University of Edinburgh
School of Informatics

Modelling BitTorrent in PEPA

4th Year Project Report
Artificial Intelligence and Computer Science

Adam Duguid
s9552712

March 1, 2005

Performance Evaluation Process Algebra (PEPA) is a high-level performance modelling language supported by software tools such as the PEPA Workbench and Choreographer. The PEPA Workbench allows performance evaluation through the use of Markov processes. Markovian modelling is subject to the well known state space explosion problem. By converting a PEPA model to cdml format, models can be evaluated within ISBJava using a variety of solvers avoiding state-space generation. Based on the reagent-centric model, the conversion process has been detailed and implemented as a module for Choreographer, the motivation being to then model the BitTorrent protocol. It will be shown that while the fluid inspired model is an approximation to the real BitTorrent protocol, the model could be enhanced to more closely follow the protocol for content distribution.

Acknowledgements

Stephen Gilmore, project supervisor
Valentin Haenel
Stephen Ramsey, Author of ISBJava at Systems Biology

All of whom contributed to the success of this project.

Contents

1. Introduction

1.1 Distributed Systems and Architectural Models

In the world of distributed systems there are numerous architectures to model the underlying system. Two of the more commonly known and used are the client-server and peer process models. The client-server model is the older of the two, its origins around 1961 when the first timesharing operating system CTSS[1] (Compatible TimeSharing System) was developed allowing remote access. This first attempt allowed multiple users to run their programs on a single machine or server. The clients at the time were nothing more than Flexowriters[2], similar in design to a typewriter but it can be seen that the client-server model's birth was here.

The client-server model as the names suggests, contains two entities, that of the server and the client. The client makes requests to the server for a known resource with the role of the server to respond to such requests, making the resource available. The resource can take many forms: a website on the WWW, a program held centrally or a printer accessible by many clients. As can be seen, the client-server model can represent everything from a small file server holding several user's data to a server on the Internet hosting one or more sites. When the resource being shared is a physical device, such as a printer, the only viable approach is the client-server model.

Despite the ability for the client-server model to handle any content distribution the model does have certain disadvantages, namely scalability and the reliability or durability of the server. In any system if the server is not powerful enough in processing speed or bandwidth via the communications link then the server becomes a bottleneck. This can be partially resolved by replacing the server with a more powerful host or in some cases such as web serving, spread the requests over multiple servers which host identical resources. Secondly, with the client-server model the server is a single point of failure that can cause cessation in the availability of the resource. Again, the use of multiple servers can alleviate this issue. However, when the resource at hand is not limited to one physical location the ability to distribute the resource even further is another possibility.

The peer process model incorporates exactly this concept. The origins of this

[1] http://web.mit.edu/afs/net/user/srz/www/multics-history.html

[2] http://www.multicians.org/mgf.html or http://www.blinkenlights.com/classiccmp/friden/

model are not quite as clear, but reach back at least as far as the creation of ARPANET[3] (Advanced Research Projects Agency NETwork) in 1968[4]. ARPANET represents the birth of the Internet and as such, a distributed system capable of losing one or more nodes while still acting as a data network.

In the peer process model every node[5] within the network has equal status to all other nodes. Thus every node is in itself a client and a server when communicating with another node. This requires the resource to be spilt over the peers, which as stated previously may not be possible. Where it is, the peers share their component of the resource with each other as required. This system works best when the whole resource is not required all the time. Then, locally stored components of the resource have no delay in being requested and multiple requests can be made to several peers to help maximize the throughput. The Internet, having grown at an exponential rate for several years could not have done so if based on the client-server model. Though still best described as a peer-to-peer network the Internet has changed since ARPANET, and currently there are 13 root DNS servers holding the information on the authoritative name servers which local name servers communicate with as follows:

> If a given name server cannot resolve the domain name, the queried name server will refer to another name server, and this process continues until a name server that can resolve the domain name is found.[6]

The delay in distributing name registration and the size of the database required are such that each node contains only partial information, but has knowledge of other nodes likely to contain the required information.

As can be seen from this brief description, communications are more complex under the peer process model. Peers require knowledge of each other to allow full access to the resource. This creates the issue of knowledge of all the peers within the system, especially in systems where peers may appear or disappear from the network. The backbone of the Internet, the 13 root DNS servers circumvent this by representing a more static network. In more dynamic systems, protocols for broadcasting new node's existence are required as is a method to discover the location of various components of the distributed resource.

---

[3] `http://www.lk.cs.ucla.edu/LK/Inet/birth.html`

[4] 1968 was when the ARPANET specification was created, the concept was formed in 1959 by Leonard Kleinrock thus preceding the creation of the first server

[5] A node herein is defined as any one computer within a network

1.2 BitTorrent

Many instances of the peer process model fall into the category of p2p filesharing, a way of distributing data of any description. For the purposes of this paper data consists of open-source software projects such as new releases of the Linux operating system and other non-copyrighted material. In the area of p2p, BitTorrent[2] is a relatively new protocol. First announced in 2001, Bram Cohen envisaged this protocol as a way to address certain bandwidth issues on the distributing server. Two major effects were intended with the creation of BitTorrent. Firstly, reduced costs of hosting content. Any business using the Internet to distribute content will incur costs for the bandwidth consumed. Secondly, to increase durability of the content distribution. If demand for one piece of content is large enough within a certain time frame then the server may not be able to cope. This can lead to a degradation of performance to the point where the server refuses to respond. This impact can be minimized by increasing the bandwidth available and utilizing a more powerful server to supply the content, both incurring a cost to the business.

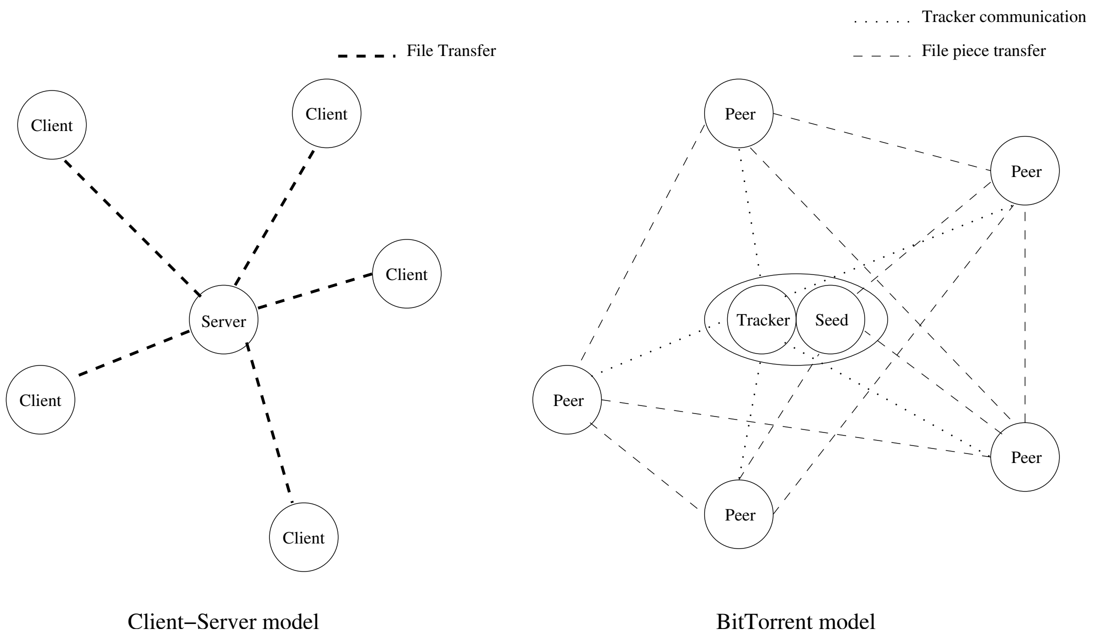


Figure 1.1: Simple Client-Server & BitTorrent models

BitTorrent works by distributing the supplying of the content amongst all the clients currently requesting the content in question. For each separate item of content (content being one or more files grouped together) there is one tracker and a torrent file associated with this particular content and tracker. The torrent file acts as the pointer to the tracker and also contains information on the content such as list of all files (including such details as size, path and name for each file), piece length and the SHA1 hash values for each piece. The SHA1 hash values allow confirmation that the piece has been correctly received, so while false data may be sent it will not corrupt the actual content, the data being discarded instead. The torrent file acts as a locater for the content and for the duration of the torrent the tracker exists to propagate knowledge of the network, requiring at least one seed (a node[6] with a complete copy of the content) to allow distribution.

The tracker behaves according to the client-server model with all peers and seeds contacting the tracker to register their current status and willingness to share. As each peer or seed connects or refreshes its status, the tracker returns a sub-list of nodes within the network. Thus this design can suffer from the problems previously discussed within the client-server model. The reasoning behind this design is twofold. The amount of information being passed by the tracker is minimal, consisting of a peer ID, IP address or DNS name and the port number for each node resulting in approximately a 5Kb text/plain document[7] sent in return to every request. With 1000 nodes querying over the space of 3 minutes this works out to an average of 28KB/s (Kilobytes per second) from the tracker requiring a 256Kbps (Kilobits per second) upload pipe assuming an even spread of GET requests from the peers. The second reason for centralizing the tracker was accountability. BitTorrent has been designed to distribute content and not as a piracy tool.

1.2.1 Protocol

Once each node has contacted the tracker, communication with other peers and seeds can begin as can be seen in figure 1.1. A simple handshake between two peers occurs at which point the status of the torrent on the clients can be exchanged via the `bitfield` message. Clients can then `request` piece-sections, send the data back via the `piece` message, or use several other messages for house-keeping purposes, all of which is happening concurrently with several peers.

As can be seen in Figure 1.2, the torrent can be viewed as one contiguous block of files. This block is then separated into identical length pieces where file borders may fall within a piece. Each piece is then seen as a block of typically 16 piece-sections. Each of these piece-sections can be requested and received from any peers this particular node is connected to. Once all the piece-sections for any one piece are received, the piece can be checked against its hash value to confirm correctness of data. After a peer has confirmed it is in possession of a

[6]Peers and Seeds are subsets of nodes such that Peers $\cap$ Seeds $= \{\}$, Peers $\cup$ Seeds $=$ Nodes

[7]5KB based on 20 character long Peer ID and 40 character long IP and responding to a request for 50 peers (default)

complete piece, it can inform the peers it is connected to via the `have` message and then upload piece-sections of the new piece as and when `request` messages are received.

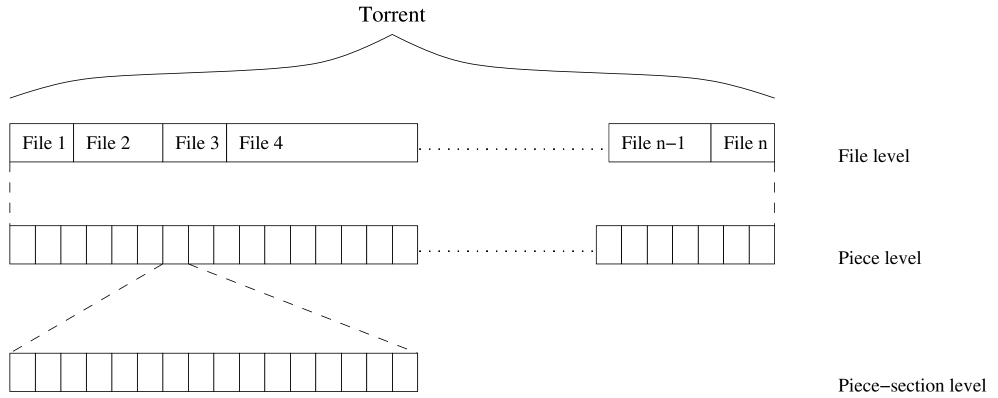


Figure 1.2: Levels of detail within a torrent

1.2.2 Behaviour

This describes the mechanics behind the protocol but does not indicate the behaviour that may be observed. BitTorrent behaviour emerges in part from the download strategy used but is dominated by what is known as choking and optimistic unchoking. Choking is what allows a tit-for-tat strategy within the protocol and the current algorithm found in many of the BitTorrent clients works as follows. Each node actively connects to x other nodes (where x is typically 4) which will initially be based on those nodes interested and with pieces which this node is missing. Then, after a certain timespan (currently recommended at 10 seconds) nodes will be choked or unchoked depending on certain criteria. The top n uploaders to this node (those nodes currently sending piece-sections) and interested in this node are unchoked. This is the tit-for-tat behaviour as previously mentioned and is intended to optimize download speeds. If this results in more interested nodes being unchoked than is specified, then those currently downloading are ranked on their upload rates and those with the slowest upload rates are choked. All nodes not interested are unchoked by default.

This algorithm is greedy in the sense of maximizing the download rate for each node and limits the ability for any one node to leech from others (to receive piece-sections without uploading, thus not contributing to the overall distribution). The overall behaviour emerging from any particular BitTorrent network is one where each node shares with its neighbours, reducing the overall demand on the original source. Optimistic unchoking is intended to remove any chance of deadlock by unchoking each node in turn, regardless of its current uploading rate.

This process is intended to happen less frequently than the normal choking and is a simple way to allow all nodes a chance.

The download strategy also impacts on the network. Two suggested strategies are either random piece selection or the preferred strategy of rarest-first. This requires each node to maintain a record of each other node and what they currently possess, updating these records as `have` messages are received. With these records each node can see locally what the rarest piece still required is and request that over any other piece. This will hopefully result in maintaining a balanced network with no dependence on any one node being too large (with the exception of the original seed in the initial phase of a torrents lifespan).

1.3 Analysing Systems

When attempting to analyse any system there are two important choices to be made prior to analysis.

- Direct measurement or modelling the system.
- Level of abstraction.

Direct measurement of the system is one approach with the advantage that any data collected will be independent of any assumptions regarding the system. It could also be suggested that the data is more indicative of the system in real world conditions - allowing for many factors that may not have been considered prior to data collection. Of course it may not be possible to measure the system directly. The system may not allow direct measurement or such measurement may disrupt the system to such an extent that the data is no longer accurate. Modelling the system of course removes both obstructions. The model also does not have to operate in real time in addition to allowing any aspect of the model to be measured. The danger lies in creating an accurate enough model such that any data collected can be shown to represent the system in some generalized sense.

The level of abstraction is more relevant if modelling the system, but still influences analysis of the system regardless of the decision to model or measure. An example in relation to BitTorrent could be the level of accuracy used in measuring the level of the content received so far. It could be stated as a simple percentage ranging from 0-100%, a list of the pieces successfully obtained so far or even to the level of piece-sections. The question is does the knowledge of piece-sections allow any new information to be derived over the more simple form of a percentage?

Currently little research has been performed on the BitTorrent protocol. Of the few papers that currently exist, one paper attempted to measure BitTorrent directly via the tracker[10] and two have attempted to model[3, 1]. The work performed by Dongyu and Srikant[3] used a fluid model whereas the talk given by Argent-Katwala[1] used PEPA to model BitTorrent.

1.3.1 Direct Measurement

The paper by Pouwelse *et al*[10] is currently the only attempt to measure and analyse the BitTorrent under real world conditions. Taking place over eight months, the authors attempted to measure four aspects of BitTorrent; availability, integrity, flashcrowd[8] handling and download performance. As the BitTorrent protocol relies on a tracker for each torrent, many sites exist purely as a hub for multiple trackers. One such site was SuprNova.org[9], known for its range and volume of content. The motivation behind the use of SuprNova was its popularity over other sites at the time. At this point one assumption has been already made by the authors - that SuprNova represented the BitTorrent community. Although it is impossible to state with any degree of confidence (the site no longer being available), almost all content linked by the site at the time was in violation of copyright laws and so it would be more exact to say SuprNova represented a certain section of the BitTorrent community, which may or may not be representative of the community as a whole.

The concept of integrity, while of concern to sites such as SuprNova can be seen as a lesser issue to BitTorrent in general. The creators of PostgreSQL, the SQL database application, offer the ability to download either via http or BitTorrent. In this case the integrity of the files is not in question as the host can be trusted and the host supplies the torrent which incorporates the hash values. Integrity only becomes an issue when content can take multiple forms (or variations on an original form) and from different sources.

The aspects of flashcrowd handling and download performance are of interest. Pouwelse *et al* conclude that SuprNova was able to efficiently handle large flashcrowds but this says little about the protocol in general and more about the specific system of mirrors employed by SuprNova. To analyse the effects of flashcrowds, measurement of a system under varying loads is required, a task better suited for a model.

The analysis performed on download performance does offer some insights to the typical peer that will be found. The authors do not state how the download speed of any one peer was calculated, but their results indicate 90% of the peers analysed had a download speed less than 520Kbps and the average was approximately

[8] The term flashcrowd merely refers to the sudden influx of a large crowd.

[9] SuprNova went off-line on the 19^{th} of December 2004, in part, due to pressure applied by the MPAA to many sites offering copyrighted material.

240Kbps. This speed may reflect the actual download speeds achieved[10] rather than the actual speed possible with the peers connection.

One valid observation made by the authors is the range of differing connections, a factor not normally expanded on in models. This simplification makes the model easier to deal with but currently at an unknown cost. The authors analysis suggests that this may not be a factor with peers still successfully completing downloads within a day even at the lower speeds.

The authors also state that the observed behaviour of the peers does not indicate a Poisson rate for many of the possible actions, a distribution used in almost all models. There is no reason to believe that this is not the case, the use of the Poisson distribution is often for ease of analysis rather than its closeness to the data. What has not been established in the paper is how far the observed behaviour deviates from the Poisson distribution or how much this may affect the models based on this distribution.

Availability was the final area of focus in the paper, in which availability is split into two types, that of the server and of peers. Again, as SuprNova dealt with thousands of torrents the issue of server availability is more relevant for a certain type of site. The availability of the peers is of interest however and the results gathered show that the typical peer will share as little as possible. Based on one torrent, over 53,000 peers were monitored via the tracker. Of those 53,000 only 17% seeded for longer than one hour after completing the download. These results may be skewed by the hit and run mentality of a casual pirate or may represent a more general behaviour of users and their downloading habits. As the results are also based on a single torrent, extrapolation is not prudent in this instance.

The data collected on availability, flashcrowd behaviour and download performance does give a clearer picture than that previously known but does so in specific circumstances which may or may not indicate real patterns. As is always the case with live experimentation, the experiment must be run multiple times to show any confidence in the results which has not been done is this instance. The paper by Pouwelse *et al* does represent a well rounded case study of SuprNova but does not allow measurement of the BitTorrent protocol itself. The ability of the BitTorrent protocol to cope in varying conditions cannot be established based on one case, the very reason why models can prove useful.

[10]given the authors access to the data of how long a peer takes to download a file and the size of the file.

1.3.2 PEPA

Process algebras are an approach to modelling communicating systems via the interactions of separate components rather than the flow of entities through a system. Of the process algebras, PEPA (Performance Evaluation Process Algebra) uses certain techniques[11] to reduce the size of the model allowing larger and more complex communicating systems to be modelled. Using the following operators; Prefix $((\alpha, r).P)$, Choice $(P + Q)$, Cooperation $(P \underset{L}{\bowtie} Q)$ and Hiding (P/L), components and activities can be combined to model complex systems. The components within a PEPA model are roughly equivalent to states within a Markov process and activities represent the transitions between states. The four operators work as follows:

Prefix $P \stackrel{def}{=} (\alpha, r).Q$ represents the state P which changes to Q by transition α with delay sampled from the negative exponential distribution with parameter r.

Choice $P \stackrel{def}{=} Q + R$ represents the component or state P and the ability to change into either a Q or R component.

Cooperation $P \underset{L}{\bowtie} Q$ requires that for both P and Q components to change on any activity with the set L, they must synchronize. An example of synchronization would be: let $A \stackrel{def}{=} (\alpha, r_1).B + (\beta, r_2).C$ and $D \stackrel{def}{=} (\alpha, r_1).E + (\beta, r_3).F$ and let $A \underset{L}{\bowtie} D$ where the set $L \in \{\{\}, \{\alpha\}, \{\beta\}, \{\alpha, \beta\}\}$. If $L = \{\}$ then $A \underset{L}{\bowtie} D$ can also be written as $A||D$ and either component can change independently of the other. If $L = \{\alpha\}$ then A or D can change via activity β independently but both must change together if via activity α. Similarly, if $L = \{\alpha, \beta\}$ then components A and D must change together either via activities α or β. See appendix A for Markov Processes for each synchronization.

Hide P/L alters component P such that any activity in the set L is hidden from the rest of the model. The main effect is that synchronization is not possible on this component for activities in set L

Using these four operators complex systems can be constructed. The following is nothing more than a toy example of the capabilities of PEPA.

[11]See [5] for details on the techniques employed in PEPA.

$$
\begin{aligned}
Server &\stackrel{def}{=} (get, \top^{a}).(download, \mu).(rel, \top).Server \\
Browser &\stackrel{def}{=} (display, p_1\lambda).(cache, m).Browser + \\
&\quad\; (display, p_2\lambda).(get, g).(download, \top).(rel, r).Browser \\
WEB &\stackrel{def}{=} (Browser||Browser) \underset{\{get,download,rel\}}{\bowtie} Server
\end{aligned}
$$

[a] $\top$ represents a passive rate and as such does not influence the rate in synchronization

This small example results in 48 states within the Markov Process. If the WEB definition is changed to analyse 4 browsers,

$$WEB \stackrel{def}{=} (Browser||Browser||Browser||Browser) \underset{\{get,download,rel\}}{\bowtie} Server$$

the state space increases to 1296 states. Increase the number of browsers to 6, a state space consisting of 3645 states is returned. The growth in the state space is the product of each components state size and so limits the size of the model that can be created.

In the context of BitTorrent, creating a realistic model will require a large number of peers. Part of this state space would be the different combinations that each peer can exist in, in relation to all other peers. If a simple representation of 0-100% is used for file completion, then the state space for representing *only* the peer status would be $100^{\#Peers}$.

1.3.2.1 BitTorrent model in PEPA

Of the three models proposed by Argent-Katwala[1], the smallest (aptly labelled smallest model) follows the previously suggested representation, again creating a state space size approximately $100^{\#Peers}$ with each peer component requiring $2 \times \#peers$ Prefixes to account for downloading and uploading to the other peers.

$$
\begin{aligned}
Peer_{i,j} &= (transfer_{1,i}, download).Peer_{i,j+1} + ... \\
&+ (transfer_{n,i}, download).Peer_{i,j+1} \\
&+ (transfer_{i,1}, upload).Peer_{i,j} + ... \\
&+ (transfer_{i,n}, upload).Peer_{i,j};
\end{aligned}
$$

where $Peer_{i,j}$ represents $Peer_i$ having completed j %
$\&transfer_{s,t}$ represents transfer of data from s to t

The other two models attempt to represent all permutations of completed pieces for each peer, creating a state space approximately $\left(2^{\#pieces}\right)^{\#peers}$ in size. The naive model simply tracks which pieces each peer has, and in doing so allows two peers to simultaneously download the same piece from each other without either peer being required to have previously acquired the piece. The huge model rectifies this with a similar solution as used within the smallest model.

$$\begin{aligned}
Piece_{i,j} &= (transfer_{1,i}, download).GotPiece_{i,j} + ... \\
&+ (transfer_{n,i}, download).GotPiece_{i,j}; \\
GotPiece_{i,j} &= (transfer_{i,1}, upload).GotPiece_{i,j} + ... \\
&+ (transfer_{i,n}, upload).GotPiece_{i,j}; \\
Peer_i &= (Piece_{i,1} \mid\mid ... \mid\mid Piece_{i,n})
\end{aligned}$$

Assuming the torrent represents one CD from any Linux distribution, the file will be 600Mb in size requiring 600 pieces if the larger piece size of 1Mb is used. Even if only 2 peers are connected the size of the space required is $(2^{600})^2$ or $2^{1,200}$ and thus intractable.

As can be seen, the work performed by Argent-Katwala was exploratory in nature rather than the creation of concrete models[12] but still clearly highlights the issues involved in creating BitTorrent models in PEPA.

1.3.3 ISBJava

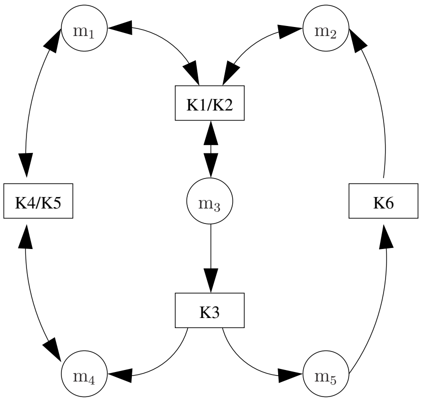
ISBJava is a Java library allowing the simulation of chemical kinetics, a tool constructed by the Institute of Systems Biology[13]. The goal of the Institute of Systems Biology is the understanding of biological systems through the relationships between each component of the system. Dizzy, the front end component written on top of ISBJava allows the modeller to define chemical species and the interactions of these species through reactions, while a range of solvers are offered to analyse the system. The individual parts to this particular modelling language are similar to that of PEPA; components/species and transitions/reactions are indeed seen in many of the languages used today. The differences lie within the level of detail or approach. Whilst PEPA modelled individual components and their transitions, ISBJava measures the flow of populations sizes of the different species through the reactions. This of course allows for large population sizes without increasing the size of the model exponentially.

A small example can be seen in Figure 1.3, as used in [8]. This small network produces the reactions to the left of the model, where the syntax is as follows $reactionname, reactants \rightarrow products, reactionrate$. Population sizes for the various species are defined in the same manner as reaction rates where the identifier

[12] The system equations for such models were not even attempted, as in reality each peer would require independent connections to all other peers.

[13] `http://www.systemsbiology.org/`

for the value is identical to the identifier for the species. What can also be seen from the example is that the reactions are not required to balance in regards to the number of reactants and products involved within the reaction. This aspect of chemical reaction modelling is not present in PEPA which by virtue of design respects the stoichiometry[14]. Care must be taken to preserve equilibrium when modelling in ISBJava.




```
k1react, A + X -> A/X, k1;
k2react, A/X -> A + B, k2;
k3react, A/X -> B + Y, k3;
k4react, B -> A, k4;
k5react, A -> B, k5;
k6react, Y -> X, k6;
```

$$\text{where}\quad \begin{array}{ll} \mathrm{A} = 1000, & \mathrm{B} = 0 \\ \mathrm{X} = 1000, & \mathrm{C} = 0 \\ \mathrm{A/X} = 0 & \end{array}$$

Figure 1.3: Small synthetic example network and reactions

For this example the population sizes are arbitrary. In comparison to ISBJava, modelling these species in PEPA would require a single state for each combination of population sizes over all species, again an intractable problem in that form.

1.3.3.1 BitTorrent as a Fluid Model

The paper by Dongyu and Srikant[3] approaches the problem of modelling BitTorrent with a similar methodology to that of ISBJava and reactions. Using a fluid model to study BitTorrent, the authors represented the network using the following eight parameters

#peers ($x(t)$)	#seeders ($y(t)$)
arrival rate of new peers (λ)	disconnection rate of downloaders (θ)
uploading bandwidth (μ)	downloading bandwidth(c)
disconnection rate of seeders (γ)	effectiveness of file sharing ($\eta,\ \eta \in [0,1]$)

Where μ and c do not represent the actual bandwidth of a peer but the rate at which a peer can transfer the entire file $\left(\frac{1}{\textit{time to download}}\right)$.

These parameters allowed the authors to concisely define equations to represent the change in number of peers and seeders over time, the rate of change of peers

[14]the relation between the quantities of substances that take part in a reaction or form a compound (typically a ratio of whole integers) taken from `http://www.hyperdictionary.com`. Component numbers do not increase/decrease in PEPA hence respect for stoichiometry.

being used as an example below.

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \lambda - \theta x(t) - min\{cx(t), \mu(\eta x(t) + y(t))\}$$

Here the change of rate is positively influenced by the arrival of new peers (λ), and negatively influenced by a combination of a percentage of the current peers quitting ($\theta x(t)$) and a certain percentage of the current peers completing the torrent based on the bottleneck being either the download bandwidth of the peers or the upload bandwidth of the peers and seeds ($min\{cx(t), \mu(\eta x(t) + y(t))\}$).

From the equation it is easy to note that the state of the peers is defined as either present or not present. No other information pertaining to the status of a peer is available and so it is impossible to estimate the duration a of specific peer within the system or the progress made downloading the content. The impact of some of the defined parameters is also questionable. The effectiveness of file sharing is presented as a parameter based on the number of pieces the content is split into (N) and the number of connections to a peer (k) and is calculated as $\eta \approx 1 - \left(\frac{log(N)}{N}\right)^k$ whose behaviour is shown in Figure 1.4. As the graph indicates, the value of η is equal to one for all but the pathological cases of its two parameters (N is typically 600+ and k approximately 40) meaning this parameter has little influence within the fluid model being.

This is not to say that the behaviour of η has no effect, but the use within the steady state has been shown by Dongyu and Srikant to negligible. The real use is during the initial period of a torrent when the number of peers vastly out numbers the number of seeders. In cases such as these many peers may be connected only to other peers and even with a large k may still not have access to the complete file.

The model created by Dongyu and Srikant does work at a high level. Having factored in download strategies and piece availability within the parameters, the equations could be seen to generalize to file sharing applications by alteration of the values used rather than just BitTorrent. Despite the level of abstraction, the results from comparison against a small real world test prove interesting. From the data presented, the model does not perform well in modelling the number of peers downloading but model inaccuracy may not be the cause. As can be seen, the number of users downloading never peaked higher than 25 during the lifespan of the injected file (approximately 3 days) and so normal variance may be the cause. The data collected by Pouwelse *et al* while still only one snapshot of BitTorrent could be seen to corroborate the model for downloaders (albeit on a different scale), despite their reservations with the fluid model. The fluid model would certainly not seem to handle the flashcrowd effect but neither was any attempt made to do so, as noted by Pouwelse *et al.*

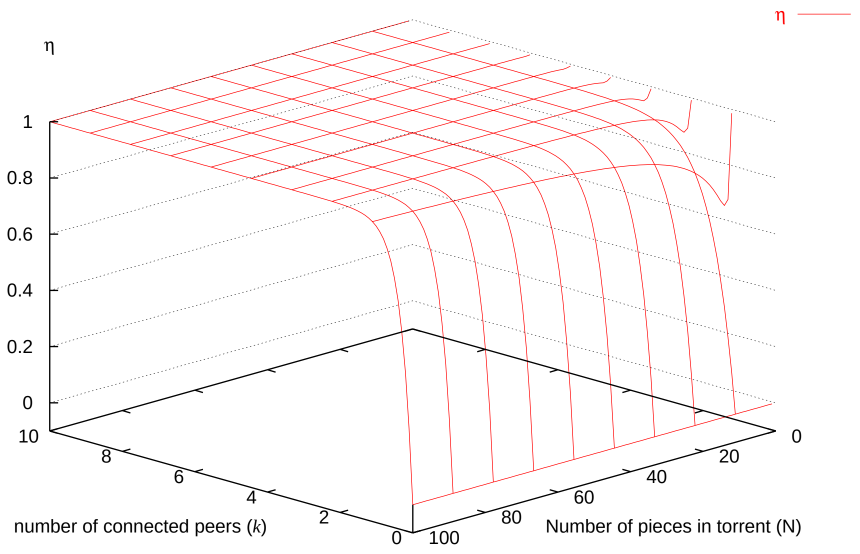


Figure 1.4: Measuring η over #pieces and #connections

The model equation for the number of seeders present in the system would seem to more accurately represent the experimental data collected by Dongyu and Srikant. Allowing for variance in the data as the authors have done the model would seem to be accurate. Comparing against the data from [10] as before it can be seen that the model may not reflect the data gathered from SuprNova. The one area of concern is the ability for seeds to increase early in the model. As the rate of change of seeds is also based on $(min\{cx(t), \mu(\eta x(t) + y(t))\})$, seeds can appear before it is realistic for any peers to have completed the download, a fact more evident from the data on SuprNova than that collected by Dongyu and Srikant.

Despite the claims by Pouwelse *et al*, the fact remains that little research has been completed on the BitTorrent protocol and of that work, the research by Dongyu and Srikant is the most complete. Many of the factors that could influence BitTorrent behaviour have been considered, including an attempt to analyse how BitTorrent copes with what Dongyu and Srikant call free-riding as well as the existence of a Nash equilibrium under certain restrictions.

1.3.4 Bridging PEPA and ISBJava

The papers by Calder *et al*[7, 8] proposes a different modelling technique, their purpose being the influence of RKIP on the ERK signalling pathways. The proposal consists of creating the original model in PEPA, either as a reagent-centric or pathway-centric model. In the reagent-centric model, every component is modelled by each activity that will lower or increase the concentration of that component as seen in Figure 1.5

$$\begin{aligned}
\mathrm{A}_H &\stackrel{\mathit{def}}{=} (k1react, k1).\mathrm{A}_L + (k5react, k5).\mathrm{A}_L \\
\mathrm{A}_L &\stackrel{\mathit{def}}{=} (k2react, k2).\mathrm{A}_H + (k4react, k4).\mathrm{A}_H \\
\mathrm{X}_H &\stackrel{\mathit{def}}{=} (k1react, k1).\mathrm{X}_L \\
\mathrm{X}_L &\stackrel{\mathit{def}}{=} (k2react, k2).\mathrm{X}_H + (k6react, k6).\mathrm{A}_H \\
\mathrm{A/X}_H &\stackrel{\mathit{def}}{=} (k2react, k2).\mathrm{A/X}_L + (k3react, k3).\mathrm{A/X}_L \\
\mathrm{A/X}_L &\stackrel{\mathit{def}}{=} (k1react, k1).\mathrm{A/X}_H \\
\mathrm{B}_H &\stackrel{\mathit{def}}{=} (k4react, k4).\mathrm{B}_L \\
\mathrm{B}_L &\stackrel{\mathit{def}}{=} (k5react, k5).\mathrm{B}_H + (k3react, k3).\mathrm{B}_H \\
\mathrm{Y}_H &\stackrel{\mathit{def}}{=} (k6react, k6).\mathrm{Y}_L \\
\mathrm{Y}_L &\stackrel{\mathit{def}}{=} (k3react, k3).\mathrm{Y}_H
\end{aligned}$$

Figure 1.5: PEPA reagent-centric model for the network in Figure 1.3

This allows the modeller to view the system from a different perspective (each component and the reactions that affect it) to that offered by ISBJava or Ordinary Differential Equation (ODE) solvers (viewable by each reaction and the components involved in that reaction). As noted by the authors, the view of high and low concentrations (signified by $_H$ & $_L$ suffixes) is a coarse discretization of the possible states for any given component to exist in. Whilst this would indicate a potential source of error within the model this is not the case. The PEPA model structure allows conversion to ODEs at which point the concept of high and low concentrations becomes clearer as the increase or decrease of a given concentration rather than simply high or low.

As the format of ISBJava is similar to that of ODEs (four of the seven solvers supplied with ISBJava are in fact ODE solvers) conversion to the ISBJava format (cdml) is possible if starting from a similar PEPA structure. The intention is then to model the BitTorrent protocol in PEPA and analyse the model by converting to ISBJava. This will allow a component centric approach to the creation of the model while still allowing efficient analysis by the solvers present in ISBJava. This conversion process will require designing and implementation before the modelling of BitTorrent can commence.

2. Design

2.1 Choreographer

Choreographer is an environment being designed at the University of Edinburgh[1] to facilitate information extraction from the Unified Modelling Language (UML) via either PEPA or LySa. The primary purpose of the platform is to allow the automatic creation of models in the PEPA and LySa languages via Extractors, at which point performance or security analysis can take place. The results are used to tag the UML model thus supplementing the original model. The platform also acts as a general purpose editor for the three languages and offers the functionality of PEPA and LySa along with syntax highlighting.

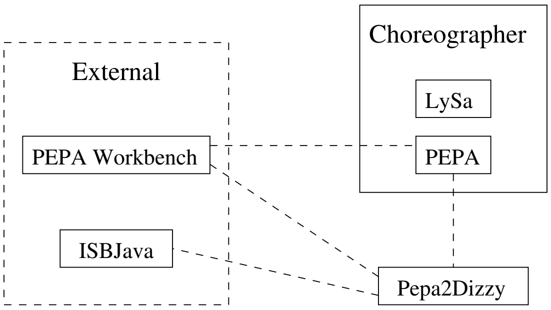


Figure 2.1: Communication channels between software modules (excluding UML module)

The use of Choreographer as a platform is driven by two objectives. To enhance the current functionality of Choreographer with the addition of machine generated cdml from PEPA to analyse via ISBJava, and secondly, allowing the use of the PEPA parse tree created by Choreographer.

As can be seen in figure 2.1, the PEPA module communicates directly with the PEPA Workbench to allow analysis. Communication with the PEPA Workbench however is only possible through file passing at present, resulting in the syntax checking of the PEPA module used for highlighting purposes only. Communication with the PEPA module allows access to the parse tree and with accessor methods, interrogation of the tree allowing conversion to take place at this stage.

Once conversion has been completed to the ISBJava representation, analysis can take place using the built in solvers before saving the resulting report to disk.

[1] In connection with the Design Environments for Global Applications (DEGAS) project

Communication with the PEPA Workbench can also happen within the conversion module allowing derivation of the state space. The intent is to make use of checks for live-locks and dead-locks within the PEPA/cdml model. As with the PEPA module, communication to the PEPA Workbench is through file passing. The reasons for direct communication rather than through the PEPA module will be made clear later.

2.2 Conversion from PEPA to CDML

2.2.1 Language Grammars

As Choreographer is an editing environment for PEPA, it must recognize valid syntax and grammar of the language concerned. Choreographer does differ from the grammar specified in the original manual[9] and as any conversion process is tied to the grammar of the languages involved, the PEPA grammar with corrections is stated below in Figure 2.2

As stated within the grammar footnotes, there have been several additions since the initial definition. The latest version of PEPA, (0.9.4b Tabasco Workbench) allows rate expressions within the PEPA Prefix operator. Choreographer extends the grammar further allowing expressions within the rate declarations themselves. This is not consistent with the latest version of PEPA but it is hoped that the workbench will support this functionality in the future. Identifiers are also more tightly defined within Choreographer as stated in the grammar. All activity and rate identifiers are denoted by beginning with a lower case letter whilst component identifiers start with an upper case letter. This stricter definition is more in keeping with the original conventions stated by Hillston[5]. It should also be noted that the grammar as it stands is ambiguous and left recursive, making it unsuitable for machine parsing. The key differences however between the original specification and the allowable grammar within Choreographer are shown and should suffice for stating the conversion process.

Figure 2.3 contains a partial cdml grammar, also suffering from ambiguity and left recursion. The full language is also more expressive; containing the ability to define multi-step or delayed reactions, perform additional mathematical operations to those already defined, use of templates and the ability to include multiple files. These aspects of the cdml language are not mirrored in PEPA and for the sake of clarity are not present in the declared grammar. Strictly speaking PEPA does not incorporate any mathematical functions or define any keywords with the exception of `infty`, and thus cannot be parsed within the Tabasco Workbench. The inclusion of the functions is only possible by the `PepaUndefinedRate` object in the PEPA module for Choreographer as will be seen later.

$$
\begin{array}{rcl}
\mathit{program} & ::= & \mathit{rate\_declaration}^{*}\ \mathit{component\_declaration}^{+}\ \mathit{composition}^{a} \\
\mathit{rate\_declaration} & ::= & \%?^{b}\ id = \mathit{rate\_expr}^{c}; \\
\mathit{component\_declaration} & ::= & \#?^{b}\ Id = \mathit{seq\_component};^{\,d} \\
\mathit{seq\_component} & ::= & \mathit{seq\_component}\ +\ \mathit{seq\_component} \\
 & | & (id, \mathit{rate\_expr}^{e}).\mathit{seq\_component} \\
 & | & Id \\
 & | & (\mathit{seq\_component}) \\
\mathit{composition} & ::= & \mathit{composition\_component} \\
 & | & \mathit{composition}\ \ <[\mathit{idseq}]>\ \ \mathit{composition} \\
 & | & \mathit{composition}\ ||\ \mathit{composition} \\
 & | & (\mathit{composition}) \\
\mathit{composition\_component}^{f} & ::= & Id \\
 & | & Id\ /\ \{[\mathit{idseq}]\} \\
\mathit{idseq} & ::= & id \mid id, \mathit{idseq} \\
\mathit{rate\_expr} & ::= & \mathit{rate} \mid \mathit{rate\_expr}\ \mathit{op}\ \mathit{rate\_expr} \mid (\mathit{rate\_expr}) \\
\mathit{rate} & ::= & id \mid \mathit{float}^{g} \mid \texttt{infty} \\
\mathit{op} & ::= & + \mid - \mid / \mid * \\
id^{h} & ::= & [a-z][a-z\ A-Z\ 0-9\ '\ \_\ -]^{*} \\
Id & ::= & [A-Z][a-z\ A-Z\ 0-9\ '\ \_\ -]^{*}
\end{array}
$$

[a] Zero or more (*) rate declarations to a program has been introduced since the original specification.

[b] The % and # signs that used to precede id has been altered to optional for compatibility with older models but are no longer required.

[c] The 0.9.4b Tabasco PEPA Workbench does not allow rate expressions in declarations despite being allowable in Choreographer. It is hoped PEPA will follow the Choreographer convention in future versions.

[d] *id* has been changed to *Id* to differentiate between rate id's and component id's.

[e] Altered to allow expressions within rates.

[f] Definition added to differentiate between operators allowed in component definitions and the system equation.

[g] Changed from *int* to more closely represent the number range permissible

[h] Definition within Choreographer follows Hillstons thesis[5] in terms of use of uppercase and lowercase within id's.

Figure 2.2: PEPA grammar as defined within Choreographer in EBNF notation

$$
\begin{array}{rcl}
program & ::= & symbol\_definition^{+}\ reaction^{+} \\
symbol\_definition & ::= & id = value; \\
reaction & ::= & id,\ species? -> species?,\ value; \\
species & ::= & id \mid id + species \\
value & ::= & [expression] \mid expression \\
expression & ::= & float \\
 & \mid & expression\ op\ expression \\
 & \mid & (expression) \\
 & \mid & function \\
 & \mid & keyword \\
op & ::= & + \mid - \mid / \mid * \\
function & ::= & exp(expression) \mid ln(expression) \mid sin(expression) \\
 & \mid & cos(expression) \mid tan(expression) \mid asin(expression) \\
 & \mid & acos(expression) \mid atan(expression) \mid abs(expression) \\
 & \mid & floor(expression) \mid ceil(expression) \mid sqrt(expression) \\
 & \mid & theta(expression) \\
 & \mid & min(expression, expression) \mid max(expression, expression)^{a} \\
keyword & ::= & \texttt{time} \mid \texttt{Navo} \\
id & ::= & [a-z\ A-Z][a-z\ A-Z\ 0-9\ \_]^{*}
\end{array}
$$

[a] *min* & *max* functions added to ISBJava 2.0.1. Patch sent to systemsbiology and incorporated as of 2.1.0

Figure 2.3: partial cdml grammar

With the two grammars defined for the languages involved, the PEPA language acceptable within the conversion requires definition. This subset of the PEPA grammar is based on the work by Calder *et al*[7]. The use of suffixes (_H,_L) will distinguish between differing states of a single component/species and will denote the direction of change within a population as seen previously in Figure 1.5. The use of the PEPA Prefix will indicate the activity/reaction taking place at the defined rate in connection to the components and the Choice operator will represent the single component/species appearing in more than one reaction.

The use of sub-declarations within a component declaration is rejected, serving no purpose in creating the reagent-centric model. The system equation or composition of the model is legally required by the workbench, and if constructed correctly will aid in showing live-locks and dead-locks within the model. However, for the purposes of creating the ISBJava model the system equation does not contain any additional information as to the relations between the components/species and their associated reactions. The Hiding operator is also not supported within the system equation. This modified grammar can be seen in Figure 2.4

$$
\begin{array}{rcl}
program & ::= & rate\_declaration^{+\,a}\ component\_declaration^{+}\ composition \\
rate\_declaration & ::= & id = rate\_expr; \\
component\_declaration & ::= & Id = seq\_component; \\
seq\_component & ::= & seq\_component\ +\ seq\_component \\
 & | & (id, rate\_expr).Id^{b} \\
composition & ::= & Id \\
 & | & composition\ <[idseq]>\ composition \\
 & | & composition \mid\mid composition \\
 & | & (composition) \\
idseq & ::= & id \mid id, idseq \\
rate\_expr & ::= & rate \mid rate\_expr\ op\ rate\_expr \mid (rate\_expr) \\
rate & ::= & id \mid float^{c} \\
op & ::= & + \mid - \mid / \mid * \\
id & ::= & [a-z][a-z\ A-Z\ 0-9\ \_]^{*} \\
Id & ::= & [A-Z][a-z\ A-Z\ 0-9\ \_]^{*}\_[L\ H]^{d}
\end{array}
$$

[a] Changed from * to $^+$. Dual functionality of PEPA rates under the conversion process will require at least one defined *id*.

[b] PEPA Prefixes cannot be sequentially combined.

[c] Passive rates are not permissible under the current conversion grammar.

[d] Altered to reflect naming convention within reagent-centric model.

Figure 2.4: PEPA grammar for PEPA-cdml conversion

2.2.2 Assigning Population Sizes

The first stage of the conversion process requires identifying all the species involved in the model and any rates associated with those species. Under the ISBJava model, every species must be coupled to a population size. As stated previously, within ISBJava the populations were declared with the rates and this approach will be adopted for the PEPA/cdml conversion. The species set will contain all unique PEPA Id's with the suffixes removed. With the example in Figure 1.5 this will lead to the species set containing $\{A, X, A/X, B, Y\}$ as A_H and A_L simply represent different states of the species A. This set is then compared against all defined rates within the model. Each rate is associated with at most one species if they share identical identifiers after the first character of the id has changed case. With the previously defined species set the rate $a = 1000$ would match to species A. $A = 1000$ is an illegal rate declaration according to the grammar and $a\_H = 1000$ would not match any of the identified species. A clearer example would be the rate $raf\_1\_RKIP = 1000$ which would match to the species Raf_1_RKIP which would have been obtained from either Id's $Raf\_1\_RKIP\_H$ or $Raf\_1\_RKIP\_L$.

Once all defined rates have been checked, any species with an undefined population is set to size 0. As rates are based on the Poisson distribution, the distribution of waiting times is $1 - e^{\lambda t}$ where λ is the defined rate and t the time until the next occurrence. When the rate is 0, $1 - e^{0t} = 0$ and so the probability of the next occurrence happening for any value of t is zero. Defining a rate of 0 therefore makes no sense and as such is illegal in PEPA. As legal population sizes are any value greater than or equal to 0 the conversion process assumes sizes of 0 if not defined. The final requirement of the conversion process is that at least one rate is associated with one species. This requirement acts as a safety measure or sanity check on the model. If all population sizes are zero then no reactions can take place suggesting a faulty model.

Within the PEPA parse tree rates can take one of five forms; specified, numeric, expression, undefined and $\top$ rates. All the initial rates are specified rates by virtue of the rate declaration in the grammar. The specified rates are composed of the id and a link to the PEPA rate it is an identifier for. The conversion process currently allows numeric and expression rates with the one condition that any rate that is associated with a species must have been defined as a numeric rate by Choreographer. Any rate not associated with a species is defined at this stage as a parameter. Most of the grammar and syntax checking has now already occurred. All component and rate declarations are known to be valid in regards to the grammar and one important syntax check has taken place. Every Id must end with one of two suffixes and no suffix may appear on both sides of any component declaration. To clarify, every component declaration must expand to one or more PEPA Prefixes by use of the Join operator on the RHS. The Id component for every PEPA Prefix on the RHS must be identical and must end with the opposite suffix to that of the Id for the component declaration on the LHS i.e.

$$\begin{aligned}\mathrm{A/X}_H &\stackrel{def}{=} (k2react, k2).\mathrm{A/X}_L + (k3react, k3).\mathrm{A/X}_L \\ \mathrm{A/X}_L &\stackrel{def}{=} (k1react, k1).\mathrm{A/X}_H\end{aligned}$$

and this declaration is syntactically illegal

$$\mathrm{A/X}_H \stackrel{def}{=} (k2react, k2).\mathrm{A/X}_L + (k3react, k3).\mathrm{A/X}_H$$

Declaring a component that shifts from a 'high' state to 'high' could be interpreted as a component required for a transition to take place but not directly used within the reaction. This could in some sense be equivalent to the $\top$ rate but simply ensuring the same activity happens in both directions will achieve the desired behaviour. For simplicity the $\top$ rate is not allowed but could be introduced in a new version of the translation process as an alternative syntax. Declaring a component as moving from a low to low state can only serve to state a component is not involved in a reaction and the same effect is achieved by simply not including that activity.

Component declarations in both directions allow a species population to increase and decrease but there is no requirement to do so. Unlike PEPA which must respect the number of components within the system the use of a reaction based approach allows populations to grow and die. The use of a component declaration in one direction simply assumes one way reactions. One possible use of such a behaviour would be a finite resource being introduced to a system, and will be seen later within the BitTorrent model.

2.2.3 Reactions and Rate Expressions

The next stage requires iterating through all of the component definitions and creating a set of all unique activities within PEPA Prefixes (the set of all α's from $(\alpha, r).Id$). This equates to the set of all reactions within the model. This set can then be used in the next stage of the conversion, that of assigning species as reactants and products to each reaction.

To complete the next stage each component's direction must be discovered using the suffixes as previously shown. A component shifting from high to low is a reactant and from low to high a product. The species Id for that component is then added to each reaction it is connected to, identified by the activities. This can be seen from the PEPA fragment here

$$\begin{aligned}
\mathrm{A/X}_H &\stackrel{\mathit{def}}{=} (k2react, k2).\mathrm{A/X}_L + (k3react, k3).\mathrm{A/X}_L \\
\mathrm{A/X}_L &\stackrel{\mathit{def}}{=} (k1react, k1).\mathrm{A/X}_H
\end{aligned}$$

which translates to

$$\begin{array}{llll}
k1react, & & - > \mathrm{A/X}, & k1 \\
k2react, & \mathrm{A/X} & - >, & k2 \\
k3react, & \mathrm{A/X} & - >, & k3
\end{array}$$

adding $\mathrm{A}_H \stackrel{\mathit{def}}{=} (k1react, k1).\mathrm{A}_L + (k5react, k5).\mathrm{A}_L$ to the PEPA model introduces a reaction labelled $k5react$ and adds a reactant to reaction $k1react$ as follows

$$\begin{array}{llll} k1react, & \text{A} & -> \text{A/X}, & k1 \\ k2react, & \text{A/X} & ->, & k2 \\ k3react, & \text{A/X} & ->, & k3 \\ k5react, & \text{A} & ->, & k5 \end{array}$$

As each component has its associated species attached to reaction α via the Prefix $(\alpha, r).Id$, the rate r is compared to the rate connected to the reaction. For all prefixes α the associated rates must agree.

Nothing has been said so far about expressions within rates. Three concerns must be addressed before continuing at this point if not before. The expressions must be verified for correctness of variable naming since the alteration and amalgamation to Id's & id's, the use of function calls defined and caught and lastly differentiate between expressions which require evaluation only once during initialization or in every time-step of the analysis.

The first concern stems from the requirement to match certain id's to species derived from Id's as per the conversion process. Neither the Id nor the ISBJava id may be used within the expression as they are deemed illegal by the Choreographer syntax checker. Therefore id's shall be used and subject to the same change as previously with rates. This is easily addressed under Choreographer. The expression rate uses a tree structure for the data with every node of the tree being a rate node itself. Traversing the tree, every specified rate can be examined and the Id's & id's checked and altered when required.

In regards to the use of functions ($sin(45)$, $theta(id)$ or any variant), these are not legal in the PEPA parser in either Choreographer or the Tabasco Workbench. The use of function calls is a useful feature but at present there is no elegant solution to their use within the current version of PEPA/ISBJava process. If the use of functions is required for a model they can be used in the form $(func * (arg))$. The secondary parser within the conversion process will extract expression nodes of this form where the left node is a `PepaUndefinedRate` object, the right node an already parsed expression encompassed by brackets. If the `PepaUndefined` object has an id equal to a keyword in the list of unary functions allowable by ISBJava, the expression is a function call. The outer set of brackets ensure the function keyword is connected to the correct argument.

The use of min & max functions are currently not allowed within the conversion process, their inclusion into ISBJava happening after the initial conversion process had been defined. It was during the conversion conceptualization that the decision to introduce min and max functions was made, their use more for controlling the reaction rates. The introduction of min/max function to ISBJava has since been fed back to Systems Biology and incorporated into version 2.1.0 and their use seen in the final stages of the conversion.

ISBJava also differentiates between expressions that require evaluation in every time step[2] in comparison to variables that can be instantiated during initialization. The current system does not make the distinction and so all expressions within the model will be evaluated every time step. Since this model is not optimal, analysis within Choreographer will incur a penalty in performance over a manually altered model.

2.2.4 Stoichiometric Information

As previously reported, chemical reactions have no requirement to respect the number of unique reactants against the number of products in any one reaction. That said, chemical equations must balance based on the number of molecules involved such that $H_2 + O_2 \neq H_2O$ as one oxygen molecule has not been accounted for, whereas $2H_2 + O_2 = 2H_2O$ has been balanced. The balancing of the reactions is known as stoichiometry. The concept of requiring x of a given entity is similar to that seen in stochastic petri nets and their use of tokens rather than PEPA (with no comparable facility). The ability to specify values of species/component in any given reaction/activity can be a valuable capability when modelling, allowing the complexity of the model to be reduced. The reduction in complexity can be seen where multiple identical entities exist. By employing the use of populations and stoichiometry, a model can request four entities for a given task from a population of 10. If each entity is modelled as a individual, to request four from 10 entities requires all permutations where $\binom{10}{4} = 210$. In this instance the use of populations can reduce the size of the problem.

Rather than place more restrictions on the grammar, the decision was taken to store all stoichiometric information separate from the original model. This keeps the PEPA model free from up to $\#reactions \times \#species$ entries. The current implementation returns a window for entry of the stoichiometric information, listing each reaction formed and so also acts as a check of the conversion process so far. The separation of stoichiometric data from model can also reinforce the division between the framework of the model and the parameters.

2.2.5 Factoring Reactants into Reaction Rates

One final aspect of behaviour of ISBJava must now be considered. Using the following reactions:

$$reaction_1,\ reactant \rightarrow product,\ 1$$

$$reaction_2,\ reactant \rightarrow product,\ [1]$$

[2]In the cdml format the expression is enclosed in [] and explicitly labelled an expression within the ISBJava model.

Within section 2.2.3 the difference in reaction rates was stated as immediate against deferred evaluation. There is an additional consequence when utilizing the deferred evaluation and that is the actual expression evaluated. The reaction rate in $reaction_1$ that will be evaluated is $1 \times \#reactant$ where ISBJava factors in population sizes for all of the species involved. In the case of $reaction_2$ the reaction rate is deferred and instead defined exactly as 1. This will lead to two very different systems where $reaction_1$ shows decay $\approx 1000 \times (1 - e^{\frac{-1}{10 \times x}})$ and $reaction_2$ linear decay disrespecting population sizes for its only reactant. As $t \to \infty$ $reaction_1 \to 0$ whereas $reaction_2 \to -\infty$. To achieve the correct rate expression for $reaction_2$ [1] should be replaced with $[1 \times \#reactant]$.

As ISBJava was originally created to model chemical reactions the default rate is based on "the numeric reaction probability density per unit time, *per distinct combination of reactant molecules*"[11]. The following reaction

$$reaction_3,\ reactant_1 + reactant_2 \to product,\ 1$$

will be associated with the reaction rate $1 \times \#reactant_1 \times \#reactant_2$ or in the case of the stoichiometric example in section 2.2.4 where four entities were required for the reaction the rate would include as part of the expression $\#entity^4$. This assumption while holding true for chemical reactions does not follow for the models based on the reagent-centric design.

There are two options at this juncture, leave addition of the population density to the rates for the modeller or mimic ISBJava's behaviour and alter the designated rates in a predefined way more acceptable to the possible types of model seen. The system finally employed was alteration of the rates using the *min* function with the following rules

1. Where a reaction has only one unique reactant the reaction rate shall be multiplied by the population size of the reactant i.e. reactant A and rate r would yield a reaction rate $r \times A$.
2. Where a reaction consists of two or more unique reactants the reaction rate shall be multiplied by the smallest of the population sizes. As the *min* function is a binary argument function multiple *min* calls must be chained together i.e. reactants A, B, C and rate r would construct the following $r \times min(A, min(B, C))$ or any variation based on reactant ordering.

When components are required to cooperate in a PEPA model, the delay in the activity taking place is defined in Hillston[5] as follows

$$r_\alpha(P \underset{L}{\bowtie} Q) = \begin{cases} min(r_\alpha(P), r_\alpha(Q)) & if \alpha \in L \\ r_\alpha(P) + r_\alpha(Q) & if \alpha \notin L \end{cases}$$

The definition accounts for the basic case where each activity is used only once within the activity set for any component. The use of the simpler definition is suitable here based on the confines of the conversion process and is the argument for using the *min* function for controlling the reaction rates.

At this stage in the conversion process, the data on the different components exceeds that required to construct the ODEs[3] for the model. Once the reactions have been derived, the data is held in a form similar to that of an Activity Matrix[7, 8] which could be used to derive the ODEs instead of the ISBJava model.

2.2.6 Completion of Conversion Process

With this last component of the process in place the conversion is complete and in a form suitable for analysis using the solvers present in ISBJava. To re-iterate the process contains the following steps

1. Identify species and population size declarations from defined components and rates.
2. Zero all population sizes not previously declared.

   At this stage verification of component declarations has occurred guaranteeing declarations have a 'direction'
3. Form reactions from component declarations.
4. Modify rate expressions to conform to rate transformations within step 1.
5. Gather stoichiometric information.
6. Reaction rate expressions altered to include delays linked to population sizes or densities.

[3]ODEs do not make use of stoichiometric data.

3. Implementation

3.1 Choreographer

The Choreographer environment as seen in Figure 3.1 is based on NetBeans 3.6[1], a Java based platform which through the use of the `org.openide` package allows extensions to itself. It is the use of this developers package that allows associations with model file types for loading, editing and saving, grammar and syntax checking of the models and calls to any external packages required. Through this package, Choreographer was created and currently consists of two modules for the process algebras PEPA and LySa.

The user interacts with the PEPA module through the PEPA menu allowing access to all of the functions associated with the Tabasco Workbench as shown in Figure 3.2. Choreographer achieves this functionality by various parameter passing including the model file location for the Workbench initialization. Feedback from Tabasco to Choreographer is currently handled through redirecting the standard output to the NetBeans output tabs. This method of invocation does result in the PEPA parse tree built within Choreographer only being used for syntax highlighting (as already noted in the design chapter).

From a user interface perspective, Choreographer restricts access to the appropriate submenu's until the pre-requisites have been met. Deriving the state space is not possible until the model has been checked for syntax errors and solving is not available until the space has been derived. This sequence does have one side affect in that once the state space has been derived, solving the state space requires that the space is derived again as each call to the Tabasco Workbench is self-contained.

3.2 PEPA to Dizzy

To implement the conversion process from chapter 2.2, a NetBeans module must be created that interacts with the PEPA module for Choreographer. Following Choreographers paradigm, the conversion process is implemented as a submenu to the existing PEPA menu (as seen in Figure 3.2) and remains inaccessible until the syntax check has been made. Once checking has occurred access to the underlying PEPA model is possible and the option of converting to Dizzy is available as seen in Figure 3.3.

[1] `http://www.netbeans.org`

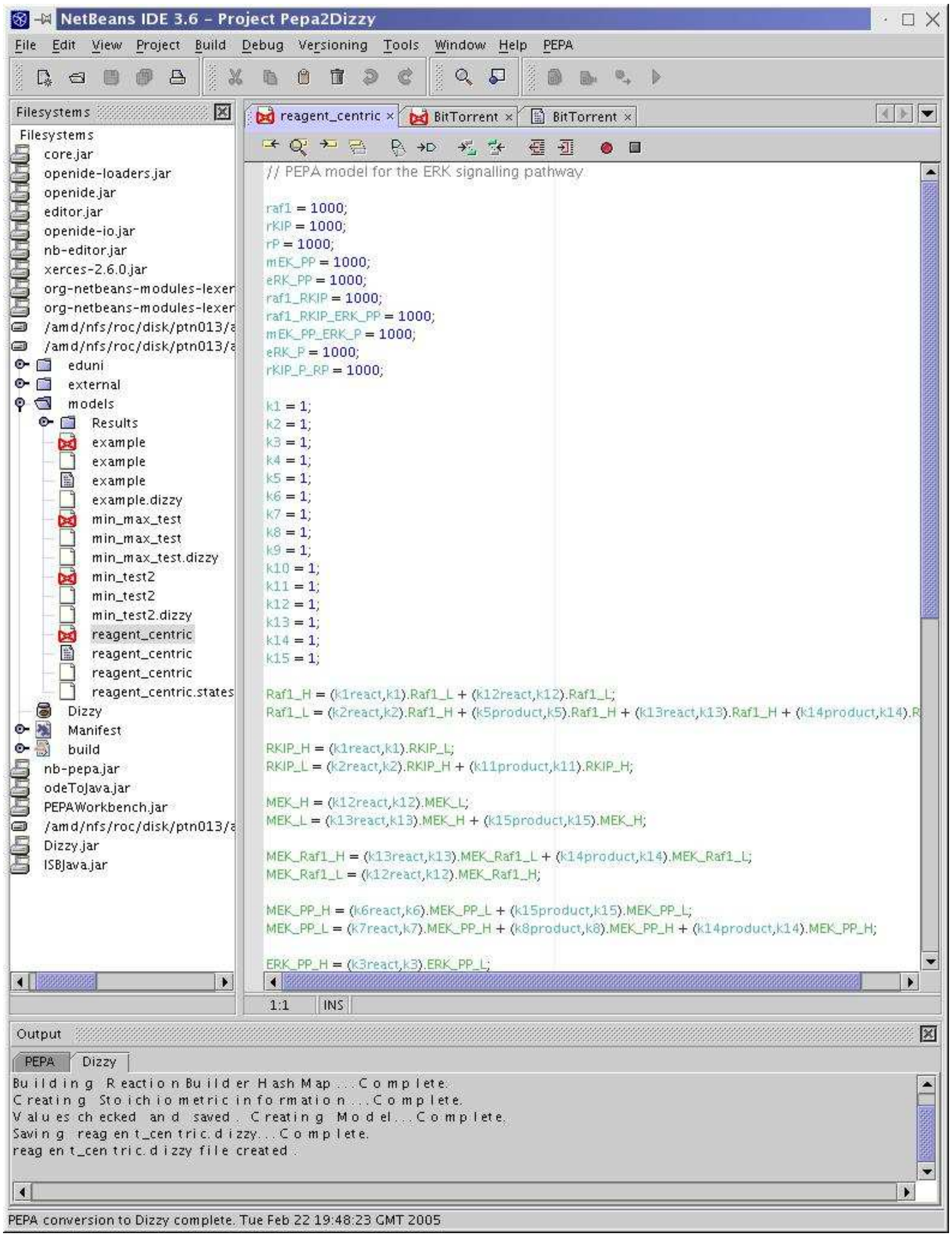


Figure 3.1: Snapshot of Choreographer

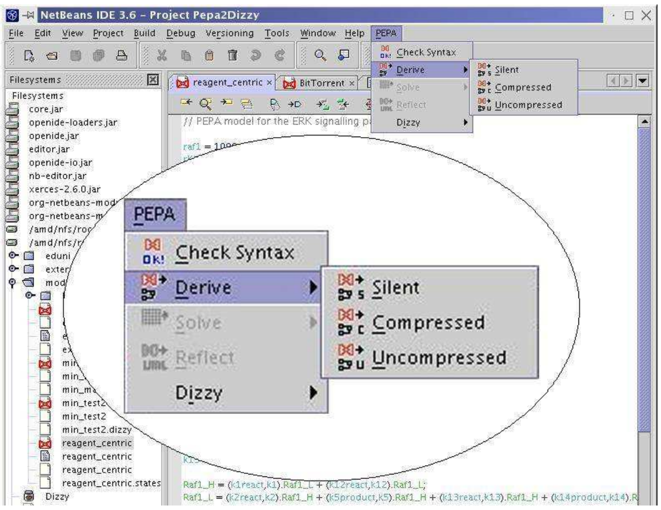


Figure 3.2: Interacting with the PEPA component in Choreographer

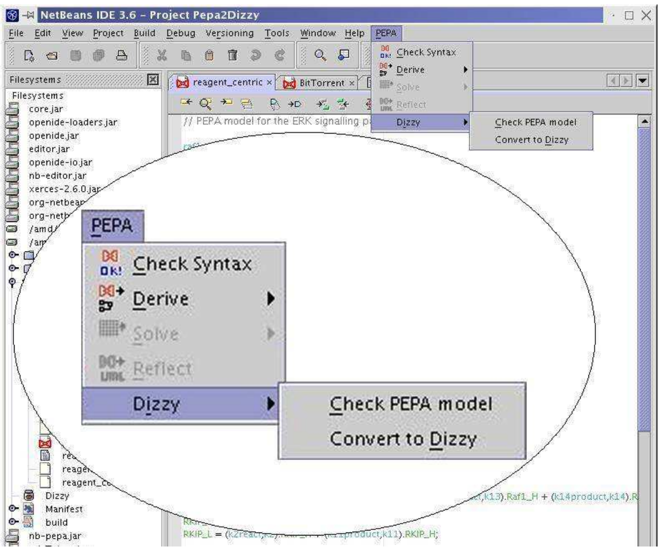


Figure 3.3: Dizzy submenu in Choreographer

3.2.1 Check PEPA model

Currently the two Dizzy related actions are Check PEPA model and Convert to Dizzy. As seen within the conversion process, the grammar for reagent-centric models differs from that of the Tabasco Workbench. The option to check the PEPA model is equivalent to deriving the state space with the expression rates replaced. As the rates do not affect the underlying structure, no detail has been lost for the purposes of validating the model and must be done for the Tabasco Workbench to accept the model.

Due to the way NetBeans currently works, access to the original file is restricted to the PEPA module. To allow this functionality in the PEPA module an accessor method, `getPepaModel()` was introduced so when the file object is passed the conversion process can access the parsed model. From here a temporary file is created whose content is equal to that of the original PEPA file but with all rates set to one.

From here the Tabasco Workbench is called in an identical manner to that of the PEPA module. It should be noted that once Choreographer reaches a release version, the module for converting will also require modification. During one of the beta releases to fix minor issues in parsing, the ability to pipe the results back from the Tabasco Workbench was lost to external use. The solution, to incorporate a modified Tabasco Workbench within the module to allow results to be piped back. This inclusion has the side effects of increasing the module size and hard wiring the process to the 0.9.4b version.

3.2.2 Convert to Dizzy

Using the access to the PEPA parse tree, the first four steps of the conversion process as summarized in section 2.2.6 are implemented as one method, `convertPepaModel()`. This separation will allow the introduction of parsing to any form of ODEs and not limit the module to just ISBJava (see future work). The method
`convertPepaModel()` currently depends heavily on hash tables for lookup. This form of storage is not optimal for creating ODEs but one iteration over the reactions hash table would allow a 2-dimensional array to be created. With this array the ODEs can be efficiently formed in the format required.

Steps five and six of the conversion process, creating the stoichiometric information and final modification to reaction rates are currently handled by the
`buildISBJavaModel(FileObject)` method, which expects all stoichiometric data to kept in a single file. The current data is read in and added to the model where possible before requesting user input for input and verification. This input comes in the form of a window as seen in Figure 3.4.

Once the values have been entered and confirmed the new data is written back to disk. With this system, alterations to rates for the purposes of experimentation can be made and there is no need to re-check the stoichiometric data. Currently there exists an implementation issue with the format used for this data. Within the file, each entry refers to a reaction and species for the purposes of identification. If a species appears as both a reactant and a product within the same reaction the system cannot differentiate and so will favour the reactant, leaving the product value blank. This error does not affect the analysis of a model,

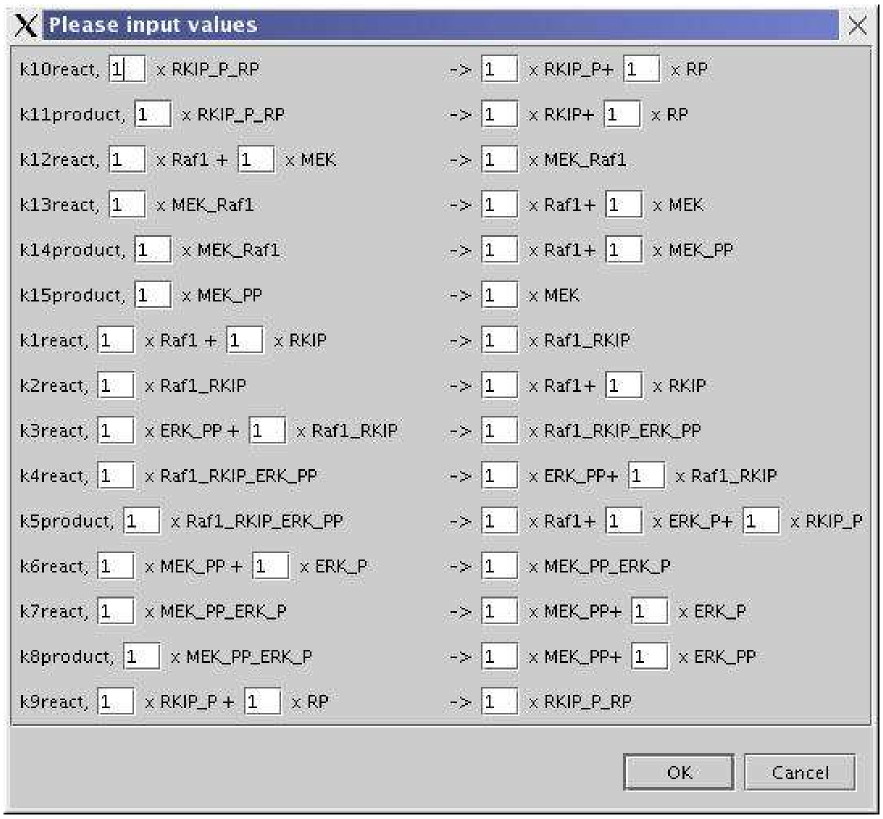


Figure 3.4: Stoichiometric input window

instead merely hindering any re-evaluation, and will be resolved with the next version.

Once finished, the ISBJava model is complete and simulations can be run. Currently the conversion action also writes a cdml version of the model to disk, thus allowing the model to be run independently. The model created does require Dizzy 1.9.0 to run. As previously noted, ISBJava has a limited number of allowable functions of which *min* and *max* were not available in 2.0.1. Hence the ISBJava within the conversion module is a modified version of 2.0.1, allowing the use of *min* and *max*. These modifications have since been sent to the Institute of Systems Biology and were incorporated in ISBJava 2.1.0, the default version included in Dizzy 1.9.0.

Access to the simulations is through the simulation window as seen in Figures 3.5 and 3.6

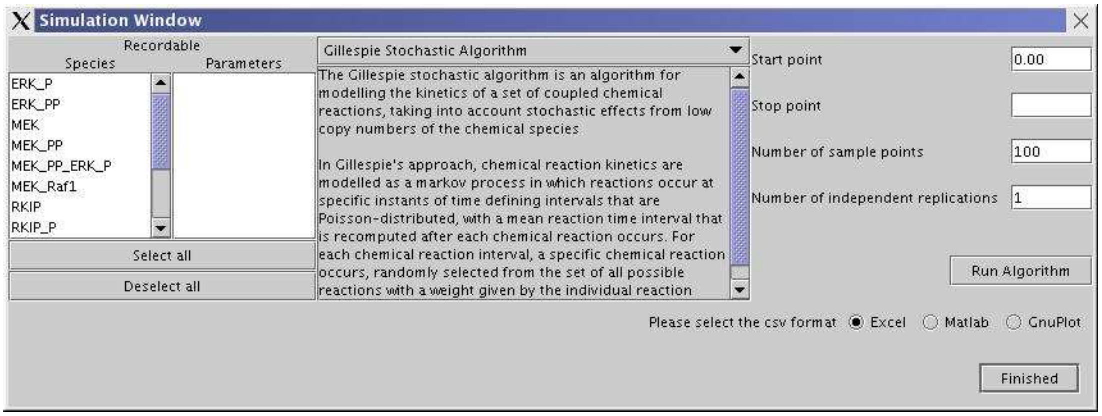


Figure 3.5:

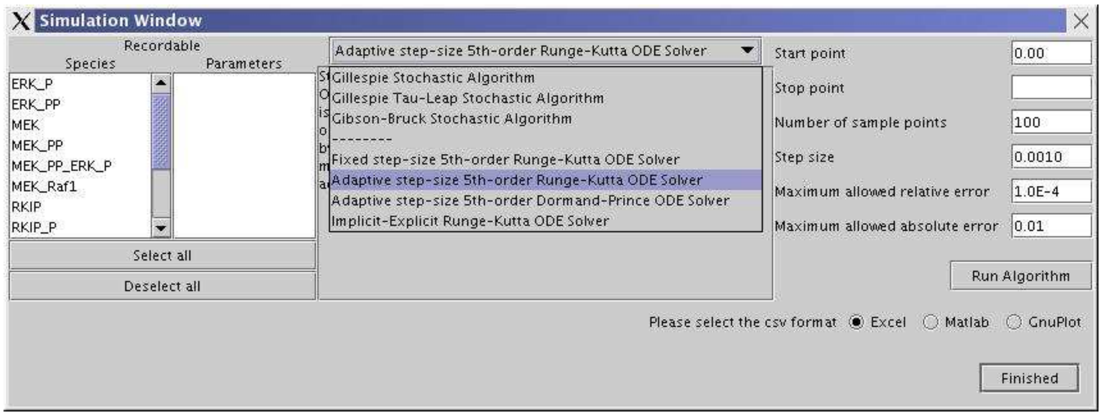


Figure 3.6:

Through the simulation window, all of the options available within Dizzy are made available here with the exception of instantly displaying the result. The modeller can select which species and parameters to record, which algorithm or solver to derive the results and set the relevant parameters for each of the solvers. The results are then automatically recorded using a combination of the model name and solver used as a filename. This does mean that currently manual renaming is required if multiple experiments are required.

4. Modelling BitTorrent

4.1 Overview

The conversion process from PEPA to ISBJava now allows a reagent-centric model of BitTorrent to be constructed. Thus model creation is centred on the component and the interactions possible compared to a reaction based approach. Adopting this approach however does impose certain restrictions to the level at which the model can operate at efficiently. With that said, these restrictions should have little impact considering previous work in this area. The modelling languages currently in use today do not lend well to modelling BitTorrent at the lower levels of detail and as noted, the ability for BitTorrent to receive and transmit pieces in an arbitrary manner creates permutations of a size that many modelling languages are not equipped to handle.

Throughout the modelling that will take place, the following assumptions will be made

- The nodes use one of the following standard connections; 1024/256, 1024/128, 512/256 or 512/128[1] and it is assumed that the entire connection is being used to acquire the content. Currently the nodes will be homogeneous, clearly unrepresentative of a real network.

- A finite number of users will represent the demand. For these modelling experiments one thousand users will be simulated to allow analysis of the BitTorrent protocol.

- The content being obtained is one 600MB file[2]

- 1% of the 600MB file is 6MB, which under optimal conditions can be download in no less than 48 seconds under a 1024 Kbps link or 96 seconds with 512 Kbps. One time unit under simulation will equate to 1 minute, with a maximum of 1% downloadable in that time. This leaves 12 seconds unaccounted for to cover overheads in transmission and less than optimal line conditions and is roughly equivalent to a download speed of 100 KB/s or 800 Kbps on a 1024 Kbps connection.

[1]Standard notation is download/upload in Kbps.
[2]An ISO image of any CD from a Linux distribution fulfils this criteria

4.2 Client-Server

To determine the effectiveness of the BitTorrent protocol there must be a base case, in this instance the client-server model. The design to be used is quite simple in most respects but should allow for a reasonable comparison with the following attributes

- The server shall provide either 1MB or 2MB upload capacity.
- The server shall provide this upload capacity at either full or half speed where full speed is defined as fully utilizing a 1024Kbps connection.
- The number of clients that can be connected at any one time is governed by the upload capacity of the server and the download speed of the clients $\frac{upload\ speed}{download\ speed}$ i.e. $\frac{2\text{MB}}{1\text{Mb}} = 16$ clients at full speed or 32 clients as half speed.
- The server cannot fail and so is always present.
- The clients once connected do not fail or pause the download.

In the case of the client-server model, the upload capacity of the clients is irrelevant making the upload capacity of the server and the download capacity of the clients the only variables requiring consideration. Allowing twice as many clients to connect at half speed has the effect of nullifying this second variable. As full speed has been defined as downloading at 1024Kbps, half speed is equivalent to 512Kbps. This implies any difference to the model by altering the download speed of the clients will only affect the start and end of the graph, where the number of clients connected is less than the threshold for half speed. For the purposes of comparison, the errors introduced by this additional simplification are acceptable.

4.2.1 Reagent-Centric PEPA Model

Based on the previous requirements and assumptions, the PEPA in Appendix B.1 was machine generated via a python script. With the number of similar components involved in the model, fewer errors will be introduced using a script, also allowing different conditions to be easily applied.

The client-server model is controlled by two rates, the initiate rate and download rate as shown in their general form.

$$\begin{aligned} initiate\_rate &= theta^3(32 - \#clients) \\ download\_rate &= 0.5 + theta(17 - \#clients) \times 0.5 \end{aligned}$$

The initiate rate restricts the number of clients that can connect, which in turn defines the upload capacity of the server. The 2MB server can host 32 clients ($32\times$ 512Kbps = 2MB) and when the uploading capacity is set to 1MB 16 clients can connect at 512Kbps. The download rate controls the speed at which the currently connected clients can connect. If the number of clients is less than or equal to 16 the rate is defined as 1, which results in downloading at 1024Kbps otherwise the rate is 0.5, which is equivalent to 512Kbps.

The use of a source component fulfils the stipulation for a finite number of peers.

$$Source_H \stackrel{def}{=} (initiate, initiate\_rate).Source_L$$

As can be seen, the transition is defined only in one direction. This dictates that the finite pool will fall to zero over time as long as the reaction is permitted to occur. The next four definitions make up the body of the model.

$$\begin{aligned} Client\text{-}0_H &\stackrel{def}{=} (download_1, download\_rate).Client\text{-}0_L \\ Client\text{-}0_L &\stackrel{def}{=} (initiate, initiate\_rate).Client\text{-}0_H \\ Client\text{-}n_H &\stackrel{def}{=} (download_{n+1}, download\_rate).Client\text{-}n_L \\ Client\text{-}n_L &\stackrel{def}{=} (download_n, download\_rate).Client\text{-}n_H \\ where &\quad n \in \{1 \ldots 99\} \end{aligned}$$

The number of clients with 0% increases by the initiate activity, the same activity within the $Source_H$ declaration. The number of clients with 0% also decreases by the $download_1$ activity which in turn increases the number of clients with 1% and so on through the n Client states. The final three declarations allow the model to track the number of clients connected at any given point and to allow analysis of the speed at which clients can complete the download.

$$\begin{aligned} Total\_Clients_H &\stackrel{def}{=} (download_{100}, download\_rate).Total\_Clients_L \\ Total\_Clients_L &\stackrel{def}{=} (initiate, initiate\_rate).Total\_Clients_H \\ Completed_L &\stackrel{def}{=} (download_{100}, download\_rate).Completed_H \end{aligned}$$

4.2.2 ISBJava Model

Through the use of the conversion process, the model seen in Appendix B.2 is generated. The rates have not changed except to account for grammar and so are not reiterated, but the flow from one state to another can be seen quite clearly

[3] The theta function returns zero if its single argument is ≤ 0 and 1 if > 0

with this different representation. The clients are introduced from the *Source* and the number of clients currently downloading is updated

$$initiate,\ Source \rightarrow Client_0 + Total\_Clients,\ [initiate\_rate \times Source]$$

Once a client has been connected it starts to download, using the following reactions

$$download_n,\ Client_n \rightarrow Client_{n+1},\ [download\_rate \times Client_n] \qquad n \in \{0 \ldots 99\}$$

The populations of each state flow in a linear fashion at a rate proportional to the number at that state and the download rate. The final step for the model, $download_{100}$ is shown below

$$download_{100},\ Client_{99} + Total\_Clients \rightarrow Completed,$$

$$at\ rate\ [download\_rate \times min(Client_{99}, Total\_Clients)]$$

As can be seen, the client having completed the download removes itself from the pool thus allowing another client to connect in conjunction to incrementing the completed counter for statistics. The conversion process introduces the use of the *min* function into the rate. This does not affect the rate in this instance as $Total\_Clients = \sum_{n=0}^{99} Client_n \Rightarrow Total\_Clients \geq Client_{99}$, and so equivalent to a rate of $Client_{99}$.

4.3 BitTorrent

Instead of modelling each piece within a torrent the percentage complete will be used, identical to the form taken for the client-server model. This approach was the smallest of the three suggested by Argent-Katwala[1] but also the least representative of the actual protocol. This abstraction can have severe consequences to the validity of the model. Without careful precautions it is possible for the model to break down catastrophically, an example being the original seed quitting before the entire file has been transmitted at least once amongst the peers. In this instance, if checks are not in place peers may still be capable of completing the torrent. Measuring the percentage of the torrent complete in itself may abstract too far from the actual system even without these issues.

Justification for the fluid style model stems mainly from the use of the rarest first download strategy. The use of this strategy within the BitTorrent clients is to ensure every piece is as prevalent as any other. Hence all peers with knowledge of only a few neighbours should still have access to all of the torrent[4]. One other restriction placed on this particular model is the permanence of the original seed.

[4] This decision is also made and analysed by Dongyu and Srikant[3].

For any distributor of open source software such as a Linux distribution, the availability of a permanent seed can be taken for granted. The introduction of these two assumptions allows a more abstract model to be created without creating flaws, and in doing so measure performance of a BitTorrent network from a global perspective.

This leaves controlling the ability to upload and download between peers. The approach uses a global pool of bandwidth based on the number of peers currently online. The justification of this decision stems from the tit-for-tat strategy present in BitTorrent. With each peer trying to maximize their connection the behaviour can safely be abstracted to this more general form. For each peer that is online and has at least part of the torrent one unit is contributed to the pool. To download 1% a peer requires a certain number of units from the same pool. Thus, anybody can download as long as enough people are online but the total number of people that can download is defined by the ratio of units added to the pool in comparison to the number required to download. This method does raise issues over the accuracy of the data that will be returned. Near the beginning of the torrent there will exist a point in time where there is one seed and n peers. If downloading requires x units then when $n = x - 1$ only one download is occurring in a single time-step, at $n = x$ 2 downloads. As x increases in size the error increases with it. Analysis of the model will hopefully indicate the seriousness of the issue.

This leads to approximately 200 states, 100 for those peers online (0-100%) and 100 for when a peer is downloading (again, 0-100%). The size of the model so far is small enough to allow modelling an off-line state resulting in just over 300 states for the peers. Once a peer has 100% they become a seed where they can either stay online, temporarily go off-line or quit permanently. As already stated, Pouwelse *et al* [10] found that only 17% of approximately 53,000 peers stayed connected longer than one hour after finishing downloading a particular torrent. This supports Bram Cohen's reasons for enforcing uploading whilst downloading and the rates for quitting should reflect this trend.

To summarize:

- The model will be based on percentage completed, 0-100%.
- The original seed is permanent and will upload at a rate of 1024Kbps.
- There are no restrictions as to how many peers are connected at any given point.
- Peers can disconnect and reconnect.
- Once a peer has obtained at least 1% of the file it contributes 1 unit to the upload pool for as long as the peer is online (either online or downloading).

- To download a peer must reserve x units (where $x = \frac{download\ speed}{upload\ speed}$).
- Additional seeders may disconnect/reconnect and quit permanently and seed at their upload rate.

4.3.1 Reagent-Centric PEPA Model

This leads to the model as shown in Appendix C.1, again using a python script to create the PEPA. As can be seen, the model created is substantially larger than the client-server model but has grown only at a linear rate to the number of states that a peer may exist in.

Many of the rates are self explanatory as to their function, such as *online* and *offline* rates for peers. The connect rate controls the number of peers that may concurrently download at any time using the same system as seen in the client-server model

$$connect\_rate \quad = \quad theta(\#upload\_pool - (x - 1))$$

This general form of the rate relies on the number of upload units required for one peer to download (x) and simply states that the rate returned is one if there are enough units left for another peer to do so. The rates for seeds quitting and disconnecting also work in the same manner enforcing the restriction of the original permanent seed.

$$\begin{aligned} seed\_offline &= theta(\#seed - 1) \times 0.05 \\ seed\_quit &= theta(\#seed - 1) \times 0.1 \end{aligned}$$

The two rates maintain the presence of one seed within the network while any additional seeds are subject to change at the specified rates (as *theta* returns 1 or 0 for additional seeds ($\#seed \geq 2$) the rate is 0.05 or 0.1 respectively).

The use of the word *connect* has been muddled, in that *connect* within the model refers to connecting to peers to allow downloading in comparison. Having the peer connect to the tracker for the first time is through the torrent action.

$$torrent\_rate \quad = \quad \frac{0.7}{source + 1}$$

This rate was found through trial and error while attempting to approximate the flashcrowd effect as observed by Pouwelse *et al.*

Now that the rates have been specified, the actual model requires definition. As stated, there are three states that a peer can exist in, online, off-line, downloading.

Upon entry to the system the peers will start in an online state.

$$\begin{aligned}
Online_Peer\text{-}0_H &\stackrel{def}{=} (connect_0, connect_rate).Online_Peer\text{-}0_L + \\
&\quad (offline_0, offline_rate).Online_Peer\text{-}0_L \\
Online_Peer\text{-}0_L &\stackrel{def}{=} (torrent, torrent_rate).Online_Peer\text{-}0_H + \\
&\quad (online_0, online_rate).Online_Peer\text{-}0_H
\end{aligned}$$

Which shows that the available actions to an online peer with no content downloaded are to connect to peers (and start downloading) or return off-line). To enter the state of an online peer with no content the peer must return from being off-line with no content or be registering with the tracker for the first time (through the torrent activity).

$$\begin{aligned}
Online_Peer\text{-}n_H &\stackrel{def}{=} (connect_n, connect_rate).Online_Peer\text{-}n_L + \\
&\quad (offline_n, offline_rate).Online_Peer\text{-}n_L \\
Online_Peer\text{-}n_L &\stackrel{def}{=} (downloaded_n, downloaded_rate).Online_Peer\text{-}n_H + \\
&\quad (online_n, online_rate).Online_Peer\text{-}n_H \\
&\quad n \in \{1 \ldots 99\}
\end{aligned}$$

The general declarations are almost identical to the initial state with the exception of the torrent activity being removed. A positive rate of change or increase in population is instead possible by the *downloaded*$_n$ activity.

The declaration for online seeds is similar in that they can go off-line and return but unlike peers can quit and only become seeds once the whole file is downloaded as can be seen below

$$\begin{aligned}
Seed_H &\stackrel{def}{=} (seed_quit, seed_quit_rate).Seed_L + \\
&\quad (seed_offline_rate).Seed_L \\
Seed_L &\stackrel{def}{=} (downloaded_{100}, downloaded_rate).Seed_H + \\
&\quad (seed_online, seed_online_rate).Seed_L
\end{aligned}$$

The online state represents one of the more complex aspects of the proposed BitTorrent model. The off-line and downloading states are limited in their transitions as is expected.

$$\begin{aligned}
Offline_Peer\text{-}n_H &\stackrel{def}{=} (online_n, online_rate).Offline_Peer\text{-}n_L \\
Offline_Peer\text{-}n_L &\stackrel{def}{=} (offline_n, offline_rate).Offline_Peer\text{-}n_H \\
&\quad n \in \{0 \ldots 99\}
\end{aligned}$$

with similar declarations for the additional seeds in the off-line state.

Downloading continues with a similar structure to allow connecting followed by downloading where the ability for the peers to progress is incorporated.

$$
\begin{aligned}
Downloading\_Peer\text{-}n_H &\stackrel{def}{=} (downloaded_{n+1}, downloaded\_rate).Downloading\_Peer\text{-}n_L \\
Downloading\_Peer\text{-}n_L &\stackrel{def}{=} (connect_n, connect\_rate).Downloading\_Peer\text{-}n_H \\
& \quad n \in \{0 \ldots 99\}
\end{aligned}
$$

The last component is that of the upload pool used to track the amount of upload capacity within the network

$$
\begin{aligned}
Upload\_Pool_H \stackrel{def}{=} & (connect_n, connect\_rate).Upload\_Pool_L + \\
& (offline_p, offline\_rate).Upload\_Pool_L + \\
& (seed\_offline, seed\_offline\_rate).Upload\_Pool_L + \\
& (seed\_quit, seed\_quit\_rate).Upload\_Pool_L \\
Upload\_Pool_L \stackrel{def}{=} & (downloaded_s, downloaded\_rate).Uploaded\_Pool_H + \\
& (online_p, online\_rate).Upload\_Pool_H + \\
& (seed\_online, seed\_online\_rate).Upload\_Pool_H \\
& \forall n, p, s \text{ where } n \in \{0 \ldots 99\},\ p \in \{1 \ldots 99\} \ \&\ s \in \{1 \ldots 100\}
\end{aligned}
$$

This large declaration allows the model to keep track of the upload pool by all of the actions that would influence the pool in either direction.

4.3.2 ISBJava Model

Through the automated conversion process the model in Appendix C.2 is generated. The model consists of roughly 5 blocks of reactions.

$$connect_n, Online\_Peer_n + Upload\_Pool \times 4 \rightarrow Downloading\_Peer_n,\ n \in \{0 \ldots 99\}$$

$$at\ rate \qquad [connect\_rate \times min(Online\_Peer_n, Upload\_Pool]$$

As upload and download bandwidths have been normalized by the higher download speed of 1024Kbps, the requirement of four upload 'tokens' allows the modeller to know this model is referring to the group of peers with the 256Kbps upload stream (it takes four peers upload to supply this peer at full speed) and transforms one online peer to a downloading peer at the given rate.

The off-line reactions and online reactions work in a near identical fashion and there is no need in detailing the reactions created. The downloaded reactions have a unique case in $downloaded_0$ and $downloaded_{100}$ as can be seen

$$\begin{aligned}
downloaded_1,\quad & Downloading_Peer_0 && \rightarrow Online_Peer_1 + Upload_Pool \times 5, \\
& && \quad \text{at rate } [downloaded_rate \times Downloading_Peer_0] \\
downloaded_n,\quad & Downloading_Peer_{n-1} && \rightarrow Online_Peer_n + Upload_Pool \times 4, \\
& && \quad \text{at rate } [download_rate \times Downloading_Peer_n] \\
downloaded_{100},\quad & Downloading_Peer_{99} && \rightarrow Seed + Completed + Upload_Pool \times 4, \\
& && \quad \text{at rate } [download_rate \times Downloading_Peer_{99}] \\
& && \quad n \in \{2 \ldots 99\}
\end{aligned}$$

Once a peer has downloaded one part of the file they are now capable of uploading, a state shown by the additional unit added to the upload pool in the first reaction. The extra components within the final reaction allow for the completion statistics to be analysed and is the point when a peer becomes a seed.

Reactions for seeds and for peers to join the torrent also exist as can be seen in Appendix C.2 and deviate little from the style of reactions already seen, therefore a detailed inspection can be omitted.

4.3.3 Rates

Within the client-server model there existed only two rates, one for controlling connections to the server and another for the speed at which a download could sustain. Due to the fixed number of clients connecting, the connection rate was simply one or zero. The downloading rate was previously determined to be equal to 1 for 1024Kbps connections and 0.5 for 512Kbps.

The model for the BitTorrent model is more complex with several rates which influence the behaviour. The rates arrived at for peers going off-line or online are not based on any prior information, in fact experimentally it would require extensive tracking to attempt and calculate averages. The rates instead were assigned to force a certain number of peers to appear off-line with $\mu = 20$ minutes for going off-line in comparison to 100 minutes to return to an online status. The same rates were used for the additional seeds and their ability to start and stop as they choose. While these rates may not equate to observable behaviour, the effect of exaggerating these rates can only have a negative effect on the time to complete, thus not biasing the results in favour of BitTorrent.

The ability for a seed to alter their status was shadowed by the rate at which seeds would disconnect completely. Setting $\lambda = 0.1$, this gives the additional seeds a $\mu = 10$ minutes before quitting and a 95% of quitting within a half hour. This is a higher rate of disconnection that observed by Pouwelse *et al*, (where a more suitable value would be $\lambda = 0.05$) but has been increased to account for the smaller size of file downloaded and also to highlight the effectiveness of the BitTorrent protocol. This figure is in stark contrast to that used by Dongyu and Srikant where the rates used were $\lambda = 0.001$ and $\mu \approx 16.5$ hours.

This leaves the rates for connecting to the tracker and connecting to other peers. As stated earlier, the rate for connecting to the tracker was found experimentally in an attempt to mimic the flashcrowd effect and the connect rate for other peers set to a value of one. The connect rate has not been designed to affect the system as such. The reaction it is associated with is an intermediate step to enforce good behaviour when downloading and so the rate is almost arbitrary. If the system was attempting to discover issues with poorly connected graphs this may allow measurement but would require careful analysis before doing so.

As can be seen, the model differs from those conceptualized by Argent-Katwala and created by Dongyu and Srikant. Though still highly abstracted, the model incorporates the idea of completeness of the download and the off-line state. As it is unknown how well ISBJava will come with large systems, a small model for BitTorrent has been devised with opportunity for growth if the solvers can cope.

5. Results and Analysis

5.1 Results

The models produced were initially run over a period of 10080 minutes or 7 days recording the current state of several variables every 30 minutes. In the case of the client-server 1MB upstream it was found that 7 days was not long enough for the model to reach the steady state of 1000 completed downloads. In this one case the duration of the simulation was increased to 14 days whilst maintaining recording every 30 minutes.

Of the solvers available, the adaptive and fixed step-size 5^{th} order Runge-Kutta ODE solvers were found to not terminate[1] during solving and so were not used. While the adaptive step-size 5^{th} order Dormand-Prince ODE solver was found to successfully solve the client-server model, suggesting a more robust error estimation function within the solver, it also failed to terminate on the BitTorrent model.

The performance of the different solvers can be seen in Table 5.1. The results from the different solvers can only be taken as an indication to general performance. It is clear that many of the solvers had trouble handling the theta function and the results could be significantly different on a range of models. The tests were performed over 10080 time units, sampling 366 times and recording one parameter.

Solver	client-server model (101 reactions)	BitTorrent model (404 reactions)	BitTorrent model (4004 reactions)
Gillespie Stochastic	7s	57s	57m
Gillespie Tau-Leap Stochastic	Use of non-differentiable functions (theta,min)		
Gibson-Bruck Stochastic	3s	1m 24s	did not finish
Fixed step-size 5^{th} order Runge-Kutta ODE solver	theta function prohibited testing		
Adaptive step-size 5^{th} order Runge-Kutta ODE solver	19m 58s	theta function prohibited testing	
Adaptive step-size 5^{th} order Dormand-Prince ODE solver	7m 47s	did not finish	
Implicit-Explicit Runge-Kutta ODE solver	did not finish		

Table 5.1: Performance of ISBJava solvers

[1] This was reported back to the Institute of Systems Biology that confirmed the error, a result from the error estimation function in the solvers reacting badly with the use of the theta function.

Instead, the Gillespie Stochastic algorithm was used in all experiments with the results averaged over 5 independent replications.

5.1.1 Client-Server Model

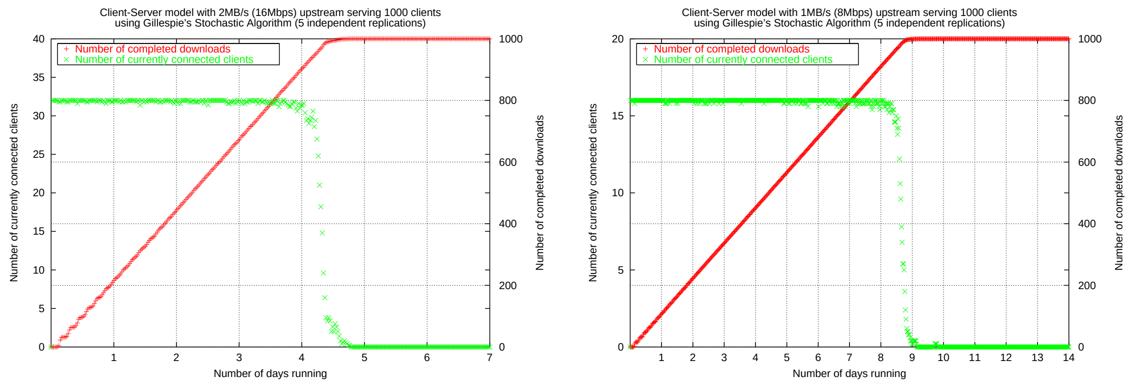


Figure 5.1: Client-Server Model

The results from the client-server model can be seen in Figure 5.1. The left axis of each graph represents the number of connected clients, with a constant number connected for the majority of the downloading lifespan. The right axis represents the number of completed downloads and as expected, the results show a linear growth over time. These results do not reflect real servers, but instead represent a possible best case situation of the client-server model under near optimal conditions. It can also be easily seen that the difference between the 2MB and 1MB upstream is double the time required to complete.

Little information can be gained from the graphs in Figure 5.1, with the exception of validating the model created and supplying the base case for comparison against the BitTorrent protocol.

5.1.2 BitTorrent Model

Figure 5.2 shows the graphs for each of the four combinations of connections used during the analysis of the BitTorrent protocol. The rows represent download speeds where the top row is 1024Kbps and the lower 512Kbps, and the columns indicate upload speeds, left 256Kbps and right 128Kbps.

As can be easily seen from the graphs, the BitTorrent model is capable of completing an identical number of downloads in a smaller period of time than the client-server model. Of the four connections used, the slowest (1024/128 Kbps) completes 1000 downloads in approximately 97 hours or 4 days and 1 hour. In

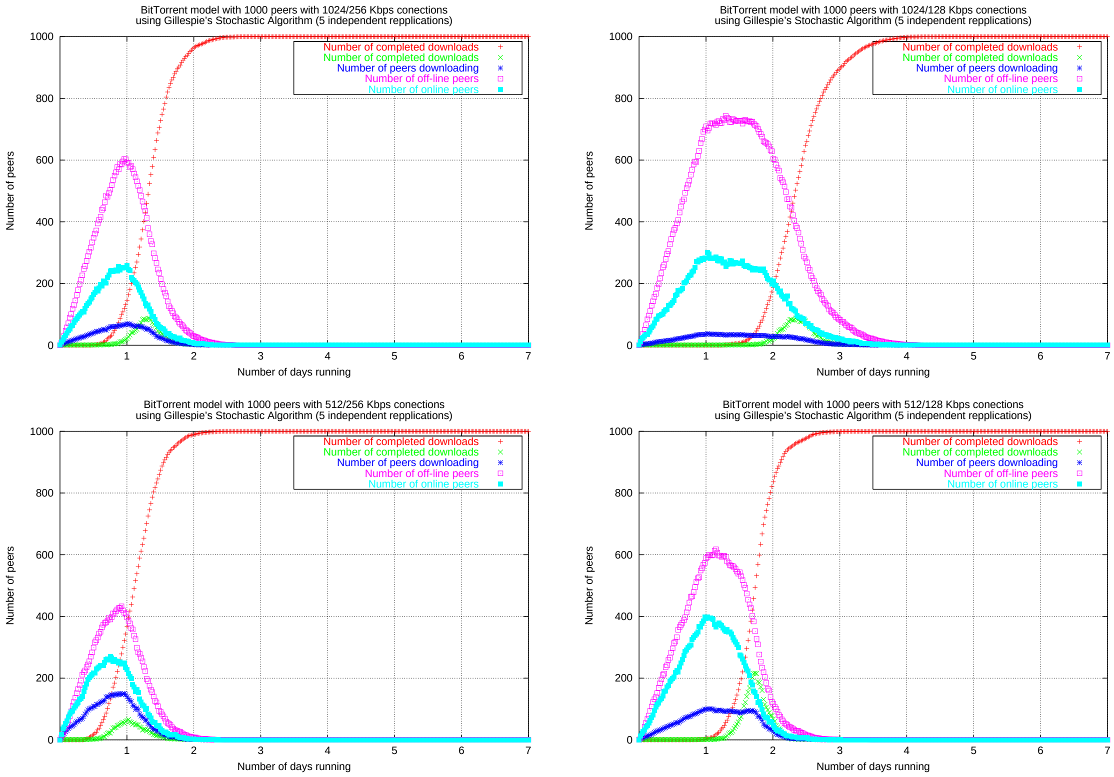


Figure 5.2: BitTorrent Model

contrast, the client-server model with 2MB upstream completes the same number of downloads in 114 hours or 4 days and 18 hours. This gives a reduction of 14 hours between the two methods of distribution, equal to a 17.5% increase in performance. The increase is less than expected based on the previous work on BitTorrent and expectations of the model. When the client-server model is compared against the 512/128 Kbps connection the results change. The 512/128Kbps model completes the same number of downloads in 77 hours or 3 days and 5 hours, 37 hours faster than the 2MB client-server model for an increase of 48% and 25% faster than the 1024/128Kbps BitTorrent model.

These results would seem to contradict both Dongyu and Srikant's analysis and the physical laws of transmission. As stated by Dongyu and Srikant, the speed at which a download can occur is the minimum of either the download ability or the upload capacity ($min\{cx(t), \mu(\eta x(t) + y(t))\}$). In the results observed from the BitTorrent model if the upload capacity was the bottleneck, then altering the download speed should not change the time to complete. If the download speed is the bottleneck decreasing the bandwidth should increase the time required, not decrease it.

The answer to this anomaly is a combination of the discretization of capacity and

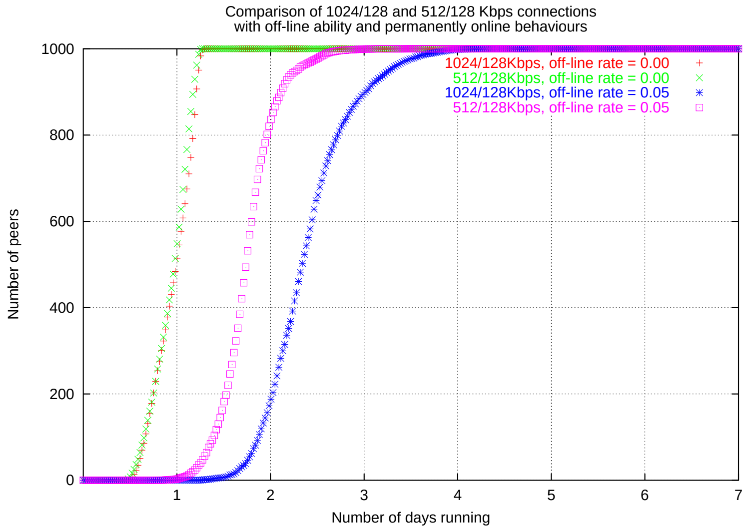


Figure 5.3: Comparing off-line rates 0.00 and 0.05 in the BitTorrent Model

the ability to go off-line. As Figure 5.3 highlights, when the ability for a peer to go off-line is removed ($\lambda = 0.00$), there is no difference between the 1024/128 and 512/128Kbps connections as was originally surmised. When the peers are allowed to disconnect from the tracker the results diverge. This was an unexpected result, but a review of the model can show where the problem lies.

The difference between downloading at 1024Kbps and 512Kbps is within the downloading rate. It takes a peer twice as long to download at 512Kbps, a valid observation and modelled as such. The effect to the implemented model is that peers with $512/x$ connections are placed in the downloading state for twice as long, a state in which a peer cannot make the transition to the off-line state. As the potential upload capacity of the two connections used in Figure 5.3 is identical, up to twice the number of peers with 512/128 connections may occupy the downloading state at any given point and so again cannot make the transition to off-line. This can also be seen in Figure 5.2 when comparing 1024Kbps download connections against it's equivalent 512Kbps connection. In these two cases, the 1024Kbps connections show a larger proportion of peers in the off-line state and a lower number of peers downloading concurrently.

While the issues surrounding the ability to go off-line may not allow a clear comparison between different bandwidths within the BitTorrent network, the results are still valid and can be used against the client-server. As already shown, even with high levels of peers appearing off-line, BitTorrent still contains the capacity to transfer at an increased rate over that of the client-server model. Not only does Figure 5.3 highlight the unexpected influence of the off-line status but also shows a substantial increase in overall performance when peers are constantly connected.

In the case of *offline_rate*[2]= 0.00, 1000 downloads are complete within 30.5 hours or 1 day and 6.5 hours. This speed of transfer offers a 273% increase over the 2MB client-server model and 670% increase over a host with 1MB upstream.

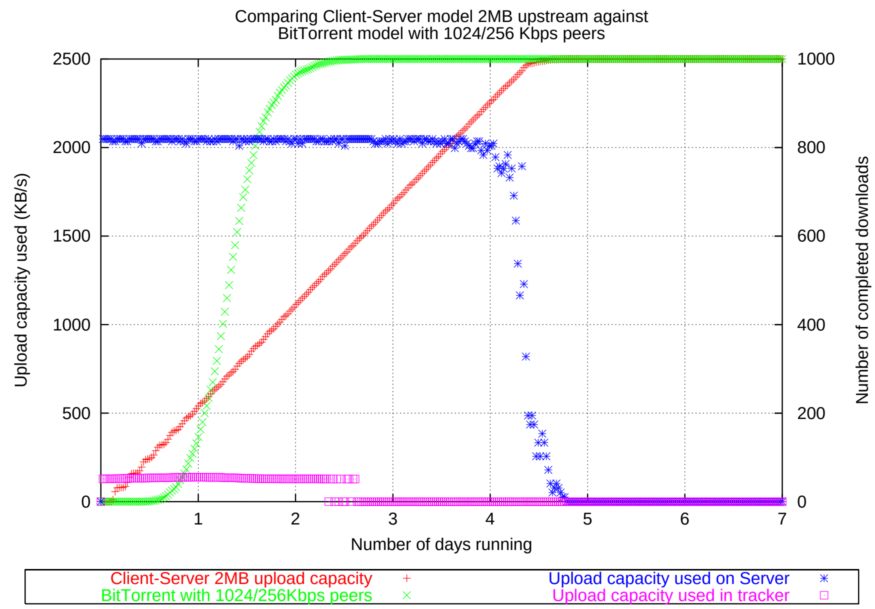


Figure 5.4: Comparing Client-Server model against BitTorrent

The graph in Figure 5.4 shows key aspects of the BitTorrent protocol over the standard server model. What is shown is the time for completion for 1000 downloads in both models and the upload capacity used each second to sustain these downloads. The right hand axis (measured in KB/s) shows current upload capacity being used by both models and the left hand axis for number of completed downloads. Unsurprisingly the server within the client-server model sustains an upload of 2MB per second for just over 3.5 days, starting to drop slightly at this point. In comparison the BitTorrent protocol performs using substantially less upload capacity at approximately 150 KB per second or 0.15 MB per second at its peak. This value accounts for a 5KB response to tracker update requests averaged over a 3 minute span in additional to the original seed being hosted on the same server as the tracker. So in each minute the server hosting the tracker is responding to one third of the seeding, online or downloading peers request messages and uploading at 1024Kbps if there is at least one peer currently downloading.

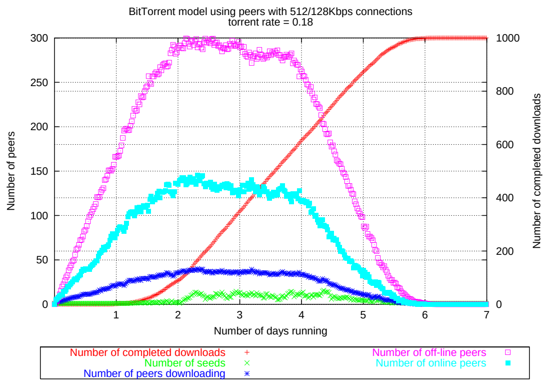
[2]peers only, the seeds off-line rate was not altered

5.1.3 Flashcrowd Effect

The speed at which the downloads occur under BitTorrent is in actuality not surprising, and the flashcrowd effect that Pouwelse *et al* measured increases this effect. In the results from the BitTorrent model using 1024/256 connections, the number of additional seeds and peers either online or currently downloading peaked at 348. This represents an upload capacity available to use of 11MB/s for a load of no more than 150KB/s, over 5 times the bandwidth available from a traditional server. This of course was peak value and not the sustained capacity within the BitTorrent network but goes some way to show why BitTorrent is so effective at distributing content.

As the flashcrowd effect is not just manageable but actually beneficial to the BitTorrent model, the question becomes how much influence does this effect exert within the model. To try and highlight the impact changes were made to the parameters over 2 simulations. The first change was to the rate at which new peers connected to the tracker for the first time ($torrent\_rate$). The rate was altered to simulate the actual connection rate seen for the client-server model where the clients were still awaiting connection after 4 days and was approximated to $torrent\_rate = \frac{0.18}{source+1}$. All other parameters were left at their initial values. As the initial rates for peers disconnecting and seeds quitting were aggressive, it is expected that the BitTorrent model will not perform as well as seen already.

The second modification changed both the peer off-line rate and the seed quit rate in addition to the new connection rate. This experiment models the seed quit rate measured by Pouwelse *et al* setting $\lambda = 0.016$ resulting in $\mu = 62.5$ minutes. The rate for peers entering the off-line state was set to zero. This change will represent the users desire to obtain the file and as such their willingness to wait for the download to complete. As before, all other rates were left at their initial values.



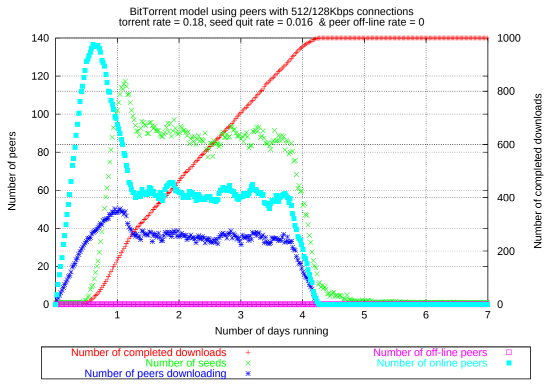


Figure 5.5: BitTorrent without the flashcrowd effect

In the previous comparison between BitTorrent and the client-server model, the BitTorrent network consisted of 1024/256Kbps peers. To study the effects of removing the flashcrowd effect the peers were changed to 512/128Kbps. Along with the alterations to the rates, the BitTorrent model can now be said to more closely mimic a real network. Figure 5.5 shows the results from altering the model to remove the flashcrowd effect. The graph on the left shows the initial effect of reducing the *torrent_rate* and the right hand side the added effect of more closely following the results found by Pouwelse *et al.* The two graphs show the same data as seen previously in Figure 5.2 with the notable exception of scale. With the flashcrowd effect, the off-line state represents the largest proportion of peers with 512/128Kbps connections, peaking at around 600 during the simulation. Without the flashcrowd effect the number of off-line users peaks at half that. For the graphs to be readable the left y axis indicates the number of seeds and peers in the different states and the right hand y axis for the number of completed downloads.

What can be seen is the flashcrowd effect does contribute heavily to the performance within a BitTorrent network. Without this effect, the time required is over double. These results are still based on the aggressive seed quit rate and peer disconnect as discussed. When these parameters are changed the results in the graph on the right hand side in Figure 5.5 are obtained. The increased number of peers and seeds within the system removes almost two entire days to complete the 1000 downloads. bringing the performance in line with the client-server.

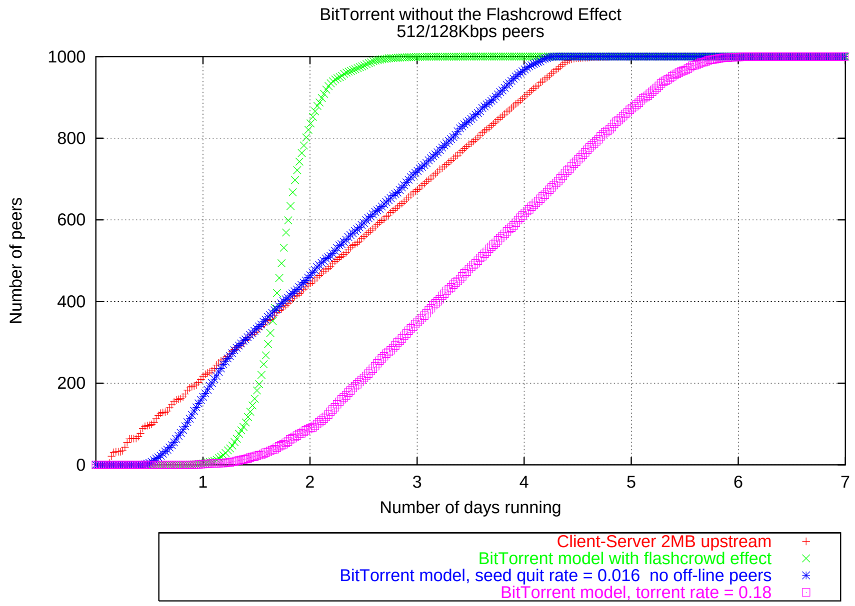


Figure 5.6: Comparing Client-Server against BitTorrent under varying network conditions

The results in Figure 5.6 show the BitTorrent protocols performance under varying conditions against the client-server model. Even under less than optimal conditions, BitTorrent performs well in comparison to the traditional server. Attempting to model the behaviour seen within measured BitTorrent networks brings BitTorrent in line with the performance of the server but at the fraction of the cost of doing so as previously seen in Figure 5.4. These results confirm one of Bram Cohens original goals, that of reduced costs for the hosts bandwidth.

The results from the flashcrowd behaviour also bring to light the strengths of BitTorrent. Not only can the BitTorrent protocol cope well with flashcrowds but gains substantially from the situation.

5.2 Conclusion and Future Work

Despite the simplicity of the models used, the results gained were positive. The BitTorrent model produced allowed a coarse grained view to the current status of a new torrent with finite number of peers whilst also validating the approach of the reagent-centric model as proposed by Calder *et al*.

Based on the proposal to convert PEPA models to ODE form[7], the conversion process between PEPA and ISBJava has been defined and implemented as a module for Choreographer. The module allows PEPA models based on the reagent-centric model to be analysed within the ISBJava environment through one of several solvers available. The module still requires work, the ISBJava within the module needs updated to the latest version and the module grammar could be altered to allow binary argument functions in a similar format to that used for the unary functions already present. The storage of the stoichiometric information also requires correction to allow storage of data in the event of a species appearing as both a product and reactant in the same reaction. In its current state the module succeeds in its main tasks, successfully converting models in the defined grammar and allowing analysis.

The BitTorrent model performed better than expected. Using a similar approach to the fluid model, it has been shown that the use of the ISBJava modelling environment allows high level abstraction of BitTorrent to be successfully simulated, highlighting the strengths of this form of content distribution. Analysis of the model supports one of the original BitTorrent claims, that of reduced hosting costs while also showing that downloading times can be reduced as well. The model also allowed inspection of the flashcrowd claims by Pouwelse *et al*, suggesting that the findings of SuprNova were not unique and that in fact BitTorrent benefits, showing increases in global performance under the increased load.

Not all the results were positive. It is clear that the design has it's flaws in how the off-line status was implemented in the model, disallowing any analysis of peer behaviour. The model also does not lend itself well to analysing specific components of the BitTorrent protocol, with several key aspects of the model based on expected behaviour of the peers as a whole. This weakness however is the cost paid by all models of BitTorrent so far to allow analysis to take place.

Despite the performance of the various solvers it may be possible to increase the complexity of the BitTorrent model. As already seen, the discretization of the download upload rates brought certain approximations, one of which allowed the peers to behave differently where it was not expected. A more fine grained approach to use of bandwidth should create a model more tolerant to change and possibly decrease analysis time (assuming the removal of the theta function). The model also does not currently allow for analysis with regards to failure, an area in which BitTorrent should show further gains over the traditional server. To allow any form of measurement would require a more detailed view of the content being held by the peers but would allow many aspects to be introduced.

Based even on this exploratory work, the BitTorrent protocol would appear to offer hosts and users gains in distributing content.

Bibliography

[1] Argent-Katwala, A. (2004) *Modelling BitTorrent with PEPA*
Third workshop on Process Algebra and Stochastically Timed Activities (PASTA) http://www.nesc.ac.uk/esi/events/290/

[2] The official BitTorrent Home Page
Retrieved June 22, 2004, from http://bittorrent.com

[3] Dongyu Q. & R. Srikant (2004) *Modelling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks*
Proceedings of the ACM Sigcomm http://www.ifp.uiuc.edu/~dqiu/paper/bittorrent.pdf

[4] Haenel, N.V. (2003) *User Guide for the Java Edition of the PEPA Workbench - Tabasco Edition*
The Laboratory for the Foundations of Computer Science (LFCS), School of Informatics, The University of Edinburgh

[5] Hillston, J. (1996) *A Compositional Approach to Performance Modelling*
Cambridge University Press

[6] Leon-Garcia, A. and Widjaja, I. (2004) *Communication Networks Fundamental Concepts and Key Architectures* (2nd ed).
McGraw Hill

[7] Calder, M., Gilmore, S. & Hillston, J. (2004) *Modelling the influence of RKIP on the ERK signalling pathway using the stochastic process algebra PEPA*
Preliminary version from Proceedings of Bio-CONCUR 2004, 2^{nd} workshop on Concurrent Models in Molecular Biology, Electronic Notes in Theoretical Computer Science

[8] Calder, M., Gilmore, S. & Hillston, J. (2004) *Automatically deriving ODEs from process algebra models of signalling pathways*
Preliminary version obtained from the LFCS, School of Informatics, The University of Edinburgh

[9] Gilmore, S. (2001) *The PEPA Workbench: User's Manual*
Retrieved June 22, 2004, from http://www.dcs.ed.ac.uk/pepa/pwb.pdf

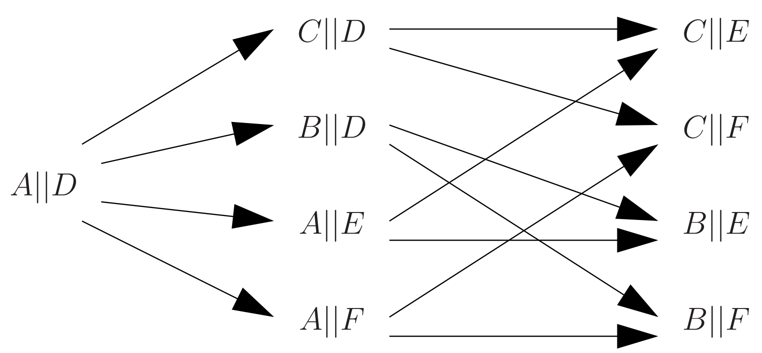
[10] Pouwelse, J.A., Garbacki, P., Epema, D.H.J., and Sips, H.J. (2004) *The BitTorrent P2P File-sharing System: Measurements and Analysis*
Retrieved December 18, 2004, from www.isa.its.tudelft.nl/~pouwelse/Bittorrent_Measurements_6pages.pdf

[11] Ramsey, S. (2004) *Dizzy User Manual* Version 1.8.2
Retrieved December 29, 2004, from `http://labs.systemsbiology.net/bolouri/Dizzy/docs/UserManual.html`

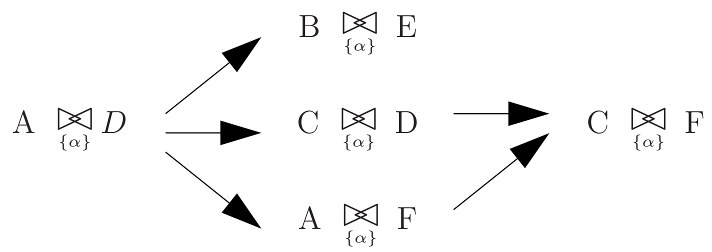
Appendix A. PEPA Synchronization

Let $A \stackrel{def}{=} (\alpha, r_1).B + (\beta, r_2).C,\ D \stackrel{def}{=} (\alpha, r_1).E + (\beta, r_3).F$ and $A \underset{L}{\bowtie} D$ where the set $L = \{\{\}, \{\alpha\}, \{\beta\}, \{\alpha, \beta\}\}$ for each of the potential synchronization sets the following Markov Processes are produced.

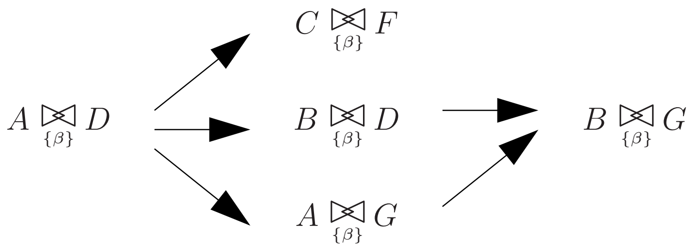
$$A \underset{\{\}}{\bowtie} D \text{ or } A||D$$



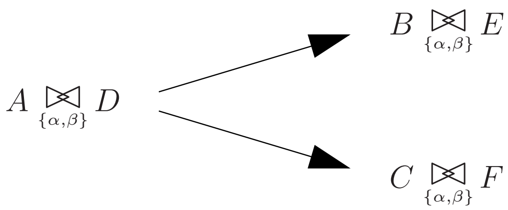
$$A \underset{\{\alpha\}}{\bowtie} D$$



$$A \underset{\{\beta\}}{\bowtie} D$$



$$A \underset{\{\alpha,\beta\}}{\bowtie} D$$



Appendix B. Client-Server Model

The models shown here and in Appendix C are abridged versions where similar states/declarations have been removed along with any superfluous bracketing within the expressions. Any alterations have been to allow readability. The PEPA models were generated by means of a python script and the ISBJava (cdml format) by the conversion process.

B.1 Client-Server as reagent-centric model in PEPA

```
source = 1000;
total_Clients = 1;
initiate_rate = (theta*(32-total_Clients));
download_rate = 0.5 + (theta*(17-total_Clients))*0.5;

Source_H = (initiate,initiate_rate).Source_L;

Client00_H = (download01,download_rate).Client00_L;
Client00_L = (initiate,initiate_rate).Client00_H;

Client01_H = (download02,download_rate).Client01_L;
Client01_L = (download01,download_rate).Client01_H;
.
.
Client99_H = (download100,download_rate).Client99_L;
Client99_L = (download99,download_rate).Client99_H;

Total_Clients_H = (download100,download_rate).Total_Clients_L;
Total_Clients_L = (initiate,initiate_rate).Total_Clients_H;

Completed_L = (download100,download_rate).Completed_H;
```

B.2 Client-Server as machine generated cdml

```
// Species populations

Client00 = 0.0;
.
.
Client99 = 0.0;
Completed = 0.0;
Source = 1000.0;
Total_Clients = 1.0;

// Rates

download_rate = [0.5 + theta(17.0-Total_Clients)*0.5];
initiate_rate = [theta(32.0-Total_Clients)];

// Reactions

download01, Client00 -> Client01, [download_rate*Client00];
.
.
download99, Client98 -> Client99, [download_rate*Client98];
download100, Client99 + Total_Clients -> Completed, [download_rate*(min(Client99, Total_Clients))];

initiate, Source -> Client00 + Total_Clients, [initiate_rate*Source];
```

Appendix C. BitTorrent Model

C.1 BitTorrent as reagent-centric model in PEPA

```
source = 1000;
seed = 1;
upload_Pool = 4;
torrent_rate = 0.7/(source+1);
connect_rate = (theta*(upload_Pool-3))*1;
downloaded_rate = 1;
online_rate = 0.01;
offline_rate = 0.05;
seed_online_rate = 0.01;
seed_offline_rate = (theta*(seed-1))*0.05;
seed_quit_rate = (theta*(seed-1))*0.1;
total_Online_Peers = online_Peer00 + online_Peer01 +..+ online_Peer99 +
                     downloading_Peer00 +..+ downloading_Peer09;
total_Downloading_Peers = downloading_Peer00 +..+ downloading_Peer99;
total_Offline_Peers = offline_Peer00 +..+ offline_Peer99;
total_Peers = total_Online_Peers + total_Offline_Peers;
Source_H = (torrent,torrent_rate).Source_L;

Online_Peer00_H = (connect00,connect_rate).Online_Peer00_L + (offline00,offline_rate).Online_Peer00_L;
Online_Peer00_L = (torrent,torrent_rate).Online_Peer00_H + (online00,online_rate).Online_Peer00_H;

Online_Peer01_H = (connect01,connect_rate).Online_Peer01_L + (offline01,offline_rate).Online_Peer01_L;
Online_Peer01_L = (downloaded01,downloaded_rate).Online_Peer01_H + (online01,online_rate).Online_Peer01_H;
.
.
Online_Peer99_H = (connect99,connect_rate).Online_Peer99_L + (offline99,offline_rate).Online_Peer99_L;
Online_Peer99_L = (downloaded99,downloaded_rate).Online_Peer99_H + (online99,online_rate).Online_Peer99_H;

Seed_H = (seed_quit,seed_quit_rate).Seed_L + (seed_offline,seed_offline_rate).Seed_L;
Seed_L = (downloaded100,downloaded_rate).Seed_H + (seed_online,seed_online_rate).Seed_H;

Offline_Peer00_H = (online00,online_rate).Offline_Peer00_L;
Offline_Peer00_L = (offline00,offline_rate).Offline_Peer00_H;
.
.
Offline_Peer99_H = (online99,online_rate).Offline_Peer99_L;
Offline_Peer99_L = (offline99,offline_rate).Offline_Peer99_H;

Offline_Seed_H = (seed_online,seed_online_rate).Offline_Seed_L;
Offline_Seed_L = (seed_offline,seed_offline_rate).Offline_Seed_H;

Downloading_Peer00_H = (downloaded01,downloaded_rate).Downloading_Peer00_L;
Downloading_Peer00_L = (connect00,connect_rate).Downloading_Peer00_H;
.
.
Downloading_Peer99_H = (downloaded100,downloaded_rate).Downloading_Peer99_L;
Downloading_Peer99_L = (connect99,connect_rate).Downloading_Peer99_H;

Completed_L = (downloaded100,downloaded_rate).Completed_H;
```

```
Upload_Pool_H = (connect00,connect_rate).Upload_Pool_L +..+ (connect99,connect_rate).Upload_Pool_L +
                (offline01,offline_rate).Upload_Pool_L +..+ (offline99,offline_rate).Upload_Pool_L +
                (seed_offline,seed_offline_rate).Upload_Pool_L +
                (seed_quit,seed_quit_rate).Upload_Pool_L;
Upload_Pool_L = (downloaded01,downloaded_rate).Upload_Pool_H +..+ (downloaded100,downloaded_rate).Upload_Pool_H +
                (online01,online_rate).Upload_Pool_H +..+ (online99,online_rate).Upload_Pool_H +
                (seed_online,seed_online_rate).Upload_Pool_H;
```

C.2 BitTorrent as machine generated cdml

```
// Species populations

Online_Peer00 = 0.0;
.
.
Online_Peer99 = 0.0;
Offline_Peer00 = 0.0;
.
.
Offline_Peer99 = 0.0;
Offline_Seed = 0.0;
Downloading_Peer00 = 0.0;
.
.
Downloading_Peer99 = 0.0;
Completed = 0.0;
Source = 1000.0;
Seed = 1.0;
Upload_Pool = 4.0;

// Rates

torrent_rate = [0.7/(Source+(1.0))];
offline_rate = 0.05;
total_Downloading_Peers = [Downloading_Peer00 +..+ Downloading_Peer99];
seed_quit_rate = [theta(Seed-1.0)*0.1];
connect_rate = [(theta(Upload_Pool-(3.0)))*(1.0)];
online_rate = 0.01;
total_Offline_Peers = [Offline_Peer00 +..+ Offline_Peer99];
seed_offline_rate = [theta(Seed-1.0)*0.05];
seed_online_rate = 0.01;
downloaded_rate = 1.0;
total_Online_Peers = [Online_Peer00 +..+ Online_Peer99 + Downloading_Peer00 +..+ Downloading_Peer99];
total_Peers = [total_Online_Peers+total_Offline_Peers];

// Reactions

connect00, Online_Peer00 + Upload_Pool x 4 -> Downloading_Peer00, [connect_rate*(min(Online_Peer00, Upload_Pool))];
.
.
connect99, Online_Peer99 + Upload_Pool x 4 -> Downloading_Peer99, [connect_rate*(min(Online_Peer99, Upload_Pool))];
downloaded01, Downloading_Peer00 -> Online_Peer01 + Upload_Pool x 5, [downloaded_rate*Downloading_Peer00];
downloaded02, Downloading_Peer01 -> Online_Peer02 + Upload_Pool x 4, [downloaded_rate*Downloading_Peer01];
.
.
downloaded100, Downloading_Peer99 -> Seed + Completed + Upload_Pool x 4, [downloaded_rate*Downloading_Peer99];
offline00, Online_Peer00 -> Offline_Peer00, [offline_rate*Online_Peer00];
offline01, Online_Peer01 + Upload_Pool -> Offline_Peer00, [offline_rate*(min(Online_Peer00, Upload_Pool))];
.
.
offline99, Online_Peer99 + Upload_Pool -> Offline_Peer99, [offline_rate*(min(Online_Peer99, Upload_Pool))];
online00, Offline_Peer00 -> Online_Peer00, [online_rate*Offline_Peer00];
online01, Offline_Peer01 -> Online_Peer01 + Upload_Pool, [online_rate*Offline_Peer00];
.
.
online99, Offline_Peer99 -> Online_Peer99 + Upload_Pool, [online_rate*Offline_Peer99];
seed_offline, Seed + Upload_Pool -> Offline_Seed, [seed_offline_rate*(min(Seed, Upload_Pool))];
seed_online, Offline_Seed -> Seed + Upload_Pool, [seed_online_rate*Offline_Seed];
seed_quit, Seed + Upload_Pool -> , [seed_quit_rate*(min(Seed, Upload_Pool))];
torrent, Source -> Online_Peer00, [torrent_rate*Source];
```