Communicating Process Architectures 2007
Alistair A. McEwan, Steve Schneider, Wilson Ifill, and Peter Welch
IOS Press, 2007

A Process-Oriented Architecture for Complex System Modelling

Carl G. RITSON and Peter H. WELCH

Computing Laboratory, University of Kent, Canterbury, Kent, CT2 7NF, England.
{cgr,phw}@kent.ac.uk

Abstract. A fine-grained massively-parallel process-oriented model of platelets (potentially artificial) within a blood vessel is presented. This is a CSP inspired design, expressed and implemented using the occam-pi language. It is part of the TUNA pilot study on nanite assemblers at the universities of York, Surrey and Kent. The aim for this model is to engineer emergent behaviour from the platelets, such that they respond to a wound in the blood vessel wall in a way similar to that found in the human body – i.e. the formation of clots to stem blood flow from the wound and facilitate healing. An architecture for a three dimensional model (relying strongly on the dynamic and mobile capabilities of occam-pi) is given, along with mechanisms for visualisation and interaction. The biological accuracy of the current model is very approximate. However, its 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. Even with the current system, simple experiments are possible and 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). 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 synchronisation patterns for which we have proven safety theorems.

Keywords. occam-pi, concurrency, CSP, complex systems

Introduction

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

This research is part of the TUNA project [3,4,5,6,7,8,9] at the universities of York, Surrey and Kent, which seeks to explore simple and formal models of emergent behaviour. Medium term applications are for the safe construction of massive numbers of nanotechnology robots (*nanites*) and their employment in a range of fields such as the dispersion of pollution and human medicine. With this goal in mind, this paper introduces our generic simulation architecture through specific details of how it has been used to simulate platelets in the human blood stream and the clotting response to injury.

1. Architecture

1.1. Dynamic Client-Servers

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 [10,11].

1.2. 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, 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 3x3x3 cube of which it forms the centre. Other interesting worlds might allow dynamic topologies – for example, the creation of *worm-holes*.

1.3. Mobile Channels and Processes

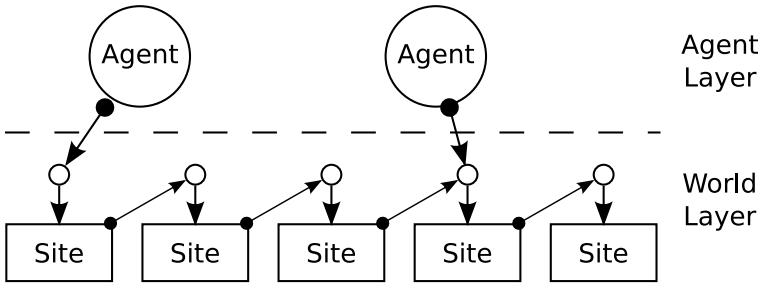

Figure 1. A simplified representation of sites and agents. Each site services an exclusive channel bundle for communicating with visiting agents. Agents obtain connections to their next site from references held by their current site.

The world layer (Figure 1) is homogeneous – only sites. The (first) agent layer is heterogeneous. There can be many kinds of agent process, 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, possi-

bly, 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 concept of channel-end mobility [12], a feature of occam-π based on items from the π-calculus [13], 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.

1.4. Barriers and Phases

Agents use *barriers* [14,15] to coordinate access to the sites into time-distinct *phases*. An occam-π `BARRIER` is (almost) the same as a multiway synchronisation event in CSP: *all* enrolled processes must reach (synchronise 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 viewing the world while it is in flux, at least two phases are required:

discovery: where agents observe the world and make decisions;
modify: where agents change the world by implenting those decisions (e.g. by moving and/or updating environmental parameters).

The basic agent logic is:

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

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

1.5. Site Occupancy and Agent Movement

In a typical simulation, 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:

first modify sub-phase: 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 to the exiting one and overwrites that if the new candidate is *better*.
second modify sub-phase: 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 optimisation can be made by including the first *modify* sub-phase in the *discovery* phase! 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 optimisation saves a whole barrier synchronisation and we employ it (section 2.5).

1.6. 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 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.

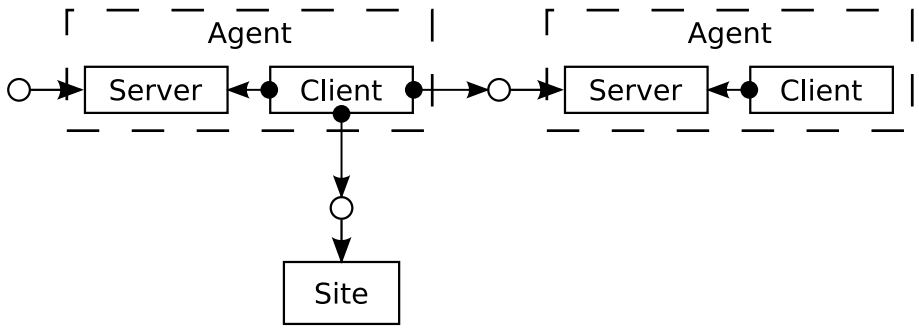

Figure 2. Agents are composed from client and server sub-processes to prevent client-server loops and maintain deadlock freedom.

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 1.7). Technically, it would be safe for the agent *server* also to communicate with the sites.

1.7. 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 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.

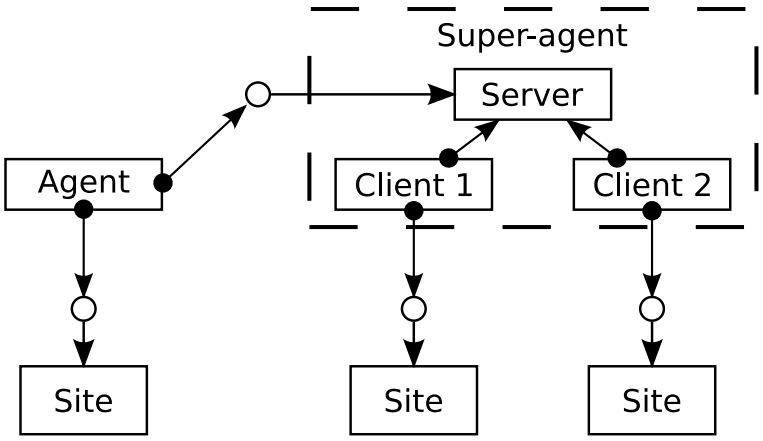

Figure 3. Super-agents as a layered composition of processes.

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 – see figure 4. Note that some processes are pure servers (the sites and mega-agents), some are pure clients (the lowest level agents) and some are servers that sometimes act as clients to fulfil their promised service (the super-agents). Note that there are no client-server cycles and that the pure clients (the lowest level agents) are the initiators of all activity.

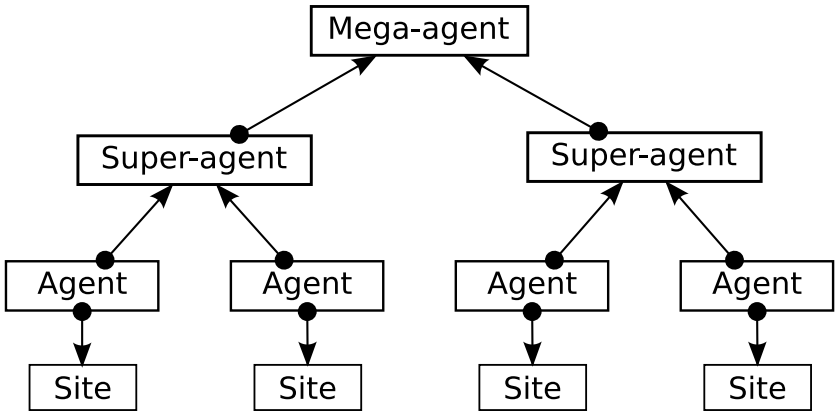

Figure 4. A hierarchy can be constructed among agents to give increasingly complex group behaviours.

2. 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 [8].

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. Summaries can be found in [16,17,18], with extensive details in [19].

The work present 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, and relying on a sufficient rate of blood flow to prevent a chain reaction until it is observed.

Clots form when sticky platelets bump together and, with some degree of probability, become permanently entangled. The velocity 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.

2.1. 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 synchronisations.

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 1.5.

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 visualise the simulation via volume rendering techniques.

2.2. Platelets (Agents)

Our simulation agents model individual platelets in the blood. As in figures 3 and 4, 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 2.3), move and update their environment. Together with the processes generating them and the processes controlling visualisation, they are enrolled and synchronise on the *discovery* and *modify* barriers – dividing the timeline into those respective phases (sections 1.4 and 2.5.1).

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 optimise those clot processes away later, introducing them only when a platelet becomes sticky. Most platelets in most simulations will not be sticky!

2.3. 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 [15], a clotting model in 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 so, arbitrarilly, 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. 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, another *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 synchronisations in the system.

The trick 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, so having low number clots win means that low number clots will tend to amass platelets. In turn, this reduces the number of times those platelets need to change 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 2.5 for more details of clot and platelet communications.

2.4. 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. 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.

2.5. Simulation Logic

To provide more detail, here is some pseudo-code (loosely based on occam-π [1,2]) for the platelet and clot processes.

2.5.1. Platelet Process

Initially, a platelet is attached to its launch site, is not `sticky`, has a clot process to which only it belongs and has no knowledge of its neighbourhood (which it assumes is empty of platelets/clots). Platelets decide whether they want to move in the *discovery* phase; however, the movement is election based (section 1.5), 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 *"channels"* `site`, `new.site` and `clot`/`clot.b`, used (illegally) in both directions below, represent `SHARED` client ends of channel bundles containing request and reply channels (flowing in opposite directions and carrying rich protocols). For further simplicity, the necessary `CLAIM` operations have also been omitted. They connect, respectively, to the current and (possible) future *site* locations of the platelet and the *clot* process of which it forms a part.

```
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

      no.move
        SYNC modify                          -- empty phase for us, in this case

      move; target
        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
    SYNC discovery                           -- must get into the right phase for last report
    clot ! terminated
```

2.5.2. 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 *"channels"* `platelets`/`platelets.b` used (illegally) in both directions, represent the server ends of two channel bundles containing request and reply channels (flowing in opposite directions and protocol rich). They service communications from and to all its component platelets (and are the opposite ends to the `clot`/`clot.b` channels shared by those platelets).

```
  WHILE n.platelets > 0
    SEQ

      -- nothing will happen till the discovery phase starts
      -- we just wait for the reports from our platelets to arrive

      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
            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
            platelets.b ! no.change

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

  TRUE
    SEQ i = 0 FOR n.platelets
      platelets.b ! no.change

target := pick.best.move.if.any (n.platelets, motion.vector)

IF
  target = no.move
    SEQ
      SEQ i = 0 FOR n.platelets
        platelets ! no.move
      -- platelets synchronise on modify barrier

  TRUE
    SEQ
      SEQ i = 0 FOR n.platelets
        platelets ! move; target
      -- platelets synchronise on modify barrier

      all.confirm := TRUE
      SEQ i = 0 FOR n.platelets
        platelets ? CASE
          ok
            SKIP
          fail
            all.confirm := FALSE
      IF
        all.confirm
          SEQ i = 0 FOR n.platelets
            platelets.b ! ok
        TRUE
          SEQ i = 0 FOR n.platelets
            platelets.b ! fail
```

2.6. Spatial Initialisation

The simulated environment must be initialised 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.

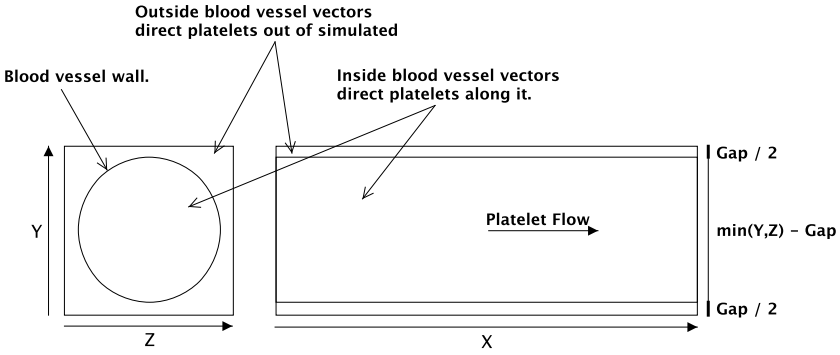

Figure 5. Layout of the simulated space in relation to blood vessel.

The blood vessel wall is placed so that it runs parallel to an axis in simulated space – the X-axis in our simulations (see figure 5). 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 initialised so that there is a 55%[1] chance of moving forward along the blood vessel, an 6% chance of moving left or right, and an 8% chance of moving up or down. A given site vector can only point in one direction per axis, so the vectors point either left or right, and either up or down, e.g. left and down. The directions are select randomly per site, with an even distribution between each. Changing the initialisation 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.

2.7. Optimisations

A few optimisations to our implementation were made to improve performance and memory usage.

Instead of giving each site an array of client-ends to neighbours, a single global array was used. This array is shared by all sites and significantly reduces memory requirement. This is safe as this connectivity information is static – we are not dealing with *worm-holes* and dynamic space topologies yet! occam-π does not yet have a language mechanism to enforce this read-only behaviour (of the connectivity) at compile time; but manual checking of our code is simple and deemed sufficient for our purposes here.

For performance enhancement, our implementation was designed so that platelets (agents) need only query their current site to discover the state of their local neighbourhood. This is accomplished in two stages. Firstly, site state data is placed into an array shared by all sites. This allows sites to retrieve data from their neighbours on behalf of an agent just by accessing (shared) memory. This is safe in our simulation because agent query and modification

[1] These are experimental values (not reflecting any biology).

are separated by barriers and individual updates to a site's state are serialised through that site's server interface. Secondly, agents now query their neighbourhood through their current site, passing it a *mobile* array of unit vectors and a *mobile* record. The site copies from the shared site state array the data for the specified vectors into the mobile memory, which it then returns along with its own state. Use of mobile data passed back and forth is very efficient and removes the need for dynamic memory allocation during normal agent run-time.

Our final optimisations were to reduce the neighbourhood searched by the agents. The previous optimisation reduced an individual search mainly to memory copies. As a first step, search is limited to (the obvious) six directions from the 26 available – although movement is permitted in any direction. When a platelet is part of a clot with other platelets, each platelet remembers the relative position of other platelets discovered around it and does not search those directions again. Futhermore, if a platelet becomes completely surrounded by platelets of the same clot, it terminates. For our simulation purposes, only the outline of clots need be maintained.

3. Support Processes

A small number of other processes complete the simulation and provide interaction and (3D) visualisation.

3.1. Platelet Generator

The platelet generator is a process that injects platelets at the *upstream* end of the blood vessel. It is enrolled on the *discovery* and *modify* barriers and restricts the injection (i.e. forking) of platelets to the *modify* phase (so that each platelet starts correctly synchronised, waiting for the *discovery* barrier). The platelet generator is programmed with a rate that can be varied at runtime. This rate (together with the cross-sectional area of the blood vessel) determines platelet density in the bloodstream. It sets a forward velocity (slightly randomised around an average of a 55% probability of movement).

At each simulation step, the number of platelets to be introduced is added to a running count; the truncated integer value of this count used to calculate the number of actual platelets to be forked. For each new platelet, two random numbers are generated: a Y and Z offset from the centre of the blood vessel. So long as these lie within the blood vessel, the platelet is injected at that position.

3.2. Wound Process

The wound process allows a user to punch a hole in the blood vessel wall. The *wound tool* is rendered as a sphere in the user interface and the user attacks the blood vessel with it. It creates a hole where there is an intersection between the sphere and the blood vessel walls. To do this, it uses the position of the sphere and its radius. If a point lies within the sphere, the corresponding site is tested to see if it is *blocking* (i.e. part of the blood vessel wall). If so, it is set to *unblocking* and four chemical factor processes are forked at its location (as a reaction to the damage). The initial movement vector of each factor process is initialised (with slight randomised jitter) so that it travels into the blood vessel.

3.3. Drawing Process

The drawing process has the task of informing the user interface when it is safe to render the voxel volume. It does this by signaling the user interface after the *discovery* barrier and before the *modify* barrier. When the user interface finishes rendering the volume, this process

synchronises on the *modify* barrier. Using this sequence, the voxels are only rendered during the stable *discovery* phase, and the user interface stays in step synchronisation with the simulation. Rendering of only *one-in-n* simulation steps is implemented by a simple counter in this process.

3.4. User Interface and Visualisation

Our simulation architecture is not tied to any specific form of visualisation or interface. We have built simulations using 2D text and graphical interfaces; however, for our 3D blood clotting simulations we choose to employ the open source Visualisation Toolkit (VTK) from Kitware [20]. Binding foreign language routines into occam-π is straightforward [21].

VTK is an open source library written in C++, with Python, Tcl/Tk and Java wrappers. It has several hundred different classes and a selection of examples illustrating their use. However, the focus of this toolkit is on loading static content from files, not the visualisation of realtime simulations (known as *tracking*).

For our visualisations, VTK is employed as a *volume renderer*. This means we can directly visualise what is in effect a 3D array of pixels. Internally, the `vtkVolumeTextureMapper2D` class is used, which turns slices of the 3D volume into 2D textures that are rendered using OpenGL. This approach is much faster than ray tracing. Two transfer functions map the byte voxel data into colour and opacity before it is rendered. In theory, and there is evidence of its use in the field, modern 3D hardware could be programmed to do this mapping in real time, reducing CPU load and improving rendering times.

Also provided by VTK is a wealth of 3D interaction tools. In practice this means that VTK handles mouse input to manipulate the camera, and the user-controllable sphere used to project wounds onto the blood vessel. Input event handlers are registered so that interaction events, including key strokes, are recorded in an overwriting ring buffer from which the occam-π user interface process can access them.

4. Results and Further Work

4.1. Emergent Behaviour

Using the architecture and 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 6), wounds to our simulated blood vessel (figures 7 and 8) triggers the formation of clots (figure 9) that eventually form a *plug* covering the wound and preventing further blood loss (figure 10). 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 blood flow – which cannot be too the 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.

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

Figure 6. Simulated blood vessel represented by the cylinder, dots are platelets.

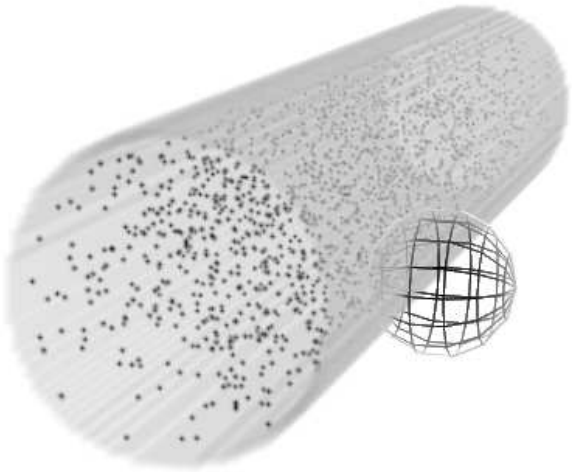

Figure 7. Simulation viewed from different angle, with wound placement tool on right.

levels of chemical and, then, clump together when they bump. Refining 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 refinement will not intefere 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. Simulations have been run with with more than 3,000,000 processes on commodity desktop hardware (P4, 3.0Ghz, 1GB 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 up an average of around 60,000 agents – actual numbers will be changing all the time. Cycling each with an average processing time of 2 microseconds (including barrier synchronisation, channel communica-

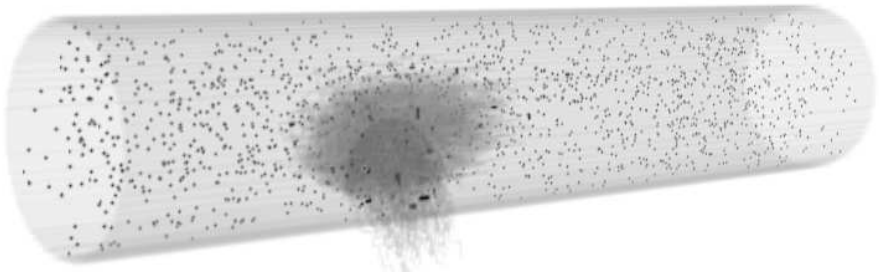

Figure 8. Having placed a wound, platelets "fall" out of the blood vessel, and chemical factors can be visualised by the darkened area.

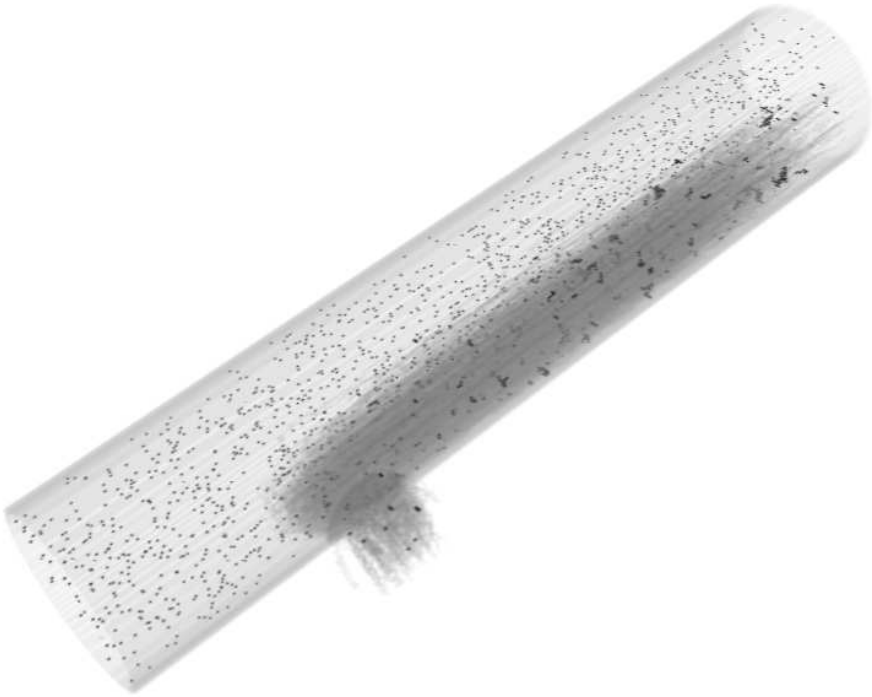

Figure 9. Given time, chemical factors flow down the blood vessel and (small) clots can be seen forming as dark blobs.

tion and cache miss overheads) still enables around 8 simuations steps per second, which is very useable.

Figure 11 shows performance for simulations on a world of size 256x96x96 (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 stablise 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. At 0.5% platelet concentration (an average of approximately 5,000 agents), we are achieving around 13 simulation/steps a second. All these results have visualisation disabled; in practice, most commodity graphics hardware has difficult 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%, performance degrades linearly. 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.

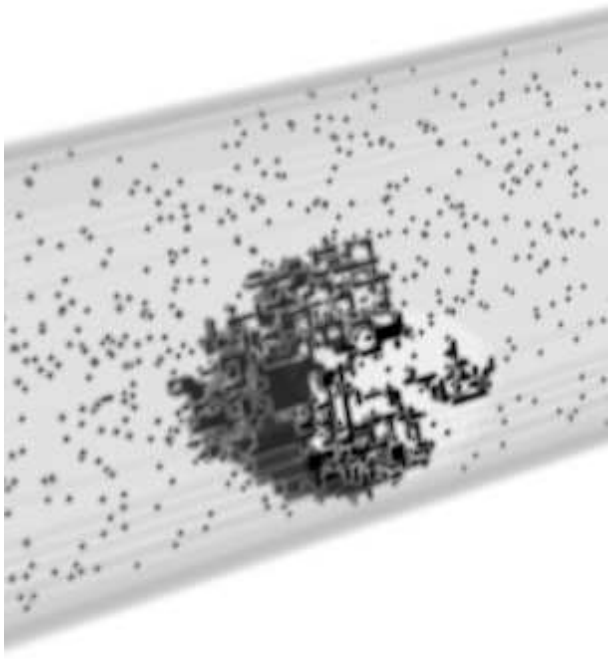

Figure 10. With sufficient time and a high enough platelet concentration a clot forms over the wound.

For the simulations whose results are shown in Figure 12, the platelets and their associated clots are initialised sticky. This is the worst case (and unrealistic) scenario where clots will form whenever two platelets collide. As expected, performance is lower than that in Figure 11, because the there are more agents. As clots form, they slow down. This means that platelets leave the simulation at a lower rate than they are entering and numbers rise. Even then, performance rates stabilise given sufficient time and the relationship between the levels of platelets is consistent.

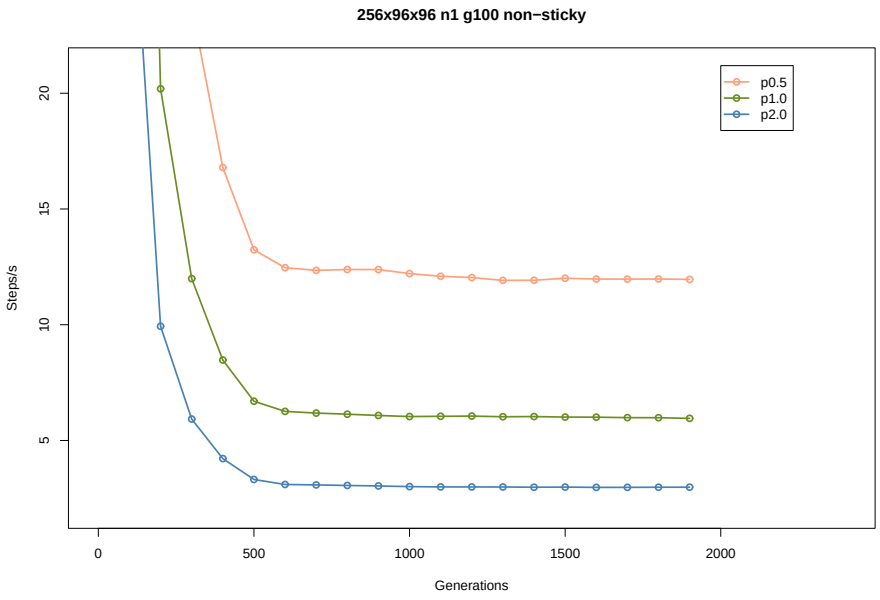

Figure 11. 256x96x96 simulations with non-sticky platelets.

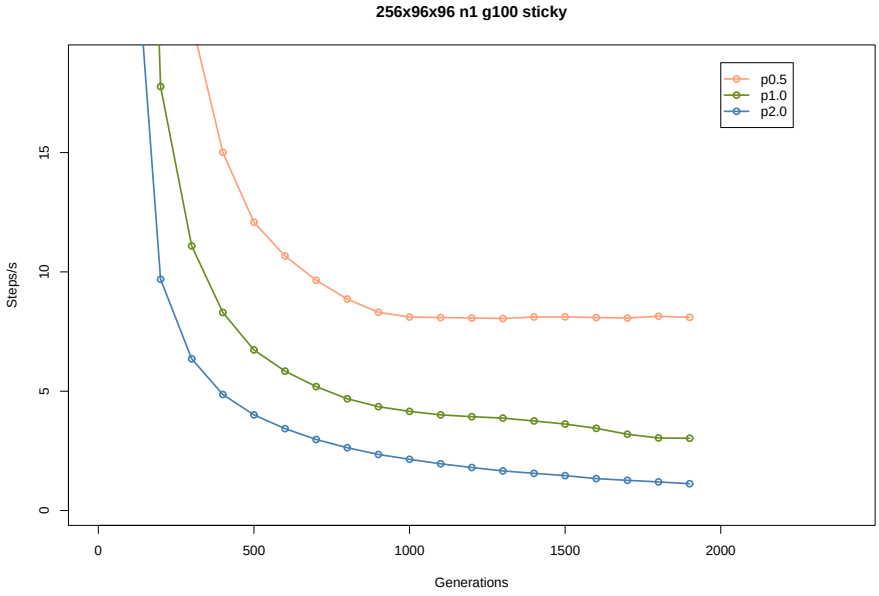

Figure 12. 256x96x96 simulations with sticky platelets.

4.3. Future Work

The next steps in our research are to expand and refine our simulations. For the former, we need to use either more powerful single machines or, more sensibly, clusters of machines. The later will be possible using pony [22], an networking environment for the occam-π runtime system. We have begun tesing a cluster-based implementation of these simulation models and initial results, not published here, are quite 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, chenical 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 and 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 targetted efficiently to modern hardware platforms. We invite others to try.

References

[1] P.H. Welch and F.R.M. Barnes. Communicating mobile processes: introducing occam-pi. In A.E. Abdallah, C.B. Jones, and J.W. Sanders, editors, *25 Years of CSP*, volume 3525 of *Lecture Notes in Computer Science*, pages 175–210. Springer Verlag, April 2005.

[2] The occam-pi programming language, June 2006. Available at: `http://www.occam-pi.org/`.

[3] S. Stepney, P.H. Welch, F.A.C. Pollack, J.C.P. Woodcock, S. Schneider, H.E. Treharne, and A.L.C. Cavalcanti. TUNA: Theory Underpinning Nanotech Assemblers (Feasibility Study), January 2005. EPSRC grant EP/C516966/1. Available from: `http://www.cs.york.ac.uk/nature/tuna/index.htm`.

[4] P.H. Welch, F.R.M. Barnes, and F.A.C. Polack. Communicating Complex Systems. In Michael G. Hinchey, editor, *Proceedings of the 11th. IEEE International Conference on Engineering of Complex Computer Systems (ICECCS-2006)*, pages 107–117, Stanford, California, August 2006. IEEE. ISBN: 0-7695-2530-X.

[5] S. Schneider, A. Cavalcanti, H. Treharne, and J. Woodcock. A Layered Behavioural Model of Platelets. In Michael G. Hinchey, editor, *ICECCS-2006*, pages 98–106, Stanford, California, August 2006. IEEE.

[6] S. Stepney, H.R. Turner, and F.A.C. Polack. Engineering Emergence *(Keynote Talk)*. In Michael G. Hinchey, editor, *ICECCS-2006*, pages 89–97, Stanford, California, August 2006. IEEE.

[7] F. Polack, S. Stepney, H. Turner, P.H. Welch, and F.R.M.Barnes. An Architecture for Modelling Emergence in CA-Like Systems. In Mathieu S. Capcarrère, Alex Alves Freitas, Peter J. Bentley, Colin G. Johnson, and Jon Timmis, editors, *Advances in Artificial Life, 8th European Conference on Artificial Life (ECAL 2005)*, volume 3630 of *Lecture Notes in Computer Science*, pages 433–442, Canterbury, UK, September 2005. Springer. ISBN: 3-540-28848-1.

[8] C. Ritson and P.H.Welch. TUNA: 3D Blood Clotting, 2006. `https://www.cs.kent.ac.uk/research/groups/sys/wiki/3D_Blood_Clotting/`.

[9] A.T. Sampson. TUNA Demos, January 2005. Available at: `https://www.cs.kent.ac.uk/research/groups/sys/wiki/TUNADemos/`.

[10] P.H. Welch, G.R.R. Justo, and C.J. Willcock. Higher-Level Paradigms for Deadlock-Free High-Performance Systems. In R. Grebe, J. Hektor, S.C. Hilton, M.R. Jane, and P.H. Welch, editors, *Transputer Applications and Systems '93*, volume 2, pages 981–1004, Aachen, Germany, September 1993. IOS Press, Netherlands. ISBN 90-5199-140-1. See also: `http://www.cs.kent.ac.uk/pubs/1993/279`.

[11] J.M.R. Martin and P.H.Welch. A Design Strategy for Deadlock-free Concurrent Systems. *Transputer Communications*, 3(4):215–232, October 1996.

[12] F.R.M. Barnes and P.H. Welch. Prioritised dynamic communicating and mobile processes. *IEE Proceedings – Software*, 150(2):121–136, April 2003.

[13] R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes – parts I and II. *Journal of Information and Computation*, 100:1–77, 1992. Available as technical report: ECS-LFCS-89-85/86, University of Edinburgh, UK.

[14] F.R.M. Barnes, P.H. Welch, and A.T. Sampson. Barrier synchronisations for occam-pi. In Hamid R. Arabnia, editor, *Parallel and Distributed Processing Techniques and Applications – 2005*, pages 173–179, Las Vegas, Nevada, USA, June 2005. CSREA press. ISBN: 1-932415-58-0.

[15] P.H. Welch and F.R.M. Barnes. Mobile Barriers for occam-pi: Semantics, Implementation and Application. In J.F. Broenink, H.W. Roebbers, J.P.E. Sunter, P.H. Welch, and D.C. Wood, editors, *Communicating Process Architectures 2005*, volume 63 of *Concurrent Systems Engineering Series*, pages 289–316. IOS Press, September 2005. ISBN: 1-58603-561-4.

[16] Hemostatis. URL: `http://en.wikipedia.org/wiki/Haemostatis`.

[17] Fibrin. URL: `http://en.wikipedia.org/wiki/Fibrin`.

[18] Disorders of Coagulation and Haemostasis. Available at: `http://www.surgical-tutor.org.uk/default-home.htm?core/preop2/clotting.%htm`.

[19] J. Griffin, S. Arif, and A. Mufti. *Immunology and Haematology (Crash Course) 2nd Edition*. C.V. Mosby, July 2003. ISBN: 0-7234-3292-9.

[20] W. Schroeder, K. Martin, and B. Lorensen. *The Visualisation ToolKit*. Kitware, 2002.

[21] D.J. Dimmich and C.L. Jacobsen. A Foreign Function Interface Generator for occam-pi. In J.F. Broenink et al., editor, *Communicating Process Architectures 2005*, volume 63 of *Concurrent Systems Engineering Series*, pages 235–248. IOS Press, September 2005. ISBN: 1-58603-561-4.

[22] M. Schweigler and A.T. Sampson. pony - The occam-pi Network Environment. In *Communicating Process Architectures 2006*, Amsterdam, The Netherlands, September 2006. IOS Press.

[23] I.V. Pivkin, P.D. Richardson, and G. Karniadakis. Blood flow velocity effects and role of activation delay time on growth and form of platelet thrombi. *Proceedings of the National Academy of Science*, 103(46):17164–17169, October 2006.