BRAHMS: Novel middleware for integrated systems computation

B. Mitchinson, T.-S. Chan, J. Chambers, M. Humphries, C. Fox, K. Gurney and T. J. Prescott

***Abstract*— Computational modellers are becoming increasingly interested in building large, eclectic, biological models. These may integrate nervous system components at various levels of description, other biological components (e.g. muscles), non-biological components (e.g. statistical discriminators or control software) and, in embodied modelling, even hardware components, all potentially with different authors. There is a need for middleware to facilitate these integrated systems. BRAHMS, a Modular Execution Framework, fills that need by defining a supervisor-process interface and an (extensible) set of process-process interfaces; authors can write to these interfaces, and processes will integrate as required. Additional benefits include reuse (never code the same model twice), cross-user readability, system-level parallelisation on multi-core or multi-node environments, cross-language integration, data logging, performance analysis, and run-stop-examine-continue execution. BRAHMS employs the nascent, and similarly general purpose, model markup language, SystemML. This will, in future, also facilitate repeatability (same answers ten years from now), transparent automatic software distribution, and interfacing with other SystemML tools.**

I. INTRODUCTION

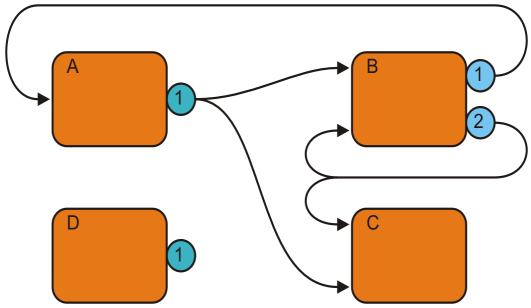

Fig. 1. Example block diagram of an integrated dynamic system built from independent processes. Rectangles represent processes, circles output ports, and arrows links between processes.

Moves are afoot in the world of computational neuroscience towards the construction of large integrated models[1], [2], [3], [4], [5], [6], [7], [8], in the spirit of Daniel Dennett's 'whole iguana'[9]. Multi-region brain models are complex, so researchers necessarily develop them as collections of sub-systems, or 'processes', which must then be 'linked' together to form a 'system' that can be computed, Fig. 1. This aside, modularity is widely considered to be a desirable trait in software design [10]. We define a Modular Execution Framework (MEF)[1] to mean any middleware that facilitates such linkage—a well-known example is Simulink[11]. Neuroscientists are wont to represent processes at different levels of abstraction (biophysical, neuronal, network, control-theoretic, etc.)[12], so it is not generally possible to compute all processes in one computation engine—rather, the task is to link *engines* together[8]. Bespoke integration serves for any single project but, as we will see, a general solution offers much more for less effort, and the startup cost is considerably outweighed by the immediate benefits.

In Section II, we run over the particular challenges and requirements for integrated computation in academic research. We go on in Section III to introduce our proposal, BRAHMS[13], beginning with an overview of its use. In Section IV, we contrast BRAHMS with existing and developing solutions and show it to be well positioned with respect to these. We report on project status in Section V and conclude in Section VI that BRAHMS already offers a solution to most of the identified challenges and will, through planned developments, meet the remainder.

Whilst BRAHMS has its roots in solving problems in and around computational neuroscience, we emphasise that it is not limited by those origins, and we expect it to be of equal interest to researchers in other fields. The problems of integration, of course, become all the more visible when crossing disciplinary boundaries.

II. CHALLENGES AND REQUIREMENTS

A. Varied Development

The primary challenge (as described above) is to integrate software processes, and the primary requirement is, therefore, to offer a middleware platform which will execute processes in concert. Processes may be developed in, on, or by different authors, labs, platforms, programming languages, human languages, programming styles and at different times. The inclusion of non-neuroscience processes generalises the problem across problem domains and technical languages. Without direct communication and refactoring, such disparate offerings will not generally be integrable. Software engineers meet such challenges by offering fixed, public, interfaces to develop against. In this context, an interface requires two facets: one between process and framework, the other between processes; these interfaces must be general (exclude no possibilities), static (or maintain backward compatibility), accessible and available in multiple programming languages on multiple platforms.

The authors are with the Adaptive Behaviour Research Group at the Department Of Psychology, The University Of Sheffield, Western Bank, Sheffield, S10 2TN, UK. Supported by European 6th Framework Grant IST 027819 ICEA and EPSRC Research Grant EP/C516303/1.

Corresponding author b.mitchinson@shef.ac.uk.

[1]We prefer 'Execution' to 'Simulation' since, in general, some processes will not be simulations.

B. Varied Deployment

High-end multiprocessing hardware is increasingly becoming available to research labs, supporting a rapid growth in the development of 'large-scale' models[14] (models with many dynamic states). At the same time, increasing focus on 'embodied modelling'[15], [16] (deployment of behavioural models on robotic hardware) is generating use cases based on low-end hardware. These two trends push the computational envelope at opposite ends, and any solution must be deployable in all these environments. A researcher may develop initially on a desktop machine, for convenience; experimental work may involve large models or parameter spaces and, thus, high-end hardware; embodied models will eventually be deployed on robots, and, particularly if there is an intellectual interest in mobile robotics *per se*, this may mean running on low-end embedded hardware. The requirements are, that a researcher should only have to develop once for such varied deployment cases, and that the middleware should be able to take advantage of the resources of high-end and desktop hardware without becoming unwieldy on low-end hardware.

C. Code Sharing

Computational researchers spend much time authoring software, and disappointingly often this work is repeated in other labs, by other researchers, or even by the same researchers, when documentation, compatible source-code and/or compatible binary code is, or becomes, unavailable. Anecdotal (and more concrete[5], [17]) evidence suggests that '...easier to rewrite it myself than try and obtain/understand/integrate their original code...' is a common story. Any solution should offer great potential for code sharing and reuse, which is to say more than that the code *could* be integrated—it must be *straightforward* to do so. This requires a (preferably, automatic) archiving/distribution mechanism, that shared code be in a form that is immediately usable (rather than having to be compiled, say), and that the solution encourages authors to document their work[18] (facilitating 'intellectual' integration).

In addition, 'background functionality' like parallelisation or data marshalling is neither trivial nor quick to author (and most researchers do not want to become software engineers), so sharing such functionality with all process developers is desirable. Therefore, as much functionality as possible should be subsumed into the middleware—'general' process code should be shared. We use the term 'supervisor' to refer to this shared code, which might, at minimum, be responsible for reading a system document, loading required processes, connecting them together, progressing them through time, collating results, and returning these to the caller.

D. Open Standards

There is more to working with systems than their execution; other possibilities include a system design GUI and an archival/retrieval tool (see also Section II-C). It is, thus, a requirement that the middleware should work with open and extensible data standards. Moreover, the needs of academic research are constantly changing and often cannot wait—the solution must be able to support these changing needs in a timely manner. In practice, therefore, it is a requirement that the solution be open source and extensible by anyone.

E. Adoption

If they are to adopt any proposal, potential users must perceive its advantages to outweigh its costs, both initially and in the long-term. The short-term requirement is met if the startup cost is sufficiently low—interfaces must be few, simple, and well documented; immediately available 'added value' will help to offset this cost. The long-term requirement is met if, in comparison with an equivalent bespoke monolithic design, overall performance does not suffer and per-process development effort is similar or less. Furthermore, an integration solution should by its nature be inclusive, so the solution must be available to all—this means affordable (preferably with no cost). Adoption will be the more willing the more freedom that is given to the developer to do things their way—this means making the interface available in multiple languages, on multiple platforms.

III. BRAHMS

A. Overview

BRAHMS represents part of our commitment to the philosophy and methodologies of neuroinformatics: developing general purpose tools that facilitate large-group working in neuroscience, and sharing and reusing resources. It is an MEF developed in-house during the course of a large-scale, multicentre project (WhiskerBot [3]) which presented many of the challenges outlined in Section II. This was a computational neuroscience project, but from the outset BRAHMS was required to integrate diverse processes (see Section III-H). The design goals of BRAHMS are performance, flexibility and extensibility. BRAHMS is open source and licensed under the GNU General Public License.

BRAHMS operates on systems, Fig. 1, progressing them through time and generating output, Fig. 2, which comprises, in large part, logs of the links between processes. Processes can be developed by dropping state initialization and update code into one of the provided templates (using a programming language that suits the developer). Systems are built from these, processes developed by other researchers, and processes provided with BRAHMS, in just a few lines of script. BRAHMS is invoked to execute these systems, taking advantage of parallel computing resources where available, and the results can easily be pulled into an analysis environment. BRAHMS is not tied to any particular interactive environment but, currently, the support offered for working in Matlab[11] is particularly strong. In time, the library of available processes will increase and additional supervisor functionality will accrue (see Section VI-B for future plans).

B. Systems

A BRAHMS system is a snapshot of a stateful dynamic system in time; that is, a collection of stateful processes and a collection of stateful links connecting them together. A system is loaded from file at invocation, and stored back to

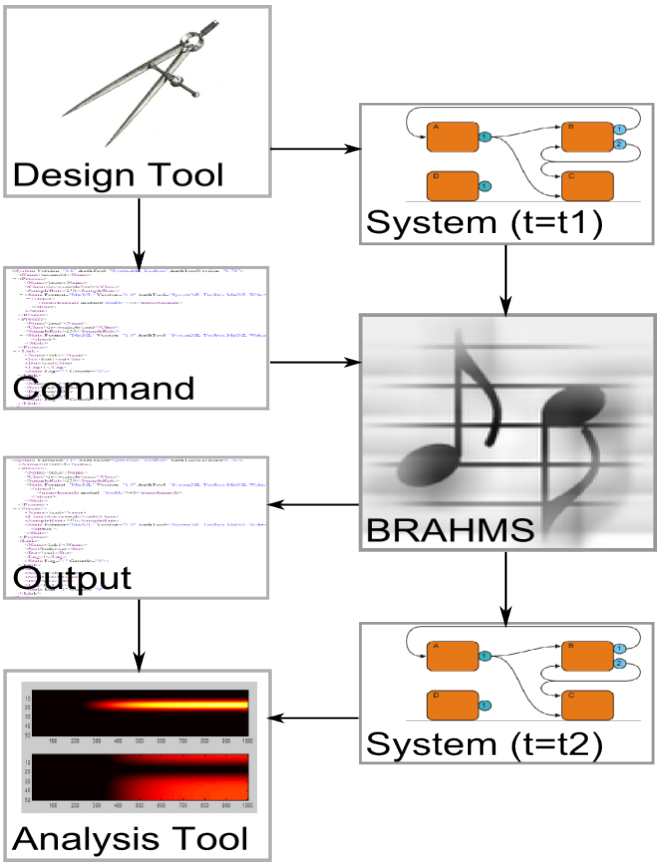

Fig. 2. Typical BRAHMS workflow: system and execution command file design, execution by BRAHMS (progression of the system through time), analysis of final system state and execution output.

file at termination (the file format used is SystemML[19], an open XML-based format for representing stateful dynamic systems). Each process is specified as a class, which indexes an extensible set of process classes each implementing some algorithm, and class-specific state data. A process may publish outputs, each of which specifies a transport protocol (periodic/aperiodic, sample rate) and a data container. Each data container is specified as a class, which indexes an extensible set of data container classes each implementing some data structure, and class-specific state data. Each link specifies a source output (implicitly, thus, a transport protocol and data container class), a destination process, and a transport delay.

The class-specific process or data container state data (termed 'StateML') is understood by design tools for and implementations of those classes (for instance, NeuroML[8] might be the StateML for some neural process classes). These class-specific data are not used by BRAHMS so the set of systems that can be represented is extensible simply by the addition of new process or data classes. Transport protocols are the responsibility of the middleware, however, so the addition of new protocols requires updating BRAHMS.

C. Interfaces

The core of BRAHMS is the supervisor-process interface—C was chosen as the language for this for its durability and interoperability with other languages. Language bindings are provided for C++ and Matlab, and additional bindings are expected to follow (Python bindings are currently under development). The interface as a whole has been designed according to modern API design principles[20], for example with an eye to minimality, extensibility, and backward compatibility. It comprises four aspects, as follows.

First, processes offer an extensible events interface through which the supervisor can invoke process operations (such as initialization, progressing through time). The remaining three aspects are callback interfaces allowing processes to invoke supervisor functionality. They are an implementation of parts of the W3C XML Document Object Model interface[21] (for manipulating StateML, amongst other things), the inter-process interface (for interacting with links), and a general interface of BRAHMS-specific functions.

In addition to the above, data container classes must offer a class-specific interface to allow manipulation of their contents by processes. Inter-process communication then proceeds as follows: the source process publishes a port during initialization, specifying the class and structure of data containers to pass over it, and this forms one end of a link; at run-time, the source process writes data into a container using the class-specific interface, and passes the container into the link; the destination process pulls the container from the other end of the link and reads the contents using the class-specific interface.

D. Modules

Process classes are implemented in modules, in one of the languages for which bindings are provided. A process module must respond to calls from the supervisor on the events interface, performing the requested operations. A simple process module might respond to only two events, first to publish an output, and second to pass data into it on each time step. Data classes are implemented in very similar modules. A data module receives a different set of events from a process module, but otherwise operation is similar. A simple data module might respond to two events, to log its current state for later retrieval, and to return its log to the supervisor. In addition, a data module must offer an interface to its content for use by processes, as mentioned above.

E. Supervisor

The BRAHMS supervisor is authored (in C++) as a standalone executable, so it does not require a virtual machine or scripting engine and is therefore resource-light. It is invoked with a command file, see Fig. 2. It then reads the system file, instantiates the system (by loading modules and passing them their state from the system file), connects processes together, then supervises the progression of the system through time, managing the transport of data through the links. At a predetermined stop time (or following cancellation by the user or by a process), it collates the state of all loaded modules (processes and data containers) and links and writes this back to a system file, also collating logs of each link specified for logging in the command file and writing these

to an output file. Most often, it is this output file of logged inter-process links that is the desired product of an execution.

Automatic parallelisation is provided at a coarse-grained (process) level. Finer-grained parallelisation can be implemented within processes, but it will generally be easier to break processes up and let BRAHMS parallelise them (models that consist of collections of similar objects, like network models, lend themselves particularly to this technique). One implementation of the supervisor, 'Solo', offers lightweight parallelisation in a shared-memory environment using multithreading. Another, 'Concerto', offers multiprocessing for parallelisation over a computing cluster, currently using either TCP/IP or MPI as the communications layer.

The data transport provided by the supervisor might comprise sharing a memory pointer (Solo), or routing a container over ethernet (Concerto), or through a stateful system file to another user, on another platform, at another time. Such transport operations are achieved without onus on the process developer. Background functionality beyond parallelisation includes inter-process data compression (Concerto), windowed data logging, distributed system performance monitoring, and a completely general 'pause & continue' execution model (allowing system snapshots at non-zero time to be coherently stored, exchanged, examined or modified).

F. Accountability

One of the dimensions of Varied Development, above, is time. This means that we should be able to integrate, today, process code that was generated many years ago. However, computing environments change, and researchers forget. BRAHMS offers accountability; that is, everything that bears on the results produced from an execution (details of each loaded library module, external library, of the run-time environment, platform, operating system, etc.) is recorded in an 'execution report'. If a repeat of the execution does not produce identical results, it is possible to identify why (thus, accountability favours repeatability by identifying sources of disagreement).

G. Software Development Kit

A BRAHMS release includes template processes and tutorial examples (corresponding to tutorials in the documentation) authored in all three currently supported languages. Thus, creating a new BRAHMS process involves little more than copying the template for the chosen programming language, and adding the content that performs the actual algorithm intended. Also included is a development version of the BRAHMS 'Standard Library', a collection of processes implementing simple operations (such as sum, product and resample) which are intended to be useful in production systems, whilst doubling as further illustrative material. This library also includes the data container class that will be most useful, 'data/numeric', which is a container for an N-dimensional array of numeric data in a comprehensive (and extensible) range of fixed-bit-width element formats.

Matlab interfaces for the command, system, and output files, and for invocation of the BRAHMS executable, are also include, such that BRAHMS can be called in Matlab using normal function call syntax. These allow the construction of systems from processes and links, and the design of the generic StateML used by the processes in the Standard Library. However, BRAHMS does not know about process state, in general, so additional tools are needed to design StateML for processes that do not use this generic StateML. Interfaces to BRAHMS from other environments are expected to follow in future.

H. Work Experience

The WhiskerBot robot is an embodied model of rat behaviour with an eclectic control model including hardware components (FPGA spiking neural simulators), neural software (leaky integrator models of Superior Colliculus and Basal Ganglia) and non-neural software (heuristic models of fixed behaviours and arithmetic/geometric modules for robot control). The model was developed on high-end desktop hardware and deployed with modification of parameters only on the low-end embedded compute platform of the robot, meeting sub-millisecond real-time constraints consistently. Large parts of the model have been adopted to contribute to the control software of the ScratchBot robot, an artefact of the ICEA project[4]. Other parts are currently in use as part of the large-scale oculomotor control model of the REVERB project[22] which deploys across a compute cluster and a separate robot-control machine. In another aspect of the ICEA project, existing WhiskerBot processes have been interfaced successfully with the Freebots[23] robot simulation environment. BRAHMS has also been chosen as the integration platform for the large European project BIOTACT[24]. Taken together, these use cases illustrate reuse, various dimensions of integration, and varied substrates of deployment.

IV. RELATED PROJECTS

We do not discuss neuroscience-only integration projects, such as NSL[25], NEOSIM and CATACOMB[26], since they do not attempt to solve the integration problem with the level of generality discussed here.

A. Simulink

Simulink[11] has a long history, and is a useful tool for learning about integrated systems. More recently, it offers multi-language support (Matlab, C, C++, Ada, Fortran) but as yet no standard support for parallelisation even within a shared-memory space, and it suffers from computational overhead. The data format is open (though not extensible, since Simulink is proprietary). In the long-term, support may improve in the technical areas where Simulink does not meet the requirements, but it is likely to remain costly, closed source and with a large resource footprint.

B. IKAROS

IKAROS[27] is a project of similar spirit to the BRAHMS project, but has rather different focus. Its positive points include the 'WebUI', which allows real-time monitoring of

system state through a browser, good documentation, and a simple developer interface. But where BRAHMS aims for to meet all the challenges listed above, IKAROS attempts to solve a much more constrained problem, albeit in a straightforward and effective way.

IKAROS is constrained to a single inter-process datatype (2D single-precision matrices), does not support dynamic creation and sizing of outputs based on connectivity, multiprocesses on a single machine only (currently), and the plugin architecture requires the whole system to be rebuilt from source when new modules are added. This last is a particular problem for accountability, since the code that executed to generate archived results is generally no longer available. There is no discussion of bindings for other languages; the IKAROS interface is authored in C++, which might become a problem in the future if the project intended to move towards dynamic loading or accountability. IKAROS also lacks some background functionality available in BRAHMS, though such features can probably, as for BRAHMS, be added without modifying the plugin codebase. In summary, we suspect that IKAROS and BRAHMS are solutions to different, if not quite orthogonal, problems.

C. Large-Scale Modeling Program and MUSIC

The International Neuroinformatics Coordinating Facility (INCF) have recently launched a program to foster infrastructure for researchers working with large-scale neural models. Attendees of the first program workshop[8] noted the large and growing set of neuron simulators available, and agreed on the importance of interoperability and component reuse. Focus was on modularity, particularly the supervisor-process interface, process-process interface, and common file format. Consequently, run-time inter-process communication was also discussed as a necessary future development to integrate computation engines into systems. They also highlighted background functionality (node allocation, communications initialisation) and marshaling of extremely large data sets.

All of these issues are addressed by our proposal. Other interoperability concerns raised in the workshop report are not applicable to BRAHMS; e.g. no application scripting is required since BRAHMS is responsible for procedure. Within the program, a communications library called MUSIC[28] is under development. The MUSIC approach leaves everything but inter-process communication to the process, in stark contrast to BRAHMS, which provides much common functionality. Each approach has its advantages—in particular, large and/or closed-source projects are more likely to suit a MUSIC interface than a BRAHMS front-end (though the latter need not be onerous). In contrast, BRAHMS allows extremely rapid development of powerful cross-platform engines, which MUSIC does not. We do not consider that BRAHMS and MUSIC will be direct competitors, and expect to offer a BRAHMS-MUSIC interface in future.

V. STATUS

BRAHMS has been public since April 2007, and is now approaching version 0.7, expected midsummer 2008. This branch will serve the ICEA, REVERB and BIOTACT projects (for three years). Interface logic is unchanged since December 2007 though minor syntax changes continue as foundations are assured for planned features. At 0.7, syntax will also freeze. Planned features will arrive in upcoming minor branches (0.8, 0.9, ...) towards version 1.0 which will end the initial development cycle. Processes authored against 0.7 will interoperate with future releases.

Solo is now fairly mature, having performed well with only minor changes for some years. Concerto still has alpha status, but is considered sufficiently mature for deployment in the REVERB project. All releases are available for Windows 32-bit, GNU/Linux 32-bit (Ubuntu) and GNU/Linux 64-bit (Debian). We anticipate offering builds for other platforms in time. Aperiodic links, pause & continue execution, and execution reports are not yet fully implemented.

VI. CONCLUSIONS AND FUTURE WORKS

A. Meeting The Challenges

BRAHMS solves the primary challenge of integration across Varied Development by specifying a common, flexible interface in multiple programming languages, against which new computational engines can be developed, and onto which existing computational engines can be imported. It meets the challenge of Varied Deployment through implementation as a lightweight standalone native executable and by allowing processes to be developed in similarly lightweight native code; a version of the supervisor is available without multiprocessing support for a further reduced footprint. BRAHMS offers extensive (and accreting) background functionality in the supervisor, meeting one aspect of the challenge of Code Sharing (a BRAHMS 'hello world' process written in C++ runs to about 25 lines of mostly boilerplate code, yet can distribute its complex processing across massively parallel resources).

BRAHMS supports the pragmatic challenges of sharing process code (by allowing the distribution of pre-compiled binaries rather than source code and by providing accountability) and goes some way to fostering documentation by defining a public process interface (development against a known interface self-documents to some extent, since a naïve reader knows at least some aspects of what a piece of code is intended to do). However, it does not directly address the challenge of sharing process code—see Section VI-B for details of how this will be addressed by future developments. BRAHMS employs Open Standards throughout. Adoption of the pre-release platform continues: the success of BRAHMS as the integration framework for the WhiskerBot project has led to its being chosen for three other major projects, involving varied use cases, and the early adopters have reported finding the workflow agreeable. Indications are that overall performance of systems executed using BRAHMS perform favourably in comparison with monolithic equivalent systems—quantitative metrics will follow in a later report.

New processes benefit from being built into the BRAHMS framework by taking advantage of services provided by the

system, and are constrained in their operation only by the requirements of the supervisor-process interface (i.e. a process is free to interact with the operating system, with hardware, with the user, as required). Integrating existing processes into BRAHMS can be achieved either by 'wrapping' the existing processing engine (contingent on cooperation and/or access to source code) or by meeting the communications interface of the existing software (Freebots was wrapped, for example, by authoring a BRAHMS process that communicated with it over TCP/IP).

None of the other proposals considered meet the requirements for general integration. Some discussed features are, as noted above, incomplete; however, users can begin using BRAHMS immediately and take advantage of feature additions as they become available.

B. Future Work

Above, we mentioned the SystemML file format, which is used by BRAHMS to represent systems. This open file format is a point of interface between BRAHMS and other tools. Beyond that, however, SystemML will also, in future, offer an infrastructure for the publication of processes represented in such systems. Process data in the infrastructure will include specification of parameterisation and of input/output interfaces, as well as of the algorithm itself. The infrastructure will provide archiving, distribution, version control and automatic patching (without breaking accountability) of published processes. The interplay of this infrastructure with accountability will ease the identification and removal of software bugs. This infrastructure will allow BRAHMS to meet the final challenge identified above, that of effective sharing of process code. It will also contribute to the reduction of the problem of algorithmic details becoming lost irrecoverably in undocumented optimised code.

Asked to execute a SystemML document, a naïve installation of BRAHMS will be able to obtain implementations of the specified processes through the SystemML infrastructure and execute the model without human intervention. Thus, processes published to the infrastructure will be immediately available to co-workers. Not only will co-workers be able to execute each other's models, but they will be able to use freely-available expertly-authored components in their own models.

We are committed to completing the development of Concerto, the BRAHMS Standard Library and the Python language bindings, and we expect to develop bindings for Java and Octave in future. In addition, we plan to develop interfaces for the use of BRAHMS from other interactive environments (Octave, for example). Other possibilities include a GUI system designer (operating within the SystemML space entirely, this is actually independent of BRAHMS) and the addition of a service equivalent to the IKAROS WebUI.

A future report will offer efficiency and scaling metrics, including comparison with equivalent monolithic systems.

VII. ACKNOWLEDGMENTS

The authors gratefully acknowledge the contributions of the other Adaptive Behaviour Research Group members in testing, and particularly the patience and advice of Martin Pearson as the main end user on the WhiskerBot project.

REFERENCES

[1] P. F. Dominey and M. A. Arbib, "A cortico-subcortical model for generation of spatially accurate sequential saccades", *Cereb Cortex*, 2:153-175, 1992.

[2] J. W. Brown, D. Bullock and S. Grossberg, "How laminar frontal cortex and basal ganglia circuits interact to control planned and reactive saccades", *Neural Networks*, 17:471-510, 2004.

[3] M. J. Pearson, A. G. Pipe, C. Melhuish, B. Mitchinson and T. J. Prescott, "Whiskerbot: A Robotic Active Touch System Modeled on the Rat Whisker Sensory System", *Adaptive Behavior*, 15:223-240, 2007.

[4] *ICEA*, European Union Framework 6 IST-027819, http://www.iceaproject.eu.

[5] J. M. Chambers, *Deciding where to look: A study of action selection in the oculomotor system*, PhD Thesis, The University Of Sheffield, 2007.

[6] J. G. Fleischer, J. A. Gally, G. M. Edelman and J. L. Krichmar, "Retrospective and prospective responses arising in a modeled hippocampus during maze navigation by a brain-based device", *Proc Natl Acad Sci U S A*, 104:3556-3561, 2007.

[7] B. Girard, D. Filliat, J. Meyer, A. Berthoz and A. Guillot, "Integration of Navigation and Action Selection Functionalities in a Computational Model of Cortico-Basal-Ganglia-Thalamo-Cortical Loops", *Adaptive Behaviour*, 13(2):115-130, 2005.

[8] M. Djurfeldt and A. Lansner, *Proceedings of 1st INCF Workshop on Large-scale Modeling of the Nervous System*, Stockholm, Sweden, 2006, *in* Nature Precedings, doi: 10.1038/npre.2007.262.1

[9] D. C. Dennett, "Why not the whole iguana?", *Behavioral and Brain Sciences*, 1:103–104, 1978.

[10] D. L. Parnas, "On the criteria to be used in decomposing systems into modules", *Communications of the ACM*, 15(12):1053-1058, 1972.

[11] Mathworks, *Matlab & Simulink*, http://www.mathworks.com.

[12] K. Gurney, T. J. Prescott, J. R. Wickens and P. Redgrave, "Computational models of the basal ganglia: from robots to membranes", *Trends in Neuroscience*, 27(8):453-459, 2004, doi: 10.1016/j.tins.2004.06.003

[13] B. Mitchinson and T.-S. Chan, *BRAHMS*, http://sourceforge.net/projects/brahms.

[14] M. Djurfeldt, Ö. Ekeberg and A. Lansner, "Large-scale modeling – a tool for conquering the complexity of the brain", *Frontiers in Neuroinformatics*, 2:1, 2008, doi: 10.3389/neuro.11.001.2008

[15] T. J. Prescott, F. M. Montes González, K. Gurney, M. D. Humphries and P. Redgrave, "A robot model of the basal ganglia: Behavior and intrinsic processing", *Neural Networks*, 19(1):31-61, 2006.

[16] B. Webb, *Biorobotics*, AAAI Press, 2001.

[17] R. Chavarriaga, T. Strösslin, D. Sheynikhovich and W. Gerstner, "A Computational model of parallel navigation systems in rodents", *Neuroinformatics*, 3(3):223-242, 2005.

[18] D. L. Parnas, "Software Aging", *in 16th international conference on Software engineering*, Sorrento, Italy, 2004, 279-287.

[19] B. Mitchinson and J. Chambers, *SystemML*, http://sourceforge.net/projects/systemml.

[20] J. Bloch, *How to Design a Good API and Why it Matters*, Javapolis, Antwerp, Belgium, December 12–16, 2005.

[21] Le Hors A. *et al.* (ed.), *W3C Document Object Model Core*, http://www.w3.org/TR/DOM-Level-3-Core/core.html.

[22] *REVERB*, EPSRC Research Grant EP/C516303/1, http://reverb.abrg.group.shef.ac.uk.

[23] ICEA Deliverable D27, http://www.iceaproject.eu.

[24] *BIOTACT*, European Union Framework 7 ICT-215910, http://www.biotact.org.

[25] A. Weitzenfeld, M. A. Arbib and A. Alexander, *The Neural Simulation Language: A System for Brain Modeling*, MIT Press, 2002.

[26] F. Howell, R. Cannon, N. Goddard, H. Bringmann, P. Rogister and H. Cornelis, "Linking computational neuroscience simulation tools : a pragmatic approach to component-based development", *Neurocomputing*, 52-54:289-294, 2003.

[27] C. Balkenius *et al.*, *IKAROS*, http://www.ikaros-project.org.

[28] Ö. Ekeberg, M. Djurfeldt, *MUSIC: Multi-Simulation Coordinator, Request For Comments*, http://www.incf.org, 2008.