

Gero JS and Brazier FMT (eds) (2002) *Agents in Design 2002*  
Key Centre of Design Computing and Cognition, University of Sydney, pp. 115-132.

## INTERACTION AND EXPERIENCE

### *Situated Agents and Sketching*

GREGORY J SMITH and JOHN S GERO  
*Key Centre of Design Computing and Cognition*  
*University of Sydney Australia*

**Abstract.** Cognitive studies of designers often involve sketching, but studies using artificial intelligence often apply a search paradigm. Sketching is an interaction between an agent and an environment. Perception influences how shape rules are applied, and the application of the rules influences future perception. One motivation behind our work is to computationally model an autonomous design agent that is based on interaction; an agent that can interact with an external representation of a developing design. We describe an interactive model of an agent. In our model an agent has six parts: sensors, perceptors, a conceptor, an action activator, a hypothesiser, and effectors. In this paper we describe our model and a trial implementation involving learning, perception and action activation

### 1. Introduction

When reading contemporary cognitive work on designers, one notices how often sketching appears directly or indirectly in discussions. Reading work on artificial intelligence, on the other hand, one often finds that a search paradigm is applied. One characteristic of sketching is that many shapes in two or three dimensions can be decomposed into constituent parts in an indefinite number of ways (Stiny 2001). Consider a design agent, human or artificial, that generates new shapes by repeatedly locating a candidate shape and then applying knowledge to change it. As computation is abstractly equivalent to a Turing machine we could consider the application of the knowledge to be abstractly equivalent to applying rules from a grammar. With a search paradigm, “locating a candidate shape” is comparing current symbols for shapes against the antecedent sides of the rules.

Classically, the computational metaphor of design achieves design goals through internal reasoning with inference rules over models in a

suitable logic or language. Such reasoning, known collectively as problem solving, is presented as search, planning or exploration (Gero 1998). In each case this reduces to an explicit search through a problem space, with designing viewed as a search through a space of candidate designs. Uninformed search techniques are exponentially complex in time or space or both (Russell and Norvig 1995), and non-optimal uninformed search eventually suffers the same fate, albeit after traversing a larger space. Further, decidability problems with planning (Chapman 1987) limit the extent to which the effects of design actions can be predicted and accounted for. The frame problem would also seem to indicate that, even without planning decidability problems, an agent would be limited in which effects of design action it could take account of. Thus making search feasible for conceptual design tasks without requiring that an agent make guesses requires strong heuristics. Additionally, the abstractness and incompleteness that are inherent in conceptual designing (Gero 1998) would restrict the use of such heuristics even if we could show how an agent could autonomously acquire them.

Researchers into cognitive aspects of humans designing have described paradigms of designing other than problem solving ones, instead employing various interactive mechanisms (Schon and Wiggins 1992; Suwa et al. 1992). One only needs to think of a human scribbling on a piece of paper to see the implications of interaction on computational models of designing. The nature of sketched shapes, however, means that shape rules apply indeterminately (Stiny 2001) and the percepts resulting from “locating a candidate shape” cannot necessarily be predicted in advance. So the search model would not be the equivalent of an agent interacting with a sketch. Perception influences how shape rules are applied, and the application of the rules influences future perception.

One motivation behind our work is to computationally model an autonomous design agent that is based on interaction; an agent that can interact with an external representation of a developing design. In this paper we consider an agent model that is based on knowledge of how an agent could interpret what it perceives from its environment, and how the agent could act on that environment so as to achieve design goals. Design goals in the agent form expectations that bias what is perceived, and the agent learns these biases from its interactive experiences.

## **2. On Interaction: A Situated Design Agent**

### **2.1 INTERACTIVE WITH THE ENVIRONMENT**

If a designer, artificial or human, is an agent then a sketch is an external representation. An external representation is an open system with which

an agent interacts as a distinct system over which it is not in direct control. The agent acts on the external representation but does not directly control it in the same way that it controls its beliefs. Using Beer's (1995) model of agency as a coupled dynamical system formed by an agent interacting with its environment, we could describe<sup>1</sup> the agent and environment using a pair of difference equations:

$$X_A(k+1) = A(X_A(k), S(X_E(k)), U_A)$$

$$X_E(k+1) = E(X_E(k), M(X_A(k)), U_E)$$

Here  $E$  is the environment,  $A$  is the agent,  $S$  is a sensory function,  $M$  is a motor function,  $X$  is a vector of state variables,  $U$  is a vector of parameters, and  $k$  is a number  $1.. \infty$ . Each of these dynamical systems continuously deforms the flow of the other, altering each other's subsequent trajectories. The point is that the agent and environment mutually perturb each other rather than having one control the other.

If our model is of a situated design agent, what is a situation? For the purposes of this work we will consider the state  $[X_A(k), X_E(k)]$  of the system to be the current situation  $S_0$ . The state of the environment,  $X_E(k)$ , is not necessarily the same as is perceived by the agent,  $S(X_E(k))$ . Indeed, it is not available to the agent. A part of the current situation  $S_0$  will be in the agent, a part will be in the environment, and a part will result from their interaction. So, if  $f_i$  is a function for entity  $i$  and  $S_i(k)$  is a situation at time  $k$  for entity  $i$ , then for the agent (and similarly for the environment),

$$f_A(S_0(k)) = S_A(k) \quad X_A(k)$$

In our model an agent has six parts: sensors, perceptors, a conceptor, a hypothesiser, an action activator, and effectors. This model builds on those presented in Gero and Fujii (1999) and Smith and Gero (2001). Figure 1 shows it in overview.

Sensors provide uninterpreted sense-data from the environment and may be biased by perceptors. Similarly, effectors provide effect-data to the environment for selected actions.

The agent uses sense-data and effect-data to construct  $S_A$ , its interpretation of the external representation. This is handled by the

---

<sup>1</sup> Functions presented in this section are intended to be descriptive only. They may not, for example, be directly computable.

perceptors and the action activator. Together these interpret objects attended in the environment.

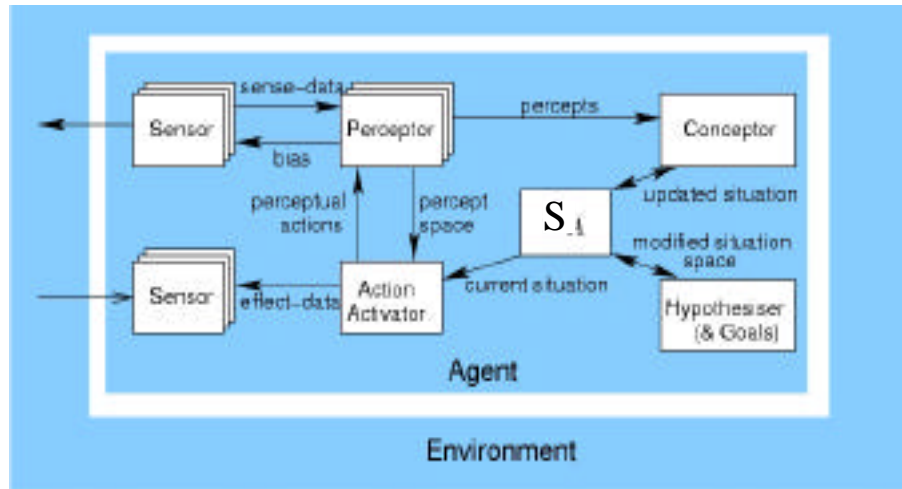


Figure 1. Overview diagram of the agent.

As we are interested in sketching we can describe this in terms of a shape grammar. To fully appreciate a shape grammar, says Stiny (2001), you need to think of its application as two processes. If the agent wishes to apply a shape rule  $A \rightarrow B$  to an attended previously sketched object  $O$ , the first task is to find a transformation that picks out some part of  $O$  that looks like  $A$ . This first process we model as perception. If it is successful, the agent can replace the part of  $O$  that looked like  $A$  with something that looks like  $B$ . This second process is controlled by the action activator, and we will return to it shortly.

Matching symbols in a symbol system uses an identity relation (Stiny 2001). Shapes exist in two or more dimensions, however, and every nonempty shape has infinitely many parts. A good example that Stiny gives is that an upper case “K” shape contains indefinitely many lower case “k” shapes. Thus perception in the agent could be described as a set of embedding relations. The particular relation that is employed at time  $k$  depends on the utility of the situation  $S_A(k)$  for the agent. The action activator controls how the embedding relations are employed and, barring any biases from the conceptor, continues the process of interpretation of the external representation by employing these relations.

Percepts as the targets of embedding relations are like the deictic references described by Agre (1997). A percept may be “the square that I am now looking at”. It is not the square itself. To change that square means acting on “the square that I am now looking at” in the external

representation. Interaction as a process is fundamental. Schon and Wiggins (1992) likely understood this when they described their subject Petra's design session as a series of move experiments – of making a move and perceiving the result. In each move experiment, Petra perceived some things and not others, and then took actions accordingly.

2.2 THE CONCEPTOR AND HYPOTHESISER

If the agent has a set of shape rules, how does it decide which to apply and how? The answer is that the action activator is biased by concepts that are constructed according to design goals of the agent and beliefs learned from previous experiences. We can think of this as being a game played between a conceptor and a hypothesiser.

We can make the following discussions more solid without restricting ourselves to a particular implementation. As Petri nets accept context sensitive languages (Peterson 1981) we can represent shape grammar rules as a Petri net. If each place in the net represents a percept or a concept (where concepts are abstractions of percepts) then transitions represent the rules. Similarly, the goals of the agent can be used to construct a Petri net of expectations. Expectations are represented as a percept or a concept but the transitions would represent relations. Figure 2 illustrates this using a very simple example.

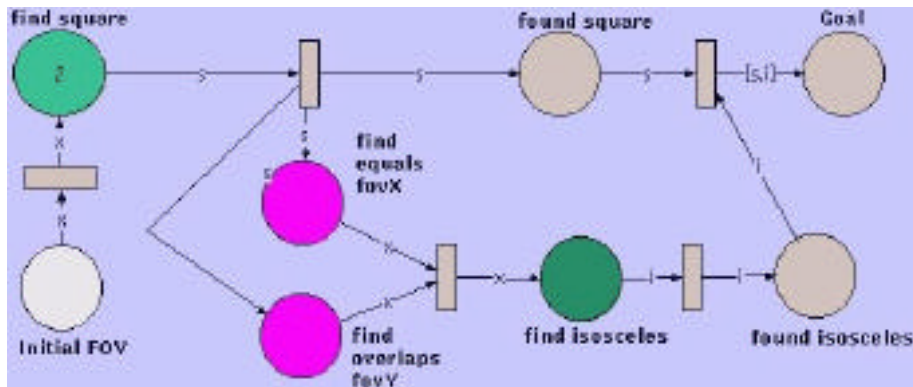


Figure 2. A very simple example expectation based on simple percepts and the rectangle algebra (Balbiani 1999). The expectation is of a square that overlaps an isosceles in the Y direction, and equals in the X direction. Note that the number 2 in the “find square” place is a token drawn by the Petri net simulator that was used to produce this figure. FOV/fov means field of vision.

The hypothesiser and conceptor operate on the agent's understanding of the situation,  $S_A(k)$  at move  $k$  in the game, as was interpreted through the efforts of the perceptor and action activator.  $S_A(k)$  also includes the

expectations of the agent. For the conceptor, one move is an update of situation  $S_A(k+1)$  from percepts constructed by the perceptors and action activator at step  $k$ , followed by selecting biases on the action activator for a new drawing action. A Petri net with no marking represents the shapes rules and expectations of the agent, and the current situation  $S_A(k)$  is the marking of the net at time  $k$ . The conceptor “plays the game” by looking at the current marking on the Petri net, finding which places are reachable from that marking, and selecting actions so as to try and effect that transition. This may mean looking on the external representation for an object, or it may mean drawing a new object on the external representation.

The hypothesiser constructs and alters the Petri net using its goals and beliefs, and judges the consequences the conceptor’s interpretation against the goals. This continues until an equilibrium is achieved (Sandholm 1999), when both the hypothesiser and conceptor agree that their utility cannot improve further without the other making a move. If nothing unexpected is discovered by the perceptors then the conceptor may continue until it can progress no further. If something interesting emerges then the hypothesiser may change the expectations.

The conceptor, then, uses expectations to bias perception. This happens by changing the space within which similarities to the percepts are measured. What the agent expects to see influences what it sees, and what it sees influences what in future it will expect.

At equilibrium a design will be depicted on the external representation that may or may not satisfy all of the design goals. That is, the agent may not be capable of satisfying all of the design goals so the process ends when no further progress can be made.

### 2.3 THE AGENT AS AN INTERACTION MACHINE

In a search model of designing, actions are bound to states as in the Turing model of computation. Our model is of a situated agent that designs by interacting with an external representation. This is better described as an interaction machine. An interaction machine has actions that are bound to input streams rather than states. Wegner (1997) shows that interaction machines are not directly expressible by or reducible to Turing machines. Streams do not have a last element, can be dynamically extended by unpredictable adversaries, and future inputs can depend on current outputs. That is, for shape grammars, the computational core of the interaction machine operates on rules, but those rules depend on a dynamic input stream rather than a static input tape. This means that the complexity of the interacting system cannot be determined by looking at the rules alone. Just as a current output from the interaction machine

effects future symbols in the input stream, and so effects future outputs, so perception changes how the rules are applied, and so effect future perceptions (Stiny 2001).

### **3. On Experience and Learning**

The model as presented so far would be sufficient for a non-autonomous computational design agent. For it to be applicable to an autonomous computational design agent, we need to be able to show that the agent can adapt. Similarly, for the model to abstractly apply to (some aspects of) human designing requires that we ground the representations of the agent in its experiences. Work by Clancey (1999), Bickhard (Bickhard and Terveen 1995), and others has shown that representations should be in the first person and grounded in the experiences of an autonomous agent, not encoded. For a design agent we aim to be able to attribute a design to an agent and not directly to encodings by its programmer. Achieving this means developing a model in which learning and interaction are fundamental. The adaptive nature inherent in such an agent allows it to be exposed to unique environments not previously encountered. Designing, by definition, is concerned with new environments. In short, we need to consider the importance of learning.

In order to test these ideas, a trial implementation examined whether the kinds of knowledge necessary to enable these interactions can be learned autonomously. The task selected for this purpose was learning to perceive and alter shapes depicted on an external representation. This task is suitable as it fits with the theory described, is interactive, involves an external representation, and requires autonomous online learning. This is, necessarily, not the complete implementation of a design agent but it does provide a platform for the investigation and illustration of our ideas. Particular representations are described for the purposes of the implementation, but the ideas should translate to other similar tasks. Details should be taken as indicative of the model rather than necessary.

The task implemented involves an external representation from which the agent tries to interpret depictions of shapes composed from line segments. The agent must learn how to interpret these shapes based on previous experiences of similar shapes, and must control this perception process by learning to take perceptual actions. The agent must also learn how to take overt actions so as to alter the external representation to make it more like expectations constructed according to its goals and prior experiences.

Learning occurs online in three phases. Firstly, learn the spaces in which interaction will be expressed. Secondly, learn when to take actions

in terms of utility. Thirdly, learn abstractions of these interactions, to act as conception biases on the other phases.

As learning is online, the three phases operate concurrently, excepting that a later phase will not learn successfully until the earlier phases have enough structure with which to represent.

### 3.1 LEARNING A STATE MODEL

The first thing needed, then, is a way for the agent to compare what it has so far interpreted with what it has experienced previously. This is handled by the capabilities. By capability we mean that part of perception that concerns perceiving a particular class of percept as well as a corresponding part of action activation. The reason for describing perception as capabilities is that perception and action are learned together. Each capability possesses a state model that can determine the similarity of itself to other percepts and expectations, as well as simple actions to draw itself.

A static set of predefined features and a strict geometric metrical model of similarity is undesirable as metrical models (such as the Euclidean metric) do not easily capture asymmetric relations. Perceptions of similarity do not always satisfy geometrical distance axioms like the triangle inequality (Tversky 1977, Santini and Jain 1999). For example, we may believe that a house cat may be like a lion and that a house cat may be like a parrot, but that a lion is not like a parrot. Or, that a particular variant three-legged cat is like a prototypical cat, but that the prototype is less like the variant. Tversky's feature contrast measure of similarity (Tversky 1977) presents set-theoretic analogs to metric space axioms. If  $x$  and  $y$  are objects then let  $similar(x, y)$  be the similarity of  $x$  to  $y$ . Tversky's representation theorem says that  $similar(x, y)$  depends on both the features that are common and the features that each has that the other does not. Tversky's model has been criticised for its reliance on a Boolean features, and an a priori fixed set of atomic features is undesirable. In the fuzzy contrast model (Santini and Jain 1999), features are no longer Boolean but are fuzzy. Rather than counting Boolean features, a fuzzy membership function  $\mu_i$  is used for each feature  $I..F$ . The behaviour of the fuzzy contrast model varies with the fuzzy membership functions used, the values of scaling factors  $\alpha$ ,  $\beta$ , and  $\gamma$ , and the fuzzy intersections and differences of each  $\mu_i(x)$  and  $\mu_i(y)$ .

Applying the fuzzy contrast model helps solve one problem, and the asymmetry and context dependence let the current situation influence how perceived objects are interpreted and how design decisions are taken. The other problem is to avoid a fixed, all encompassing set of atomic

features across all capabilities. The machine learning task of an autonomous agent finding features from unclassified feature spaces is one of online clustering. Online machine learning tasks distinguish themselves from offline ones in that training instances are presented one at a time as new experiences are encountered, rather than being collected and presented all at once. Online learning is desirable as such an agent would continue to learn as it designs.

The algorithm chosen for the implementation is adaptive vector quantization (AVQ), a type of unsupervised, online competitive learning. Details of AVQ can be found in (Dickerson 1996). It runs on  $N$  spaces and learns hyper-elliptical clusters in terms of a centre and co-variances. From these, fuzzy sets can be constructed for  $\mu_i$ .

Each capability does its own similarity measurements with respect to current expectations as well as doing its own clustering. Using the competitive AVQ approach, each cluster of the same feature space competes for the current experience. The winner shifts its centre and variances a small amount in that direction, with “small amount” being a learning rate that decays. Each new capability is initialised with no clusters. If no cluster is close to the experience and the detected error means that it should be, then a new cluster centre is started. The resulting cluster centres and variances are mapped onto fuzzy sets, and these fuzzy sets are used in the fuzzy contrast model. This means that we can generate features autonomously and online, without specifying in advance the number of clusters required, and learn incrementally. Containing the clustering within capabilities, rather than clustering globally across all instances, means that different capabilities learn different features. It is these different features that assist with the biasing of perception by expectations.

The origin of the feature space we used for this implementation is a set of histograms described by Huet and Hancock (1999). These are counts computed over all sets of line segments on the object that is currently attended to in the external representation. In order for histograms to be constructed, these ratios are discretised using equal width binning. Counts are then made over all pairs of line segments, counting in the fashion of a cumulative histogram. Clustering on a cumulative histogram is similar to discretising a multidimensional space into subspaces, each of which could be considered a feature.

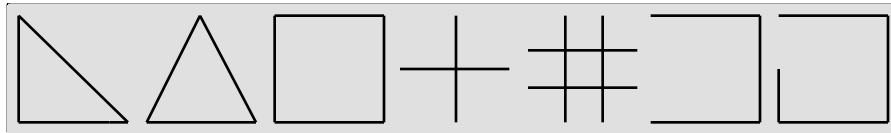
### 3.2 AN EXAMPLE OF LEARNED SIMILARITY PERCEPTION

We trained the learner on randomly generated sets of shapes; common characteristics were parametrically templated from an XML file and then randomly (affine) transformed. The shapes are presented to the known

capabilities as they are generated, and the capability with the largest similarity, if above a threshold, gets to use the instance to improve itself. If below a threshold, a base capability gets generated. For the winning capability, each time that the fuzzy match over each feature space for a shape, with itself as the expectation, is below a threshold a new cluster is started.

Based on this, a test implementation was built. We call the sense-data from the currently attended object the *subject*, and call each capability a *referent*. For this test we set  $\alpha$ ,  $\beta$ , and  $\gamma$  such that features that are distinctive in the subject decrease similarity more than features that are distinctive in a referent. The effect is of the expected capability forming a primary expectation and a referent capability forming a secondary expectation. The expectation defines the space that is used to compare the subject to each referent. In effect the fuzzy sets of the expectation provide the features that are used to judge similarity, and the judgment is done using the fuzzy contrast model.

For the purposes of this demonstration we trained state models on capabilities corresponding to: right angles, isosceles triangles, rectangles, plus symbols and hash symbols. We then tested using a similar set of subjects, including: right angle, isosceles triangle, rectangle, part missing rectangle and part short rectangle. These are illustrated in Figure 3.



*Figure 3.* Shapes such as these are parametrically templated in an XML file and are used to generate randomly affine transformed versions of the shapes for training and testing. The training shapes shown are labeled as: right angle, isosceles, rectangle, plus, hash, part missing rectangle and part short rectangle.

If we determine similarities of subjects and referents with an expectation<sup>2</sup> of “rightangle1”, for example, we get the similarity values shown in the centre column of Table 1. The fuzzy membership values all lie between 0 and 1, but the similarity measures are not normalised. This is deliberate because the measure is asymmetrical – it is neither a metric nor a fuzzy membership function. Given an expectation, we can say that

---

<sup>2</sup> For debugging we need a labeling of capabilities scheme that has some meaning to us as outside observers of the agent. We adopt the label of the training shape that first caused a capability to be construction, appended with an integer (e.g. “isosceles1” and “isosceles2”). These labels play no part in classification or debugging, and no relation based on labels is implied.

a larger value corresponds to an increasing similarity but it is not the case that the values lie between 0 and 1. Also, because the clusters act like prototypes, some clusters will be centred on mean positions that concrete shapes cannot take. Thus the distance from a cluster centre to the nearest possible shape value will not always be zero.

TABLE 1. Similarity of subject "isosceles" to referent capabilities with expectations of "rightangle1" and "isosceles1"

Referent	Expectation: "rightangle1"	Expectation: "isosceles1"
plus1	0.12	0.19
isosceles1	0.56	0.76
isosceles2	0.77	0.82
rightangle1	0.26	0.86
Square1	-0.2	-0.25
rectangle1	-0.23	-0.18

With an expectation of "rightangle1", the test subject "isosceles" is seen as being similar to base capability "isosceles2" and slightly less similar to base capability "rightangle1". The isosceles subject is more similar to a "isosceles2" referent than the "isosceles1" referent, but because the shapes are asymmetrical in these feature spaces (that is,  $\mu(\text{isosceles2}) - \mu(\text{isosceles1}) \neq \mu(\text{isosceles1}) - \mu(\text{isosceles2})$ ), then the similarity values for "isosceles1" are less than for "isosceles2". When the expectation is changed to "isosceles1", the similarity to "rightangle1" increases markedly. Similar results are found with the other subjects and expectations. For example, a part missing rectangle is seen as more similar to a rectangle when the expectation is not of a rectangle because the differences are not highlighted by the features constructed for rectangles (if we look for triangles, we are less likely to notice faults with rectangles).

The similarity values depend on the scaling factors, the fuzzy membership functions, the training instances (the agents' experiences) used to learn each clustering, and thresholds in the learner. Setting higher thresholds results in more capabilities containing fewer clusters (all else being equal), and vice-versa. Perceptions therefore vary with changes to these parameters as well as with previous experiences. That is, perceptual interpretations of depicted shapes vary according to the situation and the agent's beliefs.

### 3.3 IMPLEMENTING ACTIONS

Interacting is not only interpreting what has already been depicted, it also involves selecting and activating actions. We consider in this section those low level perceptual actions that allow capabilities to control interaction. In order to find percepts for the capabilities to determine the state by, the agent needs to know (for the capability with the greatest similarity to the current expectation) what kind of line segment to find next or, failing that, what kind of line segment to draw next. Thus for this implementation, two kinds of action are considered: for perception it is the next thing to look for and for drawing actions it is the next thing to draw.

We can restrict the action space by specifying the line segment relative to the most recently drawn or perceived line segment but this still requires finding a relative location, orientation and size. Each of these is a continuous quantity, so the number of alternatives is infinite. We could keep a finite number of actions by adopting the same discretisations as used in the state model but the number of possible actions would still be massive.

For example, consider expressing action selection in terms of utility to the agent. Finding a mapping  $S \times A$ , where  $S$  is a space of states and  $A$  a space of actions, is a reinforcement learning problem that performs well if action and state spaces are discrete and not too large. Unfortunately, however, in our case both are continuous. The way we solve this problem is by taking advantage of the distributed learning afforded by the capabilities. We let each capability learn its own action biases, and then learn the utility of selecting which base capability to apply its biases.

For a single capability, then, we need a function from the current state (the difference between what a capability has perceived and what it would have liked to perceive) onto biases of the next draw or interpret action. If a linear (or at least a lower order polynomial) function were adequate then we could let  $X$  be the state model value on each of the input histogram dimensions, let  $Y$  be the action biases for this base capability, and use an incremental least mean squares (LMS) learner to learn the model  $X \rightarrow Y$ . One such algorithm is described in Jang et al. (1997) using vector equations that update the action model according to the product of an adaption gain and the error produced by the old model.

Such an approach can be improved to allow for nonlinearities by mapping onto a higher dimensional space. Deficiencies in the state for a capability, with respect to the currently attended object, will relate to how that capability clusters its feature space. By using those clusters as Gaussian basis functions we can map into a space in which the

incremental LMS learner can be used. A weighted sum of the radial basis values is then found, with the weights learned using the incremental LMS algorithm.

With action biases provided by base capabilities, we can determine utility in terms of deciding which capability should be allowed to provide action biasing next. We implemented this using a Java reinforcement learning API<sup>3</sup>. The set of actions is discrete and includes actions such as *activate expectation* (activate the biases of the current expectation), *change expectation, reverse direction* (reverse the current scan direction), and so on. The state for the reinforcement learning model includes values such as the current actual similarity to the current expectation, the imagined similarity if the expected capability could complete its actions, and the largest imagined similarity of other capabilities. The learner gets rewarded when the similarity measure of the currently attended object crosses a threshold, at which time a successful interpretation is deemed to have occurred. This reward is then propagated to assign credit to actions taken using temporal differences.

As with the other phases of learning, this action utility learning is incremental and online. As is common with reinforcement learners, selection of actions is not done in a greedy fashion as the learner needs to explore new regions of its action space if it is to improve. In our trial implementation we used a stochastic action selector: the higher the expected reward of an action in a state, the higher the probability that the action will be selected in that state.

### 3.4 AN EXAMPLE IMPLEMENTATION

Consider, for example, two tasks. In the first the agent interprets the depiction of Figure 4(a) that is composed of shapes made from line segments. In the second the same agent draws a shape. These tasks as implemented involve an external representation from which the agent tries to interpret drawings composed of shapes made from line segments.

A trial consists of taking actions until a boundary limit number of steps is reached or until a reward state is reached. Each trial starts with a random expectation and focus of attention. For Figure 4(a), in the beginning the agent has no knowledge that one action is better than another, so actions are selected randomly. After a number of trials the agent starts to learn the idea of the relative utilities of the actions, resulting in some successful interpretations: *reset, change expectation to "plus1", activate expectation, ...*, resulting in an actual similarity value of

---

<sup>3</sup> By Pawel Cichosz at <http://tichy.ise.pw.edu.pl/~pawel/rl-java/index.html>

0.89, a reward of 1.0, and a shape that corresponds to the vertex set  $\{((100,200), (200,100)), ((100,100), (200,200))\}$ .

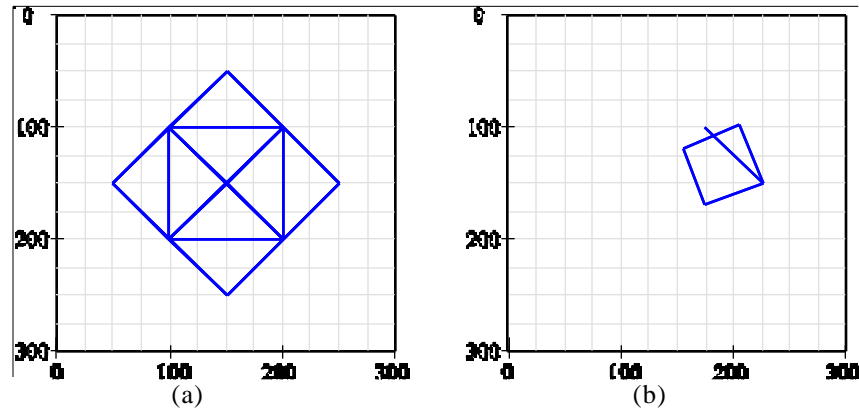


Figure 4. (a) Example depiction for action model interpretation testing, (b) example depiction drawn by agent.

The same action model can also alter the external representation. Figure 4(b) shows an example trial that was initialised with expectation “isosceles1” and a random initial location, and it illustrates how both interpreting and drawing work. Starting with the “isosceles1” expectation, it starts drawing and concurrently determines similarity values for the utility model's state values. In the example, after the first two lines are drawn the action selector changed expectations to “square1”. This was either because it determined that “square1” was more promising given the partial depiction or due to the stochastic nature of the reinforcement learner. After changing the expectation the trial continued until the square completed.

So, after interpreting a depiction based on expectations from the current configuration, the agent may decide to follow a match different from the current expectation. In this case the feature space would be recalculated and the state values recomputed. Alternatively, it may decide that the current configuration is satisfactory but that it wishes to change the depiction. Action and state models work together in reasoning and learning online, with similarities determining which action model to use and the impact of actions resulting in learning across all capabilities. The intent with respect to design tasks is to have the external representation, the partial interpretation of it and the expectations develop together.

The implementation to this point illustrates that the agent can learn to interact with its environment and that its abilities depend on its prior experiences. However, its performance is limited by its operation only at a perceptual level (based on a feature space constructed from qualitative

histograms) without any biases from conception. In order for it to be more flexible, robust and possess the constructs in which design goals could be expressed it needs concepts. The implementation is also a function of the particular set of sense-data used. Had a different set been used, different percepts and different judgments of similarity would result.

### 3.5 ABSTRACTING TO COMPOUND CAPABILITIES AND MACRO ACTIONS

We have described learning at a perceptual level, firstly as learning of the spaces in which interaction is expressed, and secondly as learning when to take actions in terms of utility. This would not of itself be sufficient for a capable agent as it needs concepts. These concepts are expressed as compound capabilities and are abstractions of perception level base capability interactions. They are referred to as compound capabilities as they bias the base capabilities.

There are different ways that these abstractions could be learned. For example, explanation based learning could be applied to sequences of actions that led to successful interpretations. By abstracting across different capabilities it could learn concepts like polygon, symmetry or closed curve, or abstracting sequences of capabilities could lead to compound capabilities that are relations.

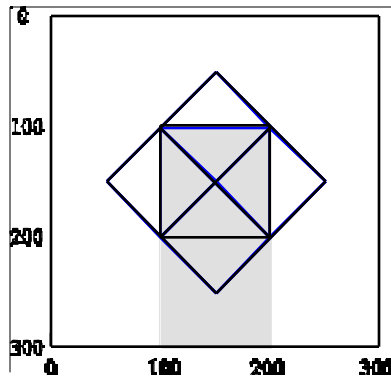
We have not implemented these abstraction mechanisms yet but a simple encoded example of a compound capability will illustrate their biasing of perception. Consider expressing the expectations of the agent as are represented abstractly by the Petri net of Figure 2.

This initial expectation is of a square and, there being no others, the agent looks for a square within the maximum field of view on the current depiction. Running this results in the square highlighted in Figure 5 being found. Given this, the compound capability triggers and biases perception so as to, in this case, restrict the field of view to that shown based on the expectation *square (equals, overlaps) isosceles*. The expectation is then changed to isosceles and the interpretation continues.

## 4. Conclusions

External representations have many advantages computationally and cognitively beyond their obvious use as an external memory. We agree with Soufi and Edmonds (1996) on the benefits of a "creative perception" that not only creates a model of the world but transforms an agent's view of the world. We differ in that, rather than variously transforming an internal representation, we maintain the external representation and transform the agent's perceptual mechanisms. Our agent starts with initial expectations constructed from design goals and

prior experiences. As the interpretation progresses, objects and relations not originally intended will be discovered, leading to revised expectations and further interpretations.



*Figure 5.* Example depiction with a located square highlighted. The restricted field of view within which to find an isosceles is shown thickened and results from applying the biases from the encoded rectangle algebra capability.

One implication of this work relates to what a suitably capable situated agent is likely to see in a particular depiction. If we start two cloned agents and expose them to different experiences then they may not interpret the same depiction at a later time in the same way. Alternatively, they may interpret the same depiction in what appears to be the same way to an external observer (a third person view of knowledge) but internally use different representations and reason differently (a first person representation).

The model of designing presented in this paper is based on interaction, non-encoding of knowledge and grounding of behaviours in experience. We discussed the theory and implementation of particular learning algorithms not because we believe those particular learning algorithms to be essential but because we believe that non-encoding of knowledge in an autonomous agent to be essential. No trial implementation can proceed using only abstract descriptions of representations and learning algorithms. We wish to be able to attribute any designs produced by an agent to that agent and not directly to implicit knowledge encoded by a programmer. The trial implementation was necessarily only of a small subtask of such an agent but it served to investigate and illustrate the ideas on interaction and learning.

### Acknowledgements

This work was supported in part by an Australian Postgraduate Award at the University of Sydney, Australia.

## References

- Agre, PE: 1997, *Computation and Human Experience*, Cambridge University Press, Cambridge, UK.
- Balbani, P, Condotta, J-F and del Cerro, LF: 1999, A new tractable subclass of the rectangle algebra, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pp. 442-447.
- Beer, RD: 1995, A dynamical systems perspective on agent-environment interaction, *Artificial Intelligence* **72**: 173-215.
- Bickhard, MH and Terveen, L: 1995, *Foundational Issues in Artificial Intelligence and Cognitive Science: Impasse and Solution*, North-Holland Elsevier Science, Amsterdam.
- Chapman, D: 1987, Planning for conjunctive goals, *Artificial Intelligence* **32**(3): 333-377.
- Clancey, WJ: 1999, *Conceptual Coordination: How the Mind Orders Experience in Time*, Lawrence Erlbaum, Mahwah, NJ.
- Dickerson, JA and Kosko, B: 1996, Fuzzy function approximation with ellipsoidal rules, *IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics* **26**(4): 542-560.
- Gero, JS: 1998, Conceptual designing as a sequence of situated acts, in I Smith (ed.), *Artificial Intelligence in Structural Engineering*, Springer, Berlin, pp. 165-177.
- Gero, JS and Fujii, H: 1999, A computational framework for concept formation for a situated design agent, in K. Hori and L. Candy (eds), *Proc Second International Workshop on Strategic Knowledge and Concept Formation*, Iwate Prefectural University, Iwate, Japan, pp. 59-71.
- Huet, B and Hancock, : 1999, Line pattern retrieval using relational histograms, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **21**(12): 1363-1370.
- Jang, J-S, Sun, C-T and Mizutani, E: 1997, *Neuro-fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*, Prentice-Hall, Upper Saddle River NJ.
- Peterson, JL: 1981, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, NJ.
- Russell, S and Norvig, P: 1995, *Artificial Intelligence: A Modern Approach*, Prentice Hall, Englewood Cliffs, NJ.
- Sandholm, TW: 1999, Distributed rational decision making, in G Weiss (ed.), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT Press, Cambridge, MA, pp. 201-258.
- Santini, S and Jain, R: 1999, Similarity measures, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **21**(9): 871-883.
- Schon, DA and Wiggins, D: 1992, Kinds of seeing and their functions in designing, *Design Studies* **13**(2): 135-156.
- Smith, GJ and Gero, JS: 2001, Emerging strategic knowledge while learning to interpret shapes, in Gero, JS and Hori (eds), K, *Strategic Knowledge and Concept Formation III*, Key Centre of Design Computing and Cognition, University of Sydney, pp. 69-86.
- Stiny, G: 2001, How to calculate with shapes, in EK Antonsson (ed.), *Formal Engineering Design Synthesis*, Cambridge University Press, Cambridge UK, pp. 20--63.
- Soufi, B and Edmonds, E: 1996, The cognitive basis of emergence: implications for design support, *Design Studies* **17**: 451-463.
- Suwa, M, Purcell, AT and Gero, JS: 1998, Macroscopic analysis of design processes based on a scheme for coding designers' cognitive actions, *Design Studies* **19**(4): 455-483.

Tversky, A: 1977, Features of similarity, *Psychological Review* **84**(4): 327-352.

Wegner, P: 1997, Why interaction is more powerful than algorithms, *CACM* **40**(5): 81-91.