Using the Object Oriented Paradigm to Model Context in Computer Go

Bruno Bouzy **Tristan Cazenave**

LAFORIA-IBP
case 169
Université Pierre et Marie Curie
4, place Jussieu
75252 PARIS CEDEX 05, FRANCE
bouzy@laforia.ibp.fr
cazenave@laforia.ibp.fr

Abstract

In this paper, we investigate the use of contextual knowledge in order to simplify knowledge representation in very complex domains and systems. In the case of a complex domain like the game of Go, we show that *context has several types*. We give some examples of temporal, goal, spatial and global contexts. We describe how Go playing programs represent and use these contexts in the Object-Oriented Paradigm (OOP). *We introduce three examples of OOP mechanisms that enable to deal with contexts*: a general class with specialized classes, a master slot of a class and slave slots, a list of goals that depend on each other. We claim that these OOP mechanisms are useful to represent context in other complex domains.

Key Words: Context, Object-Oriented Paradigm, Computer Go, Knowledge Representation, Complex Domains.

1 Introduction

In AI, context can be either a situation in the general sense of the term, a part of knowledge, or both of them. A situation contains both problem-related and environmental facts. The situation theory [Barwise & Perry 1983] has led to several works on context: for example a definition of context [Surav & Akman 1995], notes on formalizing context [McCarthy 1993] or a situation-oriented description of user interfaces [Strauss 1993]. In expert systems, context can be represented explicitly, leading to the claim that context is knowledge [Brezillon 1994]. While conventional systems do not use context, some do. For instance, Ripple Down Rules (RPR) are acquired in context and used out of it [Kang & Compton 1993]. Similarly, context can be used to classify knowledge and [McCarthy 1993] introduced the idea of "lifting rules" that activate some packets of rules. Context in rule-based formalisms can be represented by grouping rules into packets [Brezillon 1992]. Metarules then decide which packets should be activated.

In this paper, we assume that contextual knowledge in complex domains aims at organizing knowledge for an efficient use, and simplifying rules by avoiding the repetition of their preconditions. The aim of this paper is to uncover different types of contexts that arise in computer Go, and to propose general OOP mechanisms to represent context in complex systems. We support our claims by giving examples from systems that play the game of Go using a lot of knowledge : Indigo [Bouzy 1995] and Gogol [Cazenave 1996]. These systems are based on OOP and enable us to model context using OO mechanisms. Our systems are operational and have a good international ranking [Pettersen 1994].

In section 2, we introduce the rules of the game of Go and some useful Go concepts. In section 3, we model several types of contexts used in the game of Go. Section 4 is about the common object oriented architecture of our two systems. Section 5 is devoted to the representation of these contexts using OOP and explains how OOP enables interactions and synergy between contexts.

2 The game of Go

The game of Go is a 4000 years old game that is very famous in Japan, China and Korea. It is a zero-sum, two-persons, complete information game. The board is made of 19 vertical lines and 19 horizontal lines which cut themselves into 361 *intersections*. At the beginning the board is empty. Each player (Black or White) moves alternatively in adding one *stone* on an empty intersection. Two adjacent stones of the same color are *connected* and they are part of the same *string*. Empty adjacent intersections of a string are the *liberties* of the string. When a move fills the last liberty of a string, this string is removed from the board. The repetitions of positions are forbidden. According to the possibility of being captured or not, the strings may be *dead* or *alive*. A player *controls* an intersection either when he has an alive stone on it, either when the intersection is empty but adjacent to alive stones. The aim of the game is to control more intersections than the opponent. The game ends when the two players pass.

In spite of the simplicity of the rules, a Go player uses a lot of concepts to understand a position and to play a move. This paragraph briefly shows some intuitive definitions of these concepts. At the lower level, a player looks at the *safety* of the strings in performing look-ahead by playing on the liberties of the strings. When a string has enough liberties, the string is said to be safe. A player also checks if an intersection is controlled by one player or not. An *eye* is a small enclosed area, figure 1 gives an example of an eye on intersection A. In this figure, B is one of the four *diagonal* intersections of A. When searching to make an eye, it is important to control diagonal intersections.

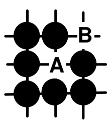

Figure 1

A virtual *connection* is a spatial configuration that enables to connect strings whatever the opponent plays. Figure 2 gives an example of a 'Bamboo join'. If the white player plays at A, black plays at B and connects its stones. If white plays at B, then black at A connects. The four stones are virtually connected.

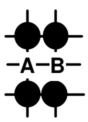

Figure 2

Using these tactical results, a Go player starts its strategic reasoning with the use of *groups*. A group is a complex concept for human players. It may be either a set of intersections that are virtually connected, either a set of intersections that gather the same properties. A group has a *status*. A status is dead or alive and it is derived from other intuitive concepts like *influence*, *fight*, *circling*, *life-base*. The reader does not need explanations of these concepts to understand the following sections.

A Go player also uses *territories* and *boundaries*. These concepts roughly correspond to the intuitive idea : a territory is a set of intersections mostly encircled with stones of the same color. A territory has a boundary.

A game of Go usually follows three stages : the *beginning*, the *middle-game* and the *endgame*. Very different strategies are associated with each stage. A score is associated to a position, either the game is *close*, or one player is *ahead* and the other one is *behind*.

3 Modeling different types of context

In this section, we show different types of context that are useful in the game of Go. We model temporal context, spatial context, global context and goal context.

3.1 Temporal context

A formal model of temporal context has been given in [McCarthy 1993] using the relation "specialize-time (t,c)". In Go, the temporal context is modeled by the three stages of the game : the beginning, the middle game and the endgame. Each stage is very different. In the beginning, the players fill the board at a big scale. In the middle game, they attack the areas which are under control of the opponent while defending their own areas. In the endgame, when the issue of each area is widely fixed, they refine the areas in order to grasp the last points. In Go playing systems, *the temporal context is the stage in which the game is*.

Rules modeling temporal context:

> If number of moves < 50 and all groups are stable
> then temporal context = beginning
>
> If some groups status are not yet fixed
> then temporal context = middle game
>
> If all groups have a fixed status
> then temporal context = endgame

Some rules using temporal context:

> beginning: If some corners or big spaces are empty then fill them
> middle game: If there are some big opponent areas then attack them
> middle game: If there are some big friend areas then defend them
> endgame: If there are some open boundaries, then close them

3.2 Goal context

When trying to achieve goals, Go players make very restricted searches. They have rules to find the good moves to look at. These rules are specialized in achieving a predefined goal, and they advise a move to try. Figure 3 give an example of a rule advising a move on intersection 'A'. This rule is very efficient when trying to achieve the goal of taking the white stone. But this rule is not useful when trying to achieve another goal.

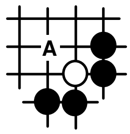

Figure 3

The goal context is the context in which a particular set of rules has to be matched.

3.3 Spatial context

Spatial information is very important in the game of Go. In the middle game, when groups are not yet settled, it is very important to know the spatial environment of a group. Figure 4 gives an example of a life-base. Black has to play on intersection 'A' in order to be sure to make the black group alive. But whether Black should play in 'A' depends on the spatial context of the black group.

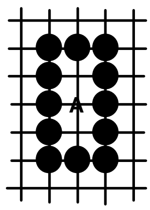

Figure 4

Figure 5 is an example where Black should play in 'A'. The spatial context is that the black group is encircled by White. In Figure 6, Black is not encircled by White. In this situation Black in 'A' is a bad move because it is an unnecessary move.

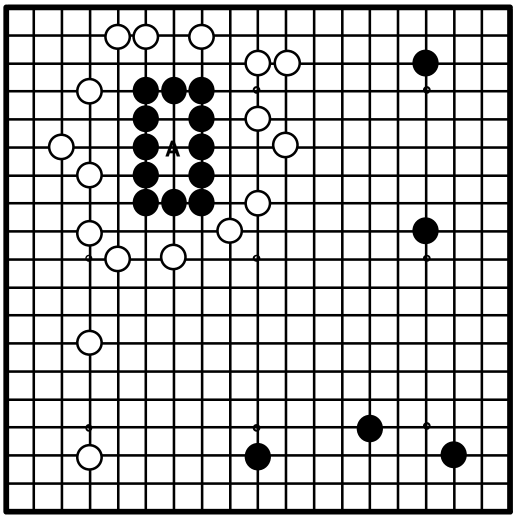

Figure 5

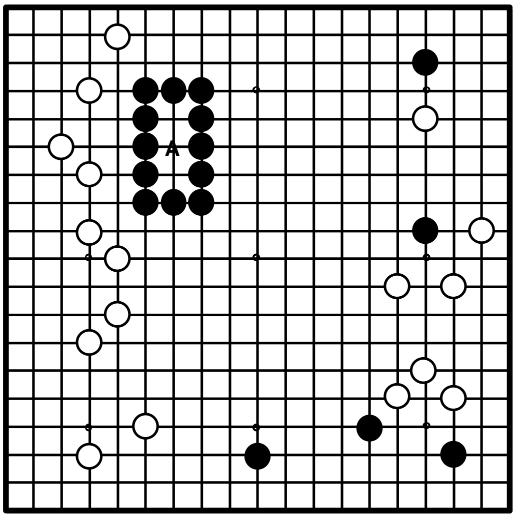

Figure 6

When a rule gives a move to make a group alive, *following the advice of the rule depends on the spatial context of the group*.

3.4 Global context

The size of the board is big (19x19). The players speak of the global situation of the board in opposition with the local situations. For example, the global situation is composed with the score of the game, the nature of the position (fight or peaceful) and the large scale vision of the board.

Global and local situations interact. Local situations are the basic elements which build the global situation. The global situation fixes the global strategy and therefore the local strategy to be played in each local situation. *We define the global context as the global situation.*

Example of a set of assertions defining the global context:

> White is ahead but the game is quite close.
> Black has less territory than White has
> Black has more influence than White has
> Black has strong groups

Example of rules using the global context:

> If ahead, defend your positions.
> If behind, attack the opponent positions.

4 Systems Architecture

In this section, we begin with a general overview of computer Go, followed by the description of the architecture of our Go playing systems, Gogol and Indigo.

4.1 Computer Go

In spite of the simplicity of its rules, playing the game of Go is a very complex task. [Robson 1983] proved that Go generalized to NxN boards is *exponential in time*. More concretely, [Allis 1994] defines the *whole game tree complexity* A. Considering the average length of actual games L and average branching factor B, we have $A = B^{L}$.

Table 1 : Complexity and H vs. C results

	Othello	**Chess**	**Go**[1]
A	10^{58}	$35^{80} \approx 10^{67}$	$200^{250} \approx 10^{575}$
H vs. C	H<C	H≈C	H>>C

Table 1 shows that A is much higher in Go than in Chess. This assertion forbids using whole game tree search as it is done in Chess. Table 1 also shows that the Computer (C) is still weak on the Human (H) scale. The best Go playing program in the world is Handtalk. Its level may be the one of a low-ranked Go club player, about 8 or 10 *kyu*[2]. In the human-computer challenge that closed the 1996 World Computer Go Championship, Handtalk lost against three dan players. In each of these games, the human players had a very large handicap, Handtalk was allowed to play the first 11 moves.

4.2 Architecture of Go playing systems

The architecture of Go playing systems, like Gogol or Indigo, is hierarchical and split in several levels. Figure 7 gives a general overview of the architecture. A bottom-up standpoint is the tactical level, the group construction level and the global level. The tactical level includes look-ahead about safety of strings, verification of eyes, connections between strings and other computations useful to build groups. The group level is the aggregation of tactical computations into the concept of group. A group has several properties that are useful to the global evaluation of the board. At least, the global level builds a global evaluation that is based on the relationships between groups and decides the move to play.

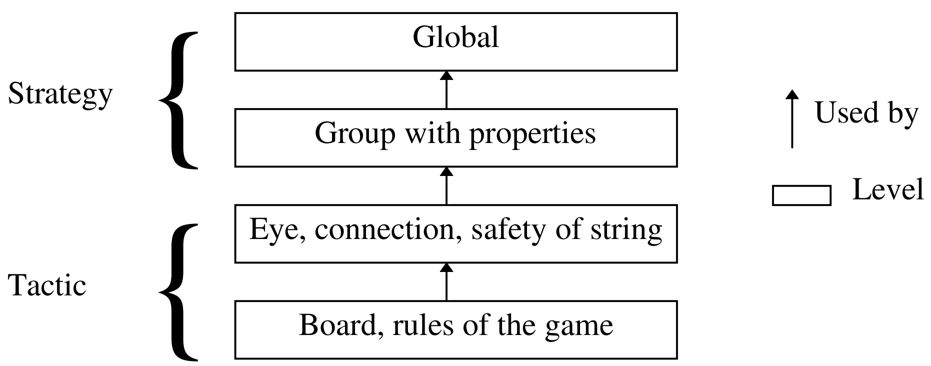

Figure 7

[1]In average, a 19x19 game lasts 250 moves and there are 200 possible moves.

[2]A complete novice is about 30 kyu, a beginner quickly reaches 20 kyu, a strong player is 1 kyu and then 1 dan until 9 dan for the strongest players in the world.

The systems begin to calculate the achievement of the tactical goals on a given board. They calculate the safety of the strings, the control of the intersections, the eyes with their diagonal intersections and the connections between strings. They use the results of the tactical searches to build the groups and their properties. Each group has a status which depends on the presence of a life-base, and of its other properties : its adjacent territories, its influence, its fighting abilities, its circling by strong or weak enemy groups. Before the move decision, the systems build the territories, the crucial areas, the boundaries and the score. The global level recognizes the stage of the game : beginning, middle-game, endgame, and then chooses its strategy.

The heterogeneity and the number of concepts naturally leads toward the use of object-oriented language. Our systems are C++ programs.

5 The representation and use of context in an object-oriented program

In this section, we show how the different types of context are represented in Go playing programs. Furthermore, we show how an OOP manages these different types of context.

5.1 Temporal context

We identified 3 stages in a game of Go. Therefore we defined the taxonomy of classes of figure 8.

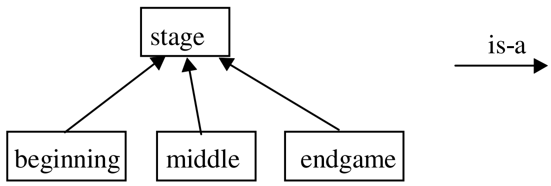

Figure 8

Each specialized class represents one specific temporal context. It contains meta-rules that allow the activation of specific rules' packets that will generate the relevant kinds of moves in the global move decision process. When the beginning of the game is finished the program stops using the packets of rules that are advised by the "beginning" meta-rules and starts to use the packets of rules advised by the "middle-game" meta-rules.

This first template - *a general class that is specialized into specific classes* - is very helpful for modeling context.

5.2 Spatial context

We showed that spatial context is very important. We gave the example of a group whose status depends on the spatial context in which it is situated. The group has no settled life-base. If a player plays first, he can delete or create the life-base according to his color. When the spatial context is constituted with opponent stones that are circling the group, the status of the group is unsettled and when the spatial context is such that the group is not well-circled, then the status is alive. Therefore the spatial context of a group strongly decides of the kinds of moves that will be used either to create or delete the life-base, either to circle the group or to make the group escape.

The fundamentals of OOP advise to represent properties (like life-base or circling) of a concept (like group here) as the slots of the associated class. Assuming we also reify the properties or slots, we use the taxonomy of figure 9.

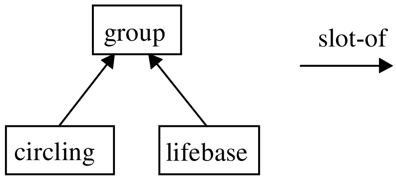

Figure 9

The meta-rules that select the relevant rules that will select the moves according to the context are simple. If the circling of a group is under a threshold, the meta-rules select the moves that are attached to the class circling only. If the circling of a group is higher than a threshold, the meta-rules select the moves that are attached to the class life-base only. Here, there is a master slot (circling) and a slave slot (life-base). The master slot fixes the context and the usefulness of the slave slot.

This template - *a class with a master slot that fixes the context for a slave slot* - is also a very helpful mechanism for modeling context.

5.3 Goal context

In order to delete or create a life-base, that is a goal, the players must delete or create eyes. In order to achieve this latter goal, the players must control the diagonal intersections. Therefore the goals are linked with dependencies as it is shown on figure 10.

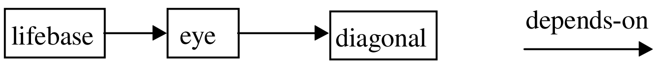

Figure 10

Programs may play moves in order to control diagonal intersections, in the context of creating or deleting eyes, in the context of creating or deleting a life-base. Each goal that is selected specifies a specific goal context. Each goal context is linked with other goal contexts.

This template - *a succession of goals* - is also a very helpful mechanism for modeling context.

5.4 Global context

We showed that an important parameter of the global context is the score of the game. This parameter is used in left part of meta-rules that select the strategic behavior of the program. Other contexts can make part of the global context : the temporal context is an example.

5.5 Synergy of contexts

We showed different kinds of representation and use of context :

explicit representation with specialized classes (example : temporal context)
use of already useful slots (example : spatial context)
use of already useful links (example : goal context)

Assuming that the global context is behind, that the stage of the game is middle-game, then the global context will advise to attack and defend groups. Given this, a local context arises:

attack or defend this specific group. Given the spatial context of the group, try to reach the goal of creating or deleting a life-base. Creating or deleting a life-base can be achieved by trying to reach the create eye or delete eye goal. The eye goal can be achieved by trying to reach the control diagonal goal.

Therefore, we see that in a complex domain, the context is also complex. It is composed with several viewpoints. The programming strategy consists in representing this viewpoints inside the taxonomy of classes of objects. It is represented with *specific master-slaves dependencies between attributes* (see circling and life-base), with *explicit definition of classes* (see middle game) or with already defined *relationships between classes* (see life-base, eye and diagonal).

5.6 OOP and context

OOP mainly uses the idea of context. In a given class which is a specialization of some general classes, the context is the set of these general classes. The basic principle of OOP is to push the definition of properties, functions and rules as high as possible in the taxonomy. This principle allows to define the smallest number of properties, functions and rules. Moreover, it simplifies the expressions of functions and rules. The programmer avoids to re-specify the properties of the general classes (the context) when he is defining a given function or packets of rules in a specialized class. Therefore the fundamental idea of OOP is that the context of a given class is contained in the classes from which the given class inherits.

6 Conclusion

In this paper, we investigate the use of contextual knowledge in order to simplify knowledge representation in very complex domains and systems. In the case of a complex domain like the game of Go, we show that *context has several types*. We give some examples of temporal, goal, spatial and global contexts. We describe how Go playing programs represent and use these contexts in an object-oriented programming environment. Our two Go playing systems, Gogol and Indigo, have achieved a good international ranking [Pettersen 1994]. We introduce *three examples of OOP mechanisms that enable to deal with contexts*:

a general class with specialized classes,
a master slot of a class and slave slots,
a list of goals that depend on each other.

We claim that these OOP mechanisms are useful to represent context in other complex domains.

Moreover, OOP seems appropriate to represent context. Actually, in complex domains, knowledge has different types and OOP is well-adapted for context-free knowledge representation. Context represents a special kind of knowledge [Brezillon 1994] and, as such, also has several types. Therefore, OOP is well-suited for context representation. We thus claim that merging a context-free knowledge representation with context representation makes it possible to organize knowledge and simplify the rules.

References

[Allis 1994] - L. V. Allis - Searching for Solutions in Games an Artificial Intelligence - Ph.D. Thesis - Vrije Universitat Amsterdam - Maastricht - September 1994.

[Barwise & Perry 1983] - J. Barwise, J. Perry - Situations and Attitudes - MIT Press - Cambridge MA - 1983.

[Bouzy 1995] - B. Bouzy - Modélisation Cognitive du Joueur de Go - Ph.D. Thesis - Paris 6 University - 1995.

[Brezillon 1992] - P. Brezillon - Architectural and contextual factors in explanation construction - Proceedings of the ECAI'92 Workshop on Improving the use of Knowledge-Based Systems with Explanations - Internal Report of LAFORIA 92/21 - Paris 6 University.

[Brezillon 1994] - P. Brezillon - Context needs in cooperative building of explanations - Internal Report of LAFORIA 94/13 - Paris 6 University.

[Cahour & Karsenty 1993] - B. Cahour, L. Karsenty - Context of Dialogue : a Cognitive point of view - Proceedings of the workshop about context at IJCAI - Chambéry 1993 - Internal Report of LAFORIA 93/13 - pp. 20-28 - Paris 6 University.

[Cazenave 1996] - T. Cazenave - Learning to Forecast by Explaining the Consequences of Actions - International Workshop on Machine Learning, Forecasting and Optimization - Madrid - 1996.

[Kang & Compton 1993] - B. Kang, P. Compton - Taking knowledge in context out of its context - Proceedings of the workshop about context at IJCAI - Chambéry 1993 - Internal Report of LAFORIA 93/13 - pp. 63-73 - Paris 6 University.

[McCarthy 1993] - J. McCarthy - Notes on Formalizing Context - Proceedings of the IJCAI - Chambéry 1993 - pp. 555-560.

[Mittal & Paris 1993] - V.B. Mittal, C.L. Paris - Context : Identifying its elements from the communication point of view - Proceedings of the workshop about context at IJCAI - Chambéry 1993 - Internal Report of LAFORIA 93/13 - pp. 87-97 - Paris 6 University.

[Pinto & al. 1995] - N. Pinto, L. Stephens, R. Bonnell - Organizing Domain theories For geographical Reasoning Using Contexts - Proceedings of the workshop about context at IJCAI - Montréal 1995 - Internal Report of LAFORIA 95/11 - pp. 109-119 - Paris 6 University.

[Pettersen 1994] - E. Pettersen - The Computer Go Ladder - http://cgl.ucsf.edu/go/ladder.html

[Robson 1983] - J. M. Robson - The Complexity of Go - Proceedings IFIP - pp. 413-417 - 1983.

[Strauss 1993] - F. Strauss - Situation oriented Description of User Interfaces - Proceedings of the workshop about context at IJCAI - Chambéry 1993 - Internal Report of LAFORIA 93/13 - pp. 121-129 - Paris 6 University.

[Surav & Akman 1995] - M. Surav, V. Akman - Modeling Context with Situations - Proceedings of the workshop about context at IJCAI - Montréal 1995 - Internal Report of LAFORIA 95/11 - pp. 145-156 - Paris 6 University.