# Representing parallelism in a control language designed for young children

Book Section

How to cite:

Whalley, Peter (2006). Representing parallelism in a control language designed for young children. In: Grundy, John and Howse, John eds. IEEE Symposium on Visual Languages and Human-Centric Computing. California: IEEE Computer Society, pp. 173–176.

Link(s) to article on publisher's website:
http://dx.doi.org/doi:10.1109/VLHCC.2006.41

# Representing Parallelism in a Control Language Designed for Young Children

Peter Whalley
*KMi, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK*
p.c.whalley@open.ac.uk

## Abstract

*Actor-lab was intended to make control problems comprehensible to young children experiencing programming for the first time, and to provide an interface around which they could have learning conversations. The design goal was to create an expressive high-level control language that could incorporate the WHEN DEMON metaphor within the intrinsically parallel actor programming paradigm. Information about the static relationship between the objects in the system, the external dynamic events and the internal message passing is provided by the visualisation. The learner-centered evolution of actor-lab is detailed in terms of how successfully it both reflects the curriculum model of control and also engenders a sense of agency within the system.*

## 1. Introduction

Young children often find it difficult to understand the relationship between the symbolic control languages and the external micro-worlds of the control technology topic, and consequently fail to develop appropriate mental models of where the 'control' actually resides in the system. The *actor-lab* interface, shown in Figure 1, was designed to resolve this problem by providing a transparent representation of the implicit *input-process-output* (*i-p-o*) model of control. Students working with *actor-lab* are also presented with the idea of viewing the actions of their program in terms of a pattern of messages between the *actors*. The dynamic visualisation of message-flow, together with the animation of the effect of meta-commands on object states, were intended to engender a sense of agency within the *actor-lab* system and support the underlying message-passing metaphor of an event-driven control language. For the control topic to work successfully in the classroom students have to be able to understand and comment on each other's projects. Consequently the design objective of the interface was that it should be able to function as a dynamic representation of control programs that young children could easily understand and have conversations around. The learner-centered evolution of *actor-lab* is detailed in terms of how successfully it both reflects the curriculum model of control and also engenders a sense of agency within the system.

## 2. Actor-lab

Parallelism, or 'multi-tasking', was identified by Papert [1] as one of the most important aspects of control programming as he made the transition from the first screen based simulations of 'turtles' to real-world devices that could move about and explore their environment. He noticed that the tasks undertaken by the children were having to be greatly simplified because realistically complex problems could not be represented in a natural way by the procedural control languages. These unfortunately still dominate the classroom. In the real world, events can happen simultaneously and this creates difficulties for these languages. In the context of the control topic a simple example of parallelism would be a program that caused a light to turn on and off repeatedly, whilst at the same time being able to respond to buttons being pressed to start and stop a motor. Complex forms of parallelism reflect higher level, *goal-orientated* behaviours, eg. for a buggy to be able to follow a line, but turn round and return if an obstacle is met.

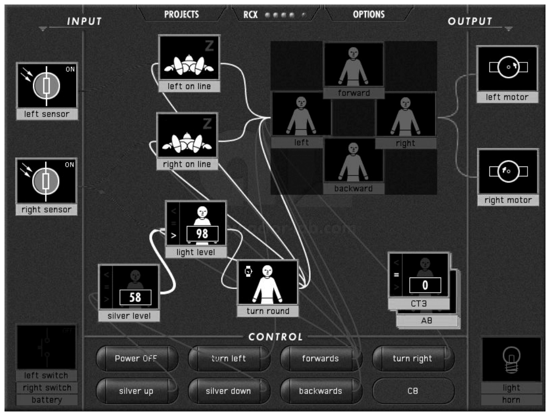


**Figure 1.** The actor-lab interface

Papert's solution to the problem of parallelism in control was to attempt to incorporate the WHEN DEMON metaphor from early models of parallel computing into LOGO. A version of this metaphor also forms the basis of the *actor-lab* control language, modified to fit the terminology of the event-driven *Actor* programming paradigm [2]. The children's role as programmer is explained to them as writing scripts for the actors, who will then carry out their play/plan for them. The actors are prompted to begin carrying out their scripts, which are made up of messages, by either a message created by an external *input* event or a message from another *process* object; an actor or counting actor. The actors in turn may send messages to *output* objects or to other actors. The relationship of the different object types is shown in Figure 2, and a simplified process model of an actor in Figure 3. A copy of *actor-lab* together with a detailed description of the language, colour images and video clips of the system in operation is available for downloading [3].

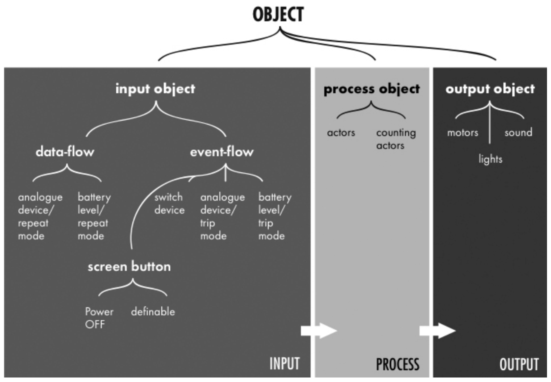


**Figure 2.** The object types in actor-lab

Adopting the *Actor* paradigm of asynchronous, and intrinsically parallel, message passing avoids the confusion found by Resnick [4] with *MultiLogo* as to which actions are carried out concurrently and which in sequence. It is made clear through examples that all messages are sent off at the same time, unless they are intentionally delayed by the program author. This implementation of the serialisation process appears counterintuitive to those with prior experience of procedural languages but Resnick found that it is the way that children expected objects to be able to act, particularly objects introduced within an anthropological metaphor. An important change to the visual representation of the object scripts was made mid-way through the developmental testing, a shaded banding effect was added to delineate groups of messages that are due to be sent off at the same time. This effect can be seen in Figure 4 but is even more apparent on a colour display. It was found to have a significant effect in reducing the number of errors made by children in misperceiving the delays given to messages as being cumulative rather than relative.

An essential aspect of the *actor-lab* visualisation is its *liveness* [5], the user interface is always active and reacting to both the micro-world and the user's interactions. Opening an object to inspect or change its script sets all outputs to an off state, but the underlying monitoring processes remain active. Input and output objects can still be tested, and most importantly the trigger levels for the analogue inputs can be set whilst viewing the same state animations as when the script editor is closed. As in *MultiLogo* [4], the actors are able to send and receive prioritising meta-commands which in the case of *actor-lab* direct them to either 'wake' and attend to messages, to 'sleep' and ignore messages, or to 'forget' their scripts. Computational objects in OOP languages can be considered metaphorically as having both physical and social aspects. Their physical properties can be used as concrete metaphors to represent abstract notions like 'state' and 'value', whilst their social properties can be related to ideas of inter-object communication. Following Travers [6], the general dynamic visualisation of *actor-lab* was intended to give this sense of agency to the whole system by the graphic signaling of any changes to the programmable internal states and by the consequent effect upon the flow of messages between actors.

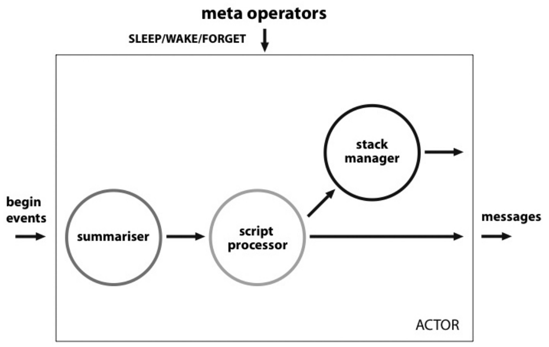
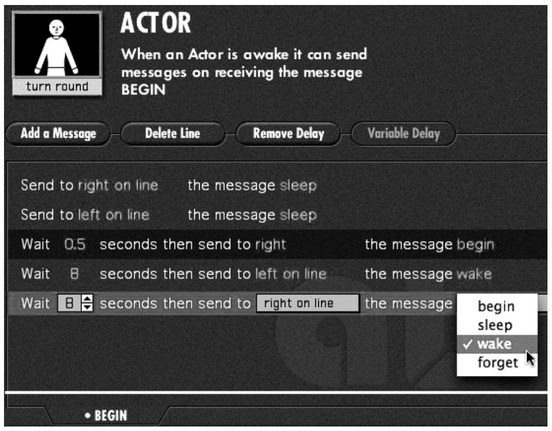


**Figure 3.** Functional model of an actor



ACTOR

When an Actor is awake it can send messages on receiving the message BEGIN

```
Send to right on line    the message sleep
Send to left on line     the message sleep
Wait 0.5 seconds then send to right          the message begin
Wait 8   seconds then send to left on line   the message wake
Wait 8   seconds then send to right on line  the message
```

**Figure 4.** Actor-lab scripting syntax

## 3. Representing the control process

Resnick [4] and Mioduser [7] reported that young children have difficulty relating the controlled objects within the micro-world, the sensors and the motors, to their representations within the command level of the procedural control languages. Resnick refers to this systematic pattern of misunderstandings as 'object-oriented bugs' and Mioduser as 'misallocation of control function'. It is suggested that these difficulties are a consequence of the procedural languages not providing a clear representation of the micro-world in relation to the other parts of the control language. Green & Petre [8] use the term 'the directness' of a programming language to describe the closeness of the mapping between the key aspects of the problem and corresponding operations within the program, and consequently one of the design goals of *actor-lab* was to provide a transparent representation of the general *i-p-o* model of the control technology topic. The graphic interface of *actor-lab* was designed to provide a static representation of the relationship between the objects, and also to display dynamically the pattern of events that occur as a program operates. Static relationships are displayed as colour coded message paths between the objects and also in the way that the movement of the different types of objects is constrained within defined spaces, as can be seen in Figure 1.

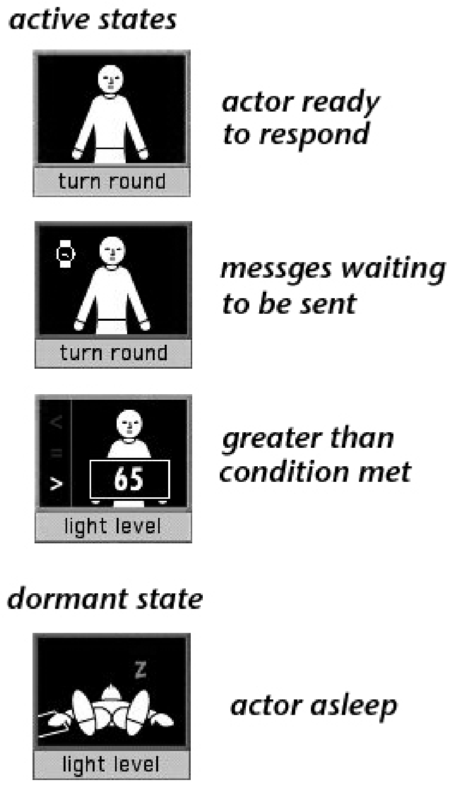


**Figure 5.** State animation of process objects

The clear delineation of input and output objects is designed to mirror the equivalent distinction made between input and output connectors that is found on most of the control units used to link computers to micro-worlds. These are usually also arranged to emphasise the underlying *i-p-o* model of control. As each element in the controlled micro-world is added they are independently named and tested by the children. The objects then appear in the drop-down menus of the script editor which effectively means that they can now send and receive messages. This process of object naming and testing is given considerable emphasis by teachers in their introductory sessions, and has become an important step in overcoming the conceptual problems found by Resnick and Mioduser. Iconic animation of the input and output objects, and also the actors, gives an indication of their current state in running programs; as can be seen in Figure 5. The underlying *i-p-o* model is dynamically reflected in the left-right flow of messages across the interface, as input events initiate a pattern of actor messages, eventually followed by output events. It is easy for teachers to emphasise this point in their introductory sessions, and the children can be seen to make reference to it when helping each other.

## 4. The message-passing metaphor

The debugging of control languages can be difficult, particularly with those systems were no indication of process state is provided once the program has been downloaded to the control unit. Brusilovsky [9] highlights the significance for the understanding and debugging of a program of being able to relate the internal operations of a program to external processes. Where "— their basic functions are carried out behind an opaque barrier — the student develops an input-output orientated understanding." The dynamic visualisation provided by *actor-lab* was intended to emphasise the process aspect of control and make the programs come to life for the children. As well as representing the communication between objects, it provides a continuous presentation of the states of objects both in the external micro-worlds and also within the control program. Figures 6 shows the progressive developments made in the representation of the message-passing metaphor over four years of developmental testing. Early attempts to provide a visualisation of the pattern of messages were found to be too complex and confusing for young children and consequently the first trial versions of *actor-lab* relied solely on icon-animation to indicate that messages were being sent, as shown in Figure 6. Debugging control tasks requires the user to simultaneously attend to both the external micro-world and the screen interface, and children acting in a mentoring role often found that rapid sequential events were difficult to point out. Animation of the message path by a brief cascade of coloured arrow heads was added, but this still required students to open up object scripts and refer to individual messages before pointing out the subsequent transient patterns as they took place.

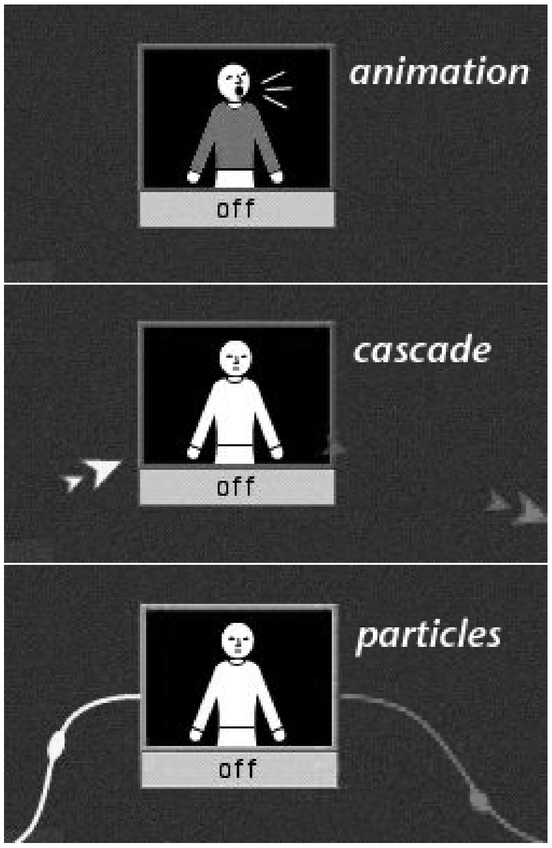


**Figure 6.** Representations of message flow

The final evolution of the visualisation, a 'particle' flowing along coloured message paths, was made possible by allowing the flexible grouping of objects. This substantially reduces the total display complexity when a large number of objects are being used, as can be seen in Figure 1, and yet individual objects can still be 'pulled out' of a group for testing if necessary. With the new display configuration, mentors could now be seen to point at particular message paths as part of their explanation before triggering events in the external micro-world. It was found that the timing of the particle transition was quite critical, too fast and they are missed by the children, too slow and they become confused with messages being sent after a delay. Given the fairly slow moving models used in control topics a time of 0.9 sec was eventually adopted, just slower than the shortest commonly used program delay of one second but long enough for the particles to be noticed. To support the particle effect a further modification was made in the last stages of testing, the dimming of message paths from objects that had not been accessed for some time. This serves to emphasise the most recent changes in the debugging cycle.

## 5. The interface as conversation piece

The role of visual representations in parallel processing languages is usually to simplify the programming task for sophisticated users by reducing the complexity of the display of the many processing events taking place. *Actor-lab* was developed for the quite different purpose of supporting the collaborative problem solving processes of naive users in a situation where almost the full complexity of their task can be presented. Trial studies [10] using the mental models approach of Mioduser [7] suggest that both the static and the dynamic visual representations of the language were successful, in that the concepts that they were designed to present were now better understood by the children. Further trials have also been made employing a protocol analysis of the conversations of Year 5 students using *actor-lab* in peer-learning situations. These showed that the main goal of the project had been achieved in that the visual representations developed could successfully support, and even become the focus of, well structured peer-mentoring dialogues.

## 6. References

1. S. Papert (1980) *Mindstorms: Children, computers and powerful ideas.* New York, Basic Books.
2. C. Hewit (1977) Viewing control structures as patterns of passing messages. *Journal of Artificial Intelligence*. **8**(3) 323-364.
3. P. Whalley (2004) *Actor-lab* . Web site, URL: http://actor-lab.open.ac.uk
4. M. Resnick (1990) MultiLogo: A study of children and concurrent programming. *Interactive Learning Environments* **1**(3) 153-170.
5. S.L.Tanimoto (1990) VIVA: A visual language for image processing *Journal of Visual Languages and Computing* **1**, 127-139.
6. M. D. Travers (1996) *Programming with Agents: New metaphors for thinking about computation*. Unpublished PhD Thesis, MIT.
7. D. Mioduser, R. L. Venezky, B. Gong (1996) Students' perceptions and designs of simple control systems. *Computers in Human Behavior* **12**, 363-388.
8. T. R. G. Green and M. Petre (1996) Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Journal of Visual Languages and Computing* **7**, 131-174.
9. P. Brusilovsky (1997) Mini-languages: a way to learn programming principles. E*ducation and Information Technologies* **2**, 65-83.
10. P. Whalley (2006) Modifying the metaphor in order to improve understanding of control languages— the little-person becomes a cast of actors. *British Journal of Educational Technology* (in press).