Res. Lett. Inf. Math. Sci., 2007, Vol. **11**, pp. 61–70
Available online at http://iims.massey.ac.nz/research/letters/

Performance Evaluation of a Distributed Integrative Architecture for Robotics

Guy K. Kloss
Computer Science
Institute of Information & Mathematical Sciences
Massey University at Albany, Auckland, New Zealand
Email: `G.Kloss@massey.ac.nz`

The field of robotics employs a vast amount of coupled sub-systems. These need to interact cooperatively and concurrently in order to yield the desired results. Some hybrid algorithms also require intensive cooperative interactions internally. The architecture proposed lends itself amenable to problem domains that require rigorous calculations that are usually impeded by the capacity of a single machine, and incompatibility issues between software computing elements. Implementations are abstracted away from the physical hardware for ease of development and competition in simulation leagues. Monolithic developments are complex, and the desire for decoupled architectures arises. Decoupling also lowers the threshold for using distributed and parallel resources. The ability to re-use and re-combine components on demand, therefore is essential, while maintaining the necessary degree of interaction. For this reason we propose to build software components on top of a Service Oriented Architecture (SOA) using Web Services. An additional benefit is platform independence regarding both the operating system and the implementation language. The robot soccer platform as well as the associated simulation leagues are the target domain for the development. Furthermore are machine vision and remote process control related portions of the architecture currently in development and testing for industrial environments. We provide numerical data based on the Python frameworks ZSI and SOAPpy undermining the suitability of this approach for the field of robotics. Response times of significantly less than 50 ms even for fully interpreted, dynamic languages provides hard information showing the feasibility of Web Services based SOAs even in time critical robotic applications.

Keywords: Robotics, Simulation, Machine Vision, Artificial Intelligence, Distributed Computing, Service Oriented Architecture

1 Introduction

Robotic Systems are often used in place of humans or biological organisms. Their design specification requires taking many interacting sub-systems into account. That usually includes: vision and other sensory systems; motor control and other actuators; action and strategy planning; adaptability; and possibly others. Each of these systems may be by itself composed of several sub-systems employing a network of algorithms.

A typical setup for such a scenario is a robot soccer league's system (FIRA MiroSot [1]) with a central vision system (Fig. 1). It is convenient that a single computer may be used for all purposes. This machine then provides all control functionality of a single team, but it may suffer from bottle necks in computational resources.

As an example the simple case in the above mentioned setup: A robot wants to move from location A to B.

1. The **vision system** detects the surrounding environment, by identifying object types and their location through the position in the image.

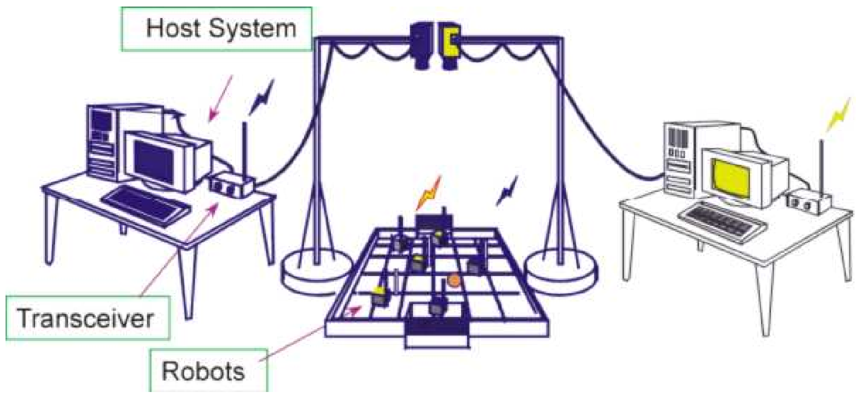

Figure 1: Physical setup in a MiroSot competition.

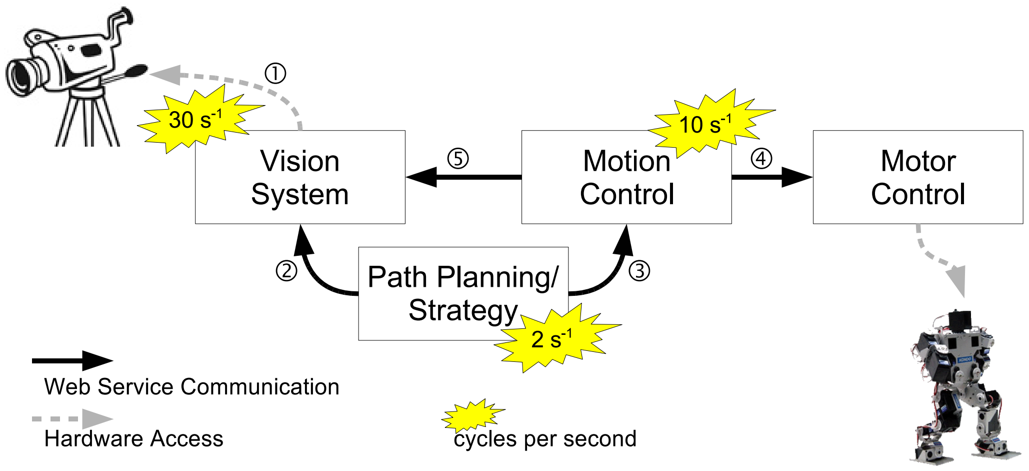

Figure 2: Distribution of workflow components on the network.

2. The **path planning** algorithm requires the robot's own position, the positions of the target location B and the positions of obstacles.

3. The **motion control** system requires the robot's own position/orientation and the target's position/orientation for the *next* way point. With the help of some physics equations, the designated individual motor speeds and/or positions for its actuators can easily be calculated.

4. The **motor control** (for the physical motor) then requires the motion control's designated actuator states to map them to the devices using the hardware control interfaces. A control loop with additional sensoric input (from the vision system) is required. Individual motor compensation movements for slippage and an adaptive learning system for compensation of constructional differences are used.

In this example, the various sub-systems ideally operate all at their individual rate (e. g. vision system with the frame rate of the camera) and communicate through interfaces on demand with other sub-systems (see Fig. 2). Thus, a complex system of concurrently acting and reacting components (employing artificial intelligence) composes a working artificial "organism" meeting it's task.

This paper outlines the design decisions currently in the implementation phase for a new robotics system, based on a Service Oriented Architecture (SOA) using Web Services. It pro-

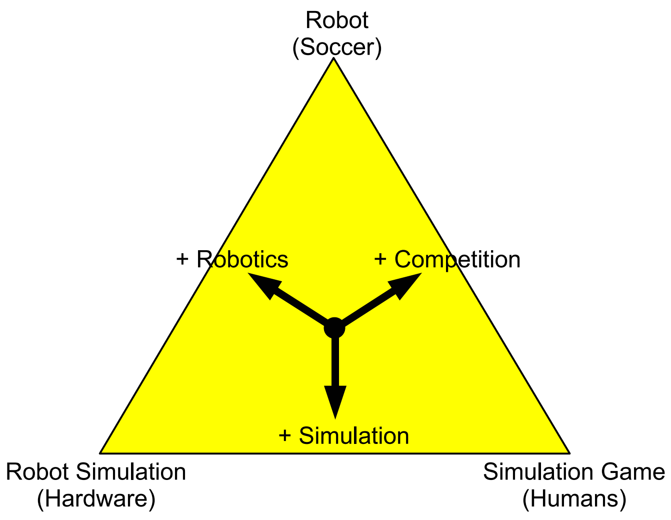

Figure 3: Difficulty: Coping with multiple – but related – problem domains.

vides implementation data and details of the previous design stages outlined in [2]. This paper covers the proposed architecture, implementation, and finally information on the performance in a real world application.

2 System Architecture

If dealing with one specific problem domain only (e. g. Robot Soccer in the MiroSot small league), the software can be built on top of the (robot) vendor supplied software tool kit only. However, ongoing alterations by multiple independent developers in this code base significantly decrease the comprehensibility. Finally an implementation for multiple domains (e. g. for MiroSot and humanoid league) will completely fail on the provided vendor's development kit.

Implementing competing designs for the same application will usually be devastating to the implementation. An implementation for multiple domains (e. g. for MiroSot and humanoid league) will completely fail on the provided vendor's development kit.

A goal was to cope with complexities introduced by the design for implementing (too many) specializations. It requires addressing multiple domains and distribution/parallelization of tasks, while still gaining independence of programming languages and the executing operating system.

Fig. 3 shows the related problem domains to be addressed at the corners of the triangle. The arrows indicate an increasing character for the domain's opposing directions. It is visible that the dynamics and modelled properties for a "real" soccer game are identified to be quite different from a simulation of robots' hardware. This needs to be accounted for in the design of the system's common architecture. The goal is to be able to abstract away from individual, domain specific implementations. The architecture and the developed components should be able to be assembled and configured depending on the problems faced.

By compartmentalising the systems, the ties between functional sections were loosened. Communication between the components is *only* possible through the defined interfaces to them. The interfaces are exposed in a Service Oriented Architecture (SOA) as Web Services to the rest of the system, so that individual components can be placed on remote systems as well as locally. A Web Service is a technology that allows applications to communicate with each other in a platform and programming language independent manner. Therefore, all components can be implemented using an implementation and platform fittest for the task. [3] gives an in depth overview of the benefits of using Web Services, and SOA in general, for robotics. This results in a building block approach, to construct larger applications from available re-usable components in the "tool box." SOAs in general can also be found in other fields of scientific computing. A distributed component approach – in some ways similar to the one proposed in this paper (but using CORBA for a SOA) – is for many years successfully used for a distributed integration environment in simulation and optimisation computations at the German Aerospace Centre [4].

This approach contrasts the tightly coupled, monolithic architecture provided by the hardware vendors. In the vendor's system all steps of the process chain are forced to run at the same beat. Next to the vision system especially the strategy and path planning components require larger amounts of computational power. Cycling through all additional components unnecessarily, thus, will waste valuable resources that could be very well used otherwise. Thus, in a decoupled system the number of executions for some sub-systems can be cut by a factor of 15.

3 Mapping to Robot Soccer Domain

The implementation needs to provide mainly two things:

- Utilise the *domain independence* of created components (robot soccer, soccer simulation, robot simulation, robot rescue and others).
- Focus the development on *one specific component,* disregarding all other neighbouring components (or mocking them by dummies).

To highlight this, Sect. 1 provides the necessary steps for an analysis. The vision system (step 1) requires computationally expensive operations at a high rate (frame rate of camera). So it will be implemented most likely in a native/compiled language (e. g. C++) on a system that is capable of interfacing the camera. Further path planning and motion/motor control can be performed on another system. These could be implemented in languages that are by far more efficient in terms of development effort (e. g. Python, Java, Visual Basic).

In Fig. 4 the communication paths between the components in our scenario are outlined. Most sub-systems are interfacing a communication layer, managing requests transparently through Web Services [5, 6]. In case of changing demands this single layer *only* needs to be modified for an alternative coupling. Only the vision system and the motor control feature a direct access to hardware. Properly designed interfaces provided, these can easily be replaced by the virtual hardware of simulation systems, or replaced by alternative specific hardware drivers without introducing further dependencies.

The uni-directional arrows indicate requests to other sub-systems. These are interfacing a communication layer (double arrows), and the requests will be transparently managed through Web Services. In case of changing demands, just this single layer needs to be modified to implement alternative links (e. g. through CORBA, direct sockets, high speed UDP connections, local method invocations, etc.). These then could be used either for dedicated links between individual components or for the whole system.

The motor control component is connected directly to the motion control. This reflects a common setup where the motor control is just an instance of a hardware controller used directly through its specified interfaces.

The path planning algorithm would request a list of vectors (positional and velocity) for all objects from the vision system. Even though the strategy and path planning may need only to be updated twice every second, it can still take advantage of the precise computations of the vision system performing synchronously with the 30 frames per second of the video capturing. This ensures accurate and current information at the time of decision making.

New/updated paths result in an update of way points for the robots. These vectors are requested on demand by the motion control, which, in turn, relays actuator settings to the motor

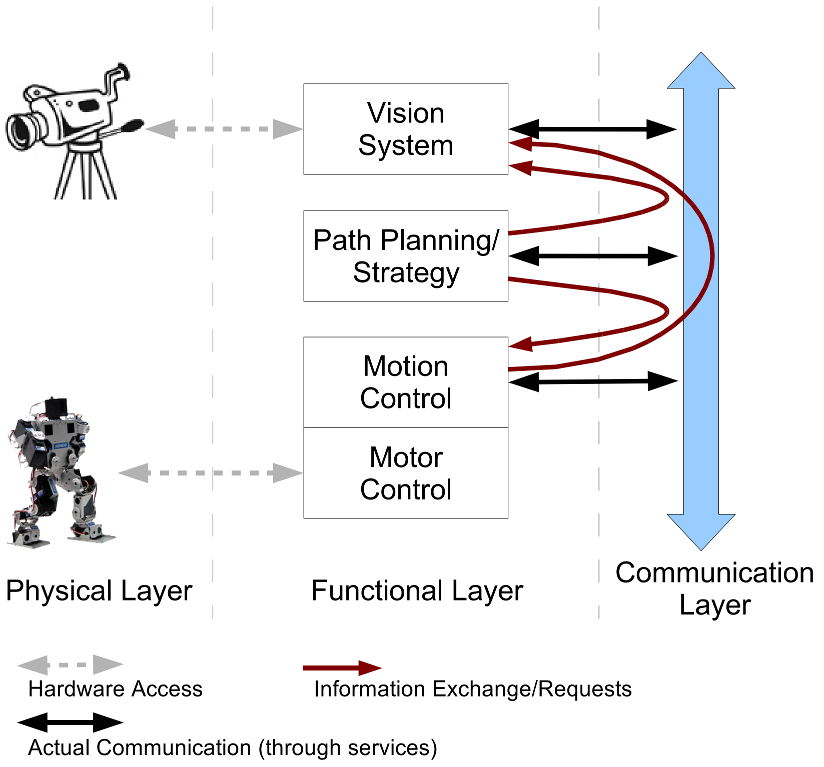

Figure 4: Communication between various components.

control [7]. The motor control (or locomotion layer [8]) represents a robot's embodiment. It converts control signals from the motion control (steering layer) into motion of the robot's "body." This motion is subject to constraints imposed by the body's physically-based model, such as the interaction of momentum and strength (limitation of forces that can be applied by the body). For flexible change of implementations accessing alternative hardware or sub systems, this component is separated from the motion control component. As this layer is dealing with the control of physical equipment it may take advantage of an increased update rate (10 Hz) due to the decoupled system.

Unfortunately, at this point reality interferes with direct steering mechanisms. Due to heterogeneous motor torques, differences in construction, wheel slippage on the surface, maximum allowed accelerations, etc. the motor control will yield individually different behaviours. For compensation a closed control loop giving accurate feedback on robot positions is needed. This is most easily accomplished by also accessing the vision system's service.

The motor control should "learn" the physical differences of the motors in the robots through adaptation. Thus the control of the motions of the robots will become more predictive, and the rate for requesting actual positions can be handled dynamically on demand to reduce computations and communication overheads.

As for the design of the implementation – and the abstraction for most flexible hardware, control and simulation – the architecture of these implementation have been inspirational: RoboCup Soccer Simulator [9], RoadNarrows Robotics' [10] robotics demonstration application "Fusion", the Python Robotics package [11] and the Java "TeamBots" package [12].

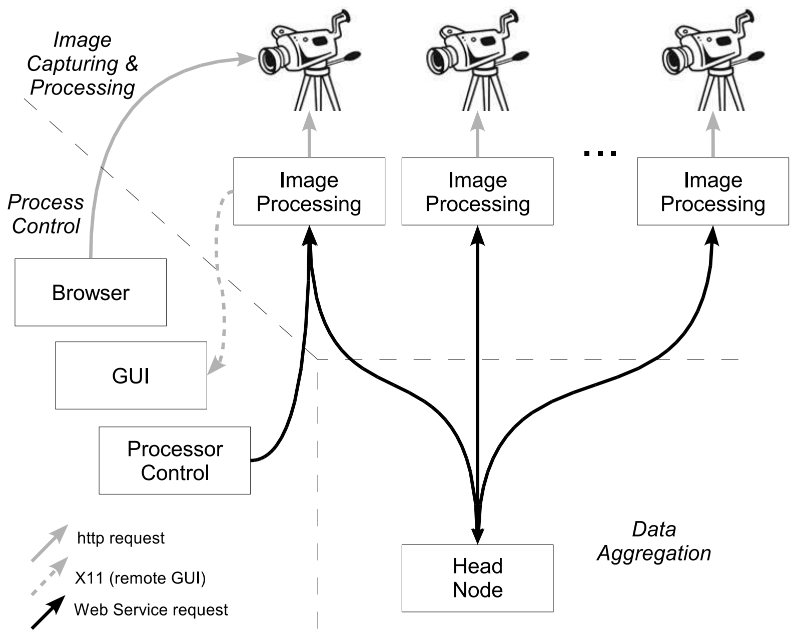

Figure 5: Industrial machine vision application of distributed image capturing/processing nodes with central control and data aggregation facilities.

4 Experiments and Analysis

4.1 Industrial Machine Vision System

The discussed architecture has been initially implemented for an industrial application employing machine vision. In this scenario several processing nodes are operating on images from a single camera each (see Fig. 5). The images are retrieved through a streaming HTTP protocol from the cameras. When certain criteria of the analysis data are met, a record of the analysis data is sent off through a Web Service call to a central head node for collection and further processing. The activity of all processing nodes can be monitored through remote GUI display and web access to the individual cameras. A Web Services server implementation within the processing nodes themselves facilitates control functions (start, stop, change of configuration and process termination). Monitoring and process control do not need to run at all times, and just need to be started on demand.

Image retrieval and processing are implemented natively in compiled code (C++) as a shared library. Threaded retrieval and processing of each individual frame per camera through the whole cascade of analysis stages takes an average of 70 ms (6 ms standard deviation). The process logic is harnessed in to the communication layer by Python bindings using ZSI [13] to implement the Web Services fabric. This combination yielded very good performance on the computationally intensive image processing, while still enabling a rapid pace and ease of development for the management layers.

The Web Service setup follows common (best) practises, by first defining Web Service Definition Language (WSDL) interfaces in XML. These are used to generate stub code for client and service implementations, which are used by the communicating instances. All complex data types have been defined verbosely with hierarchical structures using the default data types as leaf elements. We can therefore assume this implementation in an interpreted scripting language to reflect a "worst case scenario" in terms of communication performance. However, the potential for optimisation and "stream lining" of communication is quite large, many hints on this can be found in [14].

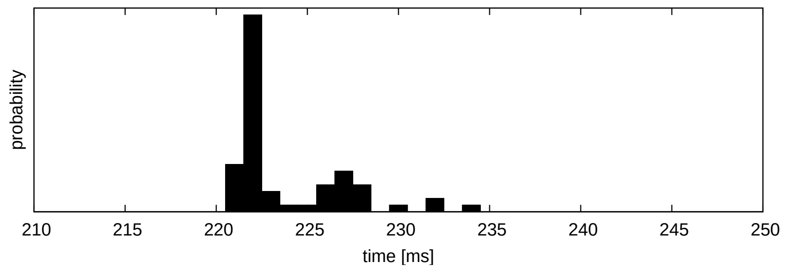

Figure 6: Typical call time distribution of the remote Web Service `sendKeyFrame()` operation.

Table 1: Median request times (and inter quartile ranges parenthesised) for Web Services operations in milliseconds.

	`isRunning()`				`sendKeyFrame()`			
	local		remote		local		remote	
idle	11	(2)	35	(2)	115	(4)	206	(4)
processing	25	(23)	60	(60)	128	(6)	224	(5)
minimum	9		31		113		203	

Due to task and thread switching, and therefore interfering other processes, the distribution of call times to complete the Web Service operations has been "smeared" (see Fig. 6). Therefore, it is relatively pointless to determine average values. The suitable value for this analysis has been the *median time,* and the width of the distribution is described by the *inter quartile range* (IQR). As it can also be seen in the figure, the distribution starts out with a very sharp rise, and a distinct minimum time for the completion of the request can be identified. This time can be safely assumed as describing the true overhead (or "cost") of the operation, as the operating system and process timing effects are accountable for the smearing towards longer times.

For testing of the implemented infrastructure, the application in question has been benchmarked while idle and under load (actively processing images) on the one hand, and locally as well as in a networked environment on the other hand. The system features an Intel Pentium D CPU, 3.20 GHz, 2 GB RAM, connected with 100 MBit/s on the same network switch. Two Web Service operations were evaluated: The first one (`isRunning()`) represents a very cheap operation both on the server and on the client. It directly returns a Boolean value stating whether the image processing is active. Therefore no complex XML encoding into the SOAP message (Simple Object Access Protocol) is necessary, and no complex dynamic structures for the results need to be built on the receiving client. The other operation (`sendKeyFrame()`) transfers a complex nested structure containing multi-dimensional arrays and various values in the SOAP message, which need to be mapped dynamically to a Python object on the receiving side (in this case the server). The data transferred by this structure is about 2 KB in size.

From the results (Table 1) we can see, that our system typically takes about 11 ms locally, and 35 ms for a remote request (`isRunning()`). The variation in this time is with 2 ms for the IQR also very low and close to the minimum times. The system is very robust in this condition. Under load – while processing images in another thread – the likelihood of being interrupted by the processing during execution rises with the time spent at it, increasing the times to 25 ms and 60 ms respectively for the remote calls. With the raw execution typically lasting 35 ms on the remote calls, an interruption of the processing lasting 70 ms occurs in an average roughly every second time, increasing the requests duration as well as the IQR well in line by the expected times.

The request times for the `sendKeyFrame()` operation indicate that sending extensive and/or complicated data structures using Web Services can be quite expensive. Striking is, that although the median times for the completed request are largely greater (compared with the `isRunning()`

Table 2: Request times for alternative Web Services communications.

	local	remote
slim	< 3.5 ms	< 10 ms
full	< 30 ms	< 35 ms

call), the IQR increases only insignificantly, and the changes between idle and load calls are only marginal. The reason for this can easily be found taking a look into the application: The request to this call is performed right after the processing of a frame within the same thread. Therefore it cannot be interrupted to increase the communication time.

Using a different type of data encoding and a more flat structure can largely improve the efficiency. As this particular application is only dependent on reasonable quick transfers (< 1 s) no need for action is given. However, it is quite valuable to gain insight into the dynamics of data structures and transfer mechanisms to be aware of potential bottle necks and strategies for resolving them. Alternative communication strategies using Web Services in Python have been tested to estimate their effects on timing. Only briefly an operation of the type `isRunning()` has been tested. Next to ZSI also the Web Service implementation SOAPpy has been included in the test. For brevity reasons of the pure estimation of possibilities only the ranges are stated in Table 2. The slimmest communication within a single host using no WSDL interface descriptions has been tested against a full-fledged WSDL request with dynamic mapping of objects (no prior stub-generation) using introspection. In one case the request's client and server were co-located on one host, in the other they were again located again on the same switched 100 MBit/s network segment.

4.2 Applied to Distributed AI for Robotics

Performance may be particularly crucial to the success of a robotics application, as many sub-systems employing sophisticated Artificial Intelligence (AI) are chained together. This is true, but only up to the point of where it needs to be "as performant as necessary." The proposed architecture demands its tributes for communication overhead (Web Services rather than a fast, custom UDP protocol or even directly linked code), using interpreted languages rather than naively compiled ones, etc.

The main advantage, however, is that it frees one from worries about other tasks, not directly related to the currently addressed problem introduced by the framework's code used for the implementation. Thus, a loosely coupled (and less static) application design will lead to more performance in terms of (the quality of) research output. Coding against a clean interface design has already given our research students the freedom to focus better on their specific topics. Using SOA interfaces takes this programming paradigm "to the next level" by possible implementations in almost arbitrary languages and enabling execution on distributed hosts. Additionally, services in the decoupled SOA can be requested on demand, rather than at the time of availability. This saves computational resources and enables one to deploy the components in distributed systems to make use of resources most suitable, yielding a better efficiency.

In many publications on robotics a time frame of 33 ms for the maximum iteration duration of the control system are given. This value has been derived from the frame rate (30 fps $\rightarrow$ 33 ms) of cameras forcing the computationally dominant vision system as well as the rest of the control chain. As a result of decoupling this force is removed from most sub-systems, giving the freedom to spend clock cycles on the architecture and more suitable implementation languages. This application continues with an implementation on top of the design considerations from [15].

Estimating from the highest determined request frequency (10/s) from Sect. 1, it can be seen that the communication overhead can be low enough to perform sufficiently. Given an implementing featuring slim and efficient interfaces, and reducing unnecessary calls in a loosely coupled system, this should be possible. Employing an architecture like this strictly leads to a dramatic reduction of code complexity, as in most developments one only needs to care about the bounding interfaces of the component, not the code beyond them. Furthermore message based architectures greatly ease the use of concurrency by eliminating many of the problems involved in multi-threaded programming.

5 Conclusions

Investment in re-architecturing the system yields an easier and more focused implementation of sub-tasks. Freedom of choice for the implementation language away from C++ *only*, and freedom not to have to worry about "neighbouring" functional components to a problem and dramatically increase the speed of development. Due to a better focus on the current core problem, more innovative and robust solutions were gained.

Someone e. g. interested in improving the path planning, therefore, would be relieved of the impact on the rest of the infrastructure. Development can be performed in any programming language that is suitable (or familiar). Additionally, a widely used and standardised protocol for a SOA has been used to further reduce system induced barriers for communication. And finally, the host and operating system used for a specific sub-task is independent from the rest of the system.

An industrial reference implementation with distinct similarities to the field of robotics has been implemented and analysed toward suitability. Through decoupling processed, it is now possible to concurrently conduct computationally expensive tasks – as sophisticated vision systems – in parallel, utilising resources easily on remote systems.

Computational efficiency of the system could be gained. This was achieved by decoupling components for distribution on hosts most suitable for them. All components work in parallel at the most appropriate rate without being forced into the "dominant" beat (e. g. the camera's frame rate), to save valuable CPU cycles and utilise parallel environments (multiple CPUs/cores or machines) most efficiently without sacrificing accuracy.

Our developments give a clear indication of feasibility for time critical robotic applications. The response time studies based on this show – in contrast to the referenced sources – that even with the case of a purely interpreted, dynamic language Python time constraints can be met. In the future this system will be employed in the field of robot soccer to loosen the ties to frame rate based processing, enable more sophisticated algorithms, and increase concurrency in the system. Loosely coupled, simple software components will ease development, and further enhance research ambitions in sub-tasks within the field of robotics.

Acknowledgements

We would like to thank the companies Baggage Sortation Management Ltd. and ecomgroup Ltd. for their support and cooperation. They gave the opportunity to develop and test the algorithms and software for live systems, and to validate its practical robustness through a real world validation.

References

[1] FIRA, "Web Site." [Online] http://www.fira.net/

[2] G. K. Kloss, N. H. Reyes, and K. A. Hawick, "Integrative Architecture for Concurrent Artificial Intelligence in Robotic Systems," in *Proceedings of the 5th New Zealand Computer Science Research Student Conference (NZ CSRSC 2007)*, Hamilton, New Zealand, April 2007.

[3] B. K. Kim, M. Miyazaki, K. Ohba, S. Hirai, and K. Tanie, "Web Services Based Robot Control Platform for Ubiquitous Functions," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2005)*, Barcelona, Spain, April 2005, pp. 691–696.

[4] T. Forkert, G. Kloss, C. Krause, and A. Schreiber, "Techniques for Wrapping Scientific Applications to CORBA Components," in *Proceeding of the 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS).* Santa Fe, NM: IEEE, April 2004, pp. 100–108.

[5] J. L. Du, U. Witkowski, and U. Rckert, "Teleoperation of a Mobile Autonomous Robot using Web Services," in *Proceedings of the 3rd International Symposium on Autonomous Minirobots for Research and Edutainment (AMiRE 2005)*, Fukui, Japan, September 2005, pp. 55–60.

[6] J. M. Vidal, P. Buhler, and H. Goradia, "The Past and Future of Multiagent Systems," in *Proceedings of AAMAS Workshop on Teaching Multi-Agent Systems*, New York, NY, July 2004.

[7] C. L. Hwang, N. W. Chang, and S. Y. Han, "A Fuzzy Decentralized Sliding-Mode Control for Car-Like Mobile Robots in a Distributed Sensor-Network Space," in *Presentation at the Third International Conference on Computational Intelligence, Robotics and Autonomous Systems.* Singapore: Elsevier, December 2005.

[8] C. W. Reynolds, "Steering Behaviors for Autonomous Characters," in *Proceedings of Game Developers Conference*, San Jose, CA, March 1999.

[9] RoboCup Federation, "RoboCup Soccer Simulator Project." [Online] http://sserver.sourceforge.net/

[10] RoadNarrows LLC, "Web Site." [Online] http://www.roadnarrowsrobotics.com/

[11] D. S. Blank, D. Kumar, L. Meeden, and H. Yanco, "The Pyro Toolkit for AI and Robotics," *AI Magazine*, vol. 27, pp. 39–50, March 2006.

[12] T. Balch, "TeamBots Project." [Online] http://www.cs.cmu.edu/~trb/TeamBots/

[13] Python Web Services Project, "Web Site," last accessed October. [Online] http://pywebsvcs.sourceforge.net/

[14] R. A. van Engelen, "Pushing the SOAP Envelope with Web Services for Scientific Computing," in *Proceedings of the International Conference on Web Services (ICWS)*, Las Vegas, NV, June 2003, pp. 346–354.

[15] G. K. Kloss and N. H. Reyes, "Unified Architecture for Concurrent Artificial Intelligence in Robotic Systems," in *Proceedings of the 6th International Conference on Hybrid Intelligent Systems (HIS 2006) and 4th Conference on Neuro-Computing and Evolving Intelligence (NCEI 2006).* Auckland, New Zealand: IEEE Computer Society, December 2006.