

## AGENT-BASED VIRTUAL INSECTS

Katherine Duczmal<sup>1</sup>, Peter Robinson<sup>1</sup>, & Jim Hanan<sup>1,2</sup>

<sup>1</sup> ARC Centre for Complex Systems,  
The University of Queensland

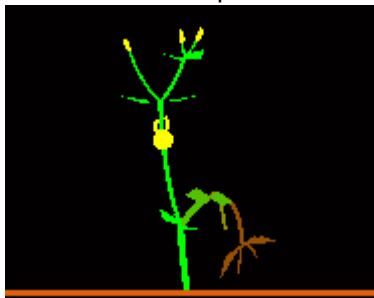
<sup>2</sup> Advanced Computational Modelling Centre,  
The University of Queensland

**ABSTRACT:** In recent years, complex systems techniques for modelling biological systems (plants, cells genes and others) have become invaluable in helping researchers developing insights into these fields. Entomologists too have been looking to apply computational modelling approaches to the field of insect behaviour. In this paper, a first-cut design of a system for creating virtual insects as agents is presented, in which behaviour is specified through an insect behaviour definition language based on Teleo-Reactive principles. These virtual insects are then simulated using a combination of a plant simulation package and an agent system that implements the insect behaviour. This system will allow entomologists to consider the validity of insect behaviour hypotheses by modelling and visualising insect behaviour.

### INTRODUCTION

There are problems associated with entomologists confirming hypotheses of insect behaviour derived from observations. It is difficult to predict the implications of hypothesised behaviours of an individual insect given the myriad of complex branching structures, flowers and tangled interconnections of plants growing in close proximity that make up its environment. This paper presents an investigation into creating virtual insects exhibiting behaviour as specified by entomologists. Simulating insects having proposed behaviour will allow entomologists to develop a better understanding of the accuracy of their models of behaviour. Refining this understanding has important applications to insect conservation, pest control, education and other related fields.

Lindenmayer Systems (L-systems) have become invaluable to biologists and others to enable these groups to describe the morphogenesis (formation of structure) of a plant [14]. Software tools have taken these L-systems and allowed users with no computer graphics or programming experience to create sophisticated 3D models of plants and animations of plant growth [14]. Entomologists desire similar sorts of capabilities to capture the interaction between plants and insects. Insects have been successfully introduced in L-systems based models (Figure 1 and [8]), however, L-systems descriptions are inappropriate for describing insects as their primary use is for describing plants – the descriptions produced are not intuitive to entomologists attempting to formalise insect behaviour. This problem has been the impetus for this project – to produce a system that can be integrated into an L-systems 3D graphical modelling environment that will allow entomologists and others to write models of insect behaviour without requiring programming knowledge.



**Figure 1 – L-studio implementation of model presented in [12]**

A side effect of creating an insect behaviour description language may be to provide a mechanism for entomologists to communicate about insect behaviours to their peers in a formalised way.

This project's aim has been to serve as a proof of concept that a system allowing entomologists to model insect behaviour could be developed. While consideration has been given to extensibility, this project implements only an initial prototype and has been modelled at the single plant/leaf, single insect level. Additionally, only crawling insects have been considered.

To produce an interface to the system that would allow entomologists to easily develop insect behaviour models it was decided to implement an insect behaviour definition language, similar to the use of the L-systems notation to describe plant growth behaviour. Existing behaviour programming

languages like JACK [1] or NetLogo [16] require the user to learn an entire programming language as well as its application to specifying behaviour. It was thought providing a simple rule based language would allow entomologists to focus on the behaviours themselves rather than how to go about achieving the desired result. Also, using a special-purpose behaviour language was consistent with the modelling and visualisation design concept within packages such as Vlab [2] or L-studio [13]. This language would be interpreted into code for insect agents that would then interact with the plant modelling program in the manner described in the open L-system environment specification [10]. In order to connect the system with the environment, it was decided to use a high-level communication mechanism. This allows for the possibility of linking the system to other plant simulation packages.

## METHODS

### *Agent Programming*

The agent programming paradigm was selected to implement virtual insects. Insects have often been associated with software agents. Indeed, much software agent theory has been based on the co-operative behaviour of social insects. However, detailed individual insect behaviour has not been given focus in the literature.

Software agents have been defined in [6] as “*software components that communicate with their peers by exchanging messages in an expressive agent communication language.*” In [3], a summary of the attributes a software agent may possess in greater or lesser quantities, as described by other authors, is presented:

- Reactivity: the ability to selectively sense and act
- Autonomy: goal-directedness, proactive and self-starting behaviour
- Collaborative behaviour: can work in concert with other agents to achieve a common goal
- “Knowledge-level” communication ability: the ability to communicate with persons and other agents with language more resembling humanlike “speech acts” than typical symbol-level program-to-program protocols
- Inferential capability: can act on abstract task specification using prior knowledge of general goals and preferred methods to achieve flexibility; goes beyond the information given, and may have explicit models of self, user, situation, and/or other agents.
- Temporal continuity: persistence of identity and state over long periods of time
- Personality: the capability of manifesting the attributes of a “believable” character such as emotion
- Adaptivity: being able to learn and improve with experience
- Mobility: being able being able to migrate in a self-directed way from one host platform to another.

It can be seen that an individual insect could be modelled as a software agent that communicates with its environment – an insect has inferential capability, temporal continuity, reactivity and autonomy. Indeed the insect itself could be modelled as many interconnected agents that control and monitor various functions of the insect.

Unlike other agent conceptual models (Belief-Desire-Intention [7] for instance), the Teleo-Reactive model represents a simpler, more clean-cut model of behaviour. This is desirable as not only does it capture the level of complexity to realise insect behaviours but also it allows the language design to be easily usable by the non-programmer. The Teleo-Reactive model [11] presents agent behaviour as an ordered series of inputs fed through a decision control function which results in a behaviour as output. A Teleo-Reactive system is likened to an analog electrical circuit; the agent’s behaviour remains the same until some combination of inputs produces new behaviour. Computationally, a Teleo-Reactive system is implemented as a series of boolean conditions (guards) that have behaviours associated with them. When a guard is satisfied, the corresponding behaviour can be executed. For example, in an insect context there might be a rule – if a leaf is tasty then eat. An input to the Teleo-Reactive system is the “tastiness” level of a leaf, the guard is whether the leaf is considered tasty by the insect and the output behaviour is that the insect eats.

The insect agents created by this system have been designed using the Teleo-Reactive model. The sensory information from the environment (percepts) represents the input to the Teleo-Reactive system; the output behaviour is reported back to the environment to be displayed in the visualiser.

### *Implementation Frameworks*

The L-system based plant modelling program called *cpfg*, a component of the Virtual Laboratory (Vlab/L-studio) was selected as the L-systems 3D modelling tool with which to implement the visualisation of insects within a plant canopy. *Cpfg* is widely used for plant simulation throughout many research centres.

Insect behaviour models (a crawling insect and butterflies) have been successfully implemented previously using L-system production rules [8]. The prior work of insect simulation provided benchmarks for comparison. A system of insect simulation using an insect behaviour description language had been developed [15], however, it contained only a 2D plant environment developed in Java which was not substantial enough to encompass all the detail of insect behaviour with respect to the environment. Models expressed using L-systems capture the subtleties of sophisticated plant-insect interactions.

Qu-Prolog was selected as the implementation language for the agents as well as the language to translate the insect behaviour language to agent code. Qu-Prolog was selected for these tasks for a number of reasons. First, the built-in backtracking of all variants of Prolog provides a useful mechanism for specifying behaviour. Second, given the language implementation and design formed a large component of this project, it was important to have a framework that would allow rapid prototyping of language design. Qu-Prolog, like most Prolog variants has a built-in definite clause grammar (DCG) interpreter. This provided an ease of use when implementing the lexical scanner and token parser. Third, Qu-Prolog is multi-threaded and also provides a high-level communication mechanism (Elvin – see Communication Mechanisms) which allows Qu-Prolog agents to communicate to the L-system environment, as well as an interprocess communication mechanism that allows Qu-Prolog threads to communicate.

### *Language Design*

The insect specification language design contains elements of procedural, functional and logic programming paradigms. The language has been designed to provide the greatest ease of use by the prospective users – entomologists. For this reason the new language must be usable without requiring understanding programming concepts. Thus, the Teleo-Reactive model (see Agent Programming) which satisfies this goal, has had a major influence on the language design.

Figure 2 is an example insect behaviour language file that produces an insect that moves in a zigzag pattern. The loops `main_behaviour_loop` and `turn_loop` represent Teleo-reactive systems. Each behaviour (move, turn, etc.) have optional guards that must be satisfied in order to select that behaviour. When a loop is executing, it will check the guards on behaviours in turn. The first behaviour for which the guard is satisfied will be executed. Once all the actions in the behaviour have been executed, control returns to the loop which will continue to execute behaviours until it reaches an END keyword at the end of a sequence of behaviour actions. A behaviour may have a call to another loop inside it, as in the case of the `turn_loop` in the turn behaviour of the `main_behaviour_loop`. This passes on control to the nested loop until an END keyword is reached in some behaviour of the nested loop. Behaviours are written sequentially in order of precedence. The first behaviour for which the guards are satisfied is always selected even if the guards for multiple behaviours are satisfied. From the prior description, it can be seen that this method of specification is essentially deterministic, however, stochastic models can be implemented using the random function call as in Figure 2. Other models such as random and deterministic walks, teleportation models (used when insects are not observed when changing position) can also be implemented using this language.

```

/*
    Represents Zig-zag insect movement model
*/
insect ZigZag {
    enum turn_type {left, right}

    initial_state {
        enum turn_type Last_Turn = right
    }

    let {
        int rand = random(1, 100),
    } in main_behaviour_loop {
        /* If the insect is at the edge of the leaf,
        turn back the other way */
        turn_at_edge::at_edge() ->
            turn_loop.
        /* Otherwise there is a 85% chance that
        the insect will continue moving forward */
        move::rand >= 16 ->
            move_forward().

        /* But there is a 15% chance that the insect
        will turn of its own accord */
        turn::rand <= 15 ->
            turn_loop.
    }

    turn_loop {
        perform_left_wheel :: Last_Turn == right ->
            turn_anticlockwise(90),
            move_forward(),
            move_forward(),
            turn_anticlockwise(90),
            Last_Turn = left,
            END.

        perform_right_wheel :: Last_Turn == left ->
            // ... like left_wheel case ...
    }
}

```

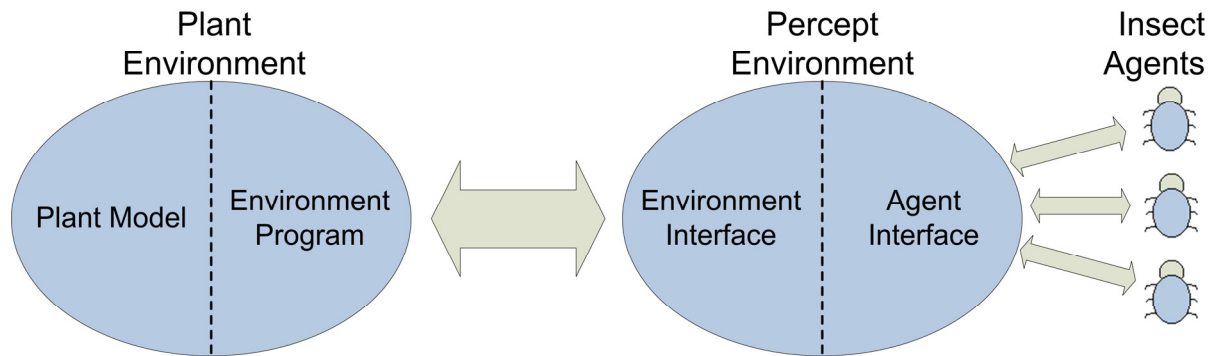
**Figure 2 – Language file that represents an insect that moves in a zigzag pattern**

Another factor that has been considered but not implemented in the language design has been predator-prey interactions. It should be the case that when a predator is detected by a prey insect, the current behaviour should be terminated, so that the prey insect can employ some counter-measure behaviour to avoid being eaten. In this case, there is a need for a mechanism to allow for some behaviours which are defined once but are implicitly at the start of each loop with appropriate guards to ensure that this behavioural interrupt is possible. It is proposed this could be implemented by defining these behaviours first at the top of the main\_behaviour\_loop prefixed with an appropriate keyword.

### *System Architecture*

The system was designed as a combination of two distinct parts – the cpfg plant model and the Qu-Prolog insect code. These communicate via a high level communication mechanism (Elvin). It was necessary to determine which side cpfg/Qu-Prolog barrier to place various functionalities.

As this project represents a proof of concept, to maximize the speed at which the system could be prototyped, it was decided that the functionality on the cpfg side should be made as simple as possible and hide the complexity on the Qu-Prolog side.



**Figure 3 – System Structure**

As can be seen in Figure 3, the agent environment is represented as a cpfg-side (plant) environment as well as a (percept) Qu-Prolog side environment. It was decided to implement a percept environment to reduce the communication complexity between the Qu-Prolog insect agents and the plant environment (it is communication intensive to synchronise distributed entities). Also, this meant the data stored in the plant environment was reduced and there was a clear division of responsibility. The plant environment is responsible for the graphical representation of the insect as well as the effects of insect habitation to L-system plants. This is communicated to the percept environment via Elvin. The percept environment is responsible for storing the state of all insects as well as communicating information from the plant environment to individual insect agents.

In order to ensure that both environments are synchronised in time, the plant environment sends a tick message that the percept environment acknowledges – the simulation will not proceed until the tick message is acknowledged. Similarly, insect agents are synchronised with the time of the percept environment. Each timestep represents a single iteration through a loop in the insect definition language code.

#### *Communication Mechanisms*

Elvin is used as the communication mechanism for messages between the percept environment and the plant environment. Elvin [5] is a publish/subscribe message based communication mechanism. Entities that register with the Elvin daemon can subscribe to a message format. Any notifications sent that match that message format will be sent to those entities. Elvin was chosen for the environment to environment communication because it is a mechanism that is easily supported in cpfg and allowed for multiple types of output (graphical representation of insects, statistics, etc).

The protocol for the environment to environment communication has been designed as simple request-reply pairs. The percept environment sends percept requests to the plant environment, which makes an appropriate reply. Additionally, there are also interactions for the percept environment to instruct the plant environment to change its state (moving the insect, inflicting damage on the plant, etc). Given the scope of the project, only a few basic percepts and state change primitives have been implemented. There is a vision percept which is limited to a whether a specific 3D grid co-ordinate is empty or has something in it. There is a taste percept which represents how “tasty” a grid co-ordinate is. State changes possible are an insect moving to a grid co-ordinate, changing its orientation and inflicting damage on a particular 3D grid co-ordinate. Some sample Elvin messages representing these are given below.

**Move state change**

```
[move = AgentID, x-co-ord = X, y-co-ord = Y, z-co-ord = Z]
```

moves the insect agent with agent ID AgentID to the position specified by the x, y and z co-ordinates.

**Vision percept request and reply**

```
[vision_percept_request=AgentID, x-co-ord = X, y-co-ord = Y, z-co-ord = Z]
```

```
[vision_percept_reply=AgentID, solid=Boolean]
```

communicates the percept of vision - whether the position specified by the x, y, and z co-ordinates is solid

Inter-Agent Communications Model (ICM) [9] is the communication mechanism used for the percept environment to communicate to individual insect agents. ICM (as provided in Qu-Prolog [4]) allows point-to-point communication of Prolog terms between named Qu-Prolog threads. Point-to-point communication is desirable for the environment to communicate to agents as each agent's percepts are different. The percept environment can then easily communicate percept information from the plant environment in a format that is easy to handle (Prolog terms). Similarly, the percept environment can communicate state information from agents to the plant environment.

**DISCUSSION**

The major outcome for this project has been to provide a proof of concept that the system and corresponding infrastructure simulating insects is feasible. Given the broad scope of this project, there are many facets that may be extended in the future. In this section areas for future work are discussed.

*Agent to L-system Communication Extensions*

The protocol specified in Communication Mechanisms will need to be progressively extended to include more sophisticated percept information than that presented here. This will potentially include sensing smell, temperature, specific nutrient content as well as toxins excreted by the plant. Percepts could potentially extend to levels of light and shade particular insects have a preference for.

*Conceptual Extensions*

The focus of this project has been on crawling insects within the context of the limited environment of a single plant. Many extensions are possible including other types of insect locomotion (hopping grasshoppers and flying moths and butterflies) as well as allowing for other animals (such as frogs) that reside within a plant ecosystem. The implementation of insects like butterflies will also imply the need to further the scope of the environment to multiple plants and interactions within an entire ecosystem.

Furthermore, practicalities such as relative sizes of insects versus plants (as well as insects versus insects) have not been addressed. Presently, the size of the insect is identical to the granularity of the 3D grid.

Attention has been predominantly focussed on the interactions of a single insect on a plant; although some considerations have been made for when multiple insects will inhabit a single environment. This will entail many alterations to the system. First, it will be necessary to add constraints for simple insect physics. Insects should not be able to walk through each other and insects should have the facility for collision avoidance. Second, co-operative behaviour (ants and the like) needs to be considered especially with respect to how co-operative insects will communicate (through percepts or some more abstract communication mechanism between agents). Third, while language constructs have been proposed to enable predator-interactions, support for predator-prey interactions will need to be implemented at all levels.

### *Output Extensions*

Currently the output of the system is the graphical representation of the insect and its environment. However, entomologists are also interested in tabular data with regards to the paths walked, time taken to perform a behaviour and other information that could be used for statistical analysis.

### *Language Improvements*

The language specification that allows entomologists to create these insect agents is relatively limited and contains little or no “syntactic sugar”. At the time of writing, only integer numbers had been implemented as part of the definition language. It is thought that floating point numbers will be added in the near future to enable further precision and also make stochastic modelling easier. Additionally, there is no string type as such – instead the enumerated type is used to control the flow of execution, this may need to be implemented if it becomes important with further use.

The language specification supports global and limited scope variables. However, it does not support higher data structures (like C-style structs and arrays, object-oriented objects or lists). For large behaviour descriptions this may be too limiting. Also there is no support for encapsulation – common functionality cannot be placed in a common file that an insect definition loads or includes as part of its definition. This may be considered to be especially useful for insects with similar, but not the same, behaviour.

### *Language Tool Improvements*

Error messages in the scanner, parser and type checker could be improved by making these more concrete and providing the line number of the error. Additionally, the code generator generates adequate Prolog agent code, however, in some cases code optimisation is possible (for example expression evaluation is handled by a specific expression evaluator rather than the inbuilt Prolog evaluation). This may become increasingly important as more agents are added to simulations.

Also, Vlab/Lstudio represents a complete integrated development environment (IDE) for the graphical display of L-system models. It is proposed that a simple editor for the insect behaviour language be developed (with syntax highlighting and the like) or a plug-in to allow insect models (editing, etc.) to be fully integrated into Vlab/Lstudio.

## CONCLUSION

Just as those interested in the morphogenesis of plants have been able to easily create sophisticated 3D representation of plants and ecosystems using L-systems tools, so too can the first steps be made in allowing entomologists to insert insects into these models using a system that affords a straight forward mechanism for developing insects as agents.

## ACKNOWLEDGEMENTS

I would like to thank my supervisors, Dr. Peter Robinson and Dr. Jim Hanan for their guidance, as well as Prof. Myron Zalucki and Dr. Marie-Louise Johnson for their advice and assistance regarding the entomological side of this project. Additionally, I would like to acknowledge Tim Rudge and Craig Harvey for extending Vlab to enable Elvin communication and communicating percept information respectively as well as acknowledging the continued support of the ARC Centre for Complex Systems.

## REFERENCES

- [1] Agent Oriented Software Group, What is JACK?, Available: <http://www.agent-software.com/shared/products/>. [Accessed 17 November 2005].
- [2] Algorithmic Botany, Available: <http://www.algorithmicbotany.org>. [Accessed 7 September 2005].
- [3] J.M. Bradshaw, *Software Agents*, MIT Press, 1997.

- [4] K. Clark, P.J. Robinson, R. Hagen, "Multi-threading and message communication in Qu-Prolog", *Theory and Practice of Logic Programming*, vol. 1, no. 3, May 2001, pp. 283-301.
- [5] "Elvin - Content Based Messaging Homepage", Available: <http://www.elvin.dstc.edu.au>. [Accessed 5 September, 2005].
- [6] M.R. Genesereth, "Software Agents", *Communications of the ACM*, vol. 37, no. 7, July 1994.
- [7] M.P. Georgeff, B. Pell, M.E. Pollack, M. Tambe, and M. Wooldridge, "The Belief-Desire-Intention Model of Agency", *5<sup>th</sup> International Workshop on Intelligent Agents, Agent Theories, Architectures, and Languages*, Springer-Verlag, 1999, pp. 1-10.
- [8] J. Hanan, P. Prusinkiewicz, M. Zalucki and D. Skirvin, "Simulation of Insect Movement with Respect to Plant Architecture and Morphogenesis", *Computers and Electronics in Agriculture*, vol. 35, no. 2, Aug. 2002, pp. 255-269.
- [9] F.G. McCabe, "ICM Reference Manual". *Fujitsu Labs of America*, 1999, <http://www.nar.fujitsulabs.com/documents/icm-manual.html>.
- [10] R. Mech and P. Prusinkiewicz, "Visual Models of Plants Interacting with Their Environment", *SIGGRAPH 96, ACM*, 1996, pp. 397-410.
- [11] N.J. Nilsson, "Teleo-Reactive Programs for Agent Control", *Journal of Artificial Intelligence Research*, vol. 1, Jan. 1994, pp. 139-158.
- [12] P. Prusinkiewicz, M. Hammel, J. Hanan, and R. Mech. "Visual models of plant development", In G. Rozenberg and A. Salomaa, editors, *Handbook of formal languages*. Springer- Verlag, 1996.
- [13] P. Prusinkiewicz, R. Karwowski, R. Mech and J. Hanan, "L-Studio/cpfg: A Software System for Modeling Plants", *Lecture Notes in Computer Science*, vol. 1779, Jan. 2000, pp. 457.
- [14] P. Room, J. Hanan and P. Prusinkiewicz, "Virtual Plants: New Perspectives for Ecologists, Pathologists and Agricultural Scientists", *Trends in Plant Science*, vol. 1, no. 1, Jan. 1996, pp. 33-38.
- [15] A. Tan, *A Virtual Insect Modelling System*, honours thesis, School of Information Technology and Electrical Engineering, University of Queensland, 2003.
- [16] U. Wilensky, NetLogo, Available: <http://ccl.northwestern.edu/netlogo>. [Accessed 17 November 2005].