

References

- [1] Bagher, M., Pour, L., (2008), *Intelligent Agent System Simulation Using Fear Emotion*, In proceedings of the International conference on World Congress of Science, Engineering and Technology, Bangkok, Thailand
- [2] Bartneck, C., (2002), *Integrating the OCC Model of Emotions in Embodied Characters*, Workshop on Virtual Conversational Characters
- [3] Bhikku Bodhi thero, (2007), *A Comprehensive Manual of Abhidhamma*, Kandy: Buddhist Publication Society
- [4] Blewitt, W., Ayesh, A., Bertelle, C., Mahboub, K., (2010), *Psychologically Grounded Emotion Model for a NPC Fuzzy Controller*
- [5] Boehner, K., DePaula, R., Dourish, P., Sengers, P., (2007), *How emotion is made and measured*, International Journal of Human-Computer Studies, 65(4), pp. 275–291
- [6] Brinol, P., Pretty, R.E., Rucker, D. D., (2006), *The role of meta-cognitive processes in emotional intelligence*, Psicothema ISSN 0214 - 9915 CODEN PSOTEG , 18, pp. 26-33
- [7] Caruso, D., Mayor, J.D., Salvoy, P., (2004), *Emotional Intelligence: Theory Findings and Implications*, Psychological Inquiry, 15(3), pp. 197-215
- [8] Cearreta, I., García, R., Gary, N., Gil, R., López, J.M., (2008), *Towards an ontology for describing emotions*, In proceedings of the 1st world summit on The Knowledge Society: Emerging Technologies and Information Systems for the Knowledge Society, Athens, Greece
- [9] Esau, N., Kleinjohann, B., Kleinjohann, L., (2005), *An Adaptable Fuzzy Emotion Model for Emotion Recognition*, In proceedings of the 4th Conference of the European Society for Fuzzy Logic and Technology , Barcelona, Spain
- [10] Freitas, J.S., Gudwin, R. R., Queiroz, J., (2008), *Emotion in Artificial Intelligence and Artificial Life Research: Facing problems*, Lecture Notes in Computer Science, ISBN:3-540-28738-8

- [11] Fujii, K. S., Koczy, T., Kovacs, Kubota, N., (2000), *Behaviour based techniques in user adaptive Kansei technology*, In proceedings of the 6th International Conference on Virtual Systems and MultiMedia
- [12] Gill, G.S., (2010), *Examining Emotional Intelligence in Sports*, MPhil thesis, University of Wolverhampton
- [13] Goleman, D., (1998), *Working with emotional intelligence*, New York: Bantam Books
- [14] Gratch, J., Mao, W., Marsella ,S., (2006), *Towards a Validated Model of “Emotional Intelligence”*, In proceedings of the 21st international conference on Artificial intelligence
- [15] Hamed, S., Hashim, A., Khalifa, O., Mustafa, A., (2008), *Adaptive Emotional Personality Model based on Fuzzy Logic Interpretation of Five Factor Theory*, Signal Processing: An International Journal (SPIJ) 2(4), pp. 1-9
- [16] Hollmann, C., Lawrence, P. J., Galea, E. R., (2010), *Introducing Emotion Modelling to Agent-Based Pedestrian Circulation Simulation*, PED 2010, NIST, Maryland USA www.lib.mrt.ac.lk
- [17] Karunananda, A. S., (2005), *Secure future through Multi-Agent System Technology*, In proceedings of the 24th National IT Conference, Colombo, pp. 104-113
- [18] Karunananda, A.S., (2007), *How to write your thesis*, ISBN978-955-98646-1-5
- [19] Karunananda A.S., Rzevski G. (2005), *Relevance of Buddhist Philosophy to ontological modeling in Information Systems and Computer Science*, In proceedings of the 2nd Annual Sessions of Sri Lanka Association for Artificial Intelligence, pp. 4-13
- [20] Kovács, S., (2002), *Fuzzy Reasoning and Fuzzy Automata in User Adaptive Emotional and Information Retrieval Systems*, In proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Hammamet, Tunisia, 02CH37349C, ISBN: 0-7803-7438-X

- [21] Laird, J.E., (2008), *Extending the Soar Cognitive Architecture*, In proceedings of the 2008 conference on Artificial General Intelligence, Amsterdam, Netherlands
- [22] Lau, N., Reis, L. P., (2008), *A Computational Study on Emotions and Temperament in Multi-Agent Systems*, Computer Science, Cornell University library
- [23] Marinier, P. R, Laird, J.E, (2006), *A Cognitive Architecture Theory of Comprehension and Appraisal*, proceedings of the 18th European Meeting on Cybernetics and System Research ,Vienna, Austria
- [24] Mason, C.L., (2008), *Human-Level AI Requires Compassionate Intelligence*, AAAI Workshop on Meta-Cognition
- [25] McCrae, R. R., (1987), *Validation of a five-factor model or personality across instruments and observers*, Journal of Personality and Social Psychology, 52, pp 81-90.
- [26] Minsky, M., (1986), *Society of Mind*, Simon & Schuster, Inc. New York, NY, USA ,ISBN:0-671-60740-5
- [27] Minsky, M., (2006), *Emotion Machine*, Simon & Schuster, Inc. New York, NY, USA , ISBN:0743276639
- [28] Mon, M. T., (1995), *Buddha Abhidhamma Ultimate Science*, International Therawady Buddhism Missionary Universtity, Union of Myanmar
- [29] Nallaperuma, S.N., Karunananda, A.S., (2011), *Towards an Emergent Model of Emotions*, In proceedings of 6th IEEE International conference on Industrial and Inforamation systems, Kandy, Sri Lanka
- [30] Narada thero, (1987), *A manual of Abhidhamma*, Budddhist Missionary Society, Kuala Lumpur, Malaysia, ISBN 967-9920-42-9
- [31] Nyanaponika thero, (1976), *Abidhamma Studies:Researches in Buddhist philosophy*, Kandy: Buddhist Publication Society
- [32] Ortony, A., Collins, A., Clore, G., (1988), *The Cognitive Structure of Emotions*. Cambridge University Press, Cambridge

- [33] Ortony, A., (2003). *On making believable emotional agents believable*, Emotions in humans and artefacts, Cambridge: MIT Press.
- [34] Pan, X., Han, C.S, Dauber, K., Law, K.H., (2007), *A multi-agent based framework for the simulation of human and social behaviors during emergency evacuations*. AI & Society, 22(2), pp.113–132
- [35] Picard, R.W., (1997) *Affective Computing*, M.I.T Media Laboratory Perceptual Computing Section Technical Report No. 321
- [36] Picard, R.W., Vyzas, E., Healey, J., (2001), *Toward machine emotional intelligence: analysis of affective physiological state*, IEEE Transactions on Pattern Analysis and Machine Intelligence 23(10), pp. 1175-1191
- [37] Polanyi, M., (1970), *Life's Irreducible Structure*, Journal of the American Scientific Affiliation, 22, pp. 123-131
- [38] Rathnapala, A., (1964), *Abidharma pradeepikawa*, ISBN 955-8873-47
- [39] Regan L., Mandryk M., Atkins, S., (2007), *A fuzzy physiological approach for continuously modeling emotion during interaction with play technologies*, The international journal of human computer studies
- [40] Robert, E. C., Frank, J. L., Chown, E., (2006) *Modeling Emotion: Arousal's Impact on Memory* , In proceedings of the 28th Annual Conference of the Cognitive Science Society pp. 1133-1138, Vancouver, British Columbia, Canada
- [41] Rosenbloom, P. S., Laird, J. E., Newell, A., (1993), *The Soar papers: Research on Integrated Intelligence*, MIT Press, Cambridge, MA.
- [42] Russell, J.A., Weiss, A., Mendelsohn, G.A., (1989), *Affect grid: a single item scale of pleasure and arousal*, Journal of personality and social psychology 57(3) pp. 493-502
- [43] Saffiotti, A., Konolige, K., Ruspini, E. H., (1995), *A multi valued logic approach to integrating planning and control*, Artificial Intelligence, 76, pp. 481-526.
- [44] Scherer, K., (2005), *Proposal for exemplars and work towards them: Theory of emotions*, Available: <http://emotion-research.net/deliverables>

- [45] Schroder, M., Pirker, H., Lamolle, M., Burkhardt, F., Peter, C., Zovato, E., (2011), *Representing Emotions and Related States in Technological Systems*
- [46] Serugendo, G. D. M., Gleizes, M. P., Karageorgos, A., (2006), *Self Organization and Emergence in MAS: An Overview*, Journal of INFORMATIASI, 30, pp. 45-54
- [47] Slater, S., (2010), *E-AI an Emotion Architecture for Agents in Games & Virtual Worlds*, University of Wolverhampton
- [48] Tanguy, E., Willis, P. J., Bryson, J. J., (2006), *A dynamic emotion representation model within a facial animation system*, International Journal of Humanoid Robotics, 3 (3), pp. 293-300
- [49] Vajrajana, thero, (1962), *Buddhist Meditation in Theory and Practice*, Kandy M D. Gunasena & Co.
- [50] Buddhadatta thero, *Concise Pali-English Dictionary*, Available: <http://www.budsas.org/ebud/dict-ep>
- [51] Yit, K.T., (2005), *Buddhist meditation experiences and the consciousness*, Science and Religion: Global Perspectives, Philadelphia, PA, USA

Appendix A

Buddhist Abhidamma Teachings on Philosophy of Mind ('Citta Sangaha')

A.1 Introduction

In this appendix section we would be describing the analysis of thoughts according to Buddhist abhidamma.

A.2 The Thought Process

The thought process is comprehensively explained in Manual of Abhidamma [30].

“The subject, the consciousness, receives objects from within and without. When a person is in a state of pro-found sleep his mind is said to be vacant, or, in other words, in a state of *Bhavañga*. We always experience such a passive state when our minds do not respond to external objects. This flow of *Bhavañga* is interrupted when objects enter the mind. Then the *Bhavañga* consciousness vibrates for one thought-moment and passes away. There-upon the sense-door consciousness arises and ceases. At this stage the natural flow is checked and is turned towards the object. Immediately after there arises and ceases the eye-consciousness, but yet knows no more about it. This sense operation is followed by a moment of reception of the object so seen (*Sampaticchana*). Next comes the investigating faculty 19 or a momentary examination of the object so received. After this comes that stage of representative cognition termed the determining consciousness (*Votthapana*). Discrimination is exercised at this stage. Freewill plays its part here. Immediately after there arises the psychologically most important stage—Impulsion or *Javana*. It is at this stage that an action is judged whether moral or immoral. *Kamma* is performed at this stage; if viewed rightly (*yoniso manasikāra*), the *Javana* becomes moral; if viewed wrongly (*ayoniso manasikāra*), it becomes immoral. In the case of an *Arahant* this *Javana* is neither moral nor immoral, but merely functional (*Kiriya*). This *Javana* stage usually lasts for seven thought-moments, or, at times of death, five. The whole process which happens in an infinitesimal part of time ends with the registering consciousness (*Tadālabhana*), lasting for two thought-moments—thus completing one thought-process at the expiration of seventeen thought-moments. The three kinds of *Bhavañga* consciousness are *Vipāka*. They are either one of the two *Santīraṇa Cittas*,

accompanied by indifference, mentioned above, or one of the eight Sobhana Vipāka Cittas. *Pancadvārāvajjana* is a *Kriyā Citta*. *Panca-Vinyana* is one of the ten moral and immoral *Vipāka Cittas*. *Sampaticchana* and *Santīraṇa* are also *Vipāka Cittas*. The *Mano-dvārāvajjana* (mind-door consciousness), a *Kriyā Citta*, functions as the *Votthapana* consciousness. One can use one's freewill at this stage. The seven *Javana* thought- moments constitute *Kamma*. The *Tadālabhāna* is a *Vipāka Citta* which is one of the three *Santīraṇa Cittas* or one of the eight *Sobhana Vipāka Cittas*. Thus, in a particular thought-process there arise various thought-moments which may be *Kamma*, *Vipāka*, or *Kriyā*" [30].

A.3 Analysis on Thoughts

The complete set of thoughts (*Cittas*) in Buddhist abhidamma is as follows.

1 *Akusala Cittas* – 12

They are 8 *lobha-mūla Cittas*, 2 *dosa-mūla Cittas* and 2 *mohamūla Cittas*.

2 *Ahetuka Cittas* – 18

They comprise 7 *akusala vipāka Cittas*, 8 *ahetuka kusala vipāka Cittas* and 3 *ahetuka kiriya Cittas*.

3 *Kāma-sobhaṇa Cittas* – 24

They are divided into 8 *mahā-kusala Cittas*, 8 *mahā-vipāka Cittas* and 8 *mahā-kiriya Cittas*.

4 *Kāmāvacara Cittas* or *kāma Cittas* – 54

They comprise 12 *akusala Cittas*, 18 *ahetuka Cittas* and 24 *kāma-sobhaṇa Cittas*.

5 *Mahaggata Cittas* – 27

The 15 *rūpāvacara Cittas* and the 12 *arūpāvacara Cittas* are collectively known as *mahaggata Cittas*. 71 'Mahaggata' literally means 'grown great', i.e., developed, exalted or supernormal. *Mahaggata Citta* is the state of 'developed consciousness' attained in the fine-material and the immaterial absorptions. *Mahaggata Cittas* are more developed or more exalted than *kāma-Cittas*.

6 *Lokiya Cittas* – 81

The 54 *kāmāvacara Cittas* and the 27 *mahaggata Cittas* are collectively known as 81 *lokiya Cittas*. *Lokiya* — *mundane* or associated with the three spheres of existence — namely, the *kāma-sphere*, the *rūpa-sphere*, and the *arūpa-sphere* (the *sense sphere*, the *fine-material sphere* and the *immaterial sphere*).

7 *Lokuttara Cittas* – 8 or 40

The 4 *magga-Cittas* (path-consciousness) and the 4 *phala-Cittas* (fruit-consciousness) constitute 8 *lokuttara Cittas*. When they are multiplied by 5 *rūpāvacara jhānas*, we get 40 *lokuttara Cittas*. *Lokuttara* – *supramundane* or beyond the three spheres of existence. The 8 *lokuttara Cittas* together with *Nibbāna* constitute the ‘9 *supramundane dhammas*’ (*nava-lokuttara-dhamma*).

8 Total number of *Cittas* 89 or 121

The 81 *lokiya Cittas* together with 8 *lokuttara Cittas* (*vipassanā-yānika* way) make up 89 *Cittas* in all. Or if we combine 81 *lokiya Cittas* with 40 *lokuttara Cittas* (*samatha-yānika* way) we get 121 *Cittas* in all.



University of Moratuwa, Sri Lanka.
Electronic Theses & Dissertations
www.lib.mrt.ac.lk

9. *Asobbhana Cittas* – 30

They comprise 12 *akusala Cittas* and 18 *ahetuka Cittas*. The *akusala Cittas* are not ‘beautiful’ (*sobhaṇa*) because they are associated with evil roots — namely *lobha* (greed), *dosa* (hatred) and *moha* (delusion). 72 The *ahetuka Cittas* are regarded as ‘not beautiful’ (*asobhaṇa*) because they are not associated with wholesome roots, namely *alobha* (generosity), *adosa* (goodwill) and *amoha* (wisdom).

10 *Sobhaṇa Cittas* – 59 or 91

If we subtract 30 *asobhaṇa* from 89 *Cittas*, we obtain 59 *sobhaṇa Cittas*. Or, if we subtract 30 *asobhaṇa Cittas* from 121 *Cittas*, we get 91 *sobhaṇa Cittas*. *Sobhaṇa Cittas* are associated with wholesome roots.

11 *Jhāna Cittas* – 67

The 27 *mahaggata Cittas* are known as the *lokiya jhāna Cittas*. Combining these with the 40 *lokuttara jhāna Cittas* we get 67 *jhāna Cittas*. Among these, there are 11 first *jhāna Cittas*, 11 second *jhāna Cittas*, 11 third *jhāna Cittas*, 11 fourth *jhāna Cittas*,

and 23 fifth *jhāna Cittas*. Note that there are 3 first *jhāna Cittas* in the *mahaggata Cittas* and 8 first *jhāna Cittas* in *lokuttara Cittas*; together they make up 11 first *jhāna Cittas*. The second, the third and the fourth *jhāna Cittas* are counted in the same way. In counting the fifth *jhāna Cittas*, all the 12 *arūpāvacara Cittas* are included in the count.

Accordingy to abhidamma the anlaysis of 54 sense sphere thoughts are as follows. As introduced above they contain 12 immoral 18 rootless and 24 beautiful *Cittas*.

12 immorals are listed below. They are again categorized as 8 greed rooted , 2 hatred rooted and 2 delusion rooted *Cittas*.

1. *One consciousness, unprompted, accompanied by pleasure, and connected with wrong view,*

2. *One consciousness, prompted, accompanied by pleasure, and connected with wrong view,*

3. *One consciousness, unprompted, accompanied by pleasure, and disconnected with wrong view,*

4. *One consciousness, prompted, accompanied by pleasure, and disconnected with wrong view,*

5. *One consciousness, unprompted, accompanied 28 by indifference, and connected with wrong view,*

6. *One consciousness, prompted, accompanied by indifference, and connected with wrong view,*

7. *One consciousness, unprompted, accompanied by indifference, and disconnected with wrong view,*

8. *One consciousness, prompted, accompanied by indifference, and disconnected with wrong view,*

These eight types of consciousness are rooted in Attachment.

9. *One consciousness, unprompted, accompanied by displeasure, and connected with illwill.*

10. *One consciousness, prompted, accompanied by displeasure, and connected with illwill.*

These two types of consciousness are connected with Illwill.

11 One consciousness, accompanied by indifference, and connected with doubts,

12. One consciousness, accompanied by indifference, and connected with restlessness.

These two types of consciousness are rooted in sheer Ignorance (delusion).

18 rootless *Cittas* are categorized as moral resultant, immoral resultant and functional *Cittas*. The list is as below.

(1) Eye-consciousness, accompanied by indifference. So are (2) Ear-consciousness, (3) Nose-consciousness, (4) Tongue-consciousness, (5) Body-consciousness, accompanied by pain, (6) Receiving consciousness, accompanied by indifference, (7) Investigating consciousness, accompanied by indifference.

These seven are the immoral resultant types of consciousness.

(8) Moral resultant Eye-consciousness, accompanied by indifference. So are (9) Ear-consciousness, (10) Nose-consciousness, (11) Tongue-consciousness, (12) Body-consciousness, accompanied by happiness, (13) Receiving consciousness, accompanied by indifference, (14) Investigating consciousness, accompanied by pleasure, (15) Investigating consciousness, accompanied by indifference.

These eight are the moral resultant types of consciousness without Hetu.

(16) Five Sense-door advertent consciousness, accompanied by indifference. So is (17) Mind-door advertent consciousness. (18) Smile-producing consciousness, accompanied by pleasure.

These three are the functional types of consciousness without Hetu. Thus end, in all, the eighteen types of consciousness without Hetu.

Set of morals are categorized into three groups as moral, resultant and functional 8 per each group.

1. One consciousness, unprompted, accompanied by pleasure, associated with knowledge,

2. One consciousness, prompted, accompanied by pleasure, associated with

knowledge,

3. *One consciousness, unprompted, accompanied by pleasure, dissociated with knowledge,*

4. *One consciousness, prompted, accompanied by pleasure, dissociated with knowledge,*

5. *One consciousness, unprompted, accompanied by indifference,²² associated with knowledge,*

6. *One consciousness, prompted, accompanied by indifference, associated with knowledge,*

7. *One consciousness, unprompted, accompanied by indifference, dissociated with knowledge,*

8. *One consciousness, prompted, accompanied by indifference, dissociated with knowledge.*

These are the eight types of moral consciousness, with Roots, of the sensuous sphere.

9. *One consciousness, unprompted, accompanied by pleasure, associated with knowledge,*

10. *One consciousness, prompted, accompanied by pleasure, associated with knowledge,*

11. *One consciousness, unprompted, accompanied by pleasure, dissociated with knowledge.*

12. *One consciousness, prompted, accompanied by pleasure, dissociated with knowledge,*

13. *One consciousness, unprompted, accompanied by indifference, associated with knowledge.*

14. *One consciousness, prompted, accompanied by indifference, associated with knowledge,*

15. *One consciousness, unprompted, accompanied by indifference, dissociated with knowledge,*

16. *One consciousness, prompted, accompanied by indifference, dissociated with knowledge.*

These are the eight types of Resultant Consciousness, with Hetus, of the sensuous sphere.

17. *One consciousness, unprompted, accompanied by pleasure, associated with knowledge,*

18. *One consciousness, prompted, accompanied by pleasure, associated with knowledge,*

19. *One consciousness, unprompted, accompanied by pleasure, dissociated with knowledge,*

20. *One consciousness, prompted, accompanied by pleasure, associated with knowledge,*

21. *One consciousness, unprompted, accompanied by indifference, dissociated with knowledge,*

22. *One consciousness, prompted accompanied by indifference, associated with knowledge,*

23. *One consciousness, unprompted, accompanied by indifference, dissociated with knowledge,*

24. *One consciousness, prompted, accompanied by indifference, dissociated with knowledge.*

These are the eight types of Functional Consciousness, with Roots, of the sensuous sphere.



University of Moratuwa, Sri Lanka
Electronic Theses & Dissertations
www.lib.mrt.ac.lk

The *Cetasiks* containment whole set of *Citta* is as described in the two charts below (Figure A.1,2). And all *Cetasik* combinations in all thoughts are presented in table below. (Figure A.3)

UNIVERSITY OF MORATUWA, SRI LANKA
ELECTRONIC THESSES & DISSERTATIONS
www.lib.mrt.ac.lk

CETASIKAS	CITAS												METHOD OF COMBINATION
	1	2	3	4	5	6	7	8	1	2	1	2	
Universals 7													
In. Application													
Sus. Application													
Decision													
Energy													
Zest													
Desire													
Unwh. Univs. 4													
Greed													
Wrong View													
Conceit													
Hate, Envy, Avarice, Worry													
Sloth, Torpor													
Doubt													
Btf. Univs. 19													
Abstinenes 3													
Illimitables 2													
Wisdom													
Totals	19	21	19	21	18	20	18	20	20	22	15	7	12
	19	21	19	21	18	20	18	20	20	22	15	7	12

Figure A.1 *Cetasiks* combination chart for thoughts first part :Source [3]

CITTA	No.	CETASIKAS	TOTAL
<i>Supramundane</i>			
1st jhāna	8	1-13, 28-49, 52	36
2nd jhāna	8	1-7, 9-13, 28-49, 52	35
3rd jhāna	8	1-7, 10-13, 28-49, 52	34
4th jhāna	8	1-7, 10, 11, 13, 28-49, 52	33
5th jhāna	8	1-7, 10, 11, 13, 28-49, 52	33
<i>Sublime</i>			
1st jhāna	3	1-13, 28-46, 50-52	35
2nd jhāna	3	1-7, 9-13, 28-46, 50-52	34
3rd jhāna	3	1-7, 10-13, 28-46, 50-52	33
4th jhāna	3	1-7, 10, 11, 13, 28-46, 50-52	32
5th jhāna	15	1-7, 10, 11, 13, 28-46, 52	30
<i>SS Beautiful</i>			
Wholesome	31, 32	1-13, 28-52	38
"	33, 34	1-13, 28-51	37
"	35, 36	1-11, 13, 28-52	37
"	37, 38	1-11, 13, 28-51	36
Resultant	39, 40	1-13, 28-46, 52	33
"	41, 42	1-13, 28-46	32
"	43, 44	1-11, 13, 28-46, 52	32
"	45, 46	1-11, 13, 28-46	31
Functional	47, 48	1-13, 28-46, 50-52	35
"	49, 50	1-13, 28-46, 50, 51	34
"	51, 52	1-11, 13, 28-46, 50-52	34
"	53, 54	1-11, 13, 28-46, 50-51	33
<i>Unwholesome</i>			
Greed-rooted	1	1-19	19
"	2	1-19, 25, 26	21
"	3	1-18, 20	19
"	4	1-18, 20, 25, 26	21
"	5	1-11, 13, 14-19	18
"	6	1-11, 13, 14-19, 25, 26	20
"	7	1-11, 13, 14-18, 20	18
"	8	1-11, 13, 14-18, 20, 25, 26	20
Hate-rooted	9	1-11, 13, 14-17, 21-24	20
"	10	1-11, 13, 14-17, 21-24, 25, 26	22
Delus.-rooted	11	1-9, 11, 14-17, 27	15
"	12	1-11, 14-17	15
<i>Rootless</i>			
Sense consness.	13-17	1-7	7
" "	20-24	1-7	7
Receiving	18, 25	1-10	10
Investigating	19, 27	1-10	10
Investigating	26	1-10, 12	11
Five door-advt.	28	1-10	10
Mind-door-advt.	29	1-11	11
Smile-producing	30	1-12	12

Figure A.3 *Cetasiks* listing for thoughts: Source [30]

History of Complex Systems and Emergence

B.1 Introduction

In this appendix section we would briefly discuss on the historical ideas on the concepts of emergence developed throughout the history in various domains, science philosophy etc. Let us briefly discuss some popular ideas developed throughout the history about emergence in complex systems.

B.2 Evolution of complex system studies

Complex systems is a new approach to science that studies how relationships between parts give rise to the collective behaviors of a system and how the system interacts and forms relationships with its environment.

The earliest precursor to modern complex systems theory can be found in the classical political economy of the Scottish Enlightenment, later developed by the Austrian school of economics, which says that order in market systems is emergence in that it is the result of human action, but not the execution of any human design.

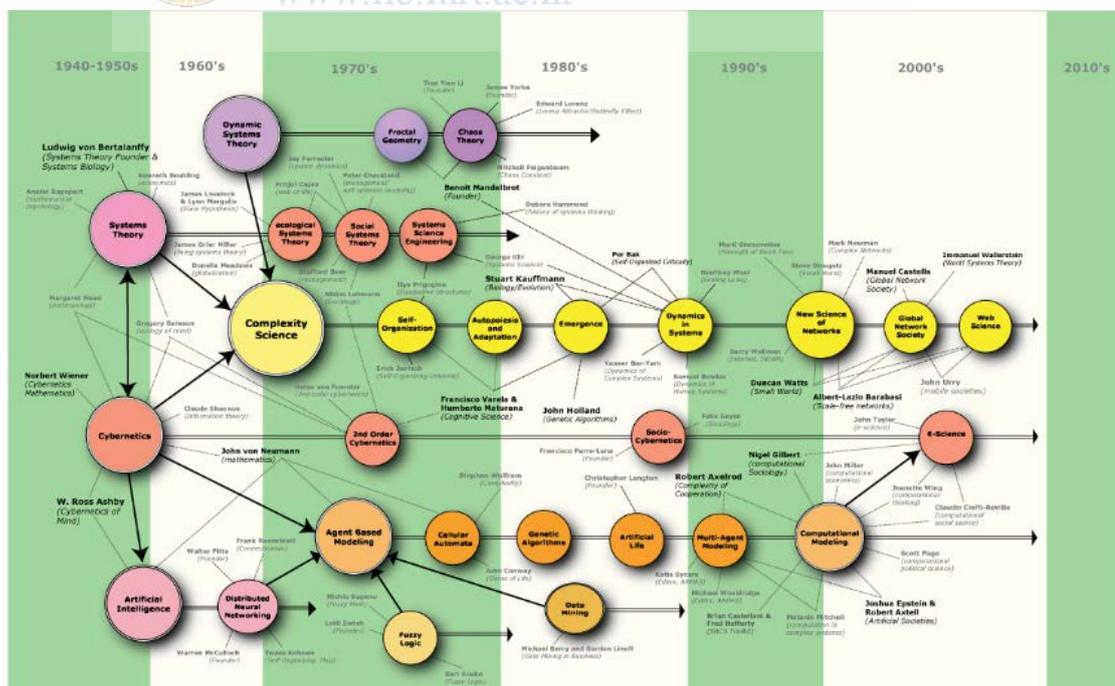


Figure B.2 History of complex systems studies

B.2 Emergence in complex systems

Real life example for emergence in complex systems could be found in an anthill of termites (Figure B.1). The distributed entities of such a system are ants, merely simple entities governed by small set of rules. These small ants perform a simple task at a time. As a result of these small tasks performed by a large set of ants continuously a large anthill emerges.

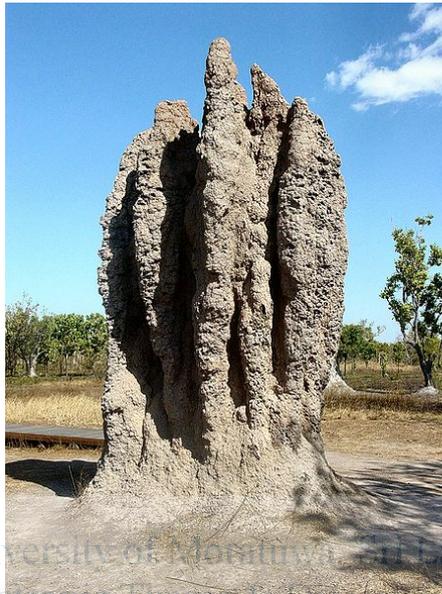


Figure B.1 anthill

B.3 Weak and strong emergence

The concept of emergence is divided into two categories based on the perspectives. They are 'weak emergence' and 'strong emergence'. Weak emergence describes new properties arising in systems as a result of the interactions at an elemental level. Emergence, in this case, is merely part of the language, or model that is needed to describe a system's behavior.

On the other hand there can be systems with qualities not directly traceable to the system's components, but rather to how those components interact, and one is willing to accept that a system supervenes on its components, then it is difficult to account for an emergent property's cause. These new qualities are irreducible to the system's constituent parts. The whole is greater than the sum of its parts. This view of emergence is called strong emergence. Some fields in which strong emergence is more widely used include etiology, epistemology and ontology.

B.4 Polanyi's hierarchical systems and levels of emergence

In [37] Polanyi argues that the information contained in the DNA molecule is irreducible to physics and chemistry. Although a DNA molecule cannot exist without physical properties, these properties are constrained by higher level ordering principles. Polanyi advocates emergence i.e. the claim that there are several levels of reality, and causality. His argument relies on the assumption that boundary conditions supply degrees of freedom that instead of being random are determined by higher level realities whose properties are dependent, but distinct, from the lower level from which they emerge. He further explains with examples from language domains how words formed with letters and property syntax emerges at word level and then sentences level semantics emerges. In these hierarchical system the whole is always greater than sum of the parts hence they are irreducible. These ideas inspired the concepts of holism.

B.5 Holism

Holism is the idea that all the properties of a given system (physical, biological, chemical, social, economic, mental, linguistic, etc.) cannot be determined or explained by its component parts alone. Instead, the system as a whole determines in an important way how the parts behave. The term holism was coined in 1926 by Jan Smuts. His definition of holism echoed the concept concisely summarized by Aristotle in the *Metaphysics*: "The whole is different from the sum of its parts".

B.6 Emergentism

In philosophy, emergentism is the belief in emergence, particularly as it involves consciousness and the philosophy of mind, and as it contrasts with reductionism. A property of a system is said to be emergent if it is in some sense more than the 'sum' of the properties of the system's parts. An emergent property is said to be dependent on some more basic properties therefore it can have no separate existence. However, a degree of independence is also asserted of emergent properties, so that they are not identical to, or reducible to, or predictable from, or deducible from their bases. The different ways in which the independence requirement can be satisfied lead to various sub-varieties of emergence.

B.7 Society of Mind theory

In his book describing natural intelligence Marvin Minsky argues that Mind can be modeled as a 'society' comprising agents [26]. The word society is explained there as a system where distributed entities interact. As a result of these interactions of the elementary entities called agents, intelligence is occurred. A later extension of this theory in Emotion machine [27] Minsky describes these agents as critics or selectors which act as a switching mechanism to generate certain emotion.

B.8 Summary

Some ideas on complex systems and emergence developed throughout the history are briefly discussed. It is observed that these ideas are built parallel in several domains such as economy, philosophy, biological sciences and computer sciences. We found these ideas inspirational in order to propose an emotion model where emotions are represented as a complex system and mind state is occurred as emergent phenomena.



University of Moratuwa, Sri Lanka.
Electronic Theses & Dissertations
www.lib.mrt.ac.lk

Implementation Details

C.1 Introduction

In this appendix section a more descriptive view of the implementation is presented in terms of class descriptions focusing on application program interface (API) of the system. Public functions in each class are listed and most important functions are described. This description is presented for module wise categorized system. The complete source is basically developed in five sub projects initially the agent library, then *EME* application, Game world and personality and finally *OCCTest* project developed for testing purposes. Game world is named as Snake and Ladders in packaging for clarity and similarly personality is developed under the name of *PlayerPersonal* packages which mimics the personality of a player. We find this inclusion of API descriptions as an appendix is essential for the reusability of the software and the extensibility of the project as a whole. A glimpse of project listing could be found below (See Figure C.1)

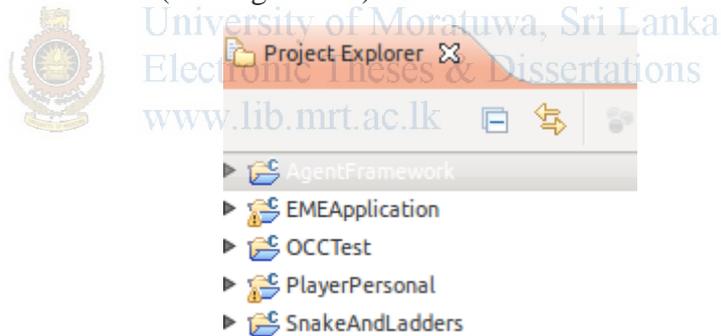


Figure C.1: Projects

Let us discuss the classes and methods in each package separately.

C.2 Agent Framework project

The list of classes in the project framework is as below (See Figure C.2). Let us briefly discuss each class in the project.

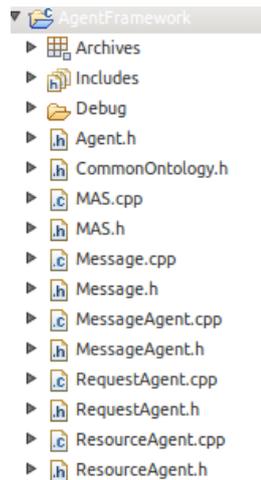


Figure C.2 Agent Framework

C.2.1 Agent

Class agent comprises of the most abstract interface for the agents in the framework. The public interface is defined as follows. (See Figure C.3)

```

class Agent
{
public:
    Agent() {};
    virtual ~Agent() {};

    void SetId(int iId) { i_Id = iId; };
    int GetId() { return i_Id; };

    void SetName(std::string zName) { z_Name = zName; };
    std::string GetName() { return z_Name; };

    virtual void SetOntology(CommonOntology* pOntology) { p_Ontology = pOntology; };

    virtual void OnTimer() = 0; // Captures timely events
    virtual void OnMessage(const Message& rMessage){};

```

Figure C.3 Public interface of Agent

The method *OnTimer* defines the operation of the agent in a timer tick. This is unique to each agent type hence declared abstract in base class. *OnMessage* method is also defined to use in different kind of implementation where agents are evoked when message is targeted, not when timely event occurred. This method is added for extensibility of implementation as this is a library which could be used in different type of agent applications.

C.2.2 Common Ontology

Class common ontology represents the ontology of the agent application. The implementation of this class is application dependant. For the framework this is added for understandability.

C.2.3 MAS

MAS is the controller class so as the initiator. The set of agents are provided to MAS and there it creates a MAS application accordingly allocating memory and put agents on run. The public interface of MAS is as below. (See Figure C.4)

```
class MAS
{
public:
    MAS(const std::list<RequestAgent*>& _listRequestAgents, const std::list<ResourceAgent*>& list_ResourceAgents, MessageAgent* _pMessageAgent);
    virtual ~MAS();

    void Run();//run the MAS application
    void OnTimer();
```

Figure C.4 Public interface of MAS

Run function is the initiator of the application where all agents are started and put to run. In onTimer of MAS timer event of each agent is called.

C.2.4 Message

Class message serves as the container passed in communication among agents. Similar to standard ACL message interface here also sender , receiver ids , sequence id, and a map to hold any other custom field in which data part has string are defined as attributes. The public interface as in Figure C.5. There library users can access these attributes via getter and setter functions defined.

```
class Message
{
public:
    Message();
    virtual ~Message();

    /*Getters and Setters*/
    void SetSequenceId(int iSeqId){i_SequenceId = iSeqId;};
    int GetSequonceId() const {return i_SequenceId;};

    int GetMessageType() const {return i_MessageType;}; //returns message type
    void SetMessageType(int iMsgType){i_MessageType = iMsgType;}; //sets message type

    std::string GetField(int iFieldId) const;
    void SetField(int iFieldId, std::string zData);

    void SendSenderId(int iId){i_SenderId = iId;};
    int GetSenderId(){return i_SenderId;};
```

Figure C.5: Public interface of Message

C.2.5 Message Agent

Class message agent represents the coordinator of an agent system. The level of control or coordination could be differed per application. That could be implemented in OnTimer function. Some applications could have a message agent with full control there in message agents onTimer it could broadcast or send messages. In another

application a message agent could be a pseudo controller and it merely passes the messages when received. For that kind of implementations the *OnMessage* method could be used. Basic interface is as below (See Figure C.6)

```
class MessageAgent: public Agent
{
public:
    MessageAgent();
    virtual ~MessageAgent();

    void RegisterRequestAgent(RequestAgent* _pAgent);
    void RegisterResourceAgent(ResourceAgent* _pAgent);
    void RegisterRequestAgents(std::list<RequestAgent*> _listAgents);
    void RegisterResourceAgents(std::list<ResourceAgent*> _listAgents);
    void RegisterResourceAgentForMessage(ResourceAgent* _pAgent, int _iMessageType);

    void OnMessage(Message & _rMessage, int _iId = 0);
    void SendTo(Message & _rMessage, int _iSequenceId);

    std::vector<Message> GetMessageSpace(int _iSequenceId);
    const Message & GetLatestMessage();
    const Message & GetSpecificMessage(int _iSequenceId);

    void OnTimer();
};
```

Figure C.6: Public interface of *MessageAgent*

Register functions are required in order register each type of agents in a MAS with the controller or the message agents. This is enabled by sending a list of agents or per single agents. The *GetMessageSpace* function is a significant method which provides a list of updated messages given a sequence id. Other getters also support variations of this to get the single latest update (*GetLatestMessage*) or to get a given message with specific sequence id (*GetSpecificMessage*).

C.2.6 Request Agent

Class Request Agent provides a template for any kind of agent who could submit messages in a form of request to message space. Within its *OnTimer* it could call *OnMessage* of message agent. For that request agent should have link to message agent. With *RegisterMessageAgent* a particular message agent instance is registered with request agent. Hence the public interface of request agent is simply one addition to super class interface (*Agent*).

C.2.7 Resource Agent

Interface of resource agent is also similar to of request agents other than having some additional functions for message receiving. *OnMessage* , *OnMessages*, *OnRegisteredMessages* are variations of this. With *OnMessage* resource agent can have particular message sent for it and with *OnMessages* a set of such messages and with

RegisteredMessages a type of message the agent has previously requested from message agent. Hence the public interface of resource agent would be like below. (See Figure C.7)

```
class ResourceAgent: public Agent
{
public:
    ResourceAgent();
    virtual ~ResourceAgent();

    void RegisterMessageAgent(MessageAgent* _pMessageAgent); //regist
    void RegisterForMessages(int _iMessageType);

    virtual void OnMessage(const Message& _rMessage) {};
    virtual void OnMessages(std::vector<Message> _vecMessages) {};
    virtual void OnRegisteredMessage(const Message& _rMessage) {};
```

Figure C.7: Public interface of *ResourceAgent*

C.3 EME Application project

The core project related to our research is *EME* application project. Here our emergent model of emotion is implemented in terms of a *MAS* application where each emotion factor is represented by agents of that *MAS*. Hence it comprises of class for 52 emotion factors as extension of request resource and message agents. The basic view of the project is as follows. (See Figure C.8). With the sub folder emotion factors the classes for the emotion factors are implemented.

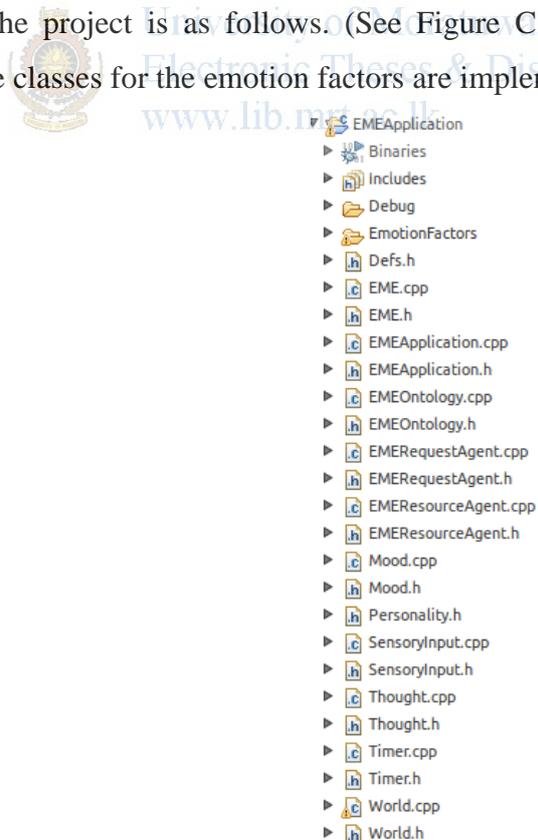


Figure C.8: *EME* application project

Let us discuss the classes in this project briefly.

C.3.1 EME

EME represents the agent architecture we designed comprising our emergent model. *EME*'s *onTimer* function calls the inside MAS timer event. For the construction of *EME* a defined personality and a world should be provided. Hence the public interface of *EME* looks like below. (See Figure C.9)

```
class EME
{
public:
    EME(Personality* _pPersonality, World* _pWorld, bool _bFeedbackEnable = true, bool _bTimeDecayEnable = true);
    virtual ~EME();

    void OnTimer();
};
```

Figure C.9: Public interface of *EME*

C.3.2 EMEApplication

This is the application class of the project. Two constructors are defined one where required attributes for the application (world and personality) could be read from a configuration file (*ReadConfigs*) or else to set as input parameters to constructor. *EMEs* could be added by providing personality or else this could be again done using setting a configuration file. *OnTimer* function of the application calls the *onTimer* functions of the set of *EME* related to application and the *onTimer* of the related world instance. Hence the public interface is defined as below (See Figure C.10)

```
class EMEApplication
{
public:
    EMEApplication(World* _pWorld, bool _bFeedbackEnable = false, bool _bTimeDecayEnable = false);
    EMEApplication();
    virtual ~EMEApplication();

    void AddEME(Personality* _pPersonality);
    void ReadConfigs();

    void OnTimer();//called by timer
    void Init(World*,std::vector<Personality*>);
};
```

Figure C.10: Public interface of *EME* Application

C.3.3 EMEOntology

EMEOntology comprises of the common information used by *EME* agents. The set of thoughts set of emotion factors and their combinations for each thought according to the Buddhist philosophy are preserved in the ontology. This knowledge is used when identifying the current thought considering the various possible emotion factor combinations. Hence the public interface of *EMEOntology* consists of utility methods to retrieve these static information. (See Figure C.11). Please note that these information are merely basic static data not like rules defined in personality.

Personality is also an ontology for the system which comprises of rules. This differs from the mere data definitions contained in the *EMEOntology*, hence implemented separately.

```
class EMEOntology: public CommonOntology
{
public:
    EMEOntology();
    virtual ~EMEOntology();
    EMEResourceAgent GetEmotionFactor(int _iFactorId){return arr_Emotions[_iFactorId];};
    Thought* GetThought(int _iId){return &arr_Thoughts[_iId];};
    std::string GetEmotionName(int _iId){return arr_Emotions[_iId].GetName();};
    std::string GetThoughtName(int _iId){return arr_Thoughts[_iId].GetName();};
    int GetThoughtCount(){return THOUGHT_COUNT;};
};
```

Figure C.11: Public interface of *EMEOntology*

C.3.4 EMERequestAgent

EMERequest agent is the derived version of request agent for our application. It observes world events in terms of sensory inputs and submits that sensory information. Therefore public methods of *EME* Request agent are *onTimer* function and setters for related word and personality instance. (See Figure C.12)

```
class EMERequestAgent: public RequestAgent
{
public:
    EMERequestAgent(int _iId);
    EMERequestAgent();
    virtual ~EMERequestAgent();
    virtual void OnTimer();
    void SetWorld(World* _pWorld){p_MyWorld = _pWorld;};
    void SetPersonalityId(int _iId){i_Personality = _iId;};
};
```

Figure C.12: Public interface of *EMERequestAgent*

C.3.5 EMEResourceAgent

Similar to *EMERequest* agent this class also consists of setters for attributes and derived functions from superclass *ResourceAgents* for message receiving. Therefore the public interface is as below. (See Figure C.13)

```
class EMEResourceAgent: public ResourceAgent
{
public:
    EMEResourceAgent();
    virtual ~EMEResourceAgent();

    virtual void OnTimer();
    virtual void OnMessage(const Message& _rMessage); //called by message age
    void OnMessages(std::vector<Message> _vecMessages);
    void OnRegisteredMessage(const Message& _rMessage); //when registered for

    void SetOntology(EMEOntology* _pOntology);
    void SetPersonality(Personality* _pPerson){p_Personality = _pPerson;};
    void SetWorld(World* _pWorld){p_World = _pWorld;};
    void SetTimeDecayEnable(bool _bEnable){b_TimeDecayEnabled = _bEnable;};
    void SetName(std::string _zName){z_Name = _zName;};
    std::string GetName(){return z_Name;};
    void Init();
};
```

Figure C.13: Public interface of *EMEResourceAgent*

C.3.6 Mood

Class *Mood* represents an important component of our system mood which is considered as the second level of emergent result. Mood is emergent result of the thoughts. Hence mood's interface consists of methods to update given thought information and to simulate timely decay and to simulate the decay of pulse generated by thought etc. Additionally this consists of access methods of various attributes of mood which are used by cognitive process of the system. Public interface of mood is as below.

```
class Mood
{
public:
    Mood();
    virtual ~Mood();
    void UpdateMood(Thought* pThought);//update mood according to recent thoughts;
    int GetIntensityFactor(int iThoughtId){return matrix_MoodIntensityFactors[iThoughtId][i_CurrentMoodType];};
    int GetCurrentMoodType(){return i_CurrentMoodType;};
    static int DeriveMoodTypeForThought(int iThoughtId);
    static int GetIntensityFactor(MOOD_TYPE etMoodType,int iThoughtId){return matrix_MoodIntensityFactors[iThoughtId][etMoodType];};
    static std::string GetMoodName(int iMoodType){return map_MoodNames[iMoodType];};
    static float GetCurrentMoodIntensity(){return f_Intensity;};
    static int GetIntensityLevel(float fIntensity);
    static void Init();
    void DecayImpulseGeneratedByThought(int iThoughtId,float fIntensity);
    void DoTimeDecay();
};
```

Figure C.14: Public interface of *Mood*

C.3.7 Personality

Abstract interface for personality declared in *EMEApplication* project. Implementation of Personalities is done as separated project which creates a dynamic link library. The public interface required to access that personality instance for the class in *EMEApplication* is provided through this abstract class. Access to retrieve personality information (*GetId*, *GetName*, *GetSkillLevel*) and other relevant data used in decision making (*GetProbableActions*) are declared here. Most importantly all the methods used by emotion factors when determining their arousal and intensities are declared in personality. Hence the abstract public interface of personality is as follows. (Please note that it's an extract the same notation continues for all methods. See Figure C.15)

```

class Personality
{
public:
    Personality(int _iId, std::string _zName){i_MyId = _iId; z_Name = _zName;};
    virtual ~Personality();};

public:
    virtual int IsLike(const Object & _rObject)= 0;
    virtual int IsDisLike(const Object & _rObject)= 0;

    //actions
    std::vector <Action*> GetProbableActions(MOOD_TYPE _etMood){return map_ProbableActions[_etMood];};
    std::string GetName(){return z_Name;};
    int GetId(){return i_MyId;};
    int GetSkillLevel(){return i_EnergyLevel;};

    //general personality

    //for universals
    virtual std::string CheckVedhana(const SensoryInput& _rInput)= 0;//feeling
    virtual std::string CheckSanna(const SensoryInput& _rInput)= 0;//sensation
    virtual int CheckChethana(const SensoryInput& _rInput)= 0;//volition
    virtual int CheckManasikara(const SensoryInput& _rInput)= 0;
    virtual int CheckEkaggata(const SensoryInput& _rInput)= 0;//focus
    //for particulars
    virtual int IsRelavantForInitialApplication(const SensoryInput& _rInput) = 0;//vitharka
    virtual int IsRelavantForSustainedApplication(const SensoryInput& _rInput) = 0;//vichara
    virtual int IsRelavantForDecision(const SensoryInput& _rInput) = 0;//adhimokkha
    virtual int IsRelavantForEffort(const SensoryInput& _rInput) = 0;//viriya
    virtual int IsRelavantForJoy(const SensoryInput& _rInput) = 0;//preethi
    virtual int IsRelavantForConation(const SensoryInput& _rInput) = 0;//chanda

```

Figure C.15 Abstract public interface of *Personality*

C.3.8 SensoryInput

SensoryInput is a container class which represents a particular sensory input. Utility methods are provided in order to serialize (*FillMessage*) sensory objects to messages and to deserialize from message (*ParseMessage*). The simple interface of sensory input is as below. (See Figure C.16)

```

class SensoryInput
{
public:
    SensoryInput();
    virtual ~SensoryInput();

    bool bRecieved;
    bool bTaken;
    bool bGiven;
    Action* pAction;
    Object* pObject;
    Person* pReceiver;
    Person* pSubject;
    bool b_Empty;

    bool IsEmpty(){return b_Empty;};
    static SensoryInput ParseMessage(const Message& _rMessage, World* _pWorld);
    int GetId(){return pAction->i_ActionId;};
    void FillMessage(Message&);

```

Figure C.16: Public interface of *SensoryInput*

C.3.9 Thought

Thought is implemented as a container class too. Other controller classes could access thought using setters and getters. Hence the public interface of thought contains a set of assessor, mutator methods. (See Figure C.17)

```

class Thought
{
public:
    Thought(std::vector<int> _vecContainingEmotionFactors, int _iId);
    Thought();
    virtual ~Thought();
    void SetId(int _iId){i_Id = _iId;};
    int GetId(){return i_Id;};
    void AddEmotionFactor(int iEmotionFactor);
    void SetIntensity(float fIntensity){f_Intensity = fIntensity;};
    float GetIntensity(){return f_Intensity;};
    int GetEmotionFactorCount(){return vec_ContainingEmotionFactors.size();};
    int GetEmotionFactorId(int iIndex);
    int GetEmotionFactor(int iIndex){return vec_ContainingEmotionFactors[iIndex];};
    void SetName(std::string _zName){z_Name = _zName;};
    std::string GetName(){return z_Name;};
    void SetContents(std::string _zContents){z_Contents = _zContents;};
    std::string GetContents(){return z_Contents;};
}

```

Figure C.17: Public interface of *Thought*

C.3.10 Timer

Timer class is the scheduler of the application. Similar to context switching mechanism done at CPU level scheduling *Timer* class does switching between running components of the application system. Hence the public interface contains the single unction called *Run* where that switching algorithm is implemented. (See Figure C.18)



```

class Timer
{
public:
    Timer(EMEApplication* _pApplication);
    virtual ~Timer();

    void Run();
}

```

Figure C.18 Public interface of *Timer*

C.3.11 World

Similar to personality world also developed as a dynamically pluggable module and the abstract interface of that library is added to *EMEApplication* project. Here we have that abstract interface with the basic functionality of the world declared. As we have designed the world as having a set of objects, actions and persons defined the interface is provided with the getter and setter functions to access those attributes. Therefore abstract interface of the world is as below (See Figure C.19)

```

class World
{
public:
    World();
    virtual ~World();

    std::vector<Event*> GetLatestEvents(int _iLastEventId);
    void AddEvent(Event& _rEvent);

    Object* GetObject(int _iId){std::map<int,Object*>::iterator it = map_Objects.find(_iId); if(it != map_Objects.end()) return it->second;else return NULL;}
    Person* GetPerson(int _iId){std::map<int,Person*>::iterator it = map_Persons.find(_iId); if(it != map_Persons.end()) return it->second;else return NULL;}

    void AddAction(Action& _rAction) { map_Actions[_rAction.i_ActionId] = new Action(_rAction); };
    Action* GetAction(int _iId) {std::map<int,Action*>::iterator it = map_Actions.find(_iId); if(it != map_Actions.end()) return it->second;else return NULL;}

    virtual std::vector<Action*> GetPossibleActions(SensoryInput& _rInput,int _iPersonId) = 0;

    virtual std::vector<Action*> GetPossibleActions() = 0;

    virtual void OnTimer();

    virtual void Init() = 0;
}

```

Figure C.19: Abstract interface of *World*

C.3.12 Emotion Factors

Set of Emotion factors are implemented as a sub package within the *EMEApplication* project. There special emotion factors having unique responsibility in the system such as *Sati* performing as system monitor *Pañña* as cognitive processor and the set of universals and *Phassa* performing as request agent are implemented in separate files. All the rest of classes representing emotion factors are packaged into a single file called *EmotionFactor*. (See Figure C.20)



University of Moratuwa, Sri Lanka.
 Electronic Theses & Dissertations
 www.lib.moratuwa.lk

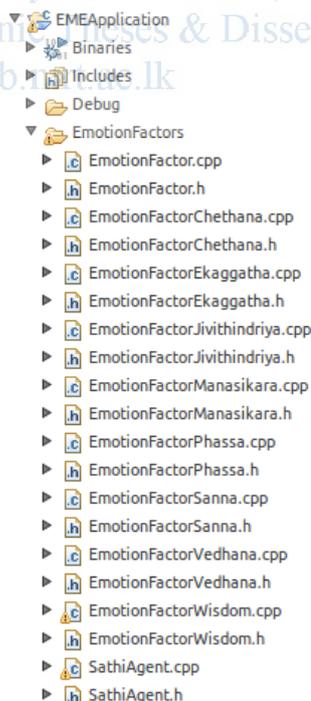


Figure C.20: Emotion factors package

All the classes within *EmotionFactor.h* and universals have common interface in which a public function named *IsRelavant()* is declared. Within this function unique

implantation for each emotion factor is defined. This function is called when to determine the emotion arousal on an input and to retrieve intensity.

EmotionFactorPhassa is a derived class from *EMERRequestAgent*. Therefore the public interface is similar to what we have described above.

EmotionFactorWisdom serves as the cognitive processor of the system, which is a resource agent of the system. Action selection and Expression updating is done there. Various methods for action selection based on the criteria are implemented in this. Public interface is as below. (See Figure C.21)

```
class EmotionFactorWisdom: public EMEResourceAgent
{
public:
    EmotionFactorWisdom();
    virtual ~EmotionFactorWisdom();

    void OnTimer();

private:
    void OnMessage(const Message & _rMessage);
    void ProcessMessage(const Message & _rMessage);
    bool IsRelavant(const Message& _rMessage);

    Action* SelectAction(SensoryInput* _pInput, MOOD_TYPE _etMood, int _iThoughtId);//Action is affected by thoughts witch are crea
    Action* SelectNonPlayAction(MOOD_TYPE _etMood,int _iThoughtId);
    std::vector<Action*> DeriveProbableActions(MOOD_TYPE _etMoodType);
    std::vector<Action*> DerivePossibleWorldActions(SensoryInput& _rInput);
    Action* FindMostSuitableAction(std::vector<Action*>& _vecProbableActions, std::vector<Action*>& _vecPossibleActions,MOOD_TYPE);

    void SelectExpression(Expression & _rExpression,MOOD_TYPE _etMoodType,float _fMoodIntensity,SensoryInput* _pInput = NULL);
};
```

Figure C.21: Public interface of *EmotionFactorWisdom*

EmotionFactorSati operates as system monitor. It is also a resource agent derived from *EMEResoureAgent*. Therefore public interface consists of base class methods for messages receiving and base *OnTimer* method defined at agent level (See Figure C.22). The monitoring is implemented in this *OnTimer* method. Another utility method is used to enable the system feedback. If this is enabled current thought is fed back to the mind again (implemented as a message submitted to message space containing current thought information)

```
class SathiAgent: public EMEResourceAgent
{
public:
    SathiAgent();
    virtual ~SathiAgent();

    void OnTimer();
    void SetFeedbackEnable(bool _bEnable){b_FeedbackEnable = _bEnable;};
};
```

Figure C.22: Public interface of Sati agent

C.4 Game World – Snake and Ladders project

GameWorld project is developed as application project supported with GUI. The abstract interface for world provided at *EMEApplication* project serves as the basis for the implementation of this project. Project package basically consists of two header files *GameWorld* containing set of classes to represent game world and *SnakeAndLadders* to represent user interface class. (See Figure C.23). Additionally main file of the program is added to here as to invoke the game application. All other projects are linked to these libraries.

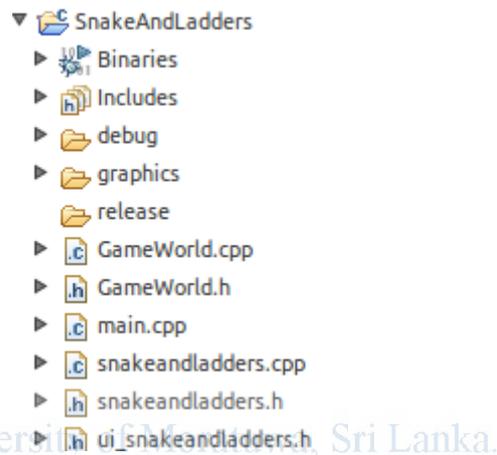


Figure C.23 Project *SnakeAndLadders*'s

C.4.1 GameWorld

Public interface of *GameWorld* consists of class *GameWorld* and several class declarations which are representing the entities in game world such as Pebble, Board and Die etc. *GameWorld* acts as the controller class of the projects which receives signals from user interface and *EMEApplicaitons* and then executes those requests and publish the results. Pebble, Board, and Die are container classes used by *Gameworld*. (See Figure C.24)

```

class Pebble
{
public:
    Pebble();
    virtual ~Pebble();

    int i_Player;
    int i_Position;
};

class Board
{
public:
    Board();
    virtual ~Board();

    int GetLinked(int _iPos){return map_LinkedPositions[_iPos];};
    void Move(Pebble* _pPebble,int _iPos){map_PebblePositions[_pPebble] = _iPos;};

    std::map <Pebble*,int> map_PebblePositions;
    std::map <int,int> map_LinkedPositions;
    std::map<int,int> map_Snakes;
    std::map<int,int> map_Ladders;
};

class Die
{
public:
    Die();
    virtual ~Die();
    enum etSide
    {
        Side_1,
        Side_2,
        Side_3,
        Side_4,

        >side >,
        Side_6
    };
    etSide et_CurrentType;
};

```

Figure C.24: Interface of *GameWorld's* container classes

The basic functionality of *GameWorld* is as defined in super class world. Pending events are added to worlds queue by other applications (in this case *EME* ,*OCCTest* or GUI class *SnakeAndLadders*) and world executes them in *OnTimer* and puts results to executed events queue. The implementation of event execution is unique to game world as in here event in snake and ladder game are considered. The public interface of *GameWorld* class is as follows. (See Figure C.25). In construction a link to related user interface (*SnakeAndLadders*) and the links to players should be provided to *GameWorld*.

```

class GameWorld:public World
{
public:
    GameWorld(SnakeAndLadders*,std::vector<Person*>);
    virtual ~GameWorld();

    void Init();
    void OnTimer();

    std::vector<Action*> GetPossibleActions(SensoryInput& _rInput,int _iPersonId);
    std::vector<Action*> GetPossibleActions();//get nonpaly actions
    virtual void OnExpression(Expression _rExpression,int _iPersonId);
    Person* GetTeamPlayer(int _iTeamId){return map_Teams[_iTeamId];};
};

```

Figure C.25: Public interface of *GameWorld*

C.5 Personality – PlayerPersonal project

Implementation of the abstract interface declared in *EMEApplication* project is done as a separate project which built a dynamically pluggable library module. This project is named *PlayerPersonal* as our application is a game. The interface is same to as we discussed in Personality class. The likes, dislikes and scenarios of emotion arousal are implemented as we develop specific kind of personality. For player personal basically objects and persons and actions are defined relevant to game world. These definitions are added to a (Defs.h) file in *EMEApplication* which is used by both world and personality.

C.6 OCCTest project

Finally let us discuss the project we developed for testing purposes. As a control experiment we use OCC. This project comprises of a single class which represents OCC agent. As OCC model states this agent generates emotions as valence reactions of three types of stimuli, objects, persons and events. Therefore the data structure defining these information is added to *OCCTest* class. The common definitions used in Defs.h which used by *GameWorld* and *PlayerPersonal* projects are used here also so that *OCCTest* agent also has emotion responses to same kind of objects, persons and events considered in *EME* also. This agent also adhered to common scheduling mechanism used throughout the development that timer tics. Therefore public interface consists of *OnTimer* function within which operational model of OCC is implemented. (See Figure C.26)

```
class OCCTest
{
public:
    OCCTest(World* _pWorld,int _iId);
    virtual ~OCCTest();
    void OnTimer();
};
```

Figure C.26: Public interface of *OCCTest*

C.6 Summary

Implementation at class level is discussed with relevance to application programming interface (API) provided by each class. Module wise decomposition is realized via separated projects built as library modules which could be linked dynamically at run time. API's are defined in simple self descriptive manner where reusability and extensibility of software is supported.