Defining the Semantics of Agent Based Modelling in InterDyne

Leo Carlos-Sandberg
Supervisor: Dr Christopher D. Clack

May 29, 2017

Abstract

This paper defines the semantics of agent based modelling within the Interdyne simulator. A converter has been created that links an agent based model step wise to a difference equation based language, this language is created in such a way that it is directly relatable to lambda calculus which can then be used as definition of the semantics of the agent based model.

Contents

1 Introduction **3**

2 InterDyne **3**
2.1 Emergent Behaviour from Interaction Dynamics . . . 3
2.1.1 Feedback Loops . . . 4
2.2 Applicability to Finance . . . 5
2.2.1 Deterministic . . . 6
2.2.2 Message Delays . . . 6
2.2.3 Message Passing . . . 7
2.2.4 Storing Passed Messages/Tracking . . . 7
2.2.5 Discrete Time . . . 7
2.2.6 Message Ordering . . . 7
2.2.7 Agents . . . 8
2.3 InterDyne Detail Operation . . . 8
2.4 Examples . . . 9
2.4.1 Hot Potato . . . 9
2.4.2 Flash Crash . . . 10

3 Description and analysis of the problem **10**
3.1 Why InterDyne is not enough . . . 10
3.1.1 One-to-Many Problem . . . 10
3.1.2 Semantics of the System . . . 10
3.1.3 Inverse Function Problem? . . . 10
3.2 Difference equations . . . 10
3.2.1 Benefits and Problems . . . 10
3.2.2 Lack of Visualisation . . . 10
3.2.3 Optimisation Problems . . . 10
3.3 Two Views Approach . . . 10

4 Converter **10**
4.1 New Language . . . 10
4.1.1 Syntax/Grammer . . . 10
4.1.2 Parser . . . 10
4.1.3 Examples . . . 10
4.1.3.1 Simple Example . . . 10
4.1.3.2 Complex Example . . . 10

5 Conclusion **10**
5.1 Further Work . . . 10

6 Appendix **10**
6.1 Appendix 1 . . . 10

References **10**

1 Introduction

This research was inspired by the "Flash Crash" of 2010, this crash can be seen as a change in phase between a stable and unstable market; it has been shown that emergent behaviour in physical systems can lead to phase transitions and postulated that this could also be true within the financial markets [1]. InterDyne hence exists to see if this flash crash and ones like it could be examples of this critical behaviour caused by emergence within the markets

2 InterDyne

InterDyne is bespoke simulator created by Clack and associates [2], it is a general-purpose simulator for exploring emergent behaviour and interaction dynamics within complex systems.
InterDyne design is that of an agent-based model interacting via a harness. This creates a structure of individual autonomous agents who interact through messages sent to one another.
Similar to other agent-based models InterDyne operates in discrete-time rather then continuous time. These quantised time chunks which move the simulation forward can be left with out proper definition, simply having things defined in a number of time steps, or they can be equated to a real time usually with the smallest time gap needed be a single time step and then all other timings being integer multiples of this. This discrete time is most important to message passing, meaning messages between agents are only sent on a integer time step.
Massages in InterDyne are just small packets of data, such as a series of numbers. Agents can only communicate via these messages, meaning that any emergent behaviour observed, that is not directly due to one agent, must be caused by linking of agents mediated by these messages. An agent can send private messages that are only received by a single other agents, one-to-one messages, or can send broadcast messages received by a number of agents, one-to-many messages. To facilitate this a communication topology can be made for InterDyne, this is done in the form of a directed graph determining which agents can communicate with each other. Due to the directional nature of these messages this topology could allow an agent to send messages to another but not be able to receive messages from that same agent. Messages have a defined order to them, an agent will, unless otherwise instructed, always process messages in the order in which they arrive. To change the order in which messages arrive delays can be added to communication paths between agents, this can be a static delay which always applies to messages sent from one agent to another, meaning this will arrive a set number of time steps later. Or a more complex dynamic delay, which is achieved by using another agent to mediate the passing of these messages delaying by an amount decided on in some internal logic. All messages in InterDyne are passed through a harness, this does not alter the messages or delay them[1], but does store the messages and their order which can be used in post analyse.
Each of the agents within an InterDyne simulation can be completely unique and modelled to different levels of complexity, allowing system components to be created to the level needed for the required experiment. As a whole InterDyne simulations are deterministic, repeated experiments will return identical results. However non-determinism can be added via the agents, making some part of an agent stochastic will lead to repeated experiments on the whole returning different results. A pseudo-random element can also be added by instructing InterDyne to randomly sort the message order for any agent receiving multiple messages in one time step. This is only pseudo-random as, as long as the same seed is used each run of the simulation will order the rearranged messages in the same way.

2.1 Emergent Behaviour from Interaction Dynamics

As mentioned previously InterDyne is designed to allow the simulation and investigation of emergent behaviour caused by interaction dynamics.

[1] Unless instructed to using the static delay.

Emergent behaviour is a term used to describe macro-behaviour of a system that is not obvious from analysis of the micro-behaviour of the system, more formally this is behaviour that can not be predicted through analysis of any single component of a system [3].
A misunderstanding of emergence can lead to the fallacy of division, this is that a property of the system as a whole most also be a property of an individual component of the system; water for example has a number of properties including being able to be cooled down to become ice and heated to become steam, saying the same must also be true of a molecule of water however is incorrect. This concept continues into economics, being called the fallacy of composition, where what is true for the whole economy my not hold for an individual and vice versa [4].
A simple way to demonstrate emergence is in the Game of Life [5], which is an example of cellular automaton; this game takes place on an infinite two-dimensional grid in which cells can either be 'alive', coloured for example green, or 'dead', a different colour usually black. Wether a cell is 'alive' or 'dead' is based on a set of simple rules:

1. 'Alive' cells will transition to be 'dead' cells in the next time step if they have few than two 'alive' neighbours.
2. 'Alive' cells with two or three 'alive' neighbours remain 'alive' at the next time step.
3. 'Alive' cells will transition to be 'dead' cells in the next time step if they have more than three 'alive' neighbours.
4. 'Dead' cells with exactly three 'alive' neighbours will transition to 'alive' at the next time step.

With this simple set up very complex patterns evolving through time can be created, these patterns can be seen as emergence, with an individual cell not being able to encapsulate this behaviour.
Emergent behaviour can be seen occurring naturally, with physics offering many well explored examples, for instance the n-body problem [6]. This historically is explained as n planets in proximity to each other, who interact via gravity in accordance with Newtons laws, this interaction gives rise to motion which is unsolvable analytically and can be viewed as an emergent property of the system. There is a certain amount of intuition with this observation, this is a very complex system and hence it seems reasonable that the interactions may not be analytically solvable. Also as likely expected a case with only one two bodies does have a solution, however the three-body case does not have a solution and generates this emergent behaviour. This outcome is not as obvious and many may assuming that such a simple system would not express emergent behaviour, the three-body problem is hence a good example of emergent behaviour in a simple system within nature.
Emergent behaviour can be generated in many-body systems, in which these bodies them selfs do not inherently create the emergent behaviour but interactions between them give rise to it. For example in the planet n-body problem interactions are via gravity which passes information about each body to the others, such as a planets mass and momentum, however if the gravity was some how removed, there would be no emergence and the planets would all move with a constant velocity [7].

2.1.1 Feedback Loops

Emergent behaviour from interaction dynamics can take a number of forms, with a prominent type being feedback loops.
Feedback loops are where the input information to an entity is dependent on the output information of that same entity, from a previous moment in time. For this to be worth investigating all entities involved must be set up so that they make decisions about their output, such as what messages they send or who they send them to, based on some input information.
A very simple example could be two algorithms who send each other messages, see Fig 1, where the message algorithm 1, A1, sends to algorithm 2, A2, is twice what it received and A2 sends A1 three times what it received. It is trivial to find that if the initial input for A1 was 1 then the subsequent four

inputs would be: 6, 36, 216, 1296. However if A2 always returned the same value, say 4, this would not be considered a feedback loop as its result would be constant and trivial.

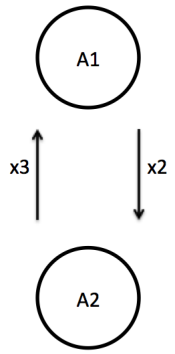

Figure 1: *Feedback loop between two algorithms A1 and A2, with each algorithm altering its output dependent on its input. A1s output is twice its input and A2s output is three times its input.*

This example was very simple and feedback loops can be much more complex, encompassing any number of entities, each of whom can have very complex algorithms for transforming their inputs. Feedback loops can operate across time, meaning that an event in the past can feedback to a present decision. A simple example being a single entity sending itself a message to process in the future effecting its actions at that time. For a feedback loop containing a large number of entities the time scale on which the feedback occurs can be come significantly large.
Not all feedback loops have a negative impact, some can be stabilising feedback loops due to a benign effect.
Feedback loops can be present in a system in two ways, either they can be a constant fixture, a static feedback loop, or they can form and change, dynamic feedback loops. A static feedback loop is present in the system from the start whether this is intentional and known, or unintentional and unknown to the members of the system. Dynamic feedback loops my not be present at the start and can form and change over time, with new entities joining or leaving them, allowing them to increase or decrease in size or effect, to split or merge, or to disappear.
Due to the potential complexity of feedback loops both in construction and in time, they can be difficult to detect therefore methods are usually used to expose them. For static loops, forms of static analysis can be used such as, analysing initial setup, this is possible since the loops do not change through out time. Dynamic loops can be much harder to observe and analyses, an important aspect to detecting these loops is the interactions, messages sent between different entities within the system. Since the loops can evolve over time being able to track and analyse these messages over a time series is vitally important for the analysis of these loops, this time dependent analyse is called dynamic analyse.

2.2 Applicability to Finance

Though InterDyne is a general purpose simulator, its main use thus far has been the exploration of financial markets. In particular InterDyne has been used to explore "Flash Crash" of 2010, during which market prices and rational valuations became disconnected, with some stocks trading as low as a penny per share, this lead to frenzied trading and irrational prices which spread between markets causing a massive price crash [8]. This event lasted around 36 minutes and has been described as"one of the most turbulent periods in their history" for the US financial markets [9].
The hypotheses for this crash which InterDyne exists to investigate, is that this crash is an emergent

phenomenon caused by the interaction between High Frequency Traders (HFTs) within the market.
HFTs are a subset of algorithmic traders who normally participate in the market as arbitrageurs or market makers, they invest in ultra-high speed technology allowing them to detect, analyses and react to market condition in nanoseconds [10]. This means HFTs can trade huge quantities of assets in very short time frames, with some estimates stating that 10-40% of all trades where initiated by them during 2016 [11].
The type of interaction between these traders suggested to have caused the crash is "Hot Potato" trading, this is when inventory imbalance is repeatedly passed between HFTs market makers. A market maker is a trader who is required to have both a bid and a ask on the order book at all times, this means in theory that they are constantly buying and selling, a high frequency market maker as expected should be buying and selling very very often. Market makers make a profit from the spread and not long positions, hence they want to keep inventories low to avoid the market moving against them. To achieve this market makers have strict inventory limits that if they pass will cause them to go into what is know as a "panic state", during this state the trader will sell of an amount of its inventory to return back into its normal trading region. This inventory now solid by the market maker can be bought by another market maker causing them to in turn go into "panic" and sell, this process is "Hot Potato" trading and can in theory continue indefinitely [12].
"Hot Potato" trading was observed in the market during the "Flash Crash" [8], this is thought to have been caused by a combination of an initial large sell order by a mutual fund and delays in communication between HFTs maker makers and the exchange on which they were operating on, causing them to buy more inventory then they wanted and go into a "panic state" and hence a "Hot Potato" feedback loop. This section explains in more detail this hypotheses and how InterDyne is set up to investigate it.

2.2.1 Deterministic

The deterministic nature of InterDyne allows for experiments to be run multiple times with the same result always returned, this allows for changes to the experiment setup to be investigated. For example changing the number of traders in the market and comparing this to a previous run allows for an investigation into how many traders are required for emergent behaviour to be observed.
This becomes particularly interesting when comparing the interactions between market makers to that of the n-body problem, like with this problem one could expect emergent behaviour might occur to some extent in a large group of market makers, however the question of wether the emergence persists in a comparable market to the three-body problem and how this compares to a larger market can be investigated.

2.2.2 Message Delays

Allowing the delaying of messages is intrinsically important to the investigation of the hypotheses since the existence of delays is proposed as one of the main aspects in the "Hot Potato" trading that occurred. Delays exist between all aspects of the market which can account for the processing time of the different elements and the transmission time of messages between them. Some of these delays will be static but it has been preposed that the delays related to the exchange actually increased during the crashing, further worsening the situation [8].
Static delays in built in InterDyne can be used to investigate the crash to see if it can occur without the need for dynamically varying delays. Dynamic delays created with agents can then be used to further investigate the events that occurred during the crash, allowing a situation to be set up where as more messages are sent to and from the exchange its delays increase. Asymmetric delays can be specified between two agents allowing further investigation into the environment in which a crash is mostly likely to occur and how delays could be altered to reduce this out come.

2.2.3 Message Passing

To observe the decided system level behaviour, a flash crash, two different methods can be use; the behaviour can be encoded into the program forcing it occur at a system level, or the system can be setup to allow the behaviour to emerge at the system level. For a true understanding of emegernt behaviour the latter approach is more relevant, this requires the different agents within the simulation to be able to communicate directly with one another. In modelling the financial market these communications are in the form of messages sent between different entities, for example a trader could send a message to an exchange detailing a limit order they wish to issue and an exchange could send back a message containing a confirmation of this order.
These messages allow interaction dynamics to occur within the simulation and hence for emergent behaviour derived from interaction dynamics to naturally present within the system.

2.2.4 Storing Passed Messages/Tracking

Due to the nature of emegernt behaviour being usually unexpected, it can be very difficult to to deduce what low level structures and operations gave rise to this system level phenomenon. This is especially true when modelling a financial system that has a large number of interacting agents all sending and receiving messages, some of which can be delayed changing their expected order of arrival. The delays in the system have been suggested to have influenced the emergence of behaviour within the system, hence in investigating these systems it is important to take into account not only message counter-parties but also message timings. InterDyne facilitates post simulation analyse into this by being able to produce a trace for all time steps of the full information of; messages sent by any agent, messages received by any agent and messages being delayed before being delayed to an agent.

2.2.5 Discrete Time

Though it is easy to assume that the financial markets operate in continuous time this is in fact not always the case. For electronic markets that trade through an exchange their time is set by the exchange, orders are not processed and messages are not sent back till the exchange decides to do so. These electronic exchanges them selfs operate in discrete time, this is unavoidable and is a product of the systems being run on computers, a computer runs based on an internal clock that ticks in discrete intervals based on a change in a square-wave oscillating voltage. This change in voltage is so fast that to a human it seems continuous, however HFTs operate themselves at such high speeds that the system clock time gaps are comparable and hence need to be considered. Therefore models simulating HFTs interactions to this detail must take account of this discrete time, hence InterDynes discrete time nature is a good match to model HFTs interactions.

2.2.6 Message Ordering

The order in which messages are processed can be very important, for an exchange, for example, it can change whose limit order has priority at a given price and whose market order executes the lowest prices. Changing these factors can make or break feedback loops within the system, meaning if message ordering is not properly dealt with the correct emegernt behaviour may not be observed. Hence InterDyne stores messages in the order they are received by an agent, taking into account delays to the messages. This however can not be done when multiple messages are received at the same time step, due to the nature of discrete time there is no way for the agents to know which message arrived first, therefore two options are presented by InterDyne; messages are ordered according to their agent identifier or messages are randomised and executed in the emerging order. This randomisation is handled in the same manner every run of the simulation [2] hence resulting in the same order and therefore not effecting the deterministic nature of the experiment.

[2] This can be changed using a new seed if so desired

2.2.7 Agents

A benefit of agent based models of other alternatives is the ability to encode agents as unique traders instead of having to model the general behaviour of a number of traders. This allows for unique behaviour of varying complexity to be given to different agents, facilitating experimentation with different trading strategies, allowing different questions to be investigated, such as, does a trading strategy need to be complex for emergence to be observed?

2.3 InterDyne Detail Operation

An InterDyne simulations structure, shown in Fig. 2, consistent of a number of independent agents, sending messages to a "Simulation Harness", this harness then (i) sends these messages to the relevant counter party or parties [3], (ii) saves these messages to a trace file.

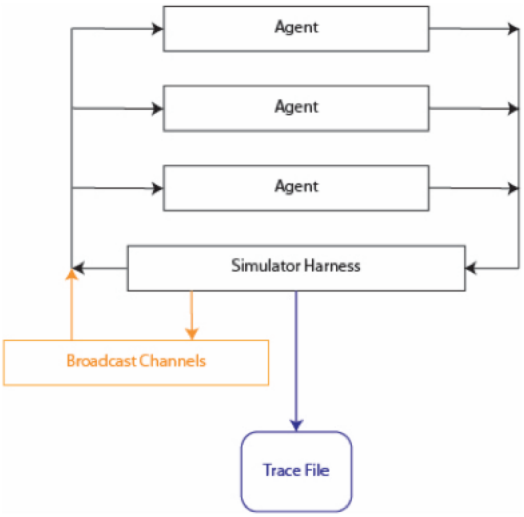

Figure 2: *Structure of an InterDyne simulation containing three agents.*

By set up each agent is required to both consume and produce a message at every time step, this is done by sending each agent a potentially-infinite list of messages and requiring it to create a potentially-infinite list of messages. The inbound list contains every message the agent will receive through out the entire simulation, ordered by time, and the outbound list contains every message the agent will ever send, order by time. This is possible due to lazy evaluation of the language that InterDyne is written in, This means that only messages being used are evaluated. Since no agent should attempted to use a message from the future [4], only messages that have been created so far are read.
Since the set up of these list requires an element for every time step and an agent my not need to send a message during a time step, empty messages can be sent, this will result in an agent either receiving or sending an empty list. It may be the case in a simulation that an agent could be generating empty messages by mistake, to differentiate between a message empty by mistake or by choice, a message can be created containing "Hiaton" demonstrating that it is empty by choice.
A InterDyne simulation once created can be run by executing the "sim" function with relevant arguments, this function has a type,

[3] In the case of broadcast messages
[4] If they do an error will be thrown.

```
sim :: Int -> [Arg_t] ->  [(Agent_t, [Int])] -> IO()
```

corresponding to, number of time steps, list of "runtime arguments", and a two tuple with the agents function and broadcast channels to which it is subscribed.
Agents within a simulation are defined by ID numbers, with the first agent having ID=1, followed by the second agent with ID=2, ID=0 is used to represent the "simulator harness". These ID numbers are used to specify which agents messages are coming from and going to. This however can be difficult to keep track of within large experiments for the experimenter, therefore InterDyne allows for agents to be referred to by a name. This example shows InterDyne being run with the use of names for identification:

```
import Simulations.RuntimeArgFunctions as RTAFuns

exampleExperiment :: IO ()
exampleExperiment
  = do
    sim 60 myargs (map snd myagents)
    where
    myargs = [ convert ]
    myagents = [ ("Trader",  (traderWrapper, [1])),
                 ("Broker",  (brokerWrapper, [3])),
                 ("Exchange",(exchangeWrapper, [2,3]))
               ]
    convert = RTAFuns.generateAgentBimapArg myagents
```

This simulation will run for 60 time steps, with three agents subscribed to a number of broadcast channels, for example the third agent is subscribed to channel 2 and 3. There is a single run time argument convert, that can convert an ID to a name and a name to an ID. This is a simplistic call of the function sim, but shows how an experiment run could be setup and executed.
In the example one can notice that the agents called all were to referred to as wrappers, this is because agents in InterDyne are typically, though not necessarily, written in two sections; "wrapper" and "logic" functions. The "wrapper" function, called in the example, can be thought of as the true agent, it handles message receiving and sending, as well as updating the local state of the function, hence this function is the one that interacts with the other agents and the harness. The "logic" function rests inside the "wrapper" function and computes the messages to be sent, hence this contains the functionality of the agent. An example of a "wrapper" function containing a "logic" function is shown here:

```
wrapper_i :: Agent_t
wrapper_i st args ((t, msgs, bcasts) : rest) myid
     =  [m]  : (wrapper_i st args rest myid)
        where
        m = logic_i (t, msgs, bcasts)
```

This agent will output a list containing m, which is the output of the "logic" function for that given time period based on the received messages. The wrapper is a recursive function with each recursion being a new time step. It takes as inputs, st the local stat variable, args the run time arguments, a list of all messages to it, and its ID. The list of messages has a three tuple for each time, containing the current time, list of all messages sent to the agent directly, and a list of all messages sent to the agent from broadcast channels.

2.4 Examples

2.4.1 Hot Potato

This feedback effect has been shown, in the InterDyne simulator, to create instabilities in market prices and even lead to crashes [13].

2.4.2 Flash Crash

3 Description and analysis of the problem

3.1 Why InterDyne is not enough

3.1.1 One-to-Many Problem

3.1.2 Semantics of the System

3.1.3 Inverse Function Problem?

3.2 Difference equations

3.2.1 Benefits and Problems

3.2.2 Lack of Visualisation

3.2.3 Optimisation Problems

3.3 Two Views Approach

4 Converter

4.1 New Language

4.1.1 Syntax/Grammer

4.1.2 Parser

4.1.3 Examples

4.1.3.1 Simple Example

4.1.3.2 Complex Example

5 Conclusion

5.1 Further Work

6 Appendix

6.1 Appendix 1

References

[1] Tsai-Ching Lu Hankyu Moon. Network catastrophe: Self-organized patterns reveal both the instability and the structure of complex networks. *Scientific Reports*, 2015.

[2] Christopher D. Clack. The interdyne simulator (2011-). http://www.resnovae.org.uk/fccsuclacuk/research, 11 2016.

[3] Gary O. Langford John S. Osmundson, Thomas V. Huynh. Emergent behavior in systems of systems. In *CONFERENCE ON SYSTEMS ENGINEERING RESEARCH(CSER)*, 2008.

[4] Norman Ehrentreich. *Agent-Based Modeling: The Santa Fe Institute Artificial Stock Market Model.* Springer, 2008.

[5] Eugene M. Izhikevich et al. Game of life. *Scholarpedia*, 10(6):1816, 2015.

[6] Douglas C. Heggie. The classical gravitational n-body problem. astro-ph/0503600, 2005.

[7] Anne Whitman Isaac Newton, I. Bernard Cohen. *The Principia: Mathematical Principles of Natural Philosophy*. Univ of California Press, 1999.

[8] U.S. Securities U.S. Commodity Futures Trading Commission and Exchange Commission. Findings regarding the market events of may 6, 2010. https://www.sec.gov/news/studies/2010/marketevents-report.pdf, September 2010.

[9] Mehrdad Samadi Tugkan Tuzun Andrei Kirilenko, Albert S. Kyle. The flash crash: The impact of high frequency trading on an electronic market. Working Paper, SSRN. http://ssrn.com/abstract=1686004 (accessed May 13, 2017)., 2014.

[10] Bus Inf Syst Peter Gomber, Martin Haferkorn. High-frequency-trading. *Business & Information Systems Engineering*, 5:97–99, April 2013.

[11] Steven Krawciw Irene Aldridge. *Real-Time Risk: What Investors Should Know About FinTech, High-Frequency Trading, and Flash Crashes*. Wiley, 2017.

[12] Elias Court. The instability of market-making algoritms. MEng Dissertation, 2013.

[13] Dmitrijs Zaparanuks Christopher D. Clack, Elias Court. Dynamic coupling and market instability. ., 2014.