(To appear in the Proceedings of the Sixth International Conference on Intelligent Data Engineering and Automated Learning (IDEAL'05))

Patterns in Complex Systems Modeling

Janet Wiles and James Watson

ARC Centre for Complex Systems
School of Information Technology and Electrical Engineering
The University of Queensland, Brisbane, 4072, Australia
http://www.itee.uq.edu.au/~patterns/

{j.wiles, j.watson}@itee.uq.edu.au

Abstract. The design, development, and use of complex systems models raises a unique class of challenges and potential pitfalls, many of which are commonly recurring problems. Over time, researchers gain experience in this form of modeling, choosing algorithms, techniques, and frameworks that improve the quality, confidence level, and speed of development of their models. This increasing collective experience of complex systems modellers is a resource that should be captured. Fields such as software engineering and architecture have benefited from the development of generic solutions to recurring problems, called *patterns*. Using pattern development techniques from these fields, insights from communities such as learning and information processing, data mining, bioinformatics, and agent-based modeling can be identified and captured. Collections of such 'pattern languages' would allow knowledge gained through experience to be readily accessible to less-experienced practitioners and to other domains. This paper proposes a methodology for capturing the wisdom of computational modelers by introducing example visualization patterns, and a pattern classification system for analyzing the relationship between micro and macro behaviour in complex systems models. We anticipate that a new field of *complex systems patterns* will provide an invaluable resource for both practicing and future generations of modelers.

1. Introduction

While complex systems models are used by a wide variety of often experienced research groups, the development of such models is far from an exact science. The design of model architecture and algorithms, the choice of hardware and software infrastructure, and methods for tracking and analyzing results, are just some of the problems often faced and often re-solved. While each research project and each simulation within it is unique, there are many underlying commonalities which, once identified, can save much 'reinvention of the wheel'.

Established fields such as architecture [1] and software engineering [2] have captured much of the experience gained by their respected practitioners. These rules of thumb are known as *patterns* in architecture and software engineering (e.g. design patterns [3], process patterns [4]). They record both commonly-recurring problems, pertaining to any aspect of the work from global structure to specific detail, and their

proven solutions. Libraries of such patterns are more than just collections of a field's heuristics. The context in which a solution is appropriate plays a critical part in the description of a pattern. Context is a recognized part of the background knowledge of any field, although in most areas it is rarely described as an integral part of a heuristic. For the field of machine learning, it is well known that no search technique will be optimal for all tasks, frequently expressed as 'no free lunch', however it is still rare for a new algorithm to be presented with an assessment of the tasks in which it is *in*-appropriate. It is important to know in which contexts a heuristic is effective, and the consequences of their use, such as tradeoffs between accuracy versus time.

Capturing commonly-recurring problems, their contexts and consequences has had multiple benefits for software engineering and architecture. First, both beginning and experienced practitioners benefit from a rapidly accessible library of tested ideas, information about the situations where they should be used and the consequences or tradeoffs of their use. Second, an appropriate and tested format (building on 40 years of patterns research in a variety of fields) facilitates the ongoing community development of the library. This format provides a shared language of tools and methods agreed upon by the community, enabling rapid communication. Third, capturing the wisdom of the field in a practical and effective resource builds confidence in the quality of systems produced.

The field of complex systems currently lacks this library of experience. Shared libraries of algorithms and heuristics exist but they lack the experience of the complex systems modeler in when to apply them.

Using the patterns approach to capture experience in the field of complex systems modeling would provide a further benefit. The complex systems community is a unique collection of researchers from many disparate backgrounds (such as machine learning, complex adaptive systems, neural networks, gene expression analysis, multi-agent systems, etc.). However, the fundamental issues studied are common to all, such as the emergence of macro-level behaviour from micro-level interactions. Insights from one branch of complex systems science can often be applied to another when taken at an appropriate level of abstraction. Patterns can be used to capture knowledge unique to one branch of the field and effectively communicate these insights to other areas.

2. Software Patterns

The software engineering field has had decades of experience developing, maintaining and using complicated systems, advancing techniques such as problem decomposition, verification, validation and project management. Software engineering provides extensive experience in pattern development, with patterns spanning the problem space from global architecture to detailed programming language specific idioms. More than a heuristic or data structure, a pattern is the solution to a problem in a specific context, balancing inherent tradeoffs [5]. Software engineering patterns have defined characteristics including name, intent, motivation, consequences, known uses, and related patterns [2] (see Section 4).

A good software engineering pattern has the following key characteristics [6]:

- solves a particular problem (not just in principle);
- is a proven solution that has been successfully employed in at least three significant scenarios; and
- has a significant human component describing how and when it is useful.

In terms of the patterns development process, the following characteristics have become accepted in the software engineering community [6]:

- a strong focus on proven solutions to recurring problems;
- those writing patterns do *not* have to be the original inventor of the solution;
- non-anonymous review, where discussions focus on how patterns should be clarified or improved upon;
- development through discussions in workshops instead of solo presentations; and
- careful editing, through which the pattern authors can incorporate feedback gained through workshops and review before presenting the patterns in their final form.

These characteristics provide a useful guide for the development of patterns in complex systems science.

3. Complex Systems Patterns

As in software engineering, the field of complex systems consists of practitioners experienced with a wide range of solutions to common problems. Modeling techniques and implementation frameworks, methods of analysis and visualization, and the most effective ways of reporting results, are just some of the techniques that form the collective experience of the field. We propose that one of the most effective methods to capture and communicate complex systems knowledge is the proven technique of pattern development from software design.

Patterns from the complex systems community can be grouped into two classes. The first is collections of proven solutions to commonly recurring problems that occur in the development and use of complex systems models. Examples in this class of patterns include hardware and software platforms, model architectures and abstractions, and analysis and visualization techniques. The second source of potential patterns are the insights generated by the complex systems models themselves. Emergent robustness, evolvability, efficient connectivity, and modular design are some of the characteristics found in complex systems that can inform the development of solutions to certain problems, and communicate insights from one class of models, such as agent-based systems, to another, such as genetic regulatory networks.

Initial places to search for such classes of patterns include the visualization of network structure and dynamics such as expression patterns [7], [8], metrics for connectivity, diameter, and cluster coherency of networks [9], [10], and micro and macro scale, temporal and spatial structures [11], [12].

Patterns vary in their level of abstraction and granularity, and consequently there needs to be a way to classify them [2]. The micro and macro level characteristics of

complex systems models can be at the level of structure, dynamics, or function. The structural level focuses on the static relationships between model entities, for example, a network of agent relationships or gene connectivity. The dynamic level is concerned with the interactions between these components, for example, a diagram of gene activations or the state space. The functional level focuses on the entire complex system functioning within an environment (e.g., a phenotype based on underlying component interactions placed within an evolutionary algorithm). A preliminary set of complex systems patterns, useful for modeling Boolean networks, and their classifications, is summarized in Table 1.

Table 1. Examples of visualizations that can inform development and use of Boolean network models

	Targeted Complex Systems Features					
Pattern	*Structure*	*Dynamics*	*Function*	*Micro Mechanics*	*Macro Behaviours*	.....
Network Diagram	✓			✓		
Activation Diagram		✓		✓	✓	
State Space Diagram		✓			✓	
.....						

4. Example Patterns

Consistent with best practice, refinements can be found at
http://www.itee.uq.edu.au/~patterns/repository/

4.1 Activation Diagram

One commonly-recurring problem in genetic regulatory network modeling is the visualization of system behaviour, where interesting behaviours span multiple levels in time and space. To provide a concrete feel for the nature and scope of a complex system pattern, this section illustrates a prototype pattern which solves this problem.

Name: Activation Diagram (Classification: Dynamics, Micro-Mechanics, Macro Behaviours)

Intent: Visualize micro level activation of components over time to see macro level characteristics.

Also known as: Gene expression diagram, gene activation diagram, expression pattern, activation signature

Motivation: Understanding the effects over time of interactions between large numbers of nodes in a network can be difficult and time-consuming. Inferring macro-level classes of behaviour is easiest when the history of system-wide activations are presented as a single diagram. The idea of this pattern is to provide a visualization of node histories for a single initial condition, allowing macro-level features such as stable, cyclic, or chaotic behaviour to be identified, and characteristics such as the length of transient periods to be measured.

Applicability: Use the Activation Diagram pattern when you want to:

- visualize the characteristics of component activations over time when components have binary or real-valued states;
- visualize characteristics of macro level behaviour such as ordered, cyclic, or chaotic activity;
- visualize both the initial transient dynamics and the longer term behaviour;
- assess the life cycle of macro level behaviours (such as the number of steps before a network settles into a certain state); or
- (variation) manually investigate robustness of macro-level behaviour

Example Visualization:

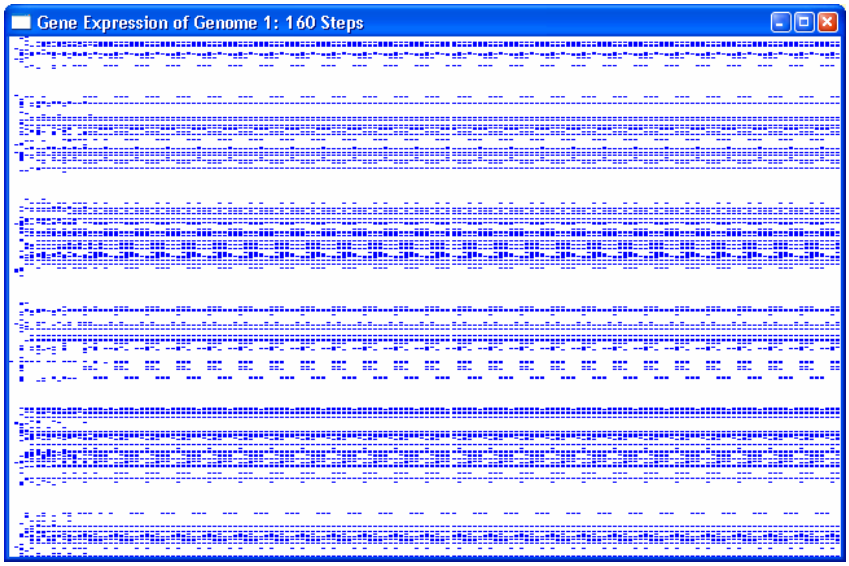

Fig. 1. Activation diagram. Time is shown along the *x* axis, and each component is positioned along the *y* axis. Active components are denoted by dark shading. This diagram shows the component activations falling into a cyclic state after a short transient period.

Consequences: The Activation Diagram has the following consequences and inherent limitations:

- it provides a clean visualization of the dynamics from a single starting state but the inherent limitation is that only a single starting state and trajectory is shown per diagram;
- it requires access to the values of all components for each time step;
- large numbers of components can make viewing difficult;
- very long cycles can appear similar to chaotic trajectories;
- the two-dimensional representation maps time into space, consequently spatial information is lost (e.g., in random Boolean networks, neural networks); this can be mitigated by using the Activation Diagram pattern together with the Network Diagram pattern; and
- spatial information is preserved if a one-dimensional representation is used (e.g., cellular automata)

Implementation: The Activation Diagram has the following important implementation variations:

- time can be expressed along the *x* or *y* axis; and
- one-dimensional interactions (e.g., cellular automata) can be visualized by ordering nodes according to their interactions

Known uses: Gene expression [13], random Boolean networks, cellular automata, neural network dynamics.

Related patterns: State Space Diagram, Network Diagram

Sample Code:
This example visualizes the expression pattern of a Boolean network of gene regulation. A C++ source code listing is available at
http://www.itee.uq.edu.au/~patterns/repository/activation-diagram.html

```
e = expression data, indexed by [step number][gene number]
s = number of steps in e
g = number of genes in e

if (s > 0) and (g > 0):
    clear the screen
    xdist = screen-width / s
    ydist = screen-height / g
    xgap = 0.12 * xdist
    ygap = 0.12 * ydist

    x = 0  /* where 0 is leftmost screen coordinate */
    for i = 1 to s:
          y = 0  /* where 0 is topmost screen coordinate */
          for j = 1 to g:
                if expression_data[i][j] is activated:
                      x1 = x + xgap
                      y1 = y + ygap
                      x2 = x + xdist – xgap
                      y2 = y + ydist – ygap
                      draw_rectangle(x1, y1, x2, y2)
                y = y + ydist
          x = x + xdist
```

4.2 Network Diagram

Name and Classification: Network Diagram (Structure, Micro Mechanics)

Intent: Visualize micro-level interactions of components at a given point in time

Also Known As: Graph

Motivation: Understanding the relationships between nodes in a network, together with their spatial information, is most intuitive with a graphical depiction. The idea of this pattern is to visualize the nature and number of interactions between nodes in a network, and any spatial relationship these nodes may have with each other.

Applicability: Use the Network Diagram pattern when you want to

- visualize the interactions of components at a given point in time
- visualize spatial relationships between network components

Consequences: The Network Diagram has the following consequences and inherent limitations:

- interactions are only shown for a single point in time
- large numbers of nodes can make viewing difficult

Implementation: Implementation issues to consider for Network Diagram include:

- Spatial arrangement of nodes is very important when identifying certain network characteristics. A random layout is the simplest to implement, but is generally unsuitable for visualizing the giant component of the network. Increasingly sophisticated network layout algorithms can incur a cost in processor time (many optimal layouts are likely to be NP-complete).

Known Uses: Boolean network visualization and design, social network visualization, neural network visualization and design

Related Patterns: State Space Diagram, Activation Diagram

Sample Code:

```
n = network, indexed by node
p = position of each node

clear the screen
for i = 1 to (number of nodes in n):
    p[i] = random position
    draw sphere at p[i]

for i = 1 to (number of nodes in n):
    r = list of nodes regulated by n[i]
    for j = 1 to (number of nodes in r):
          draw arrow from p[i] to p[r[j]]
```

A C++ source code listing is available at
http://www.itee.uq.edu.au/~patterns/repository/network-diagram.html

5. Conclusions

Patterns are proven solutions to commonly-recurring problems. Using the extensive pattern development experience of the software engineering field, the complex systems community can capture its collective experience. Patterns provide a framework that asks the right questions to extract and document knowledge gained through experience, and offer a standardized language with which to discuss this captured knowledge. This paper is an initial step towards a community-driven library of complex systems patterns. Updates and pattern contributions are available online at http://www.itee.uq.edu.au/~patterns/.

Acknowledgments

This work was funded by the ARC Centre for Complex Systems and an Australian Research Council grant to the first author.

References

1. Alexander, C., et al., *A Pattern Language*. 1977, New York: Oxford University Press.
2. Gamma, E., et al., *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. 1994: Addison-Wesley.
3. Srinivasan, S., *Design patterns in object-oriented frameworks.* Computer, 1999. **32**(2): p. 24-32.
4. Moore, J., et al. *Combining and adapting process patterns for flexible workflow*. in *11th International Workshop on Database and Expert Systems Applications*. 2000.
5. Coplien, J.O., *Software design patterns: common questions and answers*, in *The Patterns Handbook: Techniques, Strategies, and Applications*, L. Rising, Editor. 1998, Cambridge University Press, New York. p. 311-320.
6. Appleton, B., *Patterns and software: essential concepts and terminology*. 2000. http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html
7. Wuensche, A. *Genomic regulation modeled as a network with basins of attraction*. in *Pacific Symposium on Biocomputing '98*. 1998. Singapore: World Scientific.
8. Solé, R.V., P. Fernández, and S.A. Kauffman, *Adaptive walks in a gene network model of morphogenesis: insights into the Cambrian explosion.* International Journal of Developmental Biology, 2003. **47**(7/8): p. 685-693.
9. Strogatz, S.H., *Exploring complex networks.* Nature, 2001. **410**: p. 268-276.
10. Newman, M.E.J., *The structure and function of complex networks.* SIAM Review, 2003. **45**(2): p. 167-256.
11. Raff, R.A., *Evo-Devo: The evolution of a new discipline.* Nature Reviews Genetics, 2000. **1**(1): p. 74-79.
12. Hasty, J., et al., *Computational studies of gene regulatory networks: In numero molecular biology.* Nature Reviews Genetics, 2001. **2**(4): p. 268-279.
13. Reil, T. *Dynamics of gene expression in an artificial genome - implications for biological and artificial ontogeny*. in *The 5th European Conference on Artificial Life*. 1999: Springer Verlag.