(To appear in World Scientific's Advances in Natural Computation series.)

Towards a Network Pattern Language for Complex Systems

J. Watson[1,2], J. Hawkins[1,2], D. Bradley[3,4],
D. Dassanayake[5], J. Wiles[1,2] and J. Hanan[1,4,6]

[1]ARC Centre for Complex Systems,
The University of Queensland, Brisbane QLD 4072 Australia
http://www.accs.edu.au/patterns/

[2]School of Information Technology and Electrical Engineering,
The University of Queensland, Brisbane QLD 4072 Australia
Email: {jwatson, jhawkins, j.wiles}@itee.uq.edu.au

[3]Institute for Molecular Bioscience,
The University of Queensland, Brisbane QLD 4072 Australia
Email: d.bradley@imb.uq.edu.au

[4]ARC Centre in Bioinformatics,
The University of Queensland, Brisbane QLD 4072 Australia

[5]CSIRO Land & Water,
School of Science and Technology,
Charles Sturt University, Wagga Wagga NSW 2678 Australia
Email: dharma.dassanayake@csiro.au

[6]ARC Centre of Excellence for Integrative Legume Research,
The University of Queensland, Brisbane QLD 4072 Australia
Email: jim@maths.uq.edu.au

Abstract

The development and use of complex systems models can involve many common problems, problems that are solved again and again by different researchers with various backgrounds and experience. The application of the software engineering patterns paradigm to complex systems modeling will enable capture of the wisdom of the network modeling community in such a way that proven solutions to recurring challenges can be identified and tailored to the specific problem at hand. The use of networks for simulation and analysis are two areas of complex systems modeling that stand to benefit from the patterns approach. The use of networks to guide thinking, as analytic tools and as visualizations is ubiquitous in the field of complex systems. However, various methods of using networks (e.g. design, updating functions, visualization, analysis) are not always obvious to a newcomer and are often assumed as general knowledge in the literature. This paper is a first step towards a pattern language addressing these issues for the complex systems community.

1 Introduction

The field of complex systems is concerned with systems whose global behaviour cannot be predicted simply by examining components in isolation. Such studies of emergent behaviour have provided insights into disparate disciplines such as economics [2], air traffic control, and genetic regulatory networks [5]. Since complex systems are comprised of numerous entities, each interacting with some subset of the total, the use of networks is fundamental to the field. Networks can be used to analyze the entities and their relationships, and to investigate state transitions [11].

An often overlooked fact is how networks guide the way an investigator may think about a complex system, and the types of analysis and insights that may be drawn from theorizing about the system. Therefore, the way to use networks in complex systems modeling is not always obvious. Also, the finer details of a methodology – such as why a technique was or was not used, limitations and trade-offs of methods, the reasoning for discarding designs, etc. – often do not make it into the published literature. Consequently, this knowledge is often continuously rediscovered, or passed on only through informal discussion.

Patterns are a proven technique developed to deal with similar issues of reuse and knowledge capture in the fields of software engineering and architecture [3]. Far from being just documented techniques or heuristics, patterns include not only the benefits of a method, but also the inherent trade-offs and consequences. They are organized into classifications so that, for a given situation, they can be easily found and used in collaboration with other patterns as appropriate. Patterns span the range of complex systems methodology, from the abstract design of a system down to an implementation of a particular update function. Importantly, they describe proven solutions that have been discovered multiple times.

The capture of commonly recurring problems, along with the details and consequences of their solutions, can have multiple benefits. Practitioners, both novice and experienced, can benefit from a library of tested ideas. After undergoing over 40 years of development in a variety of fields, from architecture to software engineering [3], existing pattern formats facilitate the ongoing community development of a pattern library by (i) asking the right questions to appropriately derive the pattern, and (ii) by providing a shared language to rapidly communicate ideas agreed upon by the community. In addition to saving time otherwise spent reinventing solutions, using patterns in the design, development and analysis of a complex systems model can build confidence in the quality of the model produced. For more information on the use of patterns in complex systems modeling, see Wiles and Watson [10].

The goal of this paper is to introduce three network-centric prototype patterns (or proto-patterns [1]), intended to form the basis of a pattern language. A pattern language is a collection of related patterns [1] covering the use of networks in complex systems modeling. It is intended that input from the complex systems community will refine these to the state of completed patterns. The first is an abstract Network Diagram pattern, which describes the general ways a network can be used to solve complex systems problems. The second is a more concrete Synchronous System State Update pattern, which describes how a current network state may be used to produce the next simulation state. Finally, a Discrete State-Space Trajectory Diagram pattern is outlined, which offers a visualization of system state dynamics.

2 Methodology

The three proto-patterns were refined after two days of development at the first Complex Systems Patterns Workshop hosted by the ARC Centre for Complex Systems at the University of Queensland. The workshop attendees discussed commonalities among their problems and solutions, and then focused on areas in need of complex systems patterns.

Using an appropriate, agreed-upon format is critical to pattern development. By asking the right questions, a good format facilitates communication and the extraction of modeler experience at an appropriate level of detail. The pattern format used to develop the complex systems patterns is adapted from the object-oriented design patterns format used by the software engineering community. The adapted format is shown in Table 1.

In keeping with the patterns development process used in software engineering, the three proto-patterns described in this paper were developed (i) with a focus on recurring problems and their proven solutions, (ii) through development in workshop discussions instead of solo presentations, and (iii) not necessarily by the original inventor of the solution.

3 Results

The following sections describe how the three proto-patterns were developed (see Appendices A, Network Diagram, B, Synchronous System State Update, and C, Discrete State-Space Trajectory Diagram).

3.1 Development of the Network Diagram pattern

The discussion leading to the Network Diagram pattern initially centred on issues of visualizing and manipulating models of networks in complex systems simulations. However, the discussion soon elucidated a second major theme: the issue of knowing when to use a network representation in the first place. Even though in many instances a system can be naturally seen as a network due to its physical structure, in some applications the network is a significant abstraction from the system being studied. Furthermore, there are often numerous ways in which one can abstract a network structure from a representation of a physical system. For example the relationships between academic authors can be studied using either co-authorship or citations as

connections between authors.

The first point of contention in this discussion was whether we were talking about the general issue of when one should use a network to visualize a problem, or the issue of how one visualizes something as a network when a network approach has already been decided on. The two issues exist at different levels of complex systems modeling methodology. The first is an issue of deciding how to approach a problem; the second is an issue of how to implement a solution. Both issues are equally deserving of a complex systems pattern; however, focus was given to the visualization of a network structure.

This emergence of two competing notions of what the pattern should communicate illustrates an important lesson about the process of pattern development. Even though patterns are by their nature abstract, they can still exist at differing levels of abstraction. They can be relatively specific, dealing with structuring code to solve a particular design problem, or alternatively they can address conceptual aspects of the modeling process.

Two tensions emerged in the discussion of this pattern. The first was a natural consequence of the different domain knowledge among participants. These differences lead to a discussion that gradually drew out the commonalities across domains. Secondly, it was apparent during the discussion that each of the participants had a natural level of abstraction to which they would return. This resulted in contention regarding the exact level of abstraction at which the pattern exists.

The first of these issues was somewhat obvious and quickly resolved, the details of differing fields of application were put aside and the essential modeling problem distilled. However, the second issue was not readily apparent and consequently often remained unresolved until well into the discussions. While it remained unresolved it led to people talking at cross purposes, and failing to find consensus. In the end it was solved by a specific discussion of abstraction, which operated to create a balance between generality and usefulness.

It is important to recognise that the strength of a workshop approach is the drawing out of commonality from the experiences of different individuals. However, while issues remain unresolved, productive discussion can stall. We would recommend that an early part of the pattern development process involve agreeing specifically about the level of abstraction. Concrete topics such as identifying the target audience and the words that would be used to describe the particular problem solved by the pattern can help in elucidating this issue. Without deliberate targeting, the nature of the problem can

remain vague and change continuously throughout the discussion.

For several reasons the ultimate content of the pattern converged upon the more abstract issue of recognizing when to apply a network diagram. Firstly, we had originally planned to make at least one other concrete network pattern to complement the general network diagram. Hence, it seemed appropriate to begin the pattern language with the most abstract aspect of networks in modeling, 'When and why to use them?' Secondly, in developing a pattern repository, people should ideally be able to come to the repository with only a set of questions and few preconceptions about the solution. If the network pattern language began with the assumption that users already knew about and intended to use a network representation, then the repository would exclude a potentially large group if users who have problems and no inkling that they can be solved with a network representation.

3.2 Development of the Synchronous System State Update pattern

The discussion for the Synchronous System Update pattern began with the observation that some newcomers to network modeling choose naive implementations for sets of synchronously interacting entities that can be all updated simultaneously (i.e., node state updates neglect to account for the interactions of all other nodes before the updates are committed). Even though the solution to this problem may seem trivial to an experienced modeler, it is one that many people solve independently, and almost certainly causes problems for novice computer modelers. The fact that people tend to converge to one of a small set of solutions makes this problem an ideal case for an implementation level pattern.

Further discussion of the exact nature of the problem and the structure of the solution illustrated that it is a problem much broader than network models, and even complex systems. It applies to any simulated system that consists of numerous entities that must all be updated synchronously where changes of entities are affected by the current state of some of the others. For example, this situation occurs in the construction of computer games, where it is called the central game loop. Many game programming books devote an entire chapter to the subject, since it is the fundamental controller whereby the discrete temporal nature of the system is enforced.

3.3 Development of the Discrete State-Space Trajectory pattern

Much of the field of complex systems requires some understanding of the overall behavioural characteristics of a system. Researchers require methods of capturing essential behavioural properties that filter out superfluous information. This is the purpose of the Discrete State-Space Trajectory Diagram. In this pattern the states of a system are depicted as nodes of a network, with edges representing the transitions between states.

The discussion of this pattern began with its application in abstract genetic regulatory networks (GRNs). The use of state space diagrams has been extensive in this area, beginning with Kaufmann's random Boolean networks [6][7]. It has been particularly successful due to the layout approach known as the Wuensche diagram [11]. In this visualization all the nodes involved in the central attractor of the system are placed in a circle in the centre of the page. Nodes that feed into this attractor are placed at ever increasing distances from the central circle, so that the outermost nodes are those from which the system must pass through the most transient states in order to reach the attractor. Any other smaller attractors are arranged similarly on other parts of the page. This layout approach renders the overall system dynamics comprehensible to visual inspection.

Discussion of this pattern soon veered away from the particulars of its application in GRN modeling to defining the essentials of circumstances in which it can be applied. We discussed the fact that although it is most naturally applied to models with discrete states, it can be applied to continuous systems by quantising system variables. In fact such an approach could potentially lead to valuable insights into a continuous system; by investigating changes in the system dynamics as the scale of quantisation is varied.

Discusssions also dealt with the issue of state space size. If the number of system states prohibits exhaustive enumeration, it was agreed that sampling a small number of start states can still yield insight into the state space. In many instances this will illuminate the dominant attractors of the system. Secondly, one may also apply the pattern to investigate particular subsections of the state space that are of interest.

Numerous quantitative results can be produced in the application of this pattern. One may measure the number of attractors, the distribution of their sizes, the number of transients, the distribution of their lengths, the number of unreachable states and stationary points. Furthermore, this pattern leads

on to the more detailed analysis of issues such as the stability of the attractors via perturbation analysis [4].

4 Conclusions

Patterns are a means of capturing the collective experience of the complex systems community. Using the proven patterns format of software engineering to guide the development of complex systems patterns offers two main benefits. First, by posing suitable questions, the knowledge gained by experienced practitioners is extracted at an appropriate level of detail. Second, a consistent and shared language is provided, which facilitates discussion of this captured knowledge.

We have presented the development of three patterns related to the use of networks in complex systems modeling. Some of the typical issues that occur in defining patterns were highlighted through a discussion of the process used to develop these patterns.

5 Future Work

The patterns presented in this paper are progress towards a pattern language that captures essential applications of networks to complex systems. During the workshop and subsequent refinement process we identified two prominent areas for future work. First, analytical techniques for studying network structure form an important area of network modeling that hasn't been touched upon in this paper. There are numerous metrics for comparing networks, which would be served well by being organized as patterns that explain how and when to use them. Second, in our discussion we have focused on networks with static structures. There exists an extensive amount of work done on network rewiring algorithms, and on complex systems in which the network structure changes endogenously over time. We expect that formalizing the content of these two broad areas of research would be the next step along the path to a network pattern language.

Further work including a complex systems patterns repository is available online at `http://www.accs.edu.au/patterns/`.

Acknowledgements

The authors would like to thank the participants of the First Complex Systems Patterns Workshop for illuminating discussions and insights. In particular, thanks to Scott Heckbert, David Carrington and Andrew Hockey. This workshop, held 6-7 June 2005, was supported by the ARC Centre for Complex Systems (ACCS), the ARC Complex Open Systems Network (COSNet), and CSIRO's Complex Systems Science Area. The Complex Systems Patterns Project is funded by the ARC Centre for Complex Systems. Daniel Bradley was supported by Australian Research Council grant CE0348221.

References

[1] B. Appleton. Patterns and software: essential concepts and terminology, 2000.
http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html.

[2] J. Foster and W. Holzl. *Applied Evolutionary Economics and Complex Systems*. Edward Elgar, Cheltenham, England, 2004.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: elements of reusable object-oriented software*. Addison-Wesley, 1995.

[4] N. Geard, K. Willadsen, and J. Wiles. Perturbation analysis: A complex systems pattern. This volume.

[5] J. Hallinan and J. Wiles. Evolving genetic regulatory networks using an artificial genome. In Y.P. Chen, editor, *Proc. Second Asia-Pacific Bioinformatics Conference (APBC2004)*, volume 29 of *Conferences in Research and Practice in Information Technology*, pages 291–296. Australian Computer Society, 2004.

[6] S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22:437–467, 1969.

[7] S. A. Kauffman. Gene regulation networks: a theory for their global structure and behaviours. *Current Topics in Developmental Biology*, 6:145–182, 1971.

[8] A. Lindenmayer. Mathematical models for cellular interactions in development, parts I and II. *Journal of Theoretical Biology*, 18:280–315, 1968.

[9] T. Reil. Dynamics of gene expression in an artificial genome - implications for biological and artificial ontogeny. In D. Floreano, F. Mondada, and J.D. Nicoud, editors, *Proceedings of the 5th European Conference on Artificial Life*, pages 457–466. Springer Verlag, 1999.

[10] J. Wiles and J. Watson. Patterns in complex systems modeling. *Lecture Notes in Computer Science*, 3578 (June):532–539, 2005.

[11] A. Wuensche. The ghost in the machine: basin of attraction fields of random boolean networks. In C. G. Langton, editor, *Artificial Life III*, Sante Fe Institute Studies in the Sciences of Complexity, Reading, MA, 1994. Addison-Wesley.

Appendix A – Network Diagram

Submitted by: James Watson, John Hawkins, Daniel Bradley, Dharma Dassanayake, Jim Hanan, Scott Heckbert, and Andrew Hockey.

1a. Pattern name: Network Diagram

1b. Classification: Visualization (Structure, Dynamics, Function, Micro, Macro, State Space)

2. Intent. A fundamental property of complex systems is that the interactions between micro-level components result in emergent macro level behaviour. This emergent behaviour means that it is difficult to know which components are of interest. In addition, there are large numbers of possible metrics available to analyze the properties of a complex system. Thus, a common starting point is to simply generate some form of initial visualization of components and their interactions.

A network diagram provides a spatial representation of entities and their relationships. By hiding the specifics of entities and their relationships, an initial overview of the complex system is provided. For example, a tabular description of genes and their interactions makes it difficult to follow the paths of gene activation. By depicting each gene as a node and its interactions as links between nodes, we gain an immediate impression of paths of activation. This is achieved by removing superfluous information such as gene name, etc., and by providing an intuitive spatial representation.

Furthermore, numerous views of the same system can be generated by changing what the nodes and links represent. This flexibility allows researchers to home in on the pertinent features of the system.

3. Also known as: Graph

4. Motivation. *Static System:* Characterizing the nature of co-author relationships in a given academic field is difficult when browsing a textual description of their collaborations. Visualizing authors as nodes and co-authorship as links renders large amounts of information visually accessible.

Dynamic System: When investigating state spaces, an activation diagram makes it difficult to determine properties such as cyclic behaviour, length of cycles, etc. By removing information such as individual entity states for each

time step, and visualizing unique states as nodes and the transitions between them as links, such properties are immediately apparent.

5. Applicability. The Network Diagram pattern can be applied whenever you can define entities and their relationships. In many physical systems, there is a an obvious way to define the system as entities and relationships. For example, regulations between genes, interactions between people, etc. However, the Network Diagram has wider applicability than these corporal mappings. More abstract properties of a system, such as state space transitions, can also benefit from the Network Diagram visualization.

6. Structure. (Intentionally left blank).

7. Participants. Node, Links, Layout, Manipulation

8. Collaborations. Entities are represented as nodes, with their relationships represented as links between these nodes. The nodes are placed in a spatial representation defined by the layout. Manipulation provides a means of interacting with all aspects of the Network Diagram (such as layout, node states, etc.).

9. Consequences. The Network Diagram focuses on the entities and relationships of interest.

- it focuses on the relationships of interest
- layout allows different views, possibly different interpretations
- ease of view / analysis comes from reduction of information
- generality means one has to choose entities / relationships
  - this is a strength and a weakness
    - strength: flexibility
    - weakness: lack of guidance of what should be nodes and links

10. Implementation.

- there are lots of different layouts / methods of manipulation, which can largely influence usefulness
- giving user choice over layout (and manipulation) is a useful approach
- a random layout is the simplest to implement, but is generally unsuitable for (e.g.) 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).

11. 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]]
```

12. Known uses. Boolean network visualization and design, social network visualization, neural network visualization and design.

13. Related patterns. Discrete State-Space Trajectory Diagram, Activation Diagram.

Appendix B – Synchronous System State Update

Submitted by: Jim Hanan, James Watson, Daniel Bradley, John Hawkins, Dharma Dassanayake, Scott Heckbert, Andrew Hockey.

1a. Pattern name: Synchronous System State Update

1b. Classification: Model (Behavioural)

2. Intent. In many complex systems simulations, all objects that interact in the model should be updated synchronously, reflecting the parallel nature of the abstracted system. This pattern describes a method of implementing synchronous updating by creating a new set of states from current states.

3. Also known as: Synchronous update.

4. Motivation. Given the Boolean network model shown in Figure 1, you want to analyze system behaviour when all node interactions are taken into account before states are changed.

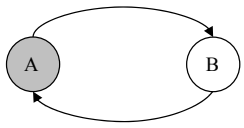

Figure 1: Simple Boolean network with node A activated.

Using the Synchronous State Update pattern, you would create and assign values to a new set of states (one for each node) according to all the interactions between nodes in the current state (see Figure 2). The new set of states then becomes the current set.

By determining the new set of states for each node in the network according to all interactions among the old states, synchronous updating is achieved.

This is different to asynchronous updating, where all nodes and their interactions are not considered before updating. From the initial conditions of Figure 1, asynchronous updating could

Node	Current State
A	active
B	inactive

Node	New State
A	inactive
B	active

Figure 2: For node A, no activation is received from its inputs, so it becomes inactive. Node B is activated by node A, so it becomes active. This process of swapping states can occur indefinitely.

- first update A according to B's state without considering A's input to B, so A and B become inactive forever
- first update B according to A's state without considering B's input to A, so A and B become active forever

5. Applicability. The Synchronous System State Update is useful whenever all components of a complex system are concurrently interacting on the same timescale. For example, this is often the case when you are interested in the influence of interactions between entities, and thus abstract away the time differences between these interactions and consider them all at the same point in time. Examples include some Boolean models of gene regulation, social networks, and L-systems plant development. It is best suited to processes with discrete time-steps, though continuous systems can be approximated with small time-steps.

6. Structure. (Intentionally left blank).

7. Participants. Entities, Relationships, Current states, Next states

8. Collaborations. Each entity has a corresponding current state. Next states are created according to the current states of the entities and the relationships between them. The next states become the current states at the end of updating.

9. Consequences.

- Since interactions are only considered at each time-step, this pattern abstracts away the actual time events occur.
  - For non-stochastic simulations, this causes the system to exhibit deterministic behaviour, which is often desirable for system-wide analysis.
  - If the system being modeled has asynchrony, this information is lost, unless modeled explicitly by incorporating delays and fine scale time steps.
  - May lose variability potentially created by asynchronous updating.
- Updating slows as the number of entities increases.
- Approximating continuous processes requires small time-steps, which requires longer processing times.

10. Implementation.

- Small time-step sizes can be used to approximate continuous time.
- Pointers to the new/current states lists can be manipulated to improve performance, e.g.:

```
free(current)
current = next
next = new (un-initialized) set of states
```

- Depending on the model's architecture, each entity (such as an agent) can handle its own updating and state, or a separate global process can maintain state lists and iterate through all entities. For very large numbers of entities, the global algorithm approach can be more efficient if simple representations are used for entities (e.g., a bit array for Boolean networks) instead of object instantiations.

11. Sample code.

```
C = current states of entities, indexed by node number
N = next states of entities, indexed by node number
t = current time-step

t = 1
while (t <= time-steps of interest)
    for i = 1 to number of nodes in the system
        N[i] = state based on inputs to C[i]

    C = N
    t = t + 1
```

12. Known uses. Network update [9], Game loop, L-system derivation process [8]

13. Related patterns. Asynchronous System State Update should be used when asynchronous updating is important to the model. Sequential State Update should be used when the ordering of the updates is inherent in the system being modeled (e.g., Chomsky grammars).

Appendix C – Discrete State-Space Trajectory Diagram

Submitted by: Daniel Bradley, John Hawkins, James Watson, Dharma Dassanayake, Scott Heckbert, Andrew Hockey.

1a. Pattern name: Discrete State-Space Trajectory Diagram

1b. Classification: Visualization (Dynamics, State Space, Function, Macro Behaviours)

2. Intent. The pattern describes a technique for visualizing a system's dynamic qualities as a discrete graph. It allows for the identification of the number, size and structure of attractors, as well as the number and length of transients leading to these attractors.

By removing the specific details about the states the system takes, a state space diagram shows an abstract map of a systems dynamics. The removal of details irrelevant to describing dynamics eases the task of analysis by visual inspection.

3. Also known as. Wuensche Diagram, Basin of attraction field

4. Motivation. In all complex systems simulations at each moment the state of the system is described by a set of variables. As the system is updated over time these variables undergo changes that are influenced by the previous state of the entire system. System dynamics can be viewed as tabular data depicting the changes in variables over time. However, it is hard to analyze system dynamics just looking at the changes in these variables, as causal relationships between variables are not readily apparent.

By removing all the details about the actual state and the actual temporal information, we can view the dynamics as a graph with nodes describing states and links describing transitions. For instance software applications can have a large number of states. Problems occur when software applications reach uncommon or unanticipated states. Being able to visualize the entire state space, and quickly comprehend the paths leading to any particular state, allows more targeted analysis. Common states can be thoroughly tested, uncommon states can be identified and artificially induced.

State space diagrams allow for numerous insights into system behaviour, in particular some states of the system can be shown to be unreachable, while others are unavoidable.

5. Applicability. Any situation in which you have a model or system which changes state over time and you want to examine the abstract dynamical qualities of these changes. For example, social network theory, gene regulatory networks, urban and agricultural water usage, and concept maps in cognition and language modeling.

6. Structure. See Figure 3.

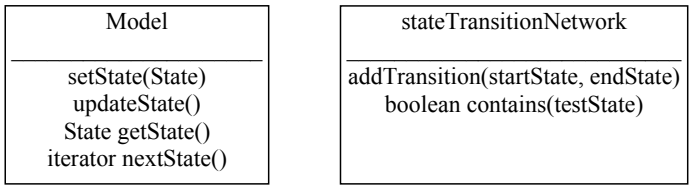

Figure 3:

7. Participants. See Figure 4.

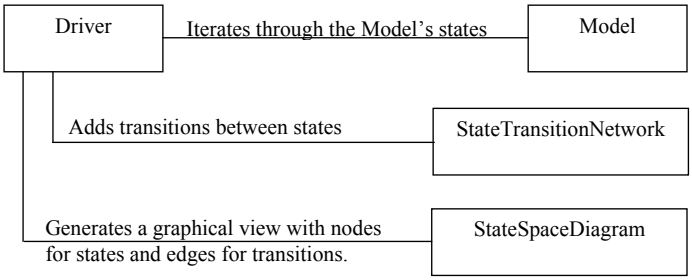

Figure 4:

8. Collaborations. Can be used with Activation Diagram to visualize the details of a given state.

9. Consequences.

- The pattern supports it's objectives by removing details unrelated to the system dynamics, then examining only states and transitions.
- There are several tradeoffs in applying this pattern. Firstly it works only for discrete state-spaces or discretely quantised continuous spaces. It is not tractable to show entire state space dynamics for systems with a large number of possible states. However, one can perform random sampling of the system and view partial state space diagrams.
- For large diagrams the layout of nodes and connections is critical for comprehending the structure.
- There can be a very large number of possible states for a given system. In these cases, visualization of the entire state space at once can be difficult.

10. Implementation.

- There are two major methods of implementing the data structure beneath the transition network. It can be written as an adjacency list where each node points to a linked list of connections. Secondly it may be stored as a complete matrix of NxN dimensions (where N is the number of states). Non-zero entries in the matrix indicate the existence of a connection between nodes.
- With an appropriate layout, e.g., Wuensche, the whole state space can be viewed at once. For large number of states, the diagram can be made easier to read by trimming off the most outlying states from transients.
- Ghosting/colour changes allow the user to examine changes to system dynamics when some underlying aspect of the system is changed.
- It is important to recognise that the graphical model is abstract only. Any spatial information contained in the underlying model is being removed.

11. Sample code. A complete state space transition network is produced thus:

```
-------------------------------------------------------------
initialiseStateTransitionNetwork(noOfStates)
For each currentState in Model
    model.setState(currentState)
    model.updateState()
    nextState = model.getState()
    StateTransitionNetwork.addTransition(currentState,nextState)
End for
-------------------------------------------------------------
```

Similarly to explore a particular trajectory through the state space:

```
-------------------------------------------------------------
initialiseStateTransitionNetwork(initialTrajectoryState)
currentState = initialTrajectoryState
nextState = model.nextState(currentState)
StateTransitionNetwork.addTransition(currentState, nextState)

While Not StateTransitionNetwork.contains(nextState)
    currentState = nextState
    model.setState(currentState)
    model.updateState()
    nextState = model.getState()
    StateTransitionNetwork.addTransition(currentState, nextState)
End for
-------------------------------------------------------------
```

These code snippets cover the generation of the data structure representing the state space diagram. Once this has been generated there are a number of layout algorithms for creating the actual visualization.

12. Known uses. Boolean models of Gene networks, NK landscape problems, Visualizing basins of attraction in random Boolean networks.

13. Related patterns. Network Diagram, Neural Networks, Activation Diagram, Hypercube

Table 1: Complex systems patterns format

Section	Description
Submitted by.	Includes name, affiliation and date.
1a. Pattern name.	The pattern's name conveys the essence of the pattern succinctly. A good name is vital, because it will become part of the design vocabulary.
1b. Classification.	The pattern's classification should reflect it's scope and use.
2. Intent.	A short statement that answers the following questions: What does the pattern do? What is its rationale and intent? What particular issue or problem does it address?
3. Also known as.	Other well-known names for the pattern, if any.
4. Motivation.	A scenario that illustrates a problem and how the pattern solves the problem. The scenario aids understanding the more abstract description of the pattern that follows.
5. Applicability.	What are the situations in which the Complex Systems pattern can be applied? What are examples of Complex Systems problems that the pattern can address? How can you recognize these situations?
6. Structure.	A graphical representation of the components of the pattern if possible (leave blank if not applicable).
7. Participants.	The components of the pattern and their responsibilities (leave blank if not applicable).
8. Collaborations.	How the participants collaborate to carry out their responsibilities (leave blank if not applicable).
9. Consequences.	How does the pattern support its objectives? What are the trade-offs and results of using the pattern?
10. Implementation.	What pitfalls, hints or techniques should one be aware of when implementing the pattern? Are there platform-specific issues?
11. Sample code.	Code fragments or pseudo-code that illustrate how you might implement the pattern.
12. Known uses.	Examples of the pattern found in real systems. Preferably at least two examples from different domains. Give references.
13. Related patterns.	Which patterns are closely related to this one? What are the important differences? With which other patterns should this one be used?