Study on OSPF Algebraic Formal Modelling Using ACP

Pedro Juan Roig[1,2], Salvador Alcaraz[1], Katja Gilly[1], Carlos Juiz[2]
[1]*Department of Physics and Computer Architecture, Miguel Hernandez University, Avda. Universidad, s/n - 03202 Elche (Alicante), Spain*
[2]*Department of Computer Science, Balearic Islands University, Ctra. Valldemossa, km 7.5 - 07122 Palma de Mallorca, Spain*
pedro.roig@graduado.umh.es

***Abstract*—OSPF is a commonly used link state routing protocol in order to interconnect network devices inside an Autonomous System. This paper focuses on achieving a detailed modelling for OSPF by performing manual algebraic derivations according to Algebra of Communicating Processes (ACP) axioms. The aim of this detailed model is to get a realistic approach of OSPF dynamics by applying the timers involved and by describing the full Link State Advertisement (LSA) exchange process.**

***Index Terms*—ACP; Formal protocol specification; Networking; OSPF.**

I. INTRODUCTION

OSPF [1] is an Interior Gateway Protocol (IGP) which may most likely be the most widespread routing protocol nowadays, mainly due to its fast converging time and its easiness of configuration.

The aim of this paper is to get an algebraic formal model for OSPF, that meaning a formal protocol specification with mathematical notation. Formal Description Techniques (FDT) [2] are largely used for modelling protocols in order to check for design flaws and to facilitate automated protocol implementation.

Most FDT are implemented by means of software tools such as mCRL2 [3] or Spin [4], but in order to get algebraic formal models, the most convenient tools are process algebras, which allow to manually reason about processes following a set of axioms.

There are some kinds of process algebras [5], but the one fitting better for algebraic protocol modelling may be Algebra of Communicating Processes (ACP) [6]. That is because ACP is the most abstract of all its counterparts [7], as it does not take into account the real nature of processes, hence allowing to just focus on their core properties [8].

Therefore, ACP is really suitable for the implementation of algebraic formal modelling, as it facilitates the study of concurrent and distributed systems [9], such as routing protocols being independently run by each particular router within certain routing domain. In addition to that, ACP may also provide algebraic tools for its further verification [10].

There is not much information concerning this area in the literature, therefore the design of a formal model of OSPF by using ACP has been undertaken, where algebraic derivations were manually performed so as to prove the correctness of the final outcome.

A routing protocol performs three basic functions, being identifying their neighbours, managing the route paths to all destinations and making dynamic decisions about where to forward user traffic getting in.

A detailed model has been designed with the premise of making it as close to real as possible, hence considering the real LSA exchange process and taking into account timers involved in sending and receiving both OSPF hello packets and LSA refresh packets.

In this paper, we focus on studying that model with different design parameters, such as the different network types available for a network segment and the number of routers within the OSPF domain.

The organisation of this paper will be as follows: first, Section II introduces the OSPF network types, then, Section III describes the kind of relationships between routers within an OSPF domain, next, Section IV presents the OSPF detailed model, after that, Section V renders a practical example of that model, later, Section VI performs a verification of that model, and finally, Section VII draws the final conclusions.

II. OSPF NETWORK TYPES

Prior to working on the expression for the OSPF models, some considerations may be done regarding the network type. The OSPF v2 standard presents four kinds of networks to be distinguished, each one with its own characteristics, as it may be seen in Table I regarding network type and in Table II with hello and dead timers.

Each network type has its own independent $TYPE_{id}$ value, but they might be associated in two different ways, either according to the binary values of the variable *NT*, standing for Network Type, or otherwise regarding the binary values of the variable *TT*, related to Timer Type.

On the one hand, two network types share the need for categorising OSPF routers within a network segment as Designated Router (DR), Backup DR (BDR) or none of them (DROther), whereas the other two of them do not.

Manuscript received 30 December, 2017; accepted 28 April, 2018.

TABLE I. NETWORK TYPES.

Network Type	Nomenclature	$TYPE_{id}$	NTvalue
Broadcast	BRC	T = 0	NT = 0
Point to Point	P2P	T = 1	NT = 1
Non Broacast MultiAccess	NBMA	T = 2	NT = 0
Point to MultiPoint	P2MP	T = 3	NT = 1

TABLE II. HELLO AND DEAD TIMERS.

Network Type	$TYPE_{id}$	NT value	TT value	Hello Timer	Dead Timer
BRC	T = 0	**NT = 0**	**TT = 0**	10 s.	40 s.
P2P	T = 1	**NT = 1**	**TT = 0**	10 s.	40 s.
NBMA	T = 2	**NT = 0**	**TT = 1**	30 s.	120 s.
P2MP	T = 3	**NT = 1**	**TT = 1**	30 s.	120 s.

As a matter of fact, network types with the value of 0 for the variable *NT* need to implement a DR and also a BDR as they are multiaccess networks, that meaning there might be more than two routers within a network segment. Those network types are Broadcast (BRC), which are the Ethernet networks, and Non Broadcast MultiAccess (NBMA), which are the Frame Relay and ATM networks with Meshed topology.

In such cases, DR is a router appointed to collect link-state information from all devices within that network segment, and in turn redistribute it properly, in order for all those devices to share the same link-state advertisements. Additionally, BDR is another router elected to collect link-state information along with DR just for backup purposes.

Alternatively, network types with *NT* value of 1 do not need to implement a DR and a BDR as they are peer to peer networks, that meaning there are just two routers within a network segment. Those network types are Point to Point (P2P), which are the Serial networks such as PPP or HDLC, and Point to MultiPoint (P2MP), which are Frame Relay or ATM networks with a Hub and Spoke topology.

However, on the other hand, another different group of two network types share the same values for the hello timers and dead timers whilst the other two of them do not.

Network types having broadcast capabilities like Ethernet or Serial point to point links share a hello timer of 10 seconds and a dead timer fourfold. This fact is reflected with the value of 0 for the variable *TT*.

On the contrary, network types with *TT* value of 1 are related to Frame Relay or ATM implementations, which do not have inherent broadcast or multicast capabilities, although this fact might be worked around through the proper configuration. They share a hello timer of 30 seconds and a dead timer four times greater.

Regarding arithmetic, the variable $TYPE_{id}$ defined in Table I takes values from 0 to 3 depending on the particular network type, but variables *NT* and *TT* may be calculated by means of the following expressions:

$$NT = (TYPE_{id}) \bmod 2, \tag{1}$$

$$TT = (TYPE_{id}) div 2. \tag{2}$$

Therefore, *NT* might be seen as the remainder of the integer division of $TYPE_{id}$ by 2, whereas *TT* might be considered as the quotient, or the integer part, of that same division.

III. OSPF NEIGHBOURHOOD – VS – OSPF ADJACENCY

It is crucial to understand both sorts of relationship present in OSPF domains. Some OSPF devices are neighbours if they are connected to the same subnet and share some configuration information such as subnet mask, authentication type, area ID and its type, hello and dead timers.

All OSPF neighbours will keep their neighbourhood relationship by means of exchanging hello packets but this fact does not imply an exchange of LSA, hence no routing information may not necessarily flow. On the contrary, some OSPF devices are adjacent if they do share LSA among them, so routing information does flow.

The requirements for a pair of OSPF neighbouring routers to be adjacent depend on the network type. This is, multiaccess networks such as BRC or NBMA need that, at least, one of them bears the role of DR or BDR, but otherwise, peer to peer networks such as P2P and P2MP need not.

In summary, an OSPF adjacency relationship will only be formed if Table III gives 1 between a particular sender, shown in rows, and a given receiver, shown in columns. This table is set up just for multiaccess networks, whilst peer to peer networks always form adjacency relationship between ends. Therefore, this table might also apply for the latter case, just considering that both ends are DR and BDR.

TABLE III. OSPF ADJACENCY RELATIONSHIP.

	r_{DR}	r_{BDR}	r_{DRO1}	r_{DRO2}	r_{DRO3}	r_{DRO4}
s_{DR}	-	1	1	1	1	1
s_{BDR}	1	-	0	0	0	0
s_{DRO1}	1	1	-	0	0	0
s_{DRO2}	1	1	0	-	0	0
s_{DRO3}	1	1	0	0	-	0
s_{DRO4}	1	1	0	0	0	-

The aforesaid table may be implemented as an arithmetic expression just by assigning the appropriate values to the different router roles present in each OSPF network types, as shown in Table IV.

TABLE IV. VALUES ASSIGNED TO ROUTER ROLES.

Network Type	Router Role	Naming	Value
NT = 0	Designated Router	DR	3
NT = 0	Backup Designated Router	BDR	2
NT = 0	DR Other	DRO	1
NT = 1	Point to Point Link Routers	-	0

Those values are assigned in a unidirectional fashion, hence each way is evaluated on its own. Therefore, for each direction within a channel, the aforesaid values are combined in the following expression, containing the addition of the integer part of a fraction with the role of the routers at both ends of a channel, where *x* represents the role of the sending end of the link and *y* is the role of the receiving end, and also containing the network type

$$k_{i,j} = \operatorname{int}\left(\frac{x + 2 \times y}{5}\right) + NT. \tag{3}$$

In a nutshell, coefficients $k_{i,j}$ capture the behaviour seen in

Table III. Those coefficients carry binary values, thus being 1 or 0, depending on whether local router *i* and neighbouring router *j* form an OSPF adjacency relationship or otherwise.

IV. OSPF DETAILED MODELLING USING ACP

In order to achieve a detailed modelling for OSPF, the main requirements to be met are the use of the proper timers and the step by step LSA exchange process.

As per the action of timers, *hello and dead timers* depend on the network type and they are given in Table II. However, just as a reminder, variable *NT*, as stated in (1), is not to be confused with variable *TT*, as stated in (2). The former identifies whether a network type will need to appoint a DR and a BDR or otherwise, whereas the latter identifies the value of the aforesaid timers.

Both timers are related to the neighbour discovery function but they are applied to different actions, as hello timers state how long a router must wait for sending a hello packet to a neighbour router, thus considering it as up, whereas dead timers state how long a router must wait for receiving a dead packet from a neighbour router prior to considering it as down.

The same router might have different network types associated to its various interfaces, therefore, the aforesaid variables, and hence their related timers, will depend on the media that each particular neighbouring router is standing.

There are another couple of timers to be taken into consideration, such as the *refresh LSA timer*, whose default value is 30 minutes, thus 1800 seconds, although it might be adjusted from 5 minutes to 59 minutes, and the *max age LSA timer*, whose value is 1 hour, thus 3600 seconds. Those timers do not depend on the network type; therefore, they will be invariant with respect of it.

Those timers are related to the refreshment of LSA entries already present on the Link State Data Base (LSDB) but they are applied to different actions, as the refresh LSA timer states how long a router must wait for sending an LSA refreshment update packet to a neighbour, hence reoriginating that LSA and forwarding on to an adjacent router, whereas max age LSA timer states how long a router must wait for receiving an LSA refreshment update packet from an adjacent router prior to removing that LSA from its LSDB.

On the other hand, it is worth taking into account that there are five different OSPF types of packets, whose functions are described in Table V.

Therefore, hello messages will be carried within OSPF type 1 packets whereas LSA exchange process will involve the rest of OSPF packet types.

As per the actual LSA exchange process, a local router *i* sends an OSPF type 2 packet to its adjacent routers *j* in order to get its LSDB synchronised with those of its adjacent routers. They will, in turn, acknowledge such packets by echoing it to the local router *i*, whilst evaluating whether each LSA header is more updated than those present onto their own LSDB.

TABLE V. OSPF PACKET TYPES.

Type	Packet Name	Function
1	Hello	Discovering and maintaining neighbors
2	Database Description	Exchanging Data Base LSA headers
3	Link State Request	Requesting LSA
4	Link State Update	Sending a requested LSA
5	Link State ACK	Sending Acknowledgements upon receiving LSA

If that is the case for any LSA header, an OSPF type 3 packet will be sent to the local router *i* in order to request the whole LSA from it, and upon receipt of it, an OSPF type 4 packet will be sent back from the local router *i* containing the LSA requested, which will in turn be acknowledged by sending an OSPF type 5 packet to the local router *i*.

Needless to say that the proper neighbouring router will be performing the complementary actions of those exposed, thus it will be receiving packets when the local router is sending them and vice versa.

In summary, terms depending on timers may be considered as synchronous, whereas terms related to LSA exchange process may be considered as asynchronous as they do not depend on time.

The synchronous terms will be called $R_{s,i}$ and they will need the implementation of a timing system, which will decrement the aforesaid timers and will reset to their default values when the proper actions may be performed.

On the contrary, the asynchronous terms will be called $R_{a,i}$ and they will be time independent, so they will not need any timing system.

All those terms may be described using ACP syntax and semantics [11], where a product stands for a sequential operator, an addition stands for an alternate operator and a conditional operator is used such as True < |Boolean| > False.

Additionally, $s_{i,j}(d)$ stands for sending a packet from *i* to *j*, whereas $r_{i,j}(d)$ stands for receiving that packet at *j* coming from *i*, and $c_{i,j}(d)$ stands for communication between them.

All of that is extended with the deployment of an appropriate timing system and some auxiliary functions to be defined later on. The different OSPF packet types will be expressed herein as PT followed by their type number (*PTt*):

$$R_{s,i} = Time(i) \times \sum_{j \neq i} \left\{ \begin{array}{l} \left(s_{i,j}(PT1) \times resetH(i) \left\langle | \, t_{hello,i} = 0 | \right\rangle \right) + \left(r_{j,i}(PT1) \times resetD(i) \left\langle | \, t_{dead,i} > 0 | \right\rangle kill(j) \right) + \\ + \left(k_{i,j} \times s_{i,j}(PT4) \times resetR(i) \left\langle | \, t_{refresh,i} = 0 | \right\rangle \right) + \left(\begin{array}{l} k_{j,i} \times r_{j,i}(PT4) \times resetM(i) \times refresh(i) \\ \left\langle | \, t_{\max Age,i} > 0 | \right\rangle remove(i) \end{array} \right) \end{array} \right\}, \quad (4)$$

$$R_{a,i} = \sum_{j \neq i} \left\{ \begin{array}{l} k_{i,j} \times s_{i,j}(PT2) \times r_{j,i}(PT2) + k_{j,i} \times r_{j,i}(PT2) \times s_{i,j}(PT2) \times \left\{ \begin{array}{l} s_{i,j}(PT3) \times r_{j,i}(PT4) \times \\ \times s_{i,j}(PT5) \left\langle | update(i) = 1 | \right\rangle \end{array} \right\} + \\ + k_{j,i} \times r_{j,i}(PT3) \times s_{i,j}(PT4) \times r_{j,i}(PT5) \end{array} \right\}. \quad (5)$$

It is to be noted that when a new route is first incorporated into the OSPF domain or any route gets updated, the LSA exchange process is asynchronous, as it gets triggered as soon as that network gets configured into the OSPF domain, regardless of the timing. However, when it comes to refreshing LSA, the process is synchronous as it depends on LSA refresh timer, as well as when any route gets deleted, it depends on max age LSA timer.

As per the timing system, it may be modelled in diverse ways, but it will herein be done by defining a variable t_i, representing the total time in seconds that local router *i* has been alive within the OSPF domain.

In addition to it, the aforesaid four timers have also been defined, going down from the maximum values previously proposed for each one of them, as stated in (6)–(9):

$$T_{hello\_MAX,i} = 10 + 20 \times TT, \tag{6}$$

$$T_{dead\_MAX,i} = 4 \times (10 + 20 \times TT), \tag{7}$$

$$T_{refreshLSA\_MAX,i} = 1800, \tag{8}$$

$$T_{\max AgeLSA\_MAX,i} = 3600. \tag{9}$$

Those four timers may be decreased one unit at a time, meaning a second has elapsed, hence simulating the effect of time passing by. That may be modelled in the local router *i* by Algorithm 1, called *time (i)*.

After executing this function, it returns the value 1 in order for the synchronous terms $R_{s,i}$ to be executed, simulating the need of a positive clock edge for running those terms.

Algorithm 1. time (i):

```
time(i){
 thello,i = thello,i – 1;
 tdead,i = tdead,i – 1;
 trefreshLSA,i = trefreshLSA,i – 1;
 tmaxAgeLSA,i = tmaxAgeLSA,i -1;
 ti = ti + 1;
 return 1;
}
```

Besides, *resetX (i)* functions reassign the maximum value to a timer, where *X* represents the initial letter of the timer involved, as it is shown in Algorithms 2-5.

Algorithm 2. resetH (i):

```
resetH(i){
 thello,i = Thello_MAX;
}
```

Algorithm 3. resetD (i):

```
resetD(i){
 tdead,i = Tdead_MAX;
}
```

Algorithm 4. resetR (i):

```
resetR(i){
 trefreshLSA,i = TrefreshLSA_MAX;
}
```

Algorithm 5. resetM (i):

```
resetM(i){
 tmaxAgeLSA,i = TmaxAgeLSA_MAX;
}
```

Furthermore, on the one hand, the *init(i)* function represents a router *i* joining the OSPF domain, which is represented by the variable $t_i = 0$ and so is the *hello timer* so as to begin the neighbour discovery straight away, whilst the rest of the timers are set to infinite as they are useless at that point of time. All of that might be seen in Algorithm 6.

Algorithm 6. init (i):

```
init(i){
 thello,i = 0;
 tdead,i = ∞;
 trefreshLSA,i = ∞;
 tmaxAgeLSA,i = ∞;
 ti = 0;
}
```

As per the hello timer, a router *i* will be constantly sending hello packets on a regular basis through all of its OSPF interfaces every time the hello timer reaches zero value. Right after that, the *resetH (i)* function will be resetting it to its highest value and the *time (i)* function will be decreasing it at each simulated second until it reaches zero value the next time, repeating this cycle over and over again as long as router *i* remains into the OSPF domain.

Later on, when a router *i* starts receiving hello packets, its dead timer will be set to its maximum value, as a new neighbour relationship has been established. Therefore, that *dead timer* will start playing its part, increasing its value according to the *resetD (i)* function and decreasing it in line with the *time (i)* function, analogously as exposed for the hello timer. This cycle will go on over and over again until the router leaves the OSPF domain.

Also, when a router *i* establishes a new adjacency relationship, it will start the process of sending and receiving LSA from its adjacent routers. At that point, the *refresh LSA timer* and the *max age LSA timer* will start playing their part when sending and receiving LSA, respectively. Both timers will be decreased by the *time (i)* function whereas the former will be increased to its maximum value by the *resetR (i)* function and the latter will be done by the *resetM (i)* function.

On the other hand, the *kill (i)* function represents a router *i* leaving the OSPF domain, which is represented by variable $t_i = \infty$ and so are the rest of the timers, effectively blocking them all. This might be seen in Algorithm 7.

Algorithm 7. kill (i)

```
kill(i){
 thello,i = ∞;
 tdead,i = ∞;
 trefreshLSA,i = ∞;
 tmaxAgeLSA,i = ∞;
 ti = ∞;
}
```

Regarding LSA exchange process, an *update (i)* function has been added up in the $R_{a,i}$ term in order to check whether each LSA header received by a local router *i* within an OSPF type 2 packet, also known as Description Data Base (DBD) packet, is more up to date than that of the corresponding LSA stored on its own LSDB copy.

For each incoming LSA headers meeting this condition, the local router *i* will be requesting the full LSA from the proper DBD sender in order to perform the LSA update out of its local LSDB. Furthermore, if the LSA is brand new, it does demand it as well. That might be seen in Algorithm 8.

For the purpose of keeping the *update (i)* function as simple as possible, both the incoming LSA header and the already existing full LSA within the local LSDB are solely identified by its LS ID field and the most up to date instance

is considered the one with a higher sequence number.

Algorithm 8. update (i)

```
update(I, DBD) {
 For Each LSAh in DBD
 Do
  If (DBD.LSAh.LSID == i.LSDB.LSA.LSID)
  Then
   If (DBD.LSAh.seqNo > i.LSDB.LSA.seqNo)
   Then
    Return 1;
   Else
    Return 0;
   EndIf
  Else
   Return 1;
  EndIf
 Done
}
```

This function has a return value in order to visualise whether that LSA update has to be performed. This is done by returning 1 if this is the case and also if the incoming LSA header does not exist within LSDB, or otherwise by returning 0 in any other case. Therefore, if the returning value is 1, then the process for getting the full updated LSA will be undertaken.

Additionally, another function is also used in order to perform the refreshment of LSA within LSDB upon receipt of a reoriginated LSA before its max age LSA timer gets to zero, as it happens in the $R_{s,i}$ term, hence avoiding the flushing of that LSA if it reaches an age of 3600 seconds.

That task will be performed by the *refresh (i)* function, which will be analogous to the *update (i)* function, but with two important differences. To start with, in the asynchronous case, the DBD argument will be considered as all of the LSA summaries received in router *i*, whereas in the synchronous case, the LSU argument will be considered as all of the set of full LSA received within an OSPF type 4 packet, also known as Link State Update (LSU), in router *i*. Additionally, this function will not have any returning value. This algorithm might be seen in Algorithm 9.

Algorithm 9. refresh (i)

```
refresh(i,LSU){
 For Each LSA in LSU
 Do
  If (LSU.LSA.LSID == i.LSDB.LSA.LSID)
  Then
   If (LSU.LSA.seqNo > i.LSDB.LSA.seqNo)
   Then
    i.LSDB.LSA = LSU.LSA;
   Else
    ;
   EndIf
  Else
   i.LSDB.LSA = LSU.LSA;
  EndIf
 Done
}
```

Apart from that, the *remove (i)* function has been added up in the $R_{s,i}$ term in order to remove those LSA from the local LSDB whose maximum aging time has been reached without them being refreshed, hence they are flushed from LSDB. That algorithm might be seen in Algorithm 10.

Algorithm 10. remove (i)

```
remove(i,FLUSH){
 For Each LSA in FLUSH
 Do
  i.LSDB.LSA = NULL;
 Done
}
```

In summary, the OSPF detailed modelling using ACP syntax and semantics for a router *i* will include both asynchronous and synchronous terms described above. Therefore,

$$R(i) = \left(R_{s,i} + R_{a,i}\right)\left\langle |t_i > 0| \right\rangle init(i). \tag{10}$$

V. PRACTICAL EXAMPLE OF OSPF MODELLING

In order to prove (10), we are going to render an example so as to show that the detailed router model for OSPF using ACP mirrors the real OSPF behavior.

This example will represent a P2P network, so n = 2 and NT = 1, although the results will be the same if n = 2 and NT = 0:

$$R_1 = time(R_1)\times$$
$$\times\left\{\begin{array}{l}\left(s_{1,2}(PT1)\times resetH(R_1)\left\langle |t_{hello,1}=0| \right\rangle\right)+\left(r_{2,1}(PT1)\times resetD(R_1)\left\langle |t_{dead,1}>0| \right\rangle kill(R_2)\right)+\\ +\left(s_{1,2}(PT4)\times resetR(R_1)\left\langle |t_{refresh,1}=0| \right\rangle\right)+\left(\begin{array}{l} r_{2,1}(PT4)\times resetM(R_1)\cdot refresh(R_1)\\ \left\langle |t_{\max Age,1}>0| \right\rangle remove(R_1)\end{array}\right)\end{array}\right\}+$$
$$+\left\{\begin{array}{l} s_{1,2}(PT2)\times r_{2,1}(PT2)+r_{2,1}(PT2)\times s_{1,2}(PT2)\times\left\{\begin{array}{l} s_{1,2}(PT3)\times r_{2,1}(PT4)\times s_{1,2}(PT5)\\ \left\langle |update(R_1)=1| \right\rangle\end{array}\right\}+\\ +r_{2,1}(PT3)\times s_{1,2}(PT4)\times r_{2,1}(PT5)\end{array}\right\},$$
$$\left\langle |t_{R1}>0| \right\rangle init(R_1), \tag{10}$$

$$R_2 = time(R_2)\times$$
$$\times\left\{\begin{array}{l}\left(s_{2,1}(PT1)\times resetH(R_2)\left\langle |t_{hello,2}=0| \right\rangle\right)+\left(r_{1,2}(PT1)\times resetD(R_2)\left\langle |t_{dead,2}>0| \right\rangle kill(R_1)\right)+\\ +\left(s_{2,1}(PT4)\times resetR(R_2)\left\langle |t_{refresh,2}=0| \right\rangle\right)+\left(\begin{array}{l} r_{1,2}(PT4)\times resetM(R_2)\times refresh(R_2)\\ \left\langle |t_{\max Age,2}>0| \right\rangle remove(R_2)\end{array}\right)\end{array}\right\}+$$

$$+\left\{\begin{array}{l} s_{2,1}(PT2)\times r_{1,2}(PT2)+r_{1,2}(PT2)\times s_{2,1}(PT2)\times \\ \times\left\{\begin{array}{l} s_{2,1}(PT3)\times r_{1,2}(PT4)\times s_{2,1}(PT5) \\ \langle|\, update(R_2)=1\,|\rangle \end{array}\right\}+r_{1,2}(PT3)\times s_{2,1}(PT4)\times r_{1,2}(PT5) \end{array}\right\},$$

$$\langle|\, t_{R2}>0\,|\rangle init(R_2), \tag{11}$$

$$\partial_H(R_1 \,||\, R_2)=time(t)\times$$

$$\times\left\{\begin{array}{l} \left(\begin{array}{l} c_{1,2}(PT1)\times resetD(R_2) \\ \langle|\, t_{dead,2}>0\,|\rangle kill(R_1) \end{array}\right)+\left(\begin{array}{l} c_{2,1}(PT1)\times resetD(R_1) \\ \langle|\, t_{dead,1}>0\,|\rangle kill(R_2) \end{array}\right)+ \\ +\left(\begin{array}{l} c_{1,2}(PT4)\times resetM(R_2)\times refresh(R_2) \\ \langle|\, t_{\max Age,2}>0\,|\rangle remove(R_2) \end{array}\right)+\left(\begin{array}{l} c_{2,1}(PT4)\times resetM(R_1)\times refresh(R_1) \\ \langle|\, t_{\max Age,1}>0\,|\rangle remove(R_1) \end{array}\right) \end{array}\right\}+$$

$$+\left\{c_{1,2}(PT2)\times c_{2,1}(PT2)\times\left\{\begin{array}{l} c_{2,1}(PT3)\times c_{1,2}(PT4)\times c_{2,1}(PT5) \\ \langle|\, update(R_2)=1\,|\rangle \end{array}\right\}\right\}+$$

$$+\left\{c_{2,1}(PT2)\times c_{1,2}(PT2)\times\left\{\begin{array}{l} c_{1,2}(PT3)\times c_{2,1}(PT4)\times c_{1,2}(PT5) \\ \langle|\, update(R_1)=1\,|\rangle \end{array}\right\}\right\}. \tag{12}$$

Simplifying the former expression by taking out the effect of algorithms and timers, it yields the following equation, which proves that all communications takes place as expected according to the OSPF v2 standard explained in this paper, either synchronous as in the first curly brackets or asynchronous as in the second ones

$$\partial_H(R_1 \,||\, R_2)=$$
$$\left\{c_{1,2}(PT1)+c_{2,1}(PT1)+c_{1,2}(PT4)+c_{2,1}(PT4)\right\}+$$
$$+\left\{\begin{array}{l} c_{1,2}(PT2)\times c_{2,1}(PT2)\times \\ \times\left\{c_{2,1}(PT3)\times c_{1,2}(PT4)\times c_{2,1}(PT5)\right\} \\ +c_{2,1}(PT2)\times c_{1,2}(PT2))\times \\ \times\left\{c_{1,2}(PT3)\times c_{2,1}(PT4)\times c_{1,2}(PT5)\right\} \end{array}\right\}. \tag{13}$$

VI. MODEL VERIFICATION

Taking into account that ACP is a sort of an abstract algebra, the verification of the model designed herein might be undertaken using proof by contradiction, in a way that if an initial proposition is stated and a logical contradiction arises whilst reasoning on that premise, then the aforesaid initial statement must be false, otherwise is true.

As stated before, the OSPF process relies on the exchange of type 1 packets for neighbor discovery and type 2 to 5 packets for LSA exchange, the former allowing neighbourhood relationships and the latter satisfying adjacency relationships.

Taking a look at the final expression at the end of the previous section, obtained for n = 2, although it might be extrapolated for any other number of routers by applying expressions (4) and (5), it is clear that exchanges of hello packets (PT1) among any pair of neighbouring routers are performed, as well as exchanges of new LSA (PT4) among any pair of adjacent routers, all of that according to the first pair of curly brackets. And by looking at the second pair of them, the synchronisation process of LSA is also performed. Hence the model designed mirrors the OSPF expected behaviour, related in the OSPFv2.

OSPF routing table depends on the correct LSA exchange. So if that process is performed properly, a couple of premises will apply, such as there will always be at least a path between any two routers located into the OSPF domain, as well as if a path goes down between a source and a destination and there is a redundant path to the same destination, then that other path may be used to get there.

According to the model, the first one is met because neighbour relationships will be established for each pair of neighbours within the OSPF domain, due to the exchange of PT1 packets. Additionally, the second one requires for the model to discover all paths from one source to one destination, and that is performed by establishing adjacency relationships for each pair of adjacent routers within the OSPF domain, due to the application of the $k_{i,j}$ coefficients and the exchange of PT2, PT3, PT4 and PT5 packets.

VII. CONCLUSIONS

In this paper, we have been working through the achievement of a formal detailed modelling of OSPF. For that purpose, ACP syntax and semantics have been used in order to perform manual algebraic derivations.

The main points taken into account to reach the aforesaid modelling have been the application of the timers involved in OSPF and the description of the LSA exchange process, as stated in the OSPF v2 standard.

A practical example has been proposed with a peer to peer network composed by 2 routers and after performing the proper algebraic derivations, its behaviour mirrors that of the real OSPF routing protocol.

That outcome may be replicated for any other kind of network with 2 routers, as an adjacency relationship will always be established, no matter the network type.

Eventually, that result might be extended for any type of multiaccess network with a higher number of routers, as the only difference will be adjacency relationships among them.

Therefore, the proposed ACP model for OSPF routing protocol meet the specifications of the standard OSPFv2.

REFERENCES

[1] *OSPF version 2 (RFC 2328)*, IETF, 1998. [Online]. Available: https://tools.ietf.org/html/rfc2328

[2] E. Gunawan, Tan Pek Tong, Shi Nansi, "Survey of formal description techniques (FDTs) for protocol converter design", in *Proc. IEEE Region 10 Int. Conf. Computers, Communications and Automation (TENCON 1993)*, Beijing, China, China, 1993, pp. 422–425. DOI: 10.1109/TENCON.1993.320017.

[3] M. H. ter Beek, E. P. de Vink, "Using mCRL2 for the analysis of software product lines", in *Proc. 2nd FME Workshop on Formal Methods in Software Engineering (FormaliSE 2014)*, Hyderabad, India, 2014, pp. 31–37. DOI: 10.1145/2593489.2593493.

[4] S. Chen, H. Fu, H. Miao, "Formal verification of security protocols using Spin", *IEEE/ACIS 15th Int. Conf. Computer and Information Science (ICIS 2016)*, Okayama, Japan, 2016, pp. 1–6. DOI: 10.1109/ICIS.2016.7550830.

[5] D. A. Padua, *Encyclopedia of Parallel Computing*. Springer, 2011. DOI: 10.1007/978-0-387-09766-4.

[6] [1] J. A. Bergstra, J. W. Klop, "Algebra of communicating processes with abstraction", *Theor. Comput. Sci*., vol. 37, pp. 77–121, 1985. DOI: 10.1016/0304-3975(85)90088-X.

[7] W. Fokkink, *Introduction to Process Algebra*. Springer, 2007.

[8] J. F. Groote, M. R. Mousavi, *Modelling and Analysis of Communicating Systems*. MIT Press, 2014.

[9] L. Lockefeer, D. M. Williams, W. J. Fokkink, "Formal specification and verification of TCP extended with the window scale option", *Formal Methods for Industrial Critical Systems, (FMICS 2014)*, 2014, pp. 63–77. DOI: 10.1007/978-3-319-10702-8_5.

[10] J. A. Bergstra, J. W. Klop, "Verification of an alternating bit protocol by means of process algebra protocol", *Mathematical Methods of Specification and Synthesis of Software Systems, (MMSSS 1985)*, 1986, pp. 9–23. DOI: 10.1007/3-540-16444-8_1.

[11] W. Fokkink, *Modelling Distributed Systems*. Springer, 2016.