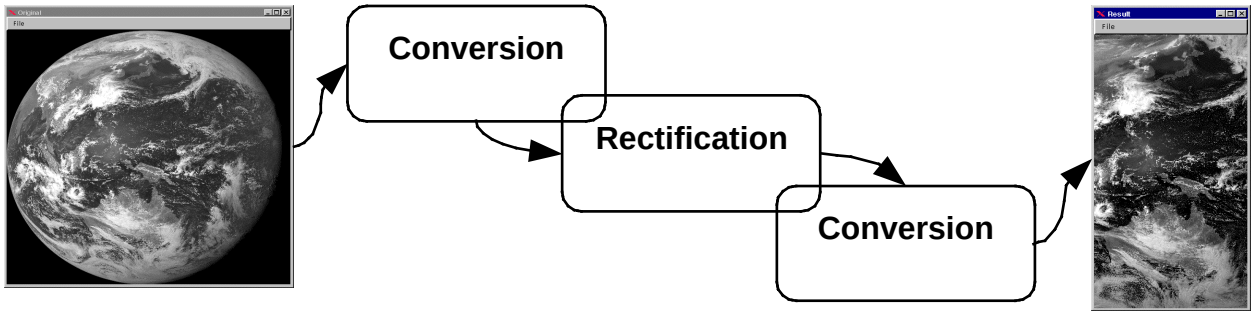
DISTRIBUTED PROCESS NETWORKS PROJECT

Progress and Directions
16th August 1999

Aims of the DPN Project

Our work is an ongoing project to develop a geographic information system (PAGIS [ref: WebWM99]) that allows a non-specialist user to define, in abstract terms, a processing network over diverse computational and data resources, and establishes distributed computations to implement the requirements thus specified. PAGIS represents computations using a high-level notation known as a "process network" (PN); a user simply selects an input image (retrieved from a repository of satellite images), then builds a network that composes various operations (drawn from an extensive library of possible operations) on the original image, defining a result image. The main advantage of the notation is that it is high-level and readily understood, and clearly expresses possible parallelism. PAGIS exploits this parallelism in a distributed environment, enabling a domain expert to construct distributed applications without expertise in distributed programming.

Here is an example of the operation of PAGIS:



In this example, we show an image (from the GMS-5 geostationary satellite) input to a simple processing chain. The chain first performs a conversion to a format suitable for processing, then performs the processing step of geo-rectification (which corrects distortions in the image due to the perspective of the camera), and then a format conversion back to the image storage format. We can readily express more complex and interesting computations. Writing the above processing chain (abbreviating the conversions, and adding an operation to apply a colour filter) as the composition:

```
conv1 . rectify . grdimage . conv2
```

we can compose that in parallel with a similar chain acting on another image, then join the two chains into an operation that, for example, forms the two processed images into a composite, and composes that with another operation to overlay a map.

In PAGIS, the operations are taken from a commonly used toolset known as GMT (the Generic Mapping Tools) [ref: WesS95]. GMT includes many useful image processing and mapping operations. We have built our prototype PAGIS around it, but our design ensures that we can incorporate toolsets for other application areas, such as processing of audio-visual data streams, or processing of geophysical datasets.

Our examples above show the syntax of process composition. We used both a graphical (the middle of the three panes in the figure above), and a textual, syntax for describing a processing chain. Our system can support a variety of front-end syntactic tools, and maintain consistency between them. For example, we have a prototype graphical user interface whereby GMT and control operations can be selected from a palette and composed into a (cyclic or acylic) network. The GUI is intended for use by GIS domain experts to construct a processing system in an intuitive manner. We also use a primitive textual interface (a linearized representation of a network), for development. A suitable programming language can also be used, such as Sisal, used by Wendelborn in earlier work with similar networks [refs: WenG93, GarW92].

To understand what these PNs compute, we use an operational semantic model introduced by Kahn [refs: KahM77]. This is a formal model that captures the computational meaning of the net as a function of the streams of data that flow on its channels. The formal model can be used to reason about properties of the net (by writing and proving theorems about its behaviour). We do not pursue that in our project, but it is important to realize that all our implementation strategies conform to the semantic model, hence any such proofs are valid for our implementation.

The semantic model must, of course, lead to a practical realization of PNs. It can be shown that an implementation satisfying some simple properties conforms to the semantics. While this restricts slightly the computations that can be directly expressed, it leads to enormous flexibility of implementation structures, a fact pivotal to our strategies. It is the practical realization of general PNs, in a distributed context and in support of systems such as PAGIS, that is currently our primary focus.

The PN model as developed so far admits a variety of implementation strategies. The nodes are arbitrary computations, and tokens of arbitrary type and structure flow on the channels. Thus, the model above can be applied using different programming languages and data types. Further, we can implement the same network, producing the same results, using different strategies. Importantly, the nodes of a PN can execute in parallel, and describe both fine and coarse-grained parallel decomposition.

There have been several realizations of the PN concept. Kahn and MacQueen [ref: KahM77] outlined an implementation based on implicit coroutines, with PNs described with a textual notation of functional composition. Bohm [ref: DeBB85] developed a programming language and Unix-based implementation. Wendelborn implemented [ref: Wen82] a technique using explicit coroutines, with a conventional programming language used to construct the PN, explored [ref: GarW92] different modes of evaluation in (related) data-flow networks, and later [ref: WenG93] implemented and measured pipelined parallelism in PNs programmed in the parallel functional language Sisal. Lee *et al*, in the Ptolemy project at the University of California at Berkeley, have carried out significant work in utilizing PNs and data-flow networks in signal processing [ref: LeeP95]. They exploit flexibility of implementation structure with a generalized software architecture that enables definition of several flow-based *domains*, with separate definition of the *computational engine* which defines the behaviour of nets in that domain. Examples are: SDF (synchronous data flow), in which networks can be analyzed for channel size and other parameters, and a static schedule constructed; DDF (dynamic data flow), with a computational engine capable of interpreting operations in the network; and PN, a process network domain based on a simple model of parallel computational threads. These domains essentially constitute a hierarchy based on granularity of parallelism (fine grain data-flow, statically scheduled in the case of SDF, throught to coarser grain PNs).

Our current work is complementary to this latter development, and extends it in several ways.

Primarily, our interest is in creating a distributed computational engine for PNs. This can be regarded as extending the Ptolemy hierarchy with a new domain, DPN, capable of implementing computations over a wide-area (nationwide) *metacomputing* system. Interestingly, such a hierarchy can be reflected in practice, with nodes of the highest level PN potentially realized as networks over a lower level domain, such as DDF.

A metacomputing environment is one in which applications can be built over multiple resource nodes (a node is typically a high-performance computer or cluster of workstations) at widespread geographic locations. Usually, a user of a metacomputing environment desires transparent access to the entire system. Typically, user access is Web-based, with transparency provided by a layer of software implemented on all nodes. Such software is often written in Java [ref: ArnG96], an object-oriented language of increasing popularity. Java is particularly relevant in the context of metacomputing because it is portable across most machines, is in common use for writing Web interface software, and is recognized as a good programming language for both general and distributed programming. Important metacomputing projects include Globus [ref: FosK97], Legion [ref: GriW96] and, at the University of Adelaide, DISCWorld [ref: Haw98]. There are many issues involved in successfully implementing a metacomputing environment, such as discovering resources in the environment appropriate to an application, decomposing an application and scheduling it over those resources, tolerating latency of long-distance communication, and utilizing variable network connections between nodes. Our work is at the level of the "glue" layer of software that provides transparent access to the facilities of a metacomputing system.

In our case, in distributing a PN, we aim to exploit freedom of implementation structure in several interesting ways:

1. Placement, scheduling and migration strategies: we can place work on any suitable node, schedule its execution as appropriate, and move it between nodes, without affecting the result;
2. Utilize flexible evaluation modes in the network, for example, data driven, in which data is "pushed" through the network as it becomes available, allowing optimization for maximum throughput (e.g. streaming a satellite image, or a video/audio feed, through several stages of processing); demand driven, whereby data is "pulled" through the network according to need for it in producing a result, which can be used to ensure that if only a small percentage of a large dataset is needed to produce a result, then only that is requested and retrieved; and hybrids of the two (for example, a portion of a dataset is "demanded", but that portion is then processed in optimized data flow mode);
3. Reconfiguration of the network: the PN model allows a network to expand and contract according to requirements of a computation, which can be used to facilitate expression of parallelism and identification of units of work for distribution;
4. A hierarchy of computational engines, as mentioned above;
5. Manipulation of behaviour: we can change behavioural parameters dynamically, without affecting the overall result, so that a PN can respond transparently to environmental changes (such as the sudden availability of a high-speed transmission link), for example, by interchanging a PN node for an equivalent version with an alternative algorithm better able to exploit the changed environment.

Our current work is concerned with improving design and architecture of our PAGIS prototype in order to provide a sound basis for achieving the above aims. The prototype is an experimental system that, due to some decisions made to benefit rapid prototyping, suffers from a design that limits experimentation and extensibility.

The first step in the redevelopment of this prototype has been the use of current object-oriented design methodologies for specification of structure and behaviour of process networks. We have developed a Process Network API (Application Program Interface) that

precisely characterizes a minimal set of operations necessary to build PNs, that is, the methods necessary to construct a representation of PN syntax. The API is very useful in that PNs can be programmed directly using it, or it can be used as a PN representation by front-end syntactic tools. Further, PNs written to this API are independent of the underlying implementation. It of itself makes a significant contribution in providing a general tool for research in PNs.

One significant aspect of the API is clarification of some aspects of reconfiguration (as defined in [ref: KahM77] and explored further by Wendelborn [ref: Wen82] in both a coroutine-based and a distributed environment). Any PN node can perform a reconfiguration step, transforming itself into a larger set of nodes "spliced" into the PN, or disappearing altogether, "tieing" its input to its output channel. It is this latter case that often proves problematic: we have devised a general method for combining the two channels of a node that vanishes. We have precisely characterized primitives needed to express reconfiguration in a computation, and implemented them in a ProActive (see below) testbed.

The API has been developed in collaboration with J. Vayssiere of INRIA Sophia-Antipolis, who is visiting Adelaide under a 1999 Small ARC Grant supporting on-going collaboration with INRIA. He is co-designer of ProActive, a transparent extension to Java to provide high-level abstractions for concurrent, parallel and distributed programming. It is notable for uniform support of both local and remote (i.e. resident on another computing node) objects, active objects (used similarly to a normal object but hosting a separate parallel activity), and a high-level abstraction for asynchronous communication.

Some future plans

We have near completion three implementations based on our API [ref: VayWW99], intended to both validate the API and provide a basis for experimental evaluation. One implementation is directly in terms of ProActive: we have used that to clarify aspects of PN implementation structure important for a distributed implementation, in particular, those aspects that are best represented as active objects (with their own thread of control), rather than conventional passive (data only) objects. We are exploring the notion that it is useful to regard both processes (nodes of a PN) and channels (the arcs of a PN graph, usually regarded as a queue of values) as possessing activity. The latter choice is novel, but it allows us to encapsulate both a complete representation of information needed for scheduling of PN nodes, and a mechanism for determining mode of evaluation ((2) above), in just the channel active objects.

Our second implementation is of PAGIS directly in terms of our PN API, and the third, PAGIS and the API using ProActive as the mechanism for distribution. These are near completion, about to be used for performance evaluation and experimental analysis of distribution mechanisms, in the context of GMT-based GIS applications.

These are some other things we would like to do in the near future:

1. Consolidation of the PAGIS prototype to a state where it can be readily used by researchers in the GIS domain, hence facilitating the establishment of application-oriented interdisciplinary projects, and possibly interesting external funding agencies;
2. Implementation of a suitable representation of "behaviour" in the prototype, to facilitate the monitoring of a PN in execution, but particularly to realize behaviour manipulation as expressed in (5) above.

With the development of the PN API, the prototype currently exists in a form difficult for demonstration or production release, and the prototype GUI does not provide all the functionality that domain experts typically expect of GIS software. In addition, the GMT software at the heart of our computational engine has recently been enhanced and re-released by its designers. We will re-evaluate the software and, as was the case with the version used

in our prototype, make modifications to the software necessary for GMT to manipulate GMS-5 satellite imagery. To achieve a more comprehensive system, we aim to add relevant GMT operations to the palette currently supported. We may also look to optimize portions of the GMT code for more efficient processing chains.

For (2), we again exploit the fact that PN nodes cannot interfere with each other's computations, meaning that we can substitute semantically equivalent implementations of that node without affecting the overall result (i.e. we can safely dynamically reconfigure a PN). To accomplish this, we are considering a technique known as *metaprogramming* [ref: KicRB91], writing essentially separate programs for the application itself (exactly as we do now), and another program (the *metaprogram*) written in terms of objects that represent pertinent aspects of the implementation itself. For example, in ProActive, an invocation of a communication method is represented as an object at this meta-level, and can thus be manipulated to customize its behaviour. Then, with suitably designed base/meta separation, we can use two interfaces: one which provides the basic functionality (the usual API), and one which can be used to ask about, monitor or adjust the functionality available through the first interface. Such techniques have not been used before in PN implementations, nor to the best of our knowledge, in metacomputing software. We believe that a carefully designed metaprogramming layer and interface could be a very powerful tool in writing computations that adapt to changes in their computing environment, a common occurrence in metacomputing applications, which are hosted on diverse resources with highly variable operational parameters. The key to success here:

- Determining, and designing an object-level representation of, important aspects of behaviour (programming with those objects uses the same techniques as normal object-oriented programming), and
- Characterization of a suitable interface (the set of behaviour-modifying commands).

An example of what we seek to do arose in an earlier project [ref: TiaWM98]. We wanted the computation to be able to use a higher-bandwidth Internet connection when it was available, which we achieved with unstable, error-prone programming in the internals of the communication system. With a suitable meta-object representation of the communication parameters and the communication call, we can program this cleanly within the metaprogram. Our PAGIS design, and experience in developing and evaluating it, provides a sound basis for formulating, implementing and evaluating a minimal, efficient behavioural metaprogram.

References

[ArnG96] Arnold, K. & Gosling, G. *The Java Programming Language*, Addison-Wesley.

[CarKV98] Caromel, D., Klauser, W. & Vayssiere, J. "Towards seamless computing and metacomputing in Java", *Concurrency: Practice and Experience,* 10(11-13):1043-1061, Sept-Nov 1998.

[DeBB85] De Bruin, A. & Böhm, W. "The Denotational Semantics of Dynamic Networks of Processes". *ACM Trans. Programming Languages and Systems*, 7(4):656-679, October 1985.

[FosK97] Foster, I. & Kesselman, C. "Globus: A metacomputing infrastructure toolkit". *Intl. J. Supercomputer Applications and High Performance Computing",* 11(2):115-128, Summer 1997.

[GarW93] Garsden, H. & Wendelborn, A.L., "Experiments with Pipelining Parallelism in SISAL", *Proc. 25th Hawaii International Conference on System Sciences,* Hawaii, January 1992. 10 pages.

[GriW96] Grimshaw, A. & Wulf, W.A. "Legion: A view from 50,000 feet", *Proc. 5th IEEE Intl. Symp. on High Performance Distributed Computing",* Los Alamitos, CA, Aug. 1996.

[Haw98] Hawick, K.A., *et al*, "DISCWorld: An Environment for Service-Based Metacomputing", *Fifth Generation Computing Systems (Special Issue on Metacomputing)*, 1998.

[KicRB91] Kiczales, G., des Rivieres, J., & Bobrow, D.G., *The Art of the Meta-Object Protocol"*, MIT Press, 1991.

[LeeP95] Lee, E.A. & Parks, T.M., "Dataflow Process Networks", *Proceedings of the IEEE*, vol. 83, no. 5, pp. 773-801, May 1995.

[TiaWM98] Tia, P.W., Wendelborn, A.L., and Maciunas, K.J. "An Investigation of Flexible Communication Infrastructure in a Distributed High Performance Computing Environment", *Proc. HPC Asia*, Singapore, Sep. 1998 10 pages.
[VayWW99] Vayssiere, J., Webb, D. & Wendelborn, A.L., "An Object-Oriented API for Process Networks" (draft) Comp Sci Technical Report 99-03, August 1999.
[WebWM99]Webb, D., Wendelborn, A.L. & Maciunas, K.J., "Process Networks as a High-Level Notation for Metacomputing", in *Parallel and Distributed Processing*, J. Rolim *et al* (eds), *Lecture Notes in Computer Science*, vol. 1586 (Springer, 1999), pp. 797-812 (presented at the IPPS/SPDP *Workshop on Java for Parallel and Distributed Computing*, Puerto Rico, April 1999.
[WenG93] Wendelborn, A.L. & Garsden, H. "Exploring the stream data type in Sisal and other languages", *Proc. IFIP WG10.3 Working Conference on Architectures and Compilation Techniques*, Orlando, Jan. 1993 (published as *IFIP Transactions*, vol. A-23, pp. 283-294).
[Wen82] Wendelborn, A.L., "Reconfiguration in the Process Networks of Kahn and MacQueen", *Australian Computer Science Communications*, vol. 4, no. 1 (Feb. 1982), pp.233-243.
[WesS95] Wessel, P. & Smith, W.H.F., *The GMT Version 3 Technical Reference*, available via http://www.soest.hawaii.edu/soest/gmt.html. University of Hawaii School of Ocean and Earth Science and Technology (SOEST).

Andrew L Wendelborn
16th August 1999