever be involved in a collision: arbitrarily, we decide the one further upstream wins.

Stepping this model up to two dimensions, multiway collisions are possible since clots can be shaped with many leading edges in the direction of movement—for example, an 'E'-shaped clot moving rightwards and bumping into an 'I' shape. Furthermore, those multiple collisions may be with just a single or many other clots. Fortunately, stepping this up to three dimensions does not introduce any further difficulties.

To resolve the decision as to which clot survives the collision, an *election* takes place involving direct communication between the clot super-agents. This is outside the client–server architecture shown in Figure 3 (for whose reasoning this election is deemed to be a bounded internal computation). The clot processes must engage in nothing else during this election and that must terminate without deadlock. Reasoning about this can then be independent from reasoning about all other synchronizations in the system.

The key to deadlock-free election is to order all the communications in a sequence that all parties know about in advance. Each clot has an `ID` number which is registered in all sites currently occupied by its constituent platelets. Each clot has had reported back to it, by its platelets, the clot `ID`s of all clots in the collision.

The platelets also place the client-end of a server channel to their clot in the site they are occupying. They report to their clot the client-ends of the other clots in the collision. Thus, each clot now has communication channels to all the other clots in its collision.

High number clots now initiate communication to low number clots. The lowest numbered clot is the winner and communicates back the election result, with communication now from low number clots to high. The choice that *low* numbered clots should win was not arbitrary. Clots are introduced into the world with increasing `ID` numbers. Clots further down the bloodstream will, therefore, have lower IDs and will tend to amass platelets as smaller (faster moving) clots (with higher IDs) catch up and collide. In turn, this reduces the number of times platelets need to change their clot super-agent after collision. Although our algorithm for ordering communication (not fully outlined here) has yet to undergo formal proof, it has so far in practice proven reliable.

Platelets communicate with their clot using the shared client-end of a server bundle. By keeping track of the number of platelet processes it contains, a clot knows how many communications to expect in each phase (and, so, does not have to be enrolled in the barriers used by the platelets to define those phases). See Section 3.5 for more details of clot and platelet communications.

3.4. Factors (agents)

The second and final type of agent in our simulation is one that models the chemical factors released into the blood by a wounded (damaged) blood vessel. Since they move and modify their environment (the sites), they must engage on the same *discovery* and *modify* barriers as the platelets.

Factors are launched (forked) into the simulation with an initial vector pointing away from the wound and into the blood vessel. At every simulation step, the factor integrates a proportion of its current site's blood flow vector with its own vector and uses the result to determine its next move. The effect is cumulative so that eventually the factor is drawn along with the blood flow. At each site it enters, the factor increases the factor strength field, and modifies the site's blood flow vector to point back to the wound. The second of these two actions simulates both the slight pressure drop from an open wound and other biological mechanisms which draw platelets to open wounds.

Finally, it should be noted that factors are not considered to take up any space—being tiny molecules as opposed to full cells. Hence, many are allowed to occupy individual sites.

3.5. Simulation logic

We illustrate with pseudo-code, closely based on occam-π [1,2], for the platelet and clot processes.

3.5.1. Platelet process

Initially, a platelet attached to its launch site, is not `sticky`, has a clot process to which only it belongs and has no knowledge of its neighbourhood. Platelets decide whether they want to move in the *discovery* phase; however, the movement is election based (Section 2.4), and the result of the election is not queried until the *modify* phase. This means that, although movement offers are made in the *discovery* phase, actual movement does not happen until the *modify* phase.

The names `site`, `new.site`, `clot` and `clot.b` represent `SHARED` client ends of channel bundles containing request and reply channels (flowing in opposite directions and carrying rich message structures). They connect, respectively, to the current and (possible) future *site* locations of the platelet and the *clot* process of which it forms a part. (The dots in occam-π names are not operators, but separators like underscore in other languages. For simplicity of presentation, we omit the request/reply field names of the channel bundles and the CLAIM operations required before shared use.)

```
SEQ

  WHILE  still in the modelled blood vessel
    SEQ

      SYNC discovery              --  all platelets and factors wait here for each other

      site !  ask for local chemical factor level and motion vector
      site ?  receive above information
      clot ! factor.vector.data;  forward above information

      IF
        sticky
          SEQ
            site !  get clot presence on neighbour sites (in directions that were previously empty)
            site ?  receive above information
            clot !  forward information only on clots different to our own (i.e. on clot collisions)
        TRUE
          SKIP

      --  clot decides either on transition to sticky state or merger of bumped clots
      clot.b ? CASE
        update; clot; clot.b      --  our clot has bumped and merged with others
          SKIP                    --  we may now belong to a different clot process
        become.sticky
          sticky := TRUE          --  accumulated chemical factors over threshold
        no.change
          SKIP

      --  clot decides which way, if any, to try and move
      clot ? CASE move; target
      IF
        target = no.move
          SYNC modify                       --  empty phase for us, in this case
        TRUE
          SEQ
            site ! get.neighbour; target    --  get the channel end of the new site
            site ? new.site
            new.site ! enter; clot          --  offer to enter new site, giving our clot reference

            SYNC modify                     --  wait for all other offers to be made

            new.site ! did.we.enter; clot   --  ask if we were successful
            new.site ? CASE
              yes
                SEQ
                  clot ! ok                 --  report ability to move
                  clot.b ? CASE
                    ok                     --  all platelets in clot can move
                      SEQ
                        site ! leave       --  leave present site
                        site := new.site   --  commit to new site
                    fail
                      new.site ! leave     --  give up attempted move
              no
                SEQ
                  clot ! fail              --  report failure to move
                  clot.b ? CASE fail       --  clot cannot move as this platelet failed

  SEQ                   --  we have exited the modelled region of space (and the loop)
    SYNC discovery      --  we must get into the right phase for last report
    clot ! terminated   --  last report
```

3.5.2. Clot process

Initially, a clot is not `sticky` and starts with a platelet count (`n.platelets`) of 1. A clot runs for as long as it has platelets. It does not need to engage in the *discovery* and *modify* barriers, deducing those phases from the messages received from its component platelets. At the start of each phase, a clot is `sticky` *if and only if* all its component platelets are `sticky`.

The names `platelets` and `platelets.b` represent server ends of channel bundles containing request and reply channels (flowing in opposite directions and carrying rich message structures). They service communications from and to all its component platelets and are the opposite ends to the `clot` and `clot.b` channel bundle ends shared by those platelets (Section 3.5.1).

```
WHILE n.platelets > 0
  SEQ
    --  nothing will happen till the discovery phase starts; wait for the reports from our platelets

    SEQ i = 0 FOR n.platelets
      platelets ? CASE
        factor.vector.data;  local chemical factor level and motion vector
          ... accumulate chemical factor level and motion vector
        terminated
          n.platelets := n.platelets - 1
    IF
      sticky
        SEQ
          SEQ i = 0 FOR n.platelets
            platelets ?  report on any bumped clots
          IF
            sufficiently hard collision anywhere
              SEQ
                ...  run clotting election to decide which clot takes over the merger
                SEQ i = 0 FOR n.platelets
                  platelets.b ! update; winner; winner.b    --  channels for winning clot
            IF
              this.clot = winner
                ... update number of platelets to new size of clot
              TRUE
                n.platelets := 0          -- i.e. terminate
          TRUE
            SEQ i = 0 FOR n.platelets     -- no platelets reported collisions
              platelets.b ! no.change     -- so, let them all know

      accumulated.chemical.factor > sticky.trigger.theshold
        SEQ
          sticky := TRUE
          SEQ i = 0 FOR n.platelets
            platelets.b ! become.sticky

      TRUE                                -- remain not sticky
        SEQ i = 0 FOR n.platelets
          platelets.b ! no.change

    target := choose.some.move.or.none (n.platelets, motion.vector)

    SEQ i = 0 FOR n.platelets             -- tell our platelets
      platelets ! move; target

    -- platelets now synchronise on modify barrier

    IF
      target = no.move                    -- we're definitely not moving
        SKIP
      TRUE                                -- move iff all platelets can move
        SEQ
          all.confirm := TRUE
          SEQ i = 0 FOR n.platelets       -- can each platelet move?
            platelets ? CASE
              ok
                SKIP
              fail
                all.confirm := FALSE
          IF
            all.confirm
              SEQ i = 0 FOR n.platelets   -- we are moving
                platelets.b ! ok
            TRUE
              SEQ i = 0 FOR n.platelets   -- we're not moving
                platelets.b ! fail
```

3.6. Spatial initialization

The simulated environment must be initialized before platelets are introduced. It needs to contain some form of bounding structure to represent the walls of the blood vessel and the vectors in the sites must direct platelets along the direction of blood flow.

The blood vessel wall is placed so that it runs parallel to an axis in simulated space—the x-axis in our simulations. Our simulated blood vessel is simple: a cylinder with wall thickness of approximately two sites. The wall is simulated by setting the sites to which it belongs to *blocking*.

Force vectors inside the blood vessel are initialized so that there are platelets likely to move forward with some lesser probability of motion in the other axes. Changing the initialization of these vectors can give subtle changes in simulation behaviour—something left largely unexplored at this time.

The vectors outside the blood vessels are programmed to draw platelets to the edges of the simulated space and beyond. This enhances the blood loss effect when the vessel wall is broken. If this were not done, platelets would continue along much the same path just outside the blood vessel.

4. RESULTS

4.1. Emergent behaviour

Using the architecture and the simple processes and behaviours described, we have been able to achieve results surprisingly similar to those in the human body. Given the *right* concentration of platelets (Figure 4), wounds to our simulated blood vessel trigger the formation of clots (Figure 5) that eventually form a *plug* covering the wound and preventing further blood loss (Figure 6). Too low a concentration and the clotting response is too weak to let sufficiently large clots form. Too high a concentration and a clot forms too early, gets stuck in the blood vessel *before* the wound and fails to seal it. The clot also gets bigger and bigger until it completely blocks all the blood flow—which cannot be good!

The concentration boundaries within which successful sealing of a wound is observed are artifacts of the current simulation model, i.e. they do not necessarily correspond with the biology. However, the fact that this region exists for our models gives us encouragement that they are beginning to reflect some reality.

In the human blood stream, clotting stimulation (and inhibition, which we have not yet modelled but is certainly needed) involves many different chemical factors, cell types (there are different types of platelet) and proteins (e.g. fibrinogen). It is encouraging that our modelling techniques have achieved some realistic results from such a simple model.

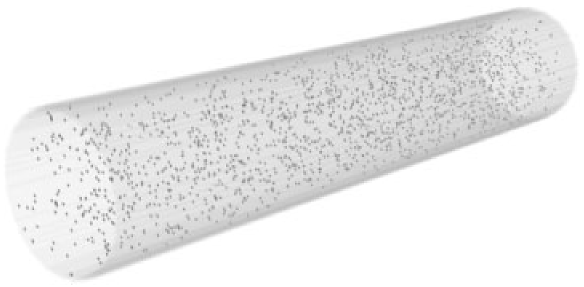

Figure 4. Simulated blood vessel represented by the cylinder, dots are platelets.

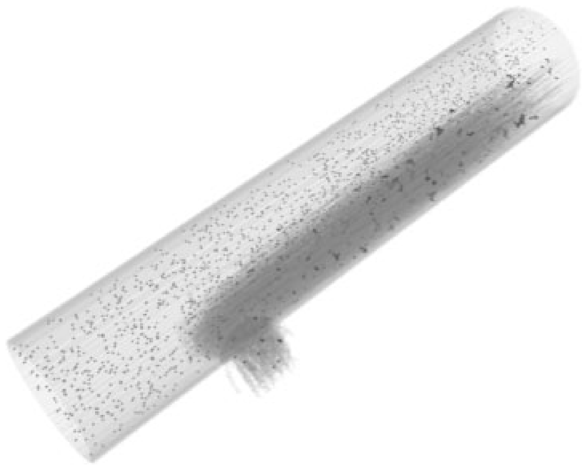
Figure 5. Having placed a wound, platelets ‘fall’ out of the blood vessel, and chemical factors can be visualized by the darkened area. Given time, chemical factors flow down the blood vessel and (small) clots can be seen forming as dark blobs.

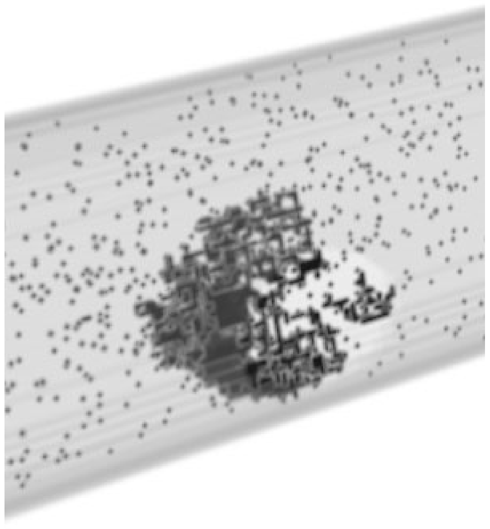
Figure 6. With sufficient time and a high enough platelet concentration, a clot forms over and seals the wound.

The clotting response we observe from our model has been engineered, but not explicitly programmed. The platelets are not programmed to spot wounds and act accordingly. They are programmed only to move with the flow of blood, become sticky on encountering certain levels of chemical and, then, clump together when they bump. Developing this so that greater and greater levels of realism emerge should be possible through the addition of processes modelling different stimulators and inhibitors of the clotting reaction, along with different platelet types and other participating agents. Because of the compositional semantics of CSP and occam-π, such extensions will not interfere with existing behaviours in ways that surprise—but should evolve to increase the stability, speed, accuracy and safety of the platelets' response to injury.

4.2. Performance

Our process-oriented model implemented in occam-π has proved stable and scalable. Further, occam-π compiled codes automatically and efficiently exploit all the available cores in a shared-memory multiprocessor system. Simulations have been run with more than 3 000 000 processes on commodity desktop hardware (P4, 3.0 GHz, 1 GB RAM). Memory places a limit on the size of our simulations. However, as our site processes only become scheduled when directly involved in the simulation, the available processing power only limits the number of active agents. Bloodstream platelet densities of up to 2% (an upper limit in healthy humans) imply an average of around 60 000 agents—actual numbers will be changing all the time. Cycling each with an average processing time of 2 μs (including barrier synchronization, channel communication and cache miss overheads) still enables around 8 simulations steps per second, which is very usable.

Figure 7 shows the performance for simulations on a world of size 256 × 96 × 96 (2.3M+ sites). The different curves are for different levels of platelet concentration (0.5, 1.0 and 2.0%). The x-axis shows simulation step numbers (*generations*), starting from an (unrealistic) bloodstream devoid of any platelets—but with them starting to arrive from upstream. Performance does not stabilize until the blood vessel is filled with platelets, which takes 500 generations. This is as expected, given a volume 256 sites in length and with a roughly even chance of any platelet moving forwards.

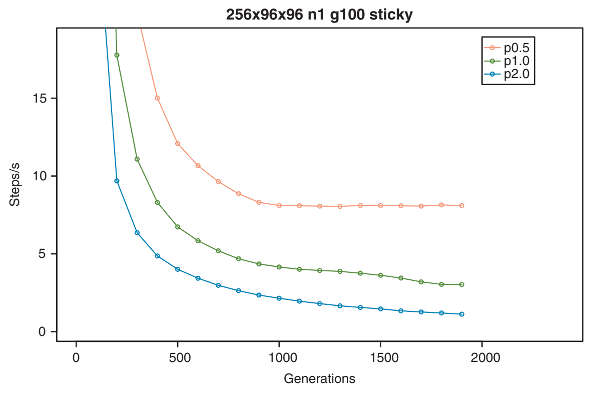

Figure 7. 256 × 96 × 96 simulations with sticky platelets. The legend values p0.5, p1.0, and p2.0 represent the concentration of platelets in the blood entering the vessel, 0.5, 1.0% and 2.0% respectively.

For the simulations whose results are shown in the platelets and their associated clots are initialized sticky. This is the worst-case (and unrealistic) scenario where clots will form whenever two platelets collide. At 0.5% platelet concentration (an average of approximately 5000 agents), we are achieving around 9 simulation/steps a second. All these results have visualization disabled; in practice, most commodity graphics hardware has difficultly rendering simulations this size at rates greater than 10 frames per second. As the number of agents doubles to 1.0%, and then 2.0%, the performance degrades almost linearly, in regular (not worst case) the degrade is linear. Again, this is expected, given that the computation load has doubled and that occam-π process management overheads are independent of the number of processes being managed. In non-worst-case simulations of the same dimensions, the performance around 13 frames per second for 0.5% plate concentrations is achieved on the same hardware.

5. FUTURE WORK

The next steps in our research are to scale up and refine our simulations. For scaling up the size of our models, we have developed a cluster-based implementation (using the pony [22] networking environment for occam-π) with performance results, not published here, that are promising.

For refining the accuracy of the model, we would like to achieve the return of our simulated blood vessel to a *normal* state once blood loss through a wound has been stemmed. We need to introduce factors that inhibit the production of further clots and *bust* existing ones (e.g. all those little ones that were washed away by the bloodstream before they could clump to the wound). So long as the wound is open, chemical factors would continue to be released, gradually lowering as the wound is closed. Inhibitor agents would also reduce clotting factor levels and correct blood flow vectors. The blood vessel wall also needs to be able to reform under the protective clot. Eventually, with the wound healed, the clot would dissipate and the factors that caused it would disappear.

Further refinement could be explored by integrating aspects of other research, both physical and simulated, into the flow of platelets within the blood stream [23]. In order to model these properties we will need to introduce aspects of fluid dynamics into our model, and allow our simulated clots to roll and sheer. By removing the rigid movement constraints on platelets within a clot and giving them a degree of individual freedom, the introduction of these new behaviours should be attainable. For example, by adding an appropriate vector (changing with time) to each of the platelets within a clot, the clot as a whole could be made to roll or tumble as it moves through the blood vessel.

Finally, we believe that the massively concurrent process-oriented architecture, outlined in this paper for this simulation framework, can be applied generically to many (or most) kinds of complex system modelling. We believe that the ideas and mechanisms are natural, easy to apply and reason about, maintainable through refinement (where the cost of change is proportional to the size of that change, not the size of the system being changed) and can be targeted efficiently to modern hardware platforms. We invite others to try.

ACKNOWLEDGEMENTS

We acknowledge all our partners in the TUNA [4] and CoSMoS [11] projects at Kent, York and Surrey and our colleagues in the Concurrency Research Group [24] at Kent, without whom this work would not have happened.

We are also grateful to the anonymous referees for their diligence and constructive criticism. This research is funded by UK EPSRC grants EP/E053505/1 ('TUNA') and EP/E049419/1 ('CoSMoS').

REFERENCES

1. Welch P, Barnes F. Communicating mobile processes: Introducing occam-pi. *25 Years of CSP* (*Lecture Notes in Computer Science*, vol. 3525), Abdallah A, Jones C, Sanders J (eds.). Springer: Berlin, 2005; 175–210.
2. The occam-pi programming language. Available at: http://www.occam-pi.org/ [April 2009].
3. Ritson CG, Welch PH. A process-oriented architecture for complex system modelling. *Communicating Process Architectures 2007* (*Concurrent Systems Engineering Series*, vol. 65), McEwan AA, Schneider S, Ifill W, Welch P (eds.). IOS Press: Amsterdam, The Netherlands, 2007; 249–266.
4. Stepney S, Welch P, Polack F, Woodcock J, Schneider S, Treharne H, Cavalcanti A. TUNA: Theory underpinning nanotech assemblers (Feasibility Study). *EPSRC EP/C516966/1*. Available at: http://www.cs.york.ac.uk/nature/tuna/index.htm [April 2009].
5. Welch P, Barnes F, Polack F. Communicating complex systems. *Proceedings of the 11th IEEE International Conference on Engineering of Complex Computer Systems* (*ICECCS-2006*). IEEE: Stanford, CA, 2006; 107–117.
6. Schneider S, Cavalcanti A, Treharne H, Woodcock J. A layered behavioural model of platelets. *ICECCS-2006*, Michael GH (ed.). IEEE: Stanford, CA, 2006; 98–106.
7. Stepney S, Turner H, Polack F. Engineering emergence (keynote talk). *ICECCS-2006*, Michael GH (ed.). IEEE: Stanford, CA, 2006; 89–97.
8. Polack F, Stepney S, Turner H, Welch P, Barnes F. An architecture for modelling emergence in CA-like systems. *Advances in Artificial Life*, *8th European Conference on Artificial Life* (*ECAL 2005*) (*Lecture Notes in Computer Science*, vol. 3630). Springer: Berlin, 2005; 433–442.
9. Ritson C, Welch P. TUNA: 3D blood clotting, 2006. Available at: https://www.cs.kent.ac.uk/research/groups/sys/wiki/3D_Blood_ Clotting/ [April 2009].
10. Sampson A. TUNA demos. Available at: https://www.cs.kent.ac.uk/research/groups/sys/wiki/TUNADemos/ [April 2009].
11. Stepney S, Welch P, Timmis J, Polack F, Barnes F, Tyrell A, Bates M, Sampson A, Andrews P. CoSMoS: Complex systems modelling and simulation infrastructure. Available at: http://www.cosmos-research.org/, EPRSC Funded Research, 2007-20011 EP/E053505/1 and EP/E049419/1 [April 2009].
12. Welch P, Justo G, Willcock C. Higher-level paradigms for deadlock-free high-performance systems. *Transputer Applications and Systems '93*, vol. 2. IOS Press: Netherlands, Aachen, Germany, 1993; 981–1004.
13. Martin J, Welch P. A design strategy for deadlock-free concurrent systems. *Transputer Communications* 1996; **3**(4): 215–232.
14. Barnes F, Welch P. Prioritised dynamic communicating and mobile processes. *IEE Proceedings—Software* 2003; **150**(2):121–136.
15. Milner R, Parrow J, Walker D. A calculus of mobile processes—Parts I and II. *Journal of Information and Computation* 1992; **100**:1–77. Available as technical report: ECS-LFCS-89-85/86 [April 2009], University of Edinburgh, U.K.
16. Barnes F, Welch P, Sampson A. Barrier synchronisations for occam-pi. *Parallel and Distributed Processing Techniques and Applications—2005*. CSREA Press: Las Vegas, NV, U.S.A., 2005; 173–179.
17. Welch P, Barnes F. Mobile barriers for occam-pi: Semantics, implementation and application. *Communicating Process Architectures 2005* (*Concurrent Systems Engineering Series*, vol. 63). IOS Press: Amsterdam, The Netherlands, 2005; 289–316.
18. Hoare C. *Communicating Sequential Processes*. Prentice-Hall: London, 1985. ISBN: 0-13-153271-5.
19. Roscoe A. *The Theory and Practice of Concurrency*. Prentice-Hall: Englewood Cliffs, NJ, 1997. ISBN: 0-13-674409-5.
20. McEwan A. Concurrent program development. *PhD Thesis*, The University of Oxford, 2004.
21. Griffin J, Arif S, Mufti A. *Immunology and Haematology* (*Crash Course*) (2nd edn). C.V. Mosby, 2003. ISBN: 0-7234-3292-9.
22. Schweigler M, Sampson A. Pony—The occam-pi network environment. *Communicating Process Architectures 2006*. IOS Press: Amsterdam, The Netherlands, 2006.
23. Pivkin I, Richardson P, Karniadakis G. Blood flow velocity effects and role of activation delay time on growth and form of platelet thrombi. *Proceedings of the National Academy of Science* 2006; **103**(46):17164–17169. DOI: 10.1073/pnas.0608546103. Available at: http://www.pnas.org/cgi/content/abstract/103/46/17164 [April 2009].
24. Welch P, Barnes F, Sampson A, Ritson C, Dimmich D, NCC Brown, Simpson J, Warren D, Bonnici E. Concurrency Research Group, Computing Laboratory, University of Kent. Available at: http://www.cs.kent.ac.uk/research/groups/sys/concur.html, EPRSC Funded Research, 2007-2011, EP/E053505/1 and EP/E049419/1.